OPERATING MODEL · TOKENOMICS · GOVERNANÇA

Do gate de validação ao Azure AI Foundry.

Como transformar "dá pra usar vários modelos e primitivos" em "usa de forma validada, com orçamento e roteamento". Um caminho didático e completo, da segunda de manhã sem infra até Redis e Foundry.

AUTORA
Paula Silva

CARGO
Software Global Black Belt

DURAÇÃO
60 a 90 minutos

DATA
2026-06-05

AGENDA

Sete partes, um modelo operacional.

PART I	Por que isso existe, multi-modelo é o começo, não o fim
PART II	O gate de validação, as cinco etapas que todo primitivo atravessa
PART III	Orçamento de tokens, as camadas de contexto hot, warm e cold
PART IV	Governança e cache, catálogo interno mais Redis
PART V	Três agentes de exemplo, Q&A, review de segurança, auditoria agendada
PART VI	Superfícies de agente, agent mode, coding agent e workflows
PART VII	A escada de maturidade, da segunda de manhã ao Foundry

PARTE

I

Por que isso existe.

Multi-modelo é o ponto de partida, não o fim. A peça que falta é o modelo operacional.

O PROBLEMA

Escolha sem governança vira billing bug.

O QUE VOCÊ GANHA DE FÁBRICA

Liberdade de escolher

No GitHub Copilot (IDE, chat, coding agent, code review) e no GitHub Copilot CLI você escolhe entre vários modelos. Via Agent HQ entram os modelos de terceiros da Anthropic, OpenAI, Google, Cognition e xAI, dentro da assinatura paga. Cada agente fixa o modelo no frontmatter, ou o Auto decide.

O QUE DÁ ERRADO

Custo sem freio

Agentes usam o modelo mais caro em tarefas triviais. Não há teto de tokens, então a cauda longa estoura o orçamento. Nada é medido antes da produção, então a qualidade é uma aposta. Liberdade sem governança é o billing bug.

O PRINCÍPIO CENTRAL

Cada primitivo é um artefato versionado, com orçamento e gate.

Antes de qualquer primitivo chegar à produção, ele passa por um gate. O awesome-copilot público é inspiração, não produção. O que vai para produção é versionado, medido e validado.

Previsível
custo, um teto conhecido por task

Auditável
qualidade, pontuada antes de publicar

Rastreável
segurança, política de modelo declarada

O QUE É UM PRIMITIVO

Cinco tipos de unidade reutilizável, um gate.

01

Agente e prompt

Um perfil com modelo, instruções e escopo fixos. Ou um template reutilizável. Os primitivos mais comuns.

02

Skill e hook

Uma capacidade encapsulada, ou um gatilho automatizado que dispara num evento. Reutilizados entre repos e times.

03

Servidor MCP

Um conector que traz sistemas externos. O mais caro de acionar, então vive na camada cold.

PARTE

II

O gate de validação.

Cinco etapas obrigatórias, na ordem. Um artefato só é validado quando atravessa todas.

O GATE DE CINCO ETAPAS

Nada chega ao dev direto.

gate-de-validacao.flow

1 **golden tasks** tarefas representativas daquele agente

2 **medir** tokens de input, output, cached, p50 e p95

3 **teto + política** teto de tokens por task e tier de modelo

4 **juiz** LLM como juiz, pareado, ordem trocada

5 **publicar** versionado no catálogo interno

ETAPA 1, GOLDEN TASKS

Tarefas reais, com respostas conhecidas.

01

Representativas

Cobrem os fluxos mais frequentes, não casos exóticos. O trabalho que o agente vai fazer de verdade em produção.

02

Estáveis

A saída esperada não muda a cada execução. Dá para comparar contra ela de forma confiável.

03

Mensuráveis

A resposta pode ser pontuada de forma objetiva ou por um juiz. Uma pergunta real com resposta correta conhecida.

ETAPA 2, MEDIR CONSUMO

O output é onde morde.

5x

O output custa cerca de cinco vezes o input.
Gerar texto é mais caro que ler texto.

USE P95 **p95**

Não só a média. A média esconde a cauda
longa que estoura o orçamento em produção.

ETAPA 3, TETO E POLÍTICA DE MODELO

Dois limites por artefato.

TETO DE TOKENS

Um limite por task

O limite acima do qual o agente corta ou recusa, baseado no p95 observado mais uma margem. Funciona como um cronômetro: passou do limite, o agente para e entrega o que tem.

POLÍTICA DE MODELO

Qual tier é permitido

Um Q&A simples não precisa do modelo mais caro. Um review de segurança talvez precise. Centralizar o acesso, descentralizar a escolha: o dev escolhe entre os modelos validados, não no mundo todo.

ETAPA 4, QUALIDADE E SEGURANÇA

Um LLM pontua as respostas.

01 Um modelo separado e capaz pontua as respostas do agente nas golden tasks.

02 Use comparações pareadas e troque a ordem entre as rodadas para eliminar viés de posição.

03 No lado GitHub, isso casa com gh aw compile --validate para os agentic workflows.

04 No lado modelo, com o Foundry Evaluations, eval de skill e de workflow.

05 Registro: o Foundry Evaluations não integra direto com o model router. O benchmark de quality, cost e latency usa o toolkit de avaliação dedicado.

ETAPA 5, PUBLICAÇÃO

Só agora ele é validado.

A publicação registra a versão, o teto de tokens, a política de modelo e o resultado da avaliação. A partir daí, o artefato é o que os desenvolvedores veem e usam.

Versão

fixada e rastreável

Teto

registrado por task

Política

tier de modelo declarado

PARTE

III

Orçamento de tokens.

Todo contexto que entra num agente tem custo. A camada certa para cada coisa é a diferença entre gasto previsível e billing bug.

HOT, WARM, COLD

Três camadas, por frequência e custo de trazer.

01

Hot, toda chamada

copilot-instructions.md, AGENTS.md, custom instructions. Pequeno e estável, porque multiplica em cada request. O imposto always-on.

02

Warm, sob demanda

GitHub Copilot Memory, cache semântico no Redis, resultados recentes de skills. Barato de trazer, entra quando relevante.

03

Cold, recuperação explícita

RAG e embeddings do repo, docs, código fora de escopo, MCP externo. Caro, então traz pouco e bem.

A REGRA DE OURO

Empurre o máximo para warm e cold, e mantenha o hot minúsculo.

Cada token na camada hot é pago em toda chamada. Cada token na camada cold só é pago quando recuperado.

PARTE

IV

Catálogo e cache.

O lado do controle e o lado do custo. Só usar do catálogo interno, e cachear o que se repete.

GOVERNANÇA DE CATÁLOGO INTERNO

Um control plane que faz cumprir a política.

LADO GITHUB

agents/ mais marketplace

Um diretório agents/ no nível de org ou enterprise, mais um marketplace interno de plugins. O control plane do Agent HQ, com allowlist de agentes e modelos, faz cumprir: o dev só vê e usa o que foi validado.

LADO FOUNDRY

Policy mais conjunto de fallback

O model router honra a mesma policy de deployment de modelo do Foundry, governada por Azure Policy no momento do deploy. O subset de modelos configurado também funciona como conjunto de fallback, impedindo prompts de serem processados por modelos não aprovados.

DOIS CACHES, NÃO UM

Prompt caching e semantic caching são complementares.

PROMPT CACHING, DO FORNECEDOR

Reuso de prefixo idêntico

Reuso de prefixo idêntico (KV cache). Anthropic por breakpoint explícito, OpenAI automático, e o Azure AI Foundry também: reuso de computação por prefixo para modelos suportados, exigindo primeiros tokens idênticos, com vida de cache de até 24 horas.

SEMANTIC CACHING, ONDE O REDIS ENTRA

Similaridade por embedding

O Redis LangCache é um cache semântico totalmente gerenciado que reduz custo de LLM e melhora tempos de resposta. O RedisVL SemanticCache é a opção self-hosted. O Redis ainda serve de store da camada warm, não só de cache.

COMO O CACHE SEMÂNTICO FUNCIONA

Checar antes de chamar o modelo.

cache-semantico.log

1 checar o cache antes da chamada ao modelo perguntar ao cache primeiro

2 hit, devolve a resposta em cache pula o modelo inteiramente

3 miss, chama o modelo guarda o par prompt-resposta

4 ajustar TTL e threshold de similaridade o cache cuida de embeddings e similaridade

ONDE O CACHE BRILHA

Agentes usam mais tokens, então o reuso compensa.

4x

Agentes usam cerca de quatro vezes mais tokens que chat. A superfície de repetição é grande.

ATÉ **90%**

Reduções citadas em cargas repetitivas: Q&A interno, enablement, suporte, o como faço X repetido. Não para geração de código única.

PARTE

V

Três agentes de exemplo.

O mesmo gate produz políticas opostas, porque a medição de cada agente leva a decisões diferentes de tier, teto e cache.

AGENTE A, AGENT MODE

docs-qa-responder

agents/docs-qa-responder.md

```
model: gpt-4o-mini · tier: economy
max_tokens_per_task: 1200
semantic_cache: enabled
hot: instructions · warm: memory, redis
cache hit de 58 a 71 por cento
```

Alta taxa de reuso, ideal para cache semântico. Uma pergunta simples não justifica modelo caro.

AGENTE B, CODING AGENT

pr-security-reviewer

agents/pr-security-reviewer.md

```
1 model: claude-sonnet-4-6 · tier: premium
2 max_tokens_per_task: 8000
3 semantic_cache: disabled
4 permissions: contents read, issues write
5 o juiz prioriza recall sobre precisão
```

Cada PR é único, então o cache fica desligado. Segurança justifica o modelo mais capaz e um teto alto.

MESMO GATE, POLÍTICAS OPOSTAS

As decisões vêm da medição, não da intuição.

DOCS-QA-RESPONDER

Economy, cache ligado

Tier economy, teto 1.200, cache semântico habilitado, alta reuso de 58 a 71 por cento, foco do juiz na precisão da fonte.

PR-SECURITY-REVIEWER

Premium, cache desligado

Tier premium, teto 8.000, cache semântico desabilitado, nenhuma reuso, foco do juiz no recall de achados. Deixar passar uma vulnerabilidade é pior que um alarme falso.

PARTE

VI

Superfícies de agente.

Existe mais de uma forma de rodar um agente no GitHub Copilot. Cada uma consome tokens de um jeito, e tratar todas igual é a forma mais rápida de estourar o orçamento.

UMA ANALOGIA PARA COMEÇAR

Três formas de trabalhar com um assistente capaz.

01

Você senta ao lado dele

Pede, ele faz, você revisa, pede a próxima coisa. Ele só trabalha enquanto você está presente. Isso é o Agent Mode.

02

Você manda a tarefa e sai

Ele trabalha sozinho por um tempo e te entrega o resultado pronto para revisão. Você não acompanha cada passo. Isso é o Coding Agent.

03

Ele age num evento

Você deixa instruções fixas. Toda vez que algo acontece, ele age, sem ninguém olhando. Isso é um Agentic Workflow.

O QUE É UM TOKEN, E POR QUE CUSTA

Dois fatos mudam tudo.

FATO UM

O output custa cerca de 5x o input

Um token é um pedaço de texto, mais ou menos três quartos de uma palavra. Gerar texto é mais caro que ler. Um agente que escreve respostas longas custa muito mais que um que responde curto, mesmo lendo a mesma coisa.

FATO DOIS

Você paga por chamada

Cada vez que o agente é acionado, a conta de tokens corre de novo. Um agente barato por chamada, acionado mil vezes, pode custar mais que um caro acionado dez vezes. Guarde isso para a discussão do gatilho.

AS TRÊS SUPERFÍCIES, LADO A LADO

Elas cobram de formas diferentes.

Agent Mode

No editor. Muitas interações curtas. Dev no loop. Cobra pela conversa.

Coding Agent

Num servidor do GitHub, sozinho. Sessão longa e autônoma. Cobra pela autonomia.

Workflows

Num runner do Actions, por evento. Recorrente. Cobra pela repetição.

A lição

Uma política para as três ou superprotege ou subprotege. Governe por superfície.

Conversa, autonomia, repetição. Três perfis de custo, três políticas.

SUPERFÍCIE 1, AGENT MODE

Muitas interações curtas, o hot multiplica.

COMO CONSOME

A conversa

O agente vive no editor, conversando com o dev em tempo real. Cada pergunta e resposta é uma chamada. As respostas são curtas, mas o volume ao longo de um dia é alto. O contexto hot se repete em cada chamada.

ONDE ECONOMIZA

Contexto hot minúsculo

Se as instruções têm 2.000 tokens, dez chamadas pagam 20.000 tokens de instrução repetida antes de contar a conversa. Multiplique por cinquenta devs. Mantenha o hot minúsculo e deixe o Auto escolher o modelo nas tarefas simples.

SUPERFÍCIE 2, CODING AGENT

Poucas sessões longas, muito output.

COMO CONSOME

Autonomia longa

Recebe uma tarefa inteira e roda sozinho num servidor do GitHub, depois abre um pull request. Ninguém está no loop durante o trabalho. Ele lê muitos arquivos (cold) e escreve muito (output, que custa 5x). Uma tarefa pode custar mais que um dia inteiro de Agent Mode.

ONDE ECONOMIZA

Escopo e teto de sessão

Uma tarefa vaga como "melhore o código" deixa ele vagar e gastar. Uma tarefa específica como "adicione validação de e-mail no cadastro" tem fim claro. O teto de tokens é um cronômetro: passou do limite, ele para.

SUPERFÍCIE 3, AGENTIC WORKFLOWS

Ele age sozinho, disparado por um evento.

COMO CONSOME

Autonomia vezes repetição

Dentro do GitHub Actions, o agente roda toda vez que um gatilho dispara, para sempre, até alguém desligar. Um workflow em todo push num repo ativo pode rodar centenas de vezes por dia. Custo é igual a custo por execução vezes número de disparos.

COMO VALIDA

Compile, valide, permissões mínimas

O gh aw compile --validate checa o workflow antes da produção. As permissões precisam ser mínimas: um agente autônomo com permissões amplas é um risco de segurança, não só de custo.

O MULTIPLICADOR ESCONDIDO, O GATILHO

Mesmo agente, custo radicalmente diferente.

300 /dia

Em todo push, num repo ativo. Custo altíssimo.
O freio humano desaparece na automação.

vs

1 /dia

Agendado uma vez por dia para trabalho não urgente. Custo baixíssimo. Aprove o agente E a frequência.

AGENTE C, AGENTIC WORKFLOW

nightly-dep-auditor

agents/nightly-dep-auditor.md

```
1 trigger: schedule · cron: 0 3 * * *
2 model: gpt-4o-mini · tier: economy
3 max_tokens_per_run: 6000 · max_runs_per_day: 1
4 permissions: contents read, issues write
5 o multiplicador escondido a seu favor
```

Auditar dependências não é urgente, então a frequência mínima corta o custo sem perder valor.

PARTE

VII

A escada de maturidade.

Quatro níveis. Cada um entrega valor sozinho e prepara o próximo.

DA SEGUNDA DE MANHÃ AO FOUNDRY

Quatro níveis de maturidade.

N3	TRIMESTRE · ROTEAMENTO E AVALIAÇÃO **Foundry**	O Model Router seleciona o LLM ótimo por query em tempo real, modos Balanced, Cost, Quality com failover. Foundry IQ como plano de retrieval. PTUs só depois de medir carga e hit rate.
N2	MÊS 1 · CACHE E TELEMETRIA **Visibilidade**	Redis LangCache nas superfícies de Q&A e enablement. Prompt caching onde há contexto grande e fixo. Telemetria de tokens e custo por agente, modelo e time.
N1	SEMANA 1 A 2 · GATE E CATÁLOGO **Disciplina**	Golden tasks, medir tokens e custo por agente, teto por task e tier de modelo, gh aw compile --validate, publicar só o validado, allowlist no control plane.
N0	SEGUNDA DE MANHÃ · ZERO INFRA **Higiene**	copilot-instructions.md enxuto, AGENTS.md com routing e guardrails, Auto como default, premium desligado em non-chat, curar GitHub Copilot Memory, budgets de créditos, exportar usage data.

OS PRIMEIROS 90 DIAS

Três fases, uma cadência.

Dias 1 a 30, baseline	Higiene de config, contexto hot enxuto, Auto como default, premium desligado em superfícies non-chat, budgets de créditos e content exclusion. Exporte o primeiro usage data.	30 dias
Dias 31 a 60, instrumente	Golden tasks por agente, medir tokens e custo, definir tetos e tiers, validar e publicar no catálogo interno, configurar a allowlist.	30 dias
Dias 61 a 90, escala	Redis LangCache no Q&A, prompt caching onde o contexto é fixo, telemetria por agente e time, preparar o plano de roteamento e avaliação do Foundry.	30 dias

ONDE CADA DECISÃO CAI

Cada superfície tem uma alavanca dominante.

01

Agent Mode, o contexto hot

Encolher o contexto always-on economiza em toda chamada, efeito multiplicado pelo volume. É a alavanca dominante aqui.

02

Coding Agent, escopo e teto

Uma tarefa bem escopada termina antes, com menos output acumulado. O teto de sessão limita a cauda longa.

03

Workflows, a frequência

Reduzir a frequência do gatilho corta o custo proporcional aos disparos evitados. Prefira eventos de conclusão a cada passo intermediário.

ERROS COMUNS

O que evitar.

01 Tratar as três superfícies com uma política. Cada uma tem um perfil de custo diferente. Governe por superfície.

02 Validar o agente, mas esquecer o gatilho. Um agente excelente rodando 300 vezes por dia é problema de custo, não vitória.

03 Dar tarefas vagas ao Coding Agent. "Melhore o código" não tem fim. Tarefas específicas têm ponto natural de parada.

04 Inchar o contexto hot. Cada token always-on é pago em toda chamada, por todo desenvolvedor.

05 Permissões amplas num agente autônomo. Risco de segurança e de custo. Permissões mínimas para a tarefa, sempre.

ENCERRAMENTO

Governe a escolha, não a liberdade.

Centralizar o acesso, descentralizar a escolha. O dev escolhe entre os validados, com orçamento e roteamento.

CONTATO

Paula Silva
Software Global Black Belt

paulasilva@microsoft.com

PRÓXIMO PASSO

Rode o gate em dois agentes
Um Q&A, um review, meça os dois

Publicado em 2026-06-05

```
// o modelo operacional, em uma linha
validar → orçar → rotear → cachear
por primitivo, por superfície, por gatilho
```