OPERATING MODEL · TOKENOMICS · GOBERNANZA

Del gate de validación a Azure AI Foundry.

Cómo transformar "se pueden usar varios modelos y primitivos" en "se usan validados, con presupuesto y enrutamiento". Un camino didáctico y completo, del lunes por la mañana sin infra hasta Redis y Foundry.

AUTORA
Paula Silva

CARGO
Software Global Black Belt

DURACIÓN
60 a 90 minutos

FECHA
2026-06-05

AGENDA

Siete partes, un modelo operativo.

PART I	Por qué existe esto, multi-modelo es el inicio, no el fin
PART II	El gate de validación, los cinco pasos que cruza cada primitivo
PART III	Presupuesto de tokens, las capas de contexto hot, warm y cold
PART IV	Gobernanza y caché, catálogo interno más Redis
PART V	Tres agentes de ejemplo, Q&A, review de seguridad, auditoría agendada
PART VI	Superficies de agente, agent mode, coding agent y workflows
PART VII	La escalera de madurez, del lunes por la mañana a Foundry

PARTE

I

Por qué existe esto.

Multi-modelo es el punto de partida, no el fin. La pieza que falta es el modelo operativo.

EL PROBLEMA

Elección sin gobernanza se vuelve un billing bug.

LO QUE RECIBES DE FÁBRICA

Libertad de elegir

En GitHub Copilot (IDE, chat, coding agent, code review) y en el GitHub Copilot CLI eliges entre varios modelos. Vía Agent HQ entran los modelos de terceros de Anthropic, OpenAI, Google, Cognition y xAI, dentro del plan pago. Cada agente fija su modelo en el frontmatter, o el Auto decide.

LO QUE SALE MAL

Costo sin freno

Los agentes usan el modelo más caro en tareas triviales. No hay techo de tokens, entonces la cola larga revienta el presupuesto. Nada se mide antes de producción, entonces la calidad es una apuesta. Libertad sin gobernanza es el billing bug.

EL PRINCIPIO CENTRAL

Cada primitivo es un artefacto versionado, con presupuesto y gate.

Antes de que cualquier primitivo llegue a producción, pasa por un gate. El awesome-copilot público es inspiración, no producción. Lo que va a producción es versionado, medido y validado.

Predecible

costo, un techo conocido por task

Auditable

calidad, puntuada antes de publicar

Rastreable

seguridad, política de modelo declarada

QUÉ ES UN PRIMITIVO

Cinco tipos de unidad reutilizable, un gate.

01

Agente y prompt

Un perfil con modelo, instrucciones y alcance fijos. O un template reutilizable. Los primitivos más comunes.

02

Skill y hook

Una capacidad encapsulada, o un disparador automatizado que se activa en un evento. Reutilizados entre repos y equipos.

03

Servidor MCP

Un conector que trae sistemas externos. El más caro de invocar, entonces vive en la capa cold.

PARTE

II

El gate de validación.

Cinco pasos obligatorios, en orden. Un artefacto solo es validado cuando cruza todos.

EL GATE DE CINCO PASOS

Nada llega al dev directo.

gate-de-validacion.flow

1 **golden tasks** tareas representativas de ese agente

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

3 **techo + política** techo de tokens por task y tier de modelo

4 **juez** LLM como juez, pareado, orden cambiado

5 **publicar** versionado en el catálogo interno

PASO 1, GOLDEN TASKS

Tareas reales, con respuestas conocidas.

01

Representativas

Cubren los flujos más frecuentes, no casos exóticos. El trabajo que el agente hará de verdad en producción.

02

Estables

La salida esperada no cambia en cada ejecución. Se puede comparar contra ella de forma confiable.

03

Medibles

La respuesta puede puntuarse de forma objetiva o por un juez. Una pregunta real con respuesta correcta conocida.

PASO 2, MEDIR CONSUMO

El output es donde muerde.

5x

El output cuesta cerca de cinco veces el input. Generar texto es más caro que leer texto.

USA P95 **p95**

No solo el promedio. El promedio esconde la cola larga que revienta el presupuesto en producción.

PASO 3, TECHO Y POLÍTICA DE MODELO

Dos límites por artefacto.

TECHO DE TOKENS

Un límite por task

El límite por encima del cual el agente corta o rechaza, basado en el p95 observado más un margen. Funciona como un cronómetro: pasado el límite, el agente para y entrega lo que tiene.

POLÍTICA DE MODELO

Qué tier se permite

Un Q&A simple no necesita el modelo más caro. Un review de seguridad quizás sí. Centralizar el acceso, descentralizar la elección: el dev elige entre los modelos validados, no en el mundo entero.

PASO 4, CALIDAD Y SEGURIDAD

Un LLM puntúa las respuestas.

01 Un modelo separado y capaz puntúa las respuestas del agente en las golden tasks.

02 Usa comparaciones pareadas y cambia el orden entre rondas para eliminar el sesgo de posición.

03 En el lado GitHub, esto encaja con gh aw compile --validate para los agentic workflows.

04 En el lado modelo, con Foundry Evaluations, eval de skill y de workflow.

05 Nota: Foundry Evaluations no integra directo con el model router. El benchmark de quality, cost y latency usa el toolkit de evaluación dedicado.

PASO 5, PUBLICACIÓN

Solo ahora es validado.

La publicación registra la versión, el techo de tokens, la política de modelo y el resultado de la evaluación. De ahí en adelante, el artefacto es lo que los desarrolladores ven y usan.

Versión

fijada y rastreable

Techo

registrado por task

Política

tier de modelo declarado

PARTE

III

Presupuesto de tokens.

Todo contexto que entra a un agente tiene costo. La capa correcta para cada cosa es la diferencia entre gasto predecible y billing bug.

HOT, WARM, COLD

Tres capas, por frecuencia y costo de traer.

01

Hot, cada llamada

copilot-instructions.md, AGENTS.md, custom instructions. Pequeño y estable, porque multiplica en cada request. El impuesto always-on.

02

Warm, bajo demanda

GitHub Copilot Memory, caché semántico en Redis, resultados recientes de skills. Barato de traer, entra cuando es relevante.

03

Cold, recuperación explícita

RAG y embeddings del repo, docs, código fuera de alcance, MCP externo. Caro, entonces trae poco y bien.

LA REGLA DE ORO

Empuja el máximo a warm y cold, y mantén el hot minúsculo.

Cada token en la capa hot se paga en cada llamada. Cada token en la capa cold solo se paga cuando se recupera.

PARTE

IV

Catálogo y caché.

El lado del control y el lado del costo. Solo usar del catálogo interno, y cachear lo que se repite.

GOBERNANZA DE CATÁLOGO INTERNO

Un control plane que hace cumplir la política.

LADO GITHUB

agents/ más marketplace

Un directorio agents/ a nivel de org o enterprise, más un marketplace interno de plugins. El control plane del Agent HQ, con allowlist de agentes y modelos, lo hace cumplir: el dev solo ve y usa lo que fue validado.

LADO FOUNDRY

Policy más conjunto de fallback

El model router honra la misma policy de deployment de modelo de Foundry, gobernada por Azure Policy al momento del deploy. El subset de modelos configurado también funciona como conjunto de fallback, impidiendo que los prompts sean procesados por modelos no aprobados.

DOS CACHÉS, NO UNO

Prompt caching y semantic caching son complementarios.

PROMPT CACHING, DEL PROVEEDOR

Reuso de prefijo idéntico

Reuso de prefijo idéntico (KV cache). Anthropic por breakpoint explícito, OpenAI automático, y Azure AI Foundry también: reuso de cómputo por prefijo para modelos soportados, exigiendo primeros tokens idénticos, con vida de caché de hasta 24 horas.

SEMANTIC CACHING, DONDE ENTRA REDIS

Similitud por embedding

Redis LangCache es un caché semántico totalmente gestionado que reduce costo de LLM y mejora tiempos de respuesta. RedisVL SemanticCache es la opción self-hosted. Redis también sirve de store de la capa warm, no solo de caché.

CÓMO FUNCIONA EL CACHÉ SEMÁNTICO

Chequear antes de llamar al modelo.

cache-semantico.log

```
1 chequear el caché antes de la llamada al modelo  preguntar al caché primero
2 hit, devuelve la respuesta en caché  salta el modelo por completo
3 miss, llama al modelo  guarda el par prompt-respuesta
4 ajustar TTL y umbral de similitud  el caché cuida embeddings y similitud
```

DONDE EL CACHÉ BRILLA

Los agentes usan más tokens, entonces el reuso compensa.

4x

Los agentes usan cerca de cuatro veces más tokens que el chat. La superficie de repetición es grande.

HASTA **90%**

Reducciones citadas en cargas repetitivas: Q&A interno, enablement, soporte, el cómo hago X repetido. No para generación de código única.

PARTE

V

Tres agentes de ejemplo.

El mismo gate produce políticas opuestas, porque la medición de cada agente lleva a decisiones diferentes de tier, techo y caché.

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 ciento
```

Alta tasa de reuso, ideal para caché semántico. Una pregunta simple no justifica modelo caro.

AGENTE B, CODING AGENT

pr-security-reviewer

agents/pr-security-reviewer.md

```
model: claude-sonnet-4-6 · tier: premium
max_tokens_per_task: 8000
semantic_cache: disabled
permissions: contents read, issues write
el juez prioriza recall sobre precisión
```

Cada PR es único, entonces el caché queda apagado. Seguridad justifica el modelo más capaz y un techo alto.

MISMO GATE, POLÍTICAS OPUESTAS

Las decisiones vienen de la medición, no de la intuición.

DOCS-QA-RESPONDER

Economy, caché encendido

Tier economy, techo 1.200, caché semántico habilitado, alto reuso de 58 a 71 por ciento, foco del juez en la precisión de la fuente.

PR-SECURITY-REVIEWER

Premium, caché apagado

Tier premium, techo 8.000, caché semántico deshabilitado, ningún reuso, foco del juez en el recall de hallazgos. Dejar pasar una vulnerabilidad es peor que una falsa alarma.

PARTE

VI

Superficies de agente.

Hay más de una forma de correr un agente en GitHub Copilot. Cada una consume tokens distinto, y tratarlas todas igual es la forma más rápida de reventar el presupuesto.

UNA ANALOGÍA PARA EMPEZAR

Tres formas de trabajar con un asistente capaz.

01

Te sientas a su lado

Pides, lo hace, revisas, pides lo siguiente. Solo trabaja mientras estás presente. Esto es el Agent Mode.

02

Mandas la tarea y te vas

Trabaja solo un rato y te entrega el resultado listo para revisar. No miras cada paso. Esto es el Coding Agent.

03

Actúa ante un evento

Dejas instrucciones fijas. Cada vez que algo pasa, actúa, sin nadie mirando. Esto es un Agentic Workflow.

QUÉ ES UN TOKEN, Y POR QUÉ CUESTA

Dos hechos cambian todo.

HECHO UNO

El output cuesta cerca de 5x el input

Un token es un pedazo de texto, más o menos tres cuartos de una palabra. Generar texto es más caro que leer. Un agente que escribe respuestas largas cuesta mucho más que uno que responde corto, aun leyendo lo mismo.

HECHO DOS

Pagas por llamada

Cada vez que el agente se acciona, la cuenta de tokens corre de nuevo. Un agente barato por llamada, accionado mil veces, puede costar más que uno caro accionado diez veces. Guarda esto para la discusión del disparador.

LAS TRES SUPERFICIES, LADO A LADO

Cobran de formas diferentes.

Agent Mode

En el editor. Muchas interacciones cortas. Dev en el loop. Cobra por la conversación.

Coding Agent

En un servidor de GitHub, solo. Sesión larga y autónoma. Cobra por la autonomía.

Workflows

En un runner de Actions, por evento. Recurrente. Cobra por la repetición.

La lección

Una política para las tres o sobreprotege o subprotege. Gobierna por superficie.

Conversación, autonomía, repetición. Tres perfiles de costo, tres políticas.

SUPERFICIE 1, AGENT MODE

Muchas interacciones cortas, el hot multiplica.

CÓMO CONSUME

La conversación

El agente vive en el editor, conversando con el dev en tiempo real. Cada pregunta y respuesta es una llamada. Las respuestas son cortas, pero el volumen a lo largo de un día es alto. El contexto hot se repite en cada llamada.

DÓNDE AHORRA

Contexto hot minúsculo

Si las instrucciones tienen 2.000 tokens, diez llamadas pagan 20.000 tokens de instrucción repetida antes de contar la conversación. Multiplica por cincuenta devs. Mantén el hot minúsculo y deja que el Auto elija el modelo en las tareas simples.

SUPERFICIE 2, CODING AGENT

Pocas sesiones largas, mucho output.

CÓMO CONSUME

Autonomía larga

Recibe una tarea entera y corre solo en un servidor de GitHub, luego abre un pull request. Nadie está en el loop durante el trabajo. Lee muchos archivos (cold) y escribe mucho (output, que cuesta 5x). Una tarea puede costar más que un día entero de Agent Mode.

DÓNDE AHORRA

Alcance y techo de sesión

Una tarea vaga como "mejora el código" lo deja vagar y gastar. Una tarea específica como "agrega validación de e-mail al registro" tiene fin claro. El techo de tokens es un cronómetro: pasado el límite, para.

SUPERFICIE 3, AGENTIC WORKFLOWS

Actúa solo, disparado por un evento.

CÓMO CONSUME

Autonomía por repetición

Dentro de GitHub Actions, el agente corre cada vez que un disparador se activa, para siempre, hasta que alguien lo apague. Un workflow en cada push en un repo activo puede correr cientos de veces al día. Costo es igual a costo por ejecución por número de disparos.

CÓMO VALIDA

Compile, valida, permisos mínimos

El gh aw compile --validate chequea el workflow antes de producción. Los permisos deben ser mínimos: un agente autónomo con permisos amplios es un riesgo de seguridad, no solo de costo.

EL MULTIPLICADOR ESCONDIDO, EL DISPARADOR

Mismo agente, costo radicalmente diferente.

300 /día

En cada push, en un repo activo. Costo altísimo. El freno humano desaparece en la automatización.

vs

1 /día

Agendado una vez al día para trabajo no urgente. Costo bajísimo. Aprueba el agente Y su frecuencia.

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 el multiplicador escondido a tu favor
```

Auditar dependencias no es urgente, entonces la frecuencia mínima corta el costo sin perder valor.

PARTE

VII

La escalera de madurez.

Cuatro niveles. Cada uno entrega valor solo y prepara el siguiente.

DEL LUNES POR LA MAÑANA A FOUNDRY

Cuatro niveles de madurez.

N3	TRIMESTRE · ENRUTAMIENTO Y EVALUACIÓN **Foundry**	El Model Router selecciona el LLM óptimo por query en tiempo real, modos Balanced, Cost, Quality con failover. Foundry IQ como plano de retrieval. PTUs solo después de medir carga y hit rate.
N2	MES 1 · CACHÉ Y TELEMETRÍA **Visibilidad**	Redis LangCache en las superficies de Q&A y enablement. Prompt caching donde hay contexto grande y fijo. Telemetría de tokens y costo por agente, modelo y equipo.
N1	SEMANA 1 A 2 · GATE Y CATÁLOGO **Disciplina**	Golden tasks, medir tokens y costo por agente, techo por task y tier de modelo, gh aw compile --validate, publicar solo lo validado, allowlist en el control plane.
N0	LUNES POR LA MAÑANA · CERO INFRA **Higiene**	copilot-instructions.md esbelto, AGENTS.md con routing y guardrails, Auto como default, premium apagado en non-chat, curar GitHub Copilot Memory, budgets de créditos, exportar usage data.

LOS PRIMEROS 90 DÍAS

Tres fases, una cadencia.

Días 1 a 30, baseline	Higiene de config, contexto hot esbelto, Auto como default, premium apagado en superficies non-chat, budgets de créditos y content exclusion. Exporta el primer usage data.	30 días
Días 31 a 60, instrumenta	Golden tasks por agente, medir tokens y costo, definir techos y tiers, validar y publicar en el catálogo interno, configurar la allowlist.	30 días
Días 61 a 90, escala	Redis LangCache en el Q&A, prompt caching donde el contexto es fijo, telemetría por agente y equipo, preparar el plano de enrutamiento y evaluación de Foundry.	30 días

DÓNDE CAE CADA DECISIÓN

Cada superficie tiene una palanca dominante.

01

Agent Mode, el contexto hot

Encoger el contexto always-on ahorra en cada llamada, efecto multiplicado por el volumen. Es la palanca dominante aquí.

02

Coding Agent, alcance y techo

Una tarea bien acotada termina antes, con menos output acumulado. El techo de sesión limita la cola larga.

03

Workflows, la frecuencia

Reducir la frecuencia del disparador corta el costo proporcional a los disparos evitados. Prefiere eventos de conclusión a cada paso intermedio.

ERRORES COMUNES

Qué evitar.

01 Tratar las tres superficies con una política. Cada una tiene un perfil de costo diferente. Gobierna por superficie.

02 Validar el agente, pero olvidar el disparador. Un agente excelente corriendo 300 veces al día es problema de costo, no victoria.

03 Dar tareas vagas al Coding Agent. "Mejora el código" no tiene fin. Las tareas específicas tienen punto natural de parada.

04 Inflar el contexto hot. Cada token always-on se paga en cada llamada, por cada desarrollador.

05 Permisos amplios en un agente autónomo. Riesgo de seguridad y de costo. Permisos mínimos para la tarea, siempre.

CIERRE

Gobierna la elección, no la libertad.

Centralizar el acceso, descentralizar la elección. El dev elige entre los validados, con presupuesto y enrutamiento.

CONTACTO

Paula Silva
Software Global Black Belt

paulasilva@microsoft.com

PRÓXIMO PASO

Corre el gate en dos agentes
Un Q&A, un review, mide ambos

Publicado el 2026-06-05

```
// el modelo operativo, en una línea
validar → presupuestar → enrutar → cachear
por primitivo, por superficie, por disparador
```