GITHUB COPILOT AGENTIC FRAMEWORK

Hierarquia de configuração: como arquivos de config moldam seus agentes.

Cinco camadas em cascata, de copilot-instructions.md a prompt files. Um briefing técnico sobre como o GitHub Copilot monta o contexto, como funciona a precedência e como desenhar uma stack de configuração que o time inteiro herda.

AUTORA
Paula Silva

FUNÇÃO
Software Global Black Belt

DURAÇÃO
45 a 60 minutos

DATA
2026-06-10

AGENDA

Cinco partes, um modelo mental.

PARTE I	Fundamento, por que configuração importa e o modelo mental da cascata
PARTE II	As cinco camadas, arquivo por arquivo, com conteúdo real para copiar hoje
PARTE III	Resolução, o algoritmo que o GitHub Copilot executa e as regras de precedência
PARTE IV	Prática, a economia de tokens, a árvore completa do projeto, cenários enterprise e a lista do que fazer ou não
PARTE V	Memória, como o GitHub Copilot aprende o repo sozinho e quando graduar para arquivos

PARTE

I

O fundamento.

Assistentes de IA modernos leem uma hierarquia de arquivos para montar o contexto por trás de cada resposta.

POR QUE CONFIGURAÇÃO IMPORTA

Configuração decide o que a IA sabe, o que pode fazer e como se comporta.

DIMENSÃO 01 · CONHECIMENTO

O que a IA sabe

Instruções do projeto, convenções de código, padrões de arquitetura e os comandos-chave do seu repositório.

DIMENSÃO 02 · PERMISSÃO

O que a IA pode fazer

Ferramentas permitidas, limites de acesso e os guardrails de segurança que mantêm os agentes dentro das linhas.

DIMENSÃO 03 · COMPORTAMENTO

Como a IA se comporta

Regras por escopo, a personalidade de agentes especializados, tom e formato das respostas.

SEM A HIERARQUIA

Times acabam com instruções duplicadas, regras conflitantes, ou uma IA que ignora convenções críticas porque o arquivo certo nunca foi criado.

O MODELO MENTAL

Pense como CSS: uma cascata de especificidade.

CASCATA CSS

`body { }`
Estilo global, vale para tudo

`.container { }`
Nível de componente, escopo menor

`#hero { }`
O mais específico vence o conflito

CASCATA DE CONFIG

`copilot-instructions.md`
Contexto global do repositório

`.instructions.md`
Escopo por pasta ou padrão de arquivo

`*.agent.md · SKILL.md`
O mais específico complementa o resto

REGRA DE OURO

Camada filha ADICIONA, nunca contradiz. Se o arquivo global diz "use TypeScript", um arquivo de escopo não pode dizer "use JavaScript". A especificidade complementa, nunca sobrescreve.

PARTE

II

As cinco camadas.

Mergulho profundo em cada tipo de arquivo, com conteúdo real, localização real e a regra de ativação de cada um.

CAMADA 1 DE 5 · SEMPRE ATIVA

copilot-instructions.md é o cérebro global do repositório.

.GITHUB/COPILOT-INSTRUCTIONS.MD

```
# Project: E-Commerce Platform

## Tech Stack
- Next.js 14 with App Router
- TypeScript strict mode
- Prisma ORM with PostgreSQL

## Conventions
- kebab-case for file names
- PascalCase for React components
- All API routes in app/api/
- Tests: Vitest, run with npm test

## Architecture
- Feature-based folder structure
- Server components by default
```

LOCALIZAÇÃO

`.github/copilot-instructions.md`

ATIVAÇÃO

Automática em toda sessão do GitHub Copilot. Sempre ativa, sem frontmatter.

POR QUE IMPORTA

O arquivo mais impactante que você pode criar. Ele define o tom de toda interação com o seu projeto.

CAMADA 2 DE 5 · ESCOPO POR GLOB

`.instructions.md` aplica regras só onde o glob faz match.

LOCALIZAÇÃO

`.github/instructions/*.instructions.md`

ATIVAÇÃO

O frontmatter applyTo carrega um glob pattern. As regras só carregam quando um arquivo no contexto faz match.

USE QUANDO

Partes diferentes do projeto pedem regras diferentes: regras React para componentes, regras de API para rotas, regras de teste para specs.

● .GITHUB/INSTRUCTIONS/REACT-COMPONENTS.INSTRUCTIONS.MD

```
---
applyTo: "src/components/**/*.tsx"
---

# React Component Rules

- Functional components with hooks
- TypeScript interfaces for all props
- JSDoc comments on public props
- Tailwind for styling, no CSS modules
- Unit tests in __tests__/ folder
- Components must stay under 200 lines
```

CAMADA 3 DE 5 · ATIVA QUANDO INVOCADA

*.agent.md define personas especializadas com ferramentas próprias.

.GITHUB/AGENTS/SECURITY-REVIEWER.AGENT.MD

```
---
description: "Security code reviewer"
tools: [codeql, semgrep, gh-advanced-security]
---

# Security Reviewer Agent

You are a security code reviewer. Analyze:
- SQL injection vulnerabilities
- XSS attack vectors
- Authentication bypass risks

Always provide:
- Severity (Critical/High/Medium/Low)
- OWASP category reference
- Remediation code example
```

IDENTIDADE

Um nome, uma descrição e um comportamento único que o modelo adota quando o agente está ativo.

FERRAMENTAS

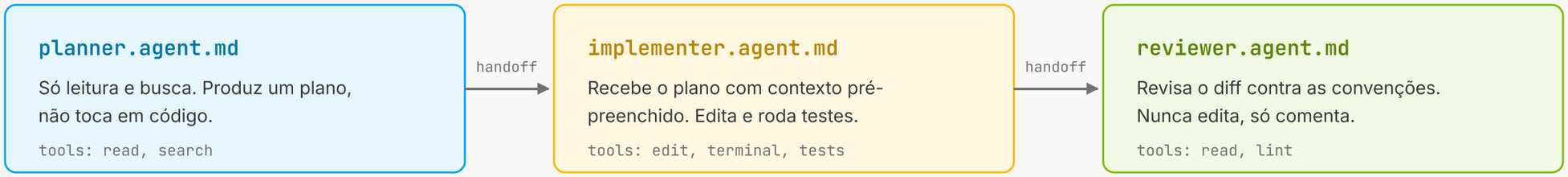
Uma allowlist explícita de ferramentas que o agente pode chamar. Todo o resto fica fora da mesa.

ESCOPO

Foco em uma área ou tipo de tarefa. O agente ativo estreita o contexto para o próprio domínio.

CAMADA 3 ESTENDIDA · HANDOFFS

Agentes se encadeiam: plan, implement, review como pipeline.



Ao final de cada agente, um botão de handoff transfere o contexto para o próximo. O fluxo vira processo, não improviso.

Custom agents (.agent.md) substituem os antigos chat modes: persona, allowlist de ferramentas e handoffs encadeados. Em planos Business e Enterprise, definições podem ser compartilhadas no nível da organização: todo repo herda os mesmos especialistas.

CAMADA 4 DE 5 · GLOB OU ALWAYSAPPLY

SKILL.md empacota um domínio completo, não só regras.

LOCALIZAÇÃO

`.github/skills/<name>/SKILL.md`

ANATOMIA

Regras mais workflows mais templates mais checklists de qualidade. A estrutura força a completude.

SKILLS VS INSTRUCTIONS

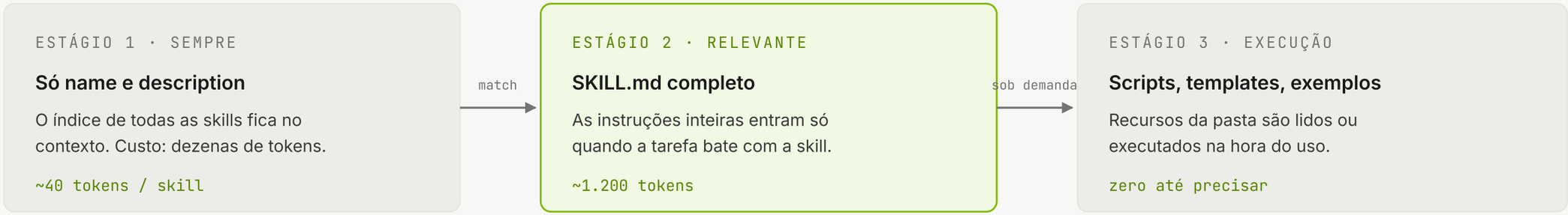
Instructions são regras simples com escopo. Skills são pacotes completos de domínio. Vá de skill quando o domínio é complexo.

.GITHUB/SKILLS/API-DESIGN/SKILL.MD

```
---
name: "REST API Design"
globs: ["src/api/**/*.ts", "src/routes/**/*.ts"]
alwaysApply: false
---

# REST API Design Skill

## Rules
- Plural nouns: /users, not /user
- Version APIs: /api/v1/resources
- Cursor-based pagination

## Workflow
1. Define resource schema  2. Route handlers
3. Validation middleware   4. Integration tests

## Quality checklist
- [ ] Proper status codes on all endpoints
- [ ] Input validation on POST and PUT
- [ ] OpenAPI docs generated
```

CAMADA 4 ESTENDIDA · CARGA PROGRESSIVA

Skills carregam em três estágios. O contexto só paga pelo que usa.



É o mesmo princípio dos três tiers de memória: pague tokens pelo que a tarefa precisa, não pelo que o repositório tem.

PADRÃO ABERTO, PORTÁVEL · VS Code · GitHub Copilot CLI · coding agent · ~/.copilot/skills pessoal

CAMADA 5 DE 5 · SOB DEMANDA

.prompt.md transforma pedidos repetidos em templates reutilizáveis.

.GITHUB/PROMPTS/CREATE-COMPONENT.PROMPT.MD

```
---
mode: "agent"
description: "Create a new React component"
variables:
  - name: "componentName"
  - name: "type"   # page, layout, widget
---

Create a React component called {{componentName}}.
Type: {{type}}

Steps:
1. Create the file in src/components/
2. Add the TypeScript props interface
3. Add a unit test in __tests__/
4. Export from the barrel index.ts
```

MODO AGENT

mode: "agent" permite que o GitHub Copilot execute ações, crie arquivos, rode testes. mode: "edit" só sugere mudanças.

VARIÁVEIS

Chaves duplas marcam parâmetros dinâmicos. O GitHub Copilot pergunta os valores antes de executar.

CICLO DE VIDA

Efêmero por design. Um prompt file roda uma vez por invocação e não persiste no contexto.

O MAPA EM UMA VISÃO

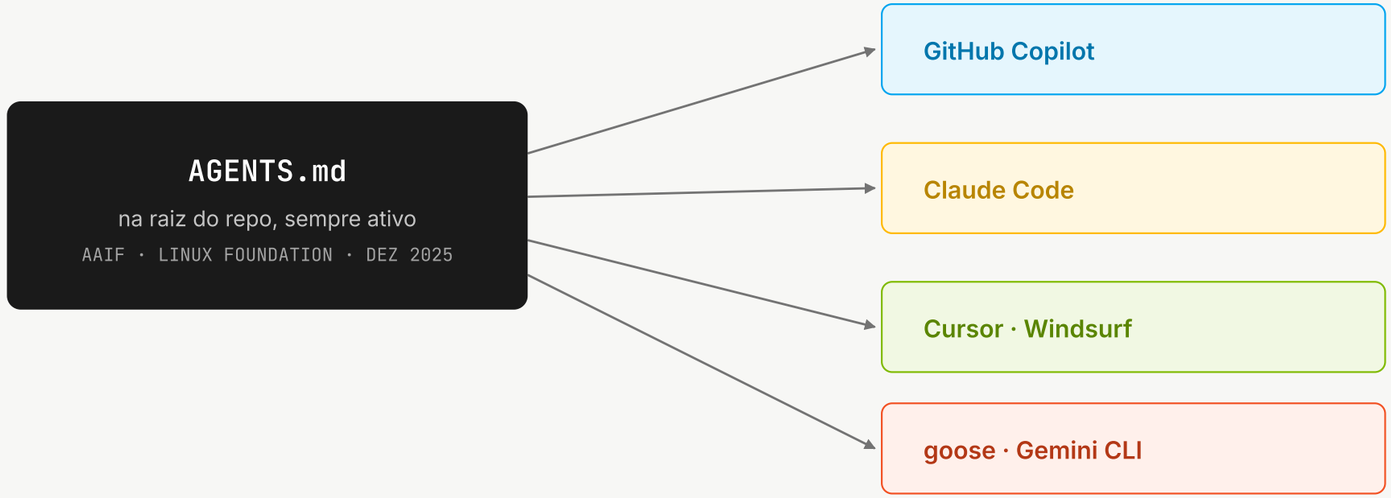
Sete tipos de arquivo, um modelo de composição.

ARQUIVO	ESCOPO	ATIVAÇÃO	PROPÓSITO	FRONTMATTER
`copilot-instructions.md`	Repo inteiro	Automática	Convenções globais	`nenhum`
`AGENTS.md`	Repo inteiro, multi-agente	Automática	Padrão aberto entre ferramentas	`nenhum`
`.instructions.md`	Glob pattern	Por glob match	Regras por área	`applyTo`
`*.agent.md`	Agente ativo	Quando invocado	Persona e ferramentas	`description, tools`
`SKILL.md`	Glob ou always	Glob ou alwaysApply	Domínio completo	`name, globs, alwaysApply`
`.prompt.md`	Invocação	Sob demanda	Template com variáveis	`mode, variables`
`.mcp.json`	Projeto	Automática	Conexões com ferramentas externas	`JSON, sem frontmatter`

Valide nomes e frontmatter contra a documentação oficial de customização do GitHub Copilot. As convenções evoluem rápido.

O MAPA EM MOVIMENTO · AGENTS.MD

AGENTS.md: um arquivo, todos os agentes. Agora padrão aberto.



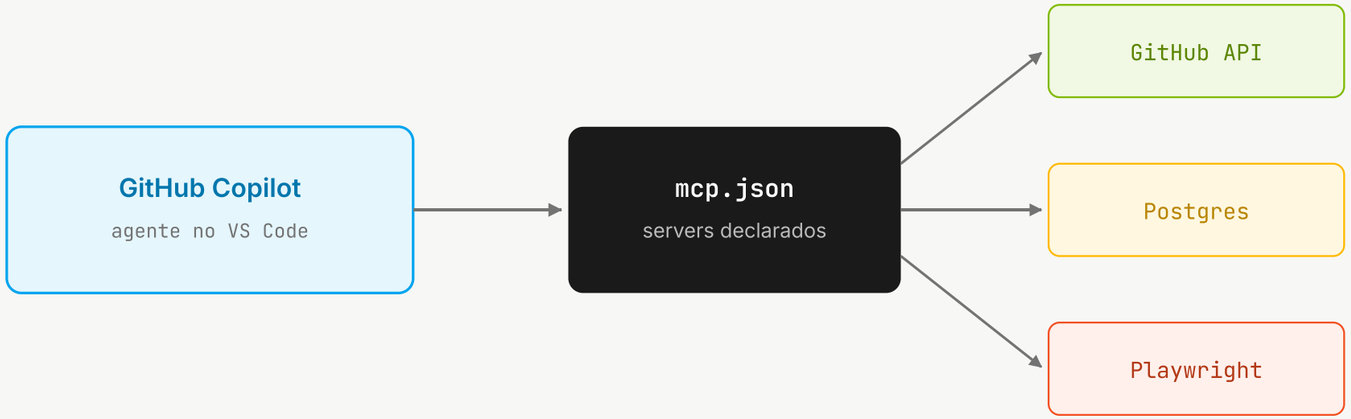
REGRA DE OURO

O que vale para qualquer agente, no AGENTS.md. O que é do GitHub Copilot, no copilot-instructions.md. Sem duplicar.

Doado pela OpenAI à Agentic AI Foundation (Linux Foundation) em dez 2025, junto com MCP e goose. O VS Code lê AGENTS.md como instrução sempre ativa lado a lado com o copilot-instructions.md: um é o padrão aberto multi-ferramenta, o outro é o canal específico do GitHub Copilot.

O MAPA EM MOVIMENTO · MCP

mcp.json conecta os agentes ao mundo. E virou infraestrutura neutra.



.VSCODE/MCP.JSON

```jsonc
// servers que o agente pode usar
{
  "servers": {
    "github": {
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "postgres": {
      "command": "npx",
      "args": ["-y", "@mcp/postgres"]
    }
  }
}
```

97M downloads SDK / mês

10.000+ servers publicados

spec 2025-11-25

Model Context Protocol: spec 2025-11-25 (async tasks, extensões, elicitation) e governança na Agentic AI Foundation desde dez 2025. Para a hierarquia, a regra é a mesma das outras camadas: declare no repo, versione em Git, e o time inteiro herda as mesmas conexões.

O MAPA EM MOVIMENTO · ÚLTIMOS 6 MESES

O que mudou desde dezembro. Valide antes de padronizar.

NOV 2025

MCP spec 2025-11-25:
async tasks, extensões

DEZ 2025

Skills no VS Code 1.108 ·
AGENTS.md e MCP vão à AAIF

Q1 2026

Chat modes viram custom agents:
.chatmode.md renomeia para .agent.md

HOJE

Skills como padrão aberto portável:
VS Code, GitHub Copilot CLI, coding agent

A lição operacional: nomes de arquivo e frontmatter mudam em ciclos de meses. Trate a doc oficial de customização como fonte de verdade, agende uma revisão trimestral da stack, e renomeie .chatmode.md para .agent.md no próximo ciclo.

PARTE

III

Resolução.

O algoritmo de sete passos que o GitHub Copilot executa para montar o contexto, e as regras de precedência que resolvem conflitos.

A ORDEM DE RESOLUÇÃO

Sete passos, cada um ADICIONA ao contexto acumulado.

01	`copilot-instructions.md`	Carregar o contexto global do repo primeiro, em toda sessão: copilot-instructions.md e, quando existir, AGENTS.md.
02	`arquivos no contexto`	Identificar quais arquivos fazem parte da conversa atual.
03	`*.instructions.md`	Aplicar todo instruction file cujo glob faz match com esses arquivos.
04	`*.agent.md`	Carregar a definição do agente ativo, quando um é invocado.
05	`SKILL.md`	Ativar skills por glob match ou alwaysApply: true.
06	`*.prompt.md`	Expandir o template de prompt com suas variáveis, quando invocado.
07	`.mcp.json`	Agregar as conexões MCP disponíveis ao contexto final.

A RESOLUÇÃO · QUATRO PEGADINHAS

Quatro comportamentos que ninguém espera, e todo time descobre tarde.

CODE REVIEW · BRANCH BASE

O review lê as instruções da branch base, não da sua

No pull request, o agente de code review usa o copilot-instructions.md da base. Regras novas na feature branch só valem depois do merge. É proposital: a branch não reescreve as próprias regras de revisão.

COMPLETIONS · FORA DO SISTEMA

O ghost text não lê nada disso

As sugestões inline enquanto você digita são um sistema separado: não leem copilot-instructions.md nem instructions files. A hierarquia governa o chat e os agentes, não o autocomplete.

STACKING · SEM VENCEDOR

Instructions empilham, não competem

Todo instructions file cujo glob bate entra junto, em união. Não existe override nem precedência entre eles. Se duas regras conflitam, as duas chegam ao modelo. Resolver conflito é trabalho seu, não do GitHub Copilot.

EXCLUDEAGENT · ALVO FINO

Dá para tirar um agente de uma regra

O frontmatter excludeAgent opta um agente para fora de um instructions file: a regra de segurança que é perfeita no code review e verbosa demais no chat fica só onde ajuda.

Comportamentos documentados na doc oficial de customização do VS Code e do GitHub Copilot. Vale colar este slide no canal do time: cada item aqui já custou uma tarde de debugging em algum lugar.

REGRAS DE PRECEDÊNCIA

Não existe override destrutivo. A composição é sempre aditiva.

Especificidade vence	Regras mais específicas complementam as genéricas e prevalecem na prática quando as duas tocam o mesmo ponto.	`.instructions.md > copilot-instructions.md`
Aditivo, não substitutivo	Camadas se acumulam, nunca se substituem. O contexto final carrega todas elas.	`regra de pasta + regra global = ambas`
O agente isola o escopo	Um agente ativo estreita o contexto para o próprio domínio, ferramentas e identidade.	`agente de segurança foca em vulnerabilidades`
O prompt é efêmero	Prompt files rodam sob demanda e não persistem. Uma invocação, uma expansão.	`template roda uma vez por invocação`

TESTE O APPLYTO AO VIVO

`src/api/payments.ts` `web/components/App.tsx` `README.md` → `copilot-instructions.md` `api.instructions.md src/api/**`

`frontend.instructions.md **/*.tsx` `docs.instructions.md **/*.md`

A RESOLUÇÃO · MONTE A SUA STACK INTERATIVO

Ligue e desligue camadas. Veja o que o agente recebe.

CENÁRIO: EDITAR SRC/API/PAYMENTS.TS E PEDIR UM ENDPOINT NOVO

- [x] **copilot-instructions.md** sempre ativo · ~800 tokens
- [ ] **AGENTS.md** sempre ativo, multi-agente · ~600
- [x] **api.instructions.md** applyTo: src/api/** bate · ~450
- [ ] **security.agent.md** se invocado · ~350
- [x] **api-design/SKILL.md** match da tarefa · ~1.200
- [ ] **new-endpoint.prompt.md** se chamado com / · ~500
- [x] **mcp.json** definições de tools · ~300

CONTEXTO MONTADO NESTA REQUEST

2.750

tokens de configuração, antes do seu prompt

LEITURA

Stack saudável: global + escopo + domínio.

Estimativas típicas por camada, para formar intuição de ordem de grandeza. O ponto: configuração também é contexto pago por request. Curadoria vale dinheiro.

O ROI DE CONTEXTO

Sem hierarquia, o time paga o mesmo contexto mil vezes por dia.

800 tok

COLADOS À MÃO A CADA CHAT: STACK, CONVENÇÕES, PADRÕES DO REPO

40/dia

CHATS POR DEV POR DIA, EM TIMES QUE RODAM AGENTES A SÉRIO

27M

TOKENS POR MÊS DESPERDIÇADOS POR UM TIME DE 20, SEM HIERARQUIA

A conta: 800 tokens de contexto repetido, vezes 40 chats, vezes 20 devs, vezes 20 dias úteis. A hierarquia transforma esses 32 milhões de tokens repetidos em arquivos versionados, carregados uma vez, herdados por todos.

O ROI DE CONTEXTO · ANTES E DEPOIS

A hierarquia não economiza só tokens. Economiza o trabalho de lembrar.

SEM HIERARQUIA

- Cada dev cola stack e convenções no prompt, toda vez.
- O contexto morre quando o chat fecha. Zero a cada sessão.
- Dois devs explicam a mesma coisa de dois jeitos diferentes.

→

COM HIERARQUIA

- ✓ O contexto vive no Git. Escrito uma vez, herdado por todos.
- ✓ Carga seletiva: o agente paga só pela camada que a tarefa aciona.
- ✓ Uma fonte de verdade. Muda no arquivo, muda para o time inteiro.

O ganho real não é o token, é a consistência. Quando o conhecimento mora no repo, o agente fica bom no primeiro dia de um dev novo, não no terceiro mês.

O ROI DE CONTEXTO · CINCO ALAVANCAS

Cada mecanismo da plataforma corta tokens de um jeito diferente.

SKILL.md	Carga progressiva em três estágios: só o índice fica no contexto até a tarefa bater.	40 tok vs 1.200
*.instructions.md	applyTo com glob: a regra de API só entra quando você edita um arquivo de API.	paga por escopo
*.agent.md	O agente estreita o contexto ao próprio domínio e a uma allowlist enxuta de tools.	contexto focado
GitHub Copilot Spaces	Curadoria: anexa os arquivos que importam em vez do repo inteiro. Busca, não despeja.	grounding curado
GitHub GitHub Copilot Memory	Aprende o repo sozinho e dispensa re-explicação: menos prompt, menos manutenção de instructions.	zero re-prompt

Cinco alavancas, um princípio: o contexto certo, no momento certo, pago uma vez. Curadoria não é estética, é a linha de FinOps que ninguém olha.

COMPOSIÇÃO EM AÇÃO

Um projeto, cada camada no seu lugar.

my-ecommerce · tree .github

```
my-ecommerce/
├── .github/
│   ├── copilot-instructions.md   # global: stack + conventions
│   ├── instructions/
│   │   ├── react.instructions.md  # scope: src/components/**
│   │   └── api.instructions.md    # scope: src/app/api/**
│   ├── agents/
│   │   ├── security.agent.md      # persona: security review
│   │   └── db-expert.agent.md     # persona: database expert
│   ├── skills/
│   │   ├── api-design/SKILL.md    # domain: REST patterns
│   │   └── testing/SKILL.md       # domain: test standards
│   └── prompts/
│       └── fix-bug.prompt.md      # template: debug + fix
├── .vscode/
│   └── mcp.json                   # external connections
└── src/
    ├── components/                # matched by react rules
    └── app/api/                   # matched by api rules
```

UM PROPÓSITO POR ARQUIVO

Cada arquivo responde uma pergunta. Nada de documento pega-tudo de 500 linhas.

COMPOSIÇÃO AUTOMÁTICA

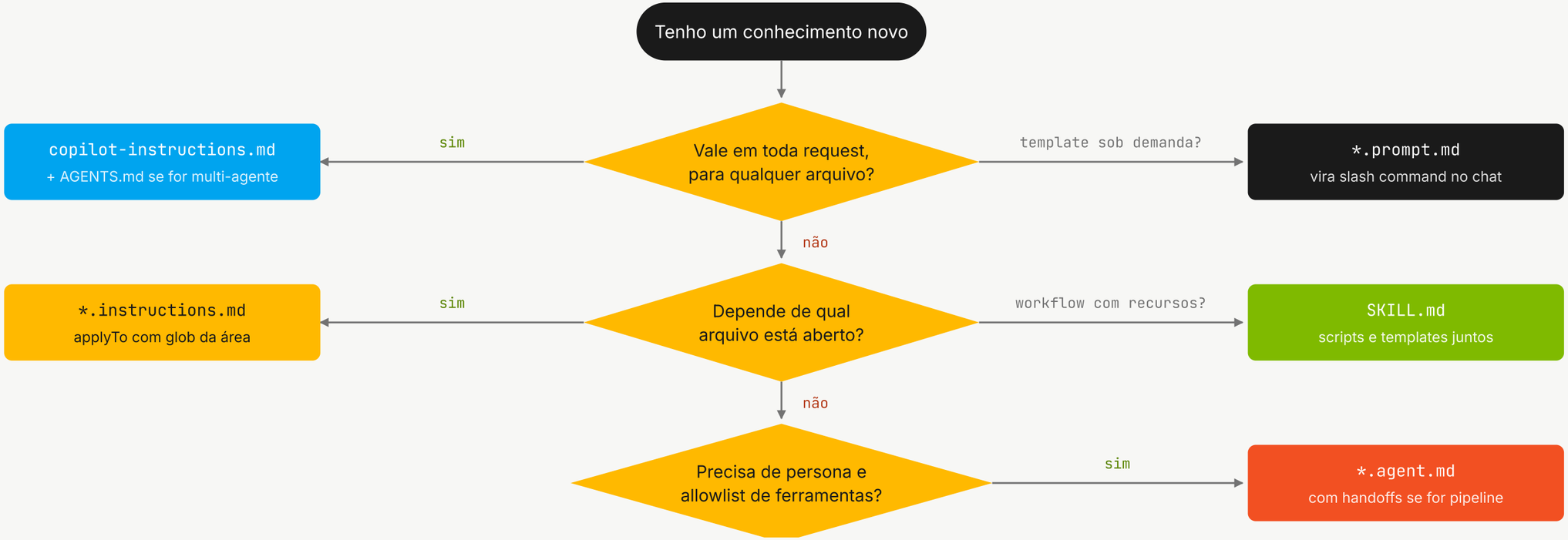
O GitHub Copilot monta o contexto certo para cada arquivo que você edita. Sem fiação manual.

VERSIONADO COM O CÓDIGO

A stack inteira vive no git. Mudança de config passa por pull request como qualquer código.

A RESOLUÇÃO · ÁRVORE DE DECISÃO

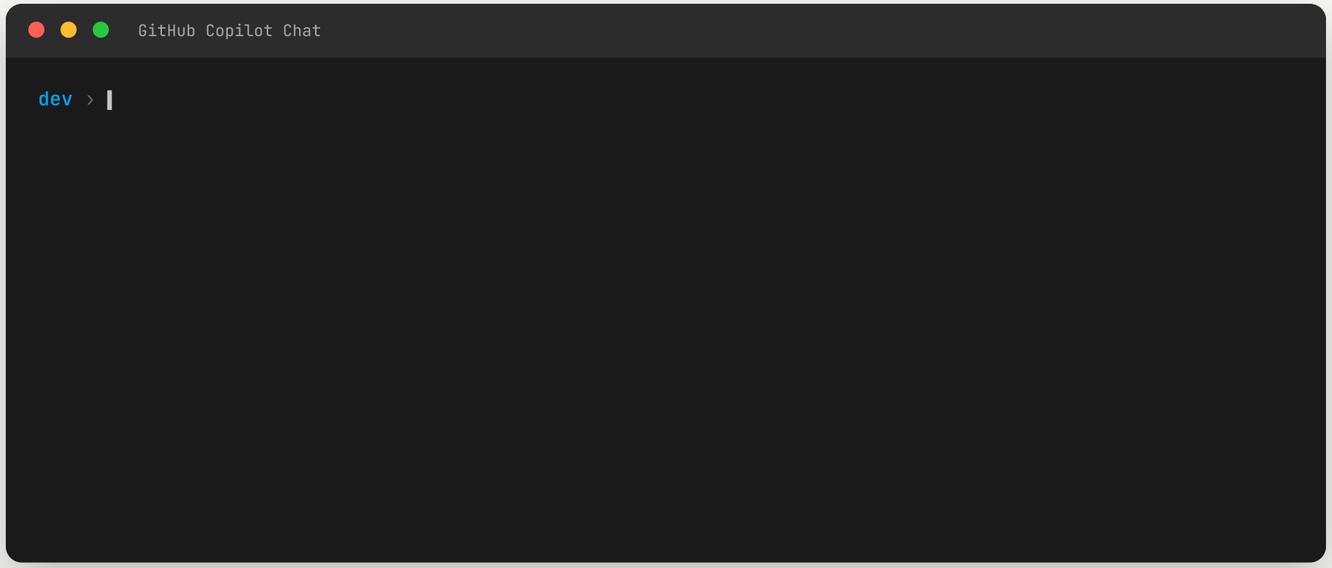
Qual arquivo criar? Quatro perguntas resolvem.



Regra de bolso: comece sempre pelo nível mais alto que ainda é verdadeiro. Conhecimento que vale para tudo sobe; conhecimento que vale para um caso desce. Duplicação entre níveis é o cheiro de que a pergunta errada foi respondida.

A HIERARQUIA TRABALHANDO

Um pedido, três camadas aplicadas, zero setup manual.



CAMADA GLOBAL

A stack e as convenções vieram do copilot-instructions.md sem ninguém colar nada.

CAMADA DE ESCOPO

As regras de API ativaram porque o caminho de destino fez match com o glob do applyTo.

CAMADA DE DOMÍNIO

A skill de design REST trouxe workflows e checklists, não só regras.

PARTE

IV

Na prática.

Três cenários enterprise, o impacto medido e a lista de faça ou não faça que mantém a stack saudável.

TRÊS TIMES, TRÊS CONFIGURAÇÕES

As mesmas cinco camadas resolvem três problemas bem diferentes.

TECH LEAD · STARTUP FINTECH

Ana padroniza 8 devs

Instruções globais definem a stack, três instruction files de escopo cobrem componentes, API e testes, um agente de segurança escaneia todo PR, e uma skill de compliance codifica o domínio financeiro.

-40%
code reviews manuais

STAFF ENGINEER · E-COMMERCE ENTERPRISE

Carlos quebra o monolito

Um arquivo global de arquitetura, instructions de escopo separadas por serviço (cart, payments, inventory), um agente migration-expert para o código legado e skills compartilhadas de event sourcing e CQRS.

18 → 10
meses de migração

PLATFORM ENGINEER · MULTINACIONAL

Maria escala para 50+ repos

Um template organizacional que todo repo herda, times adicionam regras de escopo próprias sem conflito, skills compartilhadas saem de um repositório central e prompts padronizados cobrem operações comuns.

2w → 2d
onboarding de novos devs

Números observados em adoções enterprise deste padrão. Trate como direção, e valide contra a sua própria baseline.

NA PRÁTICA · ANTI-PADRÕES

Quatro anti-padrões que você vai encontrar antes de qualquer boa prática.

ANTI-PADRÃO 01

O arquivo de 3.000 linhas

Todo o conhecimento do time despejado no copilot-instructions.md. Paga o custo em toda request, enterra o crítico no meio do irrelevante, e ninguém revisa. Quebre em camadas: o slide da árvore de decisão é o mapa.

ANTI-PADRÃO 02

Gerado e nunca curado

O /init gera um bom rascunho, mas rascunho gerado documenta o óbvio e omite o tácito. Sem um dono que corta o redundante a cada trimestre, o arquivo vira ruído pago por request.

ANTI-PADRÃO 03

A mesma regra em três camadas

"Use TypeScript strict" no global, no instructions de API e na skill. Como tudo empilha em união, a regra chega três vezes, e quando o time muda de opinião, alguém esquece uma cópia. Cada regra mora em exatamente um nível.

ANTI-PADRÃO 04

Segredo e ambiente no lugar errado

Token de API no instructions file, URL de produção no prompt file. Esses arquivos vão para o Git e para o contexto do modelo. Segredo fica em vault; configuração de ambiente fica em mcp.json com inputs, nunca hardcoded.

Os quatro saem de revisões de stack reais. O padrão comum: tratar configuração de agente como documentação, quando ela é código, com custo por execução, dono e ciclo de revisão.

FAÇA E NÃO FAÇA

Três hábitos separam stacks saudáveis de stacks ruidosas.

NÃO FAÇA

Colocar tudo em copilot-instructions.md. Um arquivo de 500 linhas vira ruído e o modelo perde o foco.

NÃO FAÇA

Escrever regras contraditórias entre camadas, como o global dizendo CSS Modules e o escopo dizendo Tailwind.

NÃO FAÇA

Despejar conhecimento de domínio complexo em instructions planas, sem estrutura, sem workflow, sem checklist.

FAÇA

Dividir em camadas: global, escopo, agentes, skills. Cada arquivo curto, focado e fácil de revisar.

FAÇA

Fazer as camadas se complementarem. Se o global define Tailwind, o arquivo de escopo adiciona regras Tailwind específicas.

FAÇA

Usar SKILL.md para domínios complexos: regras mais workflows mais templates mais checklists. A estrutura garante completude.

POR QUE VALE O ESFORÇO

Configuração automatiza o que o code review costumava pegar.

-33%

Code reviews manuais

Config automatiza padrões que antes dependiam de revisão humana.

-50%

Tempo de onboarding

Novos devs geram código consistente desde o primeiro dia, guiados pela config.

+29%

Consistência de código

Padrões uniformes em todos os repos e times da organização.

-42%

Bugs de padrão

Regras automáticas previnem erros comuns antes do commit.

Faixas observadas em adoções enterprise. A sua baseline define os seus números: meça antes e depois.

PARTE

V

The memory layer.

Os arquivos são o que você escreve. A memória é o que o GitHub Copilot aprende sozinho, e lembra entre sessões.

CAMADA DE MEMÓRIA · O QUE É

Stateless force a repetir. GitHub GitHub Copilot Memory lembra entre sessões.

TIPO 01 · FATOS DO REPOSITÓRIO

O que o GitHub Copilot descobre do código

Convenções, decisões de arquitetura, comandos de build, dependências entre arquivos. Criados por quem tem write access, disponíveis a todo o time, presos àquele repo.

TIPO 02 · PREFERÊNCIAS DO USUÁRIO

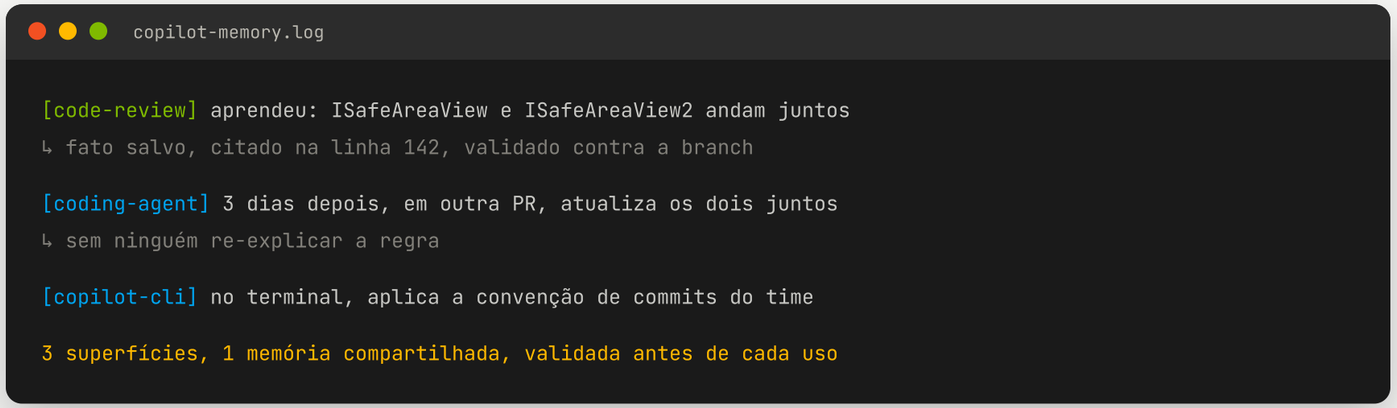
Como você gosta de trabalhar

Estilo de comunicação, stack preferida, convenções de commit. Seguem você entre repositórios, sem afetar os outros. Em Business e Enterprise, sob governança do admin.

Public preview, jan 2026, ativada por padrão em Pro e Pro+ desde março. Disponível em coding agent, code review e CLI. O que um agente aprende, outro usa.

CAMADA DE MEMÓRIA · CROSS-AGENT

O que o code review aprende, o coding agent aplica. Sem você repetir.



VALIDADO ANTES DO USO

Cada fato carrega citações. O GitHub Copilot confere contra a branch atual antes de aplicar. Stale nunca entra.

EXPIRA EM 28 DIAS

Fato não usado some sozinho. O timer reseta a cada uso validado. Nada de lixo acumulado.

ESCOPO POR REPO

Fato de um repo nunca vaza para outro. Privacidade e segurança por desenho.

CAMADA DE MEMÓRIA · MEMÓRIA OU ARQUIVO?

Memória é emergente. Arquivo é deliberado. Você precisa dos dois.

	GITHUB GITHUB COPILOT MEMORY	ARQUIVOS DA HIERARQUIA
Origem	O agente descobre sozinho enquanto trabalha	Você escreve e versiona no Git
Persistência	Expira em 28 dias se não for usada	Permanente até alguém editar
Revisão	Não passa por pull request	Revisada em PR, auditável no histórico
Melhor para	Padrões que emergem do uso real	Regras que você quer garantir, sempre

A regra de bolso: deixe a memória descobrir, depois gradue o que se repete para um arquivo. Se o GitHub Copilot reaprende a mesma coisa toda semana, isso é sinal de que falta uma instruction.

CAMADA DE MEMÓRIA · GRADUAÇÃO

O ciclo virtuoso: a memória descobre, o arquivo perpetua.

MEMÓRIA APRENDE

O agente nota um padrão no uso real e salva como fato.

→

REPETE DEMAIS

A mesma lição reaparece toda semana. Sinal claro.

→

VIRA ARQUIVO

Você promove para copilot-instructions.md ou ARCHITECTURE.md.

GRADUE QUANDO

"Sempre cheque ISafeAreaView e ISafeAreaView2 juntos" reaparece toda semana, vira instruction.

POR QUE IMPORTA

Sem graduação, o aprendizado valioso expira em 28 dias. Com graduação, o repo acumula conhecimento permanente.

Memória e arquivo não competem, se completam. A memória é o rascunho que o agente escreve sozinho. O arquivo é a versão que você decide tornar lei.

FECHAMENTO

Configuração é código para os seus agentes.

Building the future of software development with AI and Agentic DevOps.

CONTATO

Paula Silva
Software Global Black Belt

paulasilva@microsoft.com

PRÓXIMO PASSO

Revisão da stack de config
Um repositório, uma semana, cinco camadas no lugar

Publicado em 2026-06-11 · v6

TERMINAL · COMECE AQUI NA SEGUNDA

```
# 1. Create the structure
$ mkdir -p .github/instructions \
    .github/agents .github/skills .github/prompts

# 2. Start with the global brain
$ touch .github/copilot-instructions.md
  stack, convenções, comandos-chave. Curto.

# 3. Add one scoped rule where pain lives
$ touch .github/instructions/api.instructions.md
  glob no applyTo + as regras só daquela área
```