GESTIÓN DE TOKENS Y DISEÑO DE AGENTES · PLAYBOOK

La facturación por tokens recompensa la ingeniería disciplinada e instrumentada.

El 1 de junio de 2026 GitHub Copilot retiró las PRUs y migró a GitHub AI Credits. Los equipos que tratan los tokens como recurso finito, aplican context engineering e instrumentan FinOps pagan de 25 a 45 por ciento menos que los equipos que usan Copilot como caja negra. La diferencia no es talento. Es práctica.

AUTORA

Paula Silva

CARGO

Software Global Black Belt

CONTACTO

paulasilva@microsoft.com

TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

5

PARTES DEL PLAYBOOK

6

CAPÍTULOS

8

PATRONES APLICADOS

25-45%

META DE REDUCCIÓN DE COSTO

CAPÍTULOS

Chapter 00

Fundaciones: tokens, consumo, modelos

Cómo se cuentan los tokens, adónde van dentro de una sesión y cómo el model picker de Copilot moldea la cuenta. Sin esto, toda estimación es adivinanza.

Fundaciones →

Chapter 01

Disciplinas técnicas

Context engineering, prompt caching, diseño de agentes y estrategias token-eficientes. Las palancas que acumulan ahorro antes de tocar un presupuesto.

Disciplinas →

Chapter 02

El cambio de junio de 2026

PRUs retiradas, GitHub AI Credits en su lugar. Qué cambió, los tres caminos para planes anuales y cómo migrar un equipo sin desborde de overage.

Cambio →

Chapter 03

Gobernanza y FinOps

Presupuestos, alertas y atribución por sesión, agente y repositorio. La economía de Azure AI Foundry a escala enterprise.

Gobernanza →

Chapter 04

Patrones aplicados y anti-patrones

Ocho patrones aplicados de optimización y el árbol de decisión de anti-patrones. Reducción de 40 a 70 por ciento en cargas agénticas.

Aplicación →

Chapter 05

Apéndices: material de referencia

Principios de prompts optimizados, la referencia de precios de mediados de 2026, plantillas de reporte FinOps y dónde vive el código completo.

Referencia →

CONSEJOS DE LECTURA

> **ATAJOS DE TECLADO**
>
> Presiona `/` para buscar. Usa `j` y `k` para moverte entre capítulos.
> Presiona `t` para cambiar tema. Presiona `g` para ir al inicio.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PARTE I, FUNDAMENTOS

Fundamentos: tokens, consumo y la escalera de modelos.

El modelo nunca ve texto. Opera sobre secuencias de enteros, y cada entero tiene un precio. Cómo el texto se convierte en tokens, adónde van los tokens y qué modelo los multiplica: la fundación de todo lo demás en este playbook.

El token es la unidad de trabajo, y de facturación

Un token es la unidad atómica que procesa un modelo de lenguaje: un entero positivo que indexa una entrada en un vocabulario fijo, típicamente entre 32.000 y 200.000 entradas según el modelo. La mayoría de los LLMs modernos usa Byte Pair Encoding (BPE), un algoritmo de compresión creado por Philip Gage en 1994 y adaptado a NLP por Sennrich et al. en 2016, que descubre empíricamente unidades de subpalabra a partir del corpus de entrenamiento. La tokenización por subpalabras resuelve tres problemas a la vez: vocabulario abierto (las palabras nunca vistas se descomponen en subtokens conocidos), compresión eficiente y multilingüismo, porque el BPE a nivel de byte opera sobre los 256 bytes de UTF-8 y funciona en cualquier idioma, código o emoji.

Como cada proveedor descubre su propio vocabulario, el mismo texto se tokeniza distinto en cada modelo, pero las equivalencias de bolsillo se mantienen. El inglés general ronda las 0,75 palabras por token; el portugués brasileño es cerca de 25% más denso en tokens que el inglés, y el español se comporta de forma parecida; el código consume más tokens por línea que la prosa; el JSON estructurado es aún más pesado, y el Unicode raro o los emojis pueden costar de 2 a 4 tokens cada uno. En la práctica, un archivo Python de 1.000 líneas son unos 12.000 tokens, un documento markdown de 2.000 palabras en portugués, unos 3.300 tokens, y una respuesta de chat de 500 palabras en inglés, unos 670 tokens. Con estas equivalencias se estima cualquier sesión antes de ejecutarla; sin ellas, toda estimación es una adivinanza.

Input, output y caché: tres precios para tres tipos de trabajo

Cada llamada a un LLM tiene tres categorías de tokens. El input (system prompt, herramientas, historial, contexto y el mensaje actual) se procesa en paralelo mediante el mecanismo de atención; el trabajo está limitado por memoria y es la categoría más barata. El output se genera token a token en modo autorregresivo, cada token nuevo exigiendo una pasada sobre todo lo anterior, lo que lo hace típicamente de 3 a 5 veces más caro que el input. La proporción no es arbitraria; es el costo computacional real de la generación secuencial.

Los tokens en caché son la tercera categoría: el proveedor guarda el estado interno (KV cache) de un prefijo estable y lo reutiliza entre solicitudes. Las lecturas de caché cuestan el 10% del precio de input; la escritura de caché de 5 minutos cuesta 25% extra. Como system prompt, herramientas y documentos adjuntos se repiten en cada llamada en cargas conversacionales y agénticas, el caching sistemático recorta el costo de input hasta 90% y el costo total entre 30 y 50% sin cambiar nada más. Es la palanca más subutilizada entre los equipos que migran a facturación por tokens sin disciplina.

Un costo se esconde dentro del output: los tokens de razonamiento. En modelos con extended thinking, el razonamiento interno generado antes de la respuesta visible se factura como output aunque el usuario nunca lo vea. Una respuesta visible de 500 tokens puede haber consumido 5.000 tokens invisibles de razonamiento, costando 10 veces más de lo que aparentaba; los equipos que habilitan thinking sin esta conciencia reciben facturas de 5 a 10 veces más altas de lo esperado. Conviene habilitar thinking solo donde la calidad extra justifica el costo.

Adónde van los tokens: la anatomía del consumo

Los tokens no se consumen de manera uniforme. El estudio Tokenomics (arXiv 2601.14470) midió el consumo en desarrollo de software multiagente: la implementación domina con 30 a 50% del total, trabajo legítimo, mientras el refinamiento toma 15 a 30% y es el villano habitual, iteraciones que una mejor planificación, criterios de parada explícitos o el modelo correcto habrían evitado. Entender la tarea cuesta 5 a 10% cuando se hace una sola vez, la verificación y las

pruebas toman 15 a 25%, la planificación 5 a 15%, y la comunicación entre agentes, otro 5 a 15% donde hay orquestación.

El efecto más subestimado es el multiplicador agéntico: en una sesión de múltiples turnos, todo el historial se reenvía como input en cada turno, y el input acumulado crece cuadráticamente. Las sesiones de más de 10 turnos empiezan a sentirlo; pasados los 20 turnos, el costo queda dominado por el input acumulado y el caché se vuelve la única palanca práctica. Cuatro estrategias doman el multiplicador, en orden de impacto: prompt caching; compactación estructurada, que comprime el historial en un resumen y reduce tokens entre 22 y 57% sin pérdida de precisión; limpieza de resultados de herramientas, que puede liberar más de 50K tokens en sesiones largas; y fan-out de subagentes, que exploran con contexto aislado y devuelven solo resúmenes, recortando 70 a 90% en workflows de exploración.

> DEFINICIÓN
>
> Las sesiones largas sorprenden a los presupuestos porque el input crece cuadráticamente con los turnos, no linealmente. Conviene tratar cada agente de larga duración como un costo que se compone: cachear el prefijo estable, compactar el historial, limpiar los resultados obsoletos y parar cuando la tarea está terminada.

Modelos y multiplicadores en el model picker de GitHub Copilot

La elección de modelo es la palanca de costo de primer orden: bien aplicada, reduce el costo promedio entre 50 y 70% sin pérdida percibida de calidad. En el model picker de GitHub Copilot, solo la familia Claude abarca un orden de magnitud a precios de lista de mediados de 2026: Haiku 4.5 a US$ 1 por millón de tokens de input y US$ 5 de output, Sonnet 5 a US$ 3 y US$ 15 (introductorio de US$ 2 y US$ 10 hasta agosto de 2026), Opus 4.8 a US$ 5 y US$ 25, y Claude Fable 5 a US$ 10 y US$ 50. Opus cuesta cerca de 1,7 veces Sonnet; Haiku es 5 veces más barato que Opus. Los modelos bundled vienen incluidos en los planes pagos sin consumir créditos adicionales.

La heurística tiene cinco niveles. Bundled cubre cerca del 70% del trabajo diario: completions, preguntas puntuales, refactorizaciones simples. Los baratos, clase Haiku 4.5, toman 10 a 15%: clasificación, extracción, boilerplate a escala. El workhorse, Sonnet 5, toma otro 10 a 15%: debugging no trivial, refactorizaciones multiarchivo, code review profundo, el default cuando bundled no alcanza. Los premium, Opus 4.8 y Claude Fable 5, deberían ser 1 a 3% del trabajo, reservados a arquitectura, debugging distribuido y decisiones irreversibles, y solo después de que el workhorse resultó insuficiente. Long-context entra cuando el problema necesita más de 100K tokens de contexto.

Desde la transición del 1 de junio de 2026, GitHub Copilot factura este consumo en GitHub AI Credits, donde un crédito corresponde aproximadamente a un dólar de uso a precio de lista. La aritmética es directa: una llamada con 8K de input y 1K de output en Sonnet 5 cuesta US$ 0,039; un equipo de 50 desarrolladores haciendo 1.000 llamadas así por día gasta unos US$ 1.950 diarios, US$ 42.900 al mes en 22 días hábiles, cerca de US$ 515 mil al año. Bajo la facturación plana por request, todas esas llamadas costaban lo mismo; bajo tokens reflejan trabajo real. Por eso estos fundamentos son ahora una habilidad de presupuesto, no una curiosidad.

Referencias

Fuentes primarias para tokenización, anatomía del consumo y precios de modelos.

- Neural Machine Translation of Rare Words with Subword Units, Sennrich, Haddow y Birch, ACL 2016, el artículo que adaptó BPE a NLP y resolvió el problema del vocabulario abierto.
- Tokenomics, token consumption distribution in multi-agent SDLC systems, arXiv 2601.14470, la distribución empírica detrás de la anatomía del consumo.
- GitHub Docs, Available Copilot models, la lista autoritativa de modelos en el model picker de GitHub Copilot.
- Anthropic, Models and pricing, los precios de lista de la familia Claude: input, output y caché.
- Anthropic Docs, Prompt caching, la mecánica de lecturas y escrituras de caché detrás del precio de lectura del 10%.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PARTE II, DISCIPLINAS TÉCNICAS

Disciplinas técnicas: contexto, caché, agentes y eficiencia.

Cuatro disciplinas separan a los equipos que pagan el precio completo de los que pagan una fracción por el mismo trabajo: ingeniería de contexto, prompt caching, diseño de agentes y estrategias token-efficient, aplicadas bajo la lente de GitHub Copilot.

Ingeniería de contexto: curar con cirugía

La ingeniería de contexto es la disciplina de seleccionar el conjunto más pequeño de tokens de alta señal que maximiza la probabilidad del resultado deseado. La intuición común, que más contexto siempre ayuda, es errónea. La curva que relaciona volumen de contexto y calidad de respuesta es cóncava con pico: los tokens irrelevantes tienen primero un efecto neutro y luego negativo, un modo de fallo llamado context rot. Los equipos que dominan la disciplina pagan entre 30 y 50% menos por el mismo trabajo, con calidad igual o mejor.

Cuatro técnicas la hacen operativa. La compactación, donde el modelo comprime su propio historial en un resumen estructurado, logró una reducción media de 22,7% de tokens en SWE-bench Lite con Claude Haiku 4.5, con 0% de pérdida de precisión. La limpieza de tool results ataca al mayor consumidor en flujos agénticos: leer un archivo de 2.000 líneas puede inyectar 30.000 tokens, así que conserve solo los últimos 5 a 10 resultados. La memoria persistente saca los hechos estables de la ventana, con compresión de 11x demostrada sobre 4.182 conversaciones. Subagentes con fan-out y síntesis central recortan la exploración en 70% o más. En Copilot esto aterriza en AGENTS.md y .github/copilot-instructions.md: un AGENTS.md con curaduría humana redujo el tiempo de ejecución en 28,64% y los tokens en 16,58%, mientras que uno generado por LLM rindió 3% peor que no tener archivo.

Prompt caching: la palanca del 90%

El prompt caching almacena el estado de atención calculado sobre su prompt para que las solicitudes con el mismo prefijo de tokens se salten ese trabajo. La caché opera por hash de prefijo, byte a byte: cualquier cambio en cualquier byte la invalida desde ese punto en adelante. La regla es arquitectónica: contenido estable primero, contenido dinámico después. System prompt, definiciones de herramientas y contexto estático, como el contenido de AGENTS.md, van arriba con cache_control; el historial de conversación y el mensaje actual quedan al final, nunca en caché. Hay mínimos: 1.024 tokens de prefijo cacheable en modelos clase Sonnet y Opus y 2.048 en Haiku 4.5.

La economía es simple. Una lectura de caché cuesta el 10% del input regular. Una escritura de caché con el TTL por defecto de 5 minutos cuesta 25% más, y cada solicitud reactiva el TTL mientras la sesión sigue activa. En cargas conversacionales, con hit rates típicos de 70 a 85% después del primer turno, es posible ahorrar hasta el 90% del costo de input sin cambiar una sola línea de prompt. GitHub Copilot en VS Code gestiona esta caché automáticamente; usted lo ayuda manteniendo estables .github/copilot-instructions.md y AGENTS.md, cambiados en pull requests y no ad hoc.

Antipatrones que rompen la caché

Cinco errores recurrentes destruyen el beneficio. Un timestamp dentro del prefijo en caché cambia el hash en cada solicitud: usted paga la prima de escritura de 25% cada vez y nunca lee. cache_control en un bloque que cambia, como el mensaje del usuario, hace lo mismo. Reordenar bloques entre sesiones produce un miss completo. La variación cosmética, un espacio doble o una palabra intercambiada en un prompt armado inline, lo invalida todo en silencio; mantenga los system prompts en archivos versionados. Un prefijo por debajo del tamaño mínimo se ignora silenciosamente.

DEFINICIÓN

La lectura de caché cuesta el 10% del input regular; la escritura de caché de 5 minutos cuesta 25% más. Con un prefijo estable, hasta el 90% del costo de input desaparece sin tocar una línea de prompt. El prerrequisito es disciplina: estable primero, dinámico después, nada volátil dentro del prefijo en caché.

Diseño de agentes: el default es workflow

La taxonomía de Anthropic separa los workflows, donde el código decide la secuencia y el LLM hace partes del trabajo, de los agentes, donde el modelo dirige su propio proceso y el uso de herramientas. Un workflow tiene costo predecible; un agente puede costar desde US$ 0,30 hasta más de US$ 5 en la misma tarea. Alrededor del 70% de los casos empresariales se resuelven con workflows a una fracción del costo, así que el default es workflow y la agencia debe justificarse. Cinco patrones componibles cubren el espacio: prompt chaining, routing, paralelización, orchestrator-workers y evaluator-optimizer. Reserve los agentes verdaderos, en Copilot el agent mode y el Coding Agent, para espacios amplios donde la secuencia varía por problema.

Cuando la agencia se justifica, restrínjala. Agentes personalizados definidos en .agents/ deben seguir responsabilidad única, ser read-only por defecto, llevar allowlists explícitas de herramientas y declarar presupuestos como max_iterations y max_tokens_per_session. Aplique el mismo tratamiento al GitHub Coding Agent en la nube: limite tokens e iteraciones por tarea, planifique con un modelo más fuerte y ejecute con uno más barato, y exija aprobación humana en cada PR. Los servidores MCP están entre las mayores fuentes de overhead de tokens; una definición esbelta de unos 80 tokens vence a una verbosa de 500. Hooks alrededor de las llamadas a herramientas, con caché de resultados repetidos y truncado de salidas grandes, reducen el costo agéntico en 30 a 50%.

Estrategias token-efficient y el model picker

Menos es más. Recortar el 39% del cuerpo de una skill y el 48% de las descripciones mejoró la calidad funcional en 2,8%: los prompts compactos concentran la señal que la atención realmente alcanza. El progressive disclosure aplica la misma idea estructuralmente: un núcleo pequeño siempre cargado, con checklists detallados y ejemplos traídos bajo demanda. El diseño esbelto de herramientas completa el conjunto: nombres cortos, descripciones de una línea, pocos parámetros, resultados por rango en lugar de volcados completos, JSON denso en lugar de prosa. Todo bucle evaluator-optimizer necesita criterios de parada explícitos: máximo de iteraciones, umbral de score, delta mínimo de mejora y presupuesto de tokens.

La última disciplina es el enrutamiento por complejidad, y en Copilot vive en el model picker. Usar un modelo premium por defecto cuesta en promedio de 3 a 5 veces más; un paso de clasificación que cuesta unos US$ 0,00005 por llamada reduce el costo promedio en 50 a 70% sin pérdida percibida de calidad. Mapee los niveles a los precios de mediados de 2026: trabajo rutinario a Haiku 4.5, a US$ 1 de input y US$ 5 de output por millón de tokens, trabajo complejo a Sonnet 5, a US$ 3 y US$ 15, trabajo experto a Opus 4.8, a US$ 5 y US$ 25, y solo el razonamiento de frontera a Fable 5, a US$ 10 y US$ 50. Cada elección premium consume AI Credits con su propio multiplicador: el picker es un instrumento de presupuesto, no una preferencia. Codifique la política en custom prompts en .github/prompts/, con modelo explícito y patrones applyTo.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PARTE III, EL CAMBIO

El cambio de junio de 2026: de premium requests a GitHub AI Credits.

Las PRUs fueron retiradas, la facturación por uso está activa y un dólar en créditos compra un dólar de trabajo a precio de lista. Qué cambió, qué decidir y cómo llevar a un equipo por la transición.

Qué cambió el 1 de junio de 2026

El 1 de junio de 2026, GitHub Copilot retiró las Premium Request Units, la unidad de facturación por solicitud de los modelos premium, y migró todos los planes de pago a GitHub AI Credits, facturación basada en uso en la que un dólar en créditos corresponde a cerca de un dólar de uso al precio de lista de la API de cada modelo. Un AI Credit equivale a US$ 0,01, y cada llamada premium consume créditos en proporción a los tokens de input, output y caché procesados. El cambio estructural cabe en una frase: bajo PRUs, el costo era por intención; bajo tokens, es por trabajo realizado. La causa, según Mario Rodriguez, CPO de GitHub, es arquitectónica, no comercial: una pregunta rápida y una sesión autónoma de horas costaban lo mismo, y ese subsidio implícito dejó de ser sostenible.

El precio base de los planes no cambió. Copilot Pro sigue en US$ 10 por mes con 1.000 AI Credits incluidos, Pro+ en US$ 39 con 3.900, Business en US$ 19 por usuario con 1.900 y Enterprise en US$ 39 por usuario con 3.900. Las code completions inline, Next Edit Suggestions y Copilot Chat con modelos bundled como GPT-4.1 y GPT-5 mini siguen incluidos y no consumen créditos. Dos cambios merecen atención especial. Primero, el fallback automático desapareció: con créditos en cero y la política de paid usage deshabilitada, los modelos premium dejan de funcionar, lo que convierte el presupuesto en control de continuidad, no solo de costo. Segundo, Copilot Code Review ahora factura en dos dimensiones, AI Credits por el modelo y minutos de GitHub Actions por el runner. Los créditos

se reinician el día 1 de cada mes a las 00:00 UTC y no se acumulan: se usan o se pierden.

> DEFINICIÓN
>
> El ancla primaria de este capítulo: el 1 de junio de 2026, GitHub Copilot retiró las PRUs y migró los planes de pago a GitHub AI Credits, cambio anunciado en el blog oficial de GitHub como GitHub Copilot is moving to usage-based billing. Un dólar en créditos compra aproximadamente un dólar de uso a precio de lista. Todo lo demás se deriva de ese único hecho.

Tres caminos para suscriptores anuales de Pro y Pro+

Quien se suscribió a Pro o Pro+ anual antes del 1 de junio de 2026 recibió una transición diferenciada: permanece en PRUs hasta el vencimiento del plan, pero con multiplicadores bastante mayores. Claude Opus 4.7 pasó de 7,5x a 27x y Claude Sonnet 4.6 de 1,25x a 4,5x, ambos un aumento de 3,6x, mientras que GPT-5.3 Codex y Gemini 3.1 Pro pasaron de 1x a 6x. La capacidad real cae en la misma proporción: 100 sesiones con Opus por mes se vuelven unas 28, y 100 con Codex, unas 17. GitHub ofreció tres caminos, cada uno con plazo en mayo.

Camino A, mantener el plan anual hasta el vencimiento

Cero acción requerida y el precio anual se mantiene, pero la capacidad premium real cae de 3,6x a 6x y no hay visibilidad granular de costos, porque la facturación sigue en PRUs. Tiene sentido para quien trabaja casi siempre en modelos bundled como GPT-4.1 y GPT-5 mini, usa premium solo de forma ocasional o tiene un plan anual que vence en menos de tres meses.

Camino B, convertir a mensual

GitHub ofrece créditos prorrateados por el tiempo restante del plan anual. Un suscriptor de Pro+ anual que pagó US$ 390 y tiene cuatro meses restantes recibe US$ 130 como 13.000 AI Credits de bono y entra a junio con el Pro+ mensual activo

más el bono, 16.900 créditos en total; desde julio valen los 3.900 créditos mensuales estándar. El plazo de conversión era el 25 de mayo. Es el camino para power users que dependen de premium y quieren visibilidad a nivel de token desde el primer día.

Camino C, cancelar con reembolso prorrateado

Cancelar antes del 20 de mayo activaba un reembolso proporcional al tiempo no utilizado, US$ 130 en el mismo ejemplo de cuatro meses restantes. Recupera valor y deja al usuario libre para elegir otro proveedor, al costo de perder el acceso a Copilot, salvo que vuelva al plan mensual. Sirve para quien deja el ecosistema Copilot o pierde el acceso al cambiar de empresa. El mapa por persona es directo: el dev que vive en el bundled toma el Camino A, el power user de premium el B y quien se va el C.

Migrar un equipo: la ventana de 16 semanas

Para las organizaciones, la transición es un proceso de unas 16 semanas, desde el bill preview que GitHub puso a disposición a principios de mayo de 2026 hasta el costo estable en septiembre. Mayo fue descubrimiento y preparación: analizar el bill preview en Billing Overview, identificar a los mayores consumidores, decidir el destino de los planes anuales antes de los plazos del 20 y el 25 de mayo, activar la política de premium request paid usage con un tope del 20% al 30% sobre el consumo histórico, configurar alertas al 50%, 75%, 90% y 100% del presupuesto, designar un dueño explícito de FinOps y entrenar a los tech leads antes del 1 de junio. Esa política es la configuración más importante: habilitada, el trabajo continúa más allá de los créditos incluidos como overage controlado; sin ella, los modelos premium se detienen al agotarse los créditos.

Junio es el D-Day más la estabilización: monitorear el consumo a diario en la primera semana, validar las alertas y aplicar los quick wins, ruteo de modelos por complejidad y prompt caching en las cargas más pesadas. Julio es optimización sistemática, con showback semanal a los tech leads, prompts personalizados y observabilidad enterprise. Agosto es maduración, y septiembre entrega el primer mes completo de facturación realista. El bill preview marca la urgencia: una proyección igual o menor que la factura actual en PRUs significa que la disciplina ya existe; un aumento del 30% al 100% señala premium en workflows agénticos largos

y pide ruteo y caching inmediatos; por encima del 100%, en general es Opus en agent mode sin disciplina y exige corrección antes de que se vuelva un riesgo serio. Los equipos que aplicaron esta disciplina capturaron una reducción del 25% al 45% frente al uso sin disciplina. Quien actuó temprano capturó ahorros; quien esperó pagó overage.

Referencias

Fuentes para el cambio de junio de 2026, los GitHub AI Credits y los caminos de migración.

- GitHub Copilot is moving to usage-based billing, GitHub Blog, abril de 2026, el anuncio oficial que retiró las PRUs e introdujo los GitHub AI Credits.
- About billing for GitHub Copilot, GitHub Docs, precios de los planes, AI Credits incluidos y la política de paid usage.
- GitHub Copilot documentation, GitHub Docs, tablas de precios por modelo y multiplicadores de los planes anuales, siempre verifique los valores actualizados.
- FinOps Foundation, el marco detrás del cronograma de migración, el showback y la propiedad del presupuesto en este capítulo.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PARTE IV

Gobernanza y FinOps: opere el gasto en IA como un recurso de nube.

Presupuestos, alertas y atribución por sesión, agente y repositorio en GitHub Copilot, y la economía de Azure AI Foundry que hace viable la gobernanza a escala enterprise.

Inform, Optimize, Operate: el ciclo FinOps aplicado a Copilot

Sin datos, optimizar es adivinar. La FinOps Foundation define tres fases iterativas. Inform es visibilidad granular: quién usa IA, cuánto, con qué modelo, en qué feature. El stack Microsoft es concreto: Copilot emite telemetría vía OpenTelemetry hacia Application Insights, Azure Cost Management exporta datos en el estándar FOCUS, el Microsoft FinOps Hub los procesa, y Power BI o Grafana los visualizan. Optimize convierte visibilidad en oportunidades con payback estimable: prompts ineficientes, modelos mal elegidos, falta de caché, equipos sin capacitación. Operate institucionaliza el ciclo: reuniones recurrentes, dashboards de KPI vivos, ajustes de presupuesto basados en datos.

La madurez se mide en meses, no en días. El mes 1 establece la línea de base: cuánto consumimos hoy, y dónde. Los meses 2 y 3 entregan los primeros ahorros con quick wins. Los meses 4 a 6 institucionalizan la mejora continua. Del mes 6 en adelante, el showback evoluciona a chargeback y las decisiones pasan a estar guiadas por datos.

Cinco métricas para instrumentar desde el primer día

El costo por 1K tokens dice si el precio promedio es justo para su mix; la meta es una tendencia decreciente: más caché, ya que las lecturas de caché cuestan el 10% del precio de input, enrutamiento hacia bundled y context engineering. El costo por feature dice si la capacidad devuelve su ROI. El costo por usuario expone outliers: el 10% superior consume típicamente del 40 al 50% del gasto, y el 1% superior por sí

solo del 10 al 15%. La tasa de aciertos de caché debe superar el 60% en workloads conversacionales. La razón output/input es saludable entre 0,1 y 0,3; por encima de 0,5 es una señal roja.

Presupuestos, alertas y atribución por sesión, agente y repositorio

Hay cuatro niveles de control, del más suave al más invasivo. Las alertas notifican al 50, 75 y 90% del presupuesto mensual; son señal para reaccionar, no parada. Los caps interrumpen los modelos premium al 100% mientras los bundled siguen: con un cap mensual de US$ 5.000 en la política de premium requests de GitHub Copilot, los desarrolladores pierden Sonnet, Opus y Gemini Pro hasta el mes siguiente, pero conservan GPT-4.1 y GPT-5 mini. Los límites duros detienen todo y deben ser raros: presupuestos fijos de POC o contratos con clientes. La detección de anomalías es el nivel proactivo: se compara el consumo con un histórico móvil de 30 días y se marca cualquier día por encima de 3 sigma. Un drill-down típico va del día anómalo a los mayores usuarios, luego a las sesiones de un usuario, y encuentra un Coding Agent que corrió 50 veces en loop por un bug.

Nada de esto funciona si cada AI Credit no es atribuible. Cada solicitud debe llevar modelo, modo (ask, edit, agent o plan), tokens por tipo, costo en USD, AI Credits e identificadores de usuario, equipo, feature, sesión y organización o repositorio. Esa atribución alimenta el showback, que muestra a cada equipo el costo generado sin cobrar, y el chargeback, que debita de verdad el centro de costos. Secuencie: tech leads en los meses 1 a 3, engineering managers en los meses 4 a 6, público entre equipos después del mes 6. Pase a chargeback solo con 6 o más meses de showback confiable y tagging consistente, en transición gradual: 25, 50, 75 y 100% en cuatro meses.

DEFINICIÓN

El comportamiento cambia cuando el número se ve. El showback crea visibilidad sin fricción financiera; el chargeback convierte la elección de modelo en decisión económica. Menos de 3 meses de showback, más del 10% de llamadas sin tag o equipos aún resistentes significan chargeback prematuro, sea cual sea la presión de finanzas.

La economía de Azure AI Foundry a escala enterprise

GitHub Copilot resuelve la codificación en el IDE; Azure AI Foundry resuelve los workloads de producción y la gobernanza regulatoria. Es un complemento, no una alternativa, y la elección es por workload, no por organización. Foundry trae un catálogo con más de 1.800 modelos de Anthropic, OpenAI, Meta, Mistral y otros, autenticación con Microsoft Entra ID, RBAC integrado, logs de auditoría en Azure Monitor, aislamiento de red con private endpoints y certificaciones como HIPAA, SOC 2 Type II, ISO 27001, FedRAMP High, GDPR y LGPD. La codificación se queda en Copilot, los workloads regulados y el RAG sensible van a Foundry, las POC rápidas usan la API directa.

PTU versus PAYG: la decisión de facturación con mayor impacto

PAYG cobra por token, sin mínimo ni compromiso, para carga variable o con picos. PTU, Provisioned Throughput Units, reserva capacidad: se paga por capacidad, no por token, con throughput garantizado y latencia consistente. La regla: debajo de US$ 5 mil por mes en un modelo, PAYG siempre; entre US$ 5 mil y US$ 30 mil, depende de la latencia; sobre US$ 30 mil, evalúe PTU en serio; sobre US$ 100 mil, PTU casi siempre gana. La escala lo cambia todo: 50 millones de tokens por mes en GPT-5 cuestan cerca de US$ 500 por mes en PAYG. Con 5.000 millones de tokens por mes, la factura PAYG llega a US$ 50 mil por mes, y las PTU reservadas con un 30% de descuento anual resultan un 75% más baratas.

La gobernanza tiene precio, y es pequeño

Los filtros de Azure AI Content Safety no son gratuitos, pero son baratos frente al riesgo: moderación de texto a US$ 0,40 por 1.000 llamadas, prompt shields a US$ 0,30, protected materials a US$ 0,60 y groundedness a US$ 0,75. Para 100 mil llamadas por mes con todos los filtros, cerca de US$ 200 por mes, un error de redondeo frente a un incidente de compliance. Vigile el logging de prompts y completions completos, que puede salir caro; defina retención. El export FOCUS deja la facturación de GitHub Copilot y el consumo de Foundry en los mismos datos de Azure Cost Management, y el CFO lee un dashboard, no dos.

Referencias

Fuentes para el ciclo FinOps, los niveles de control y la economía de Foundry.

- FinOps Framework, FinOps Foundation, las fases Inform, Optimize, Operate.
- FOCUS Specification, FinOps Foundation, el esquema abierto de costo y uso detrás del export unificado.
- Microsoft FinOps Toolkit, Microsoft, el FinOps Hub open source con plantillas de Power BI y queries.
- Azure Cost Management Documentation, Microsoft, presupuestos, alertas y exports FOCUS.
- Azure AI Foundry Documentation, Microsoft, catálogo de modelos, content safety y gobernanza enterprise.
- Provisioned Throughput Concepts, Microsoft Learn, dimensionamiento de PTU y modelo de precios.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 PARTE V, APLICACIÓN PRÁCTICA

Patrones aplicados y antipatrones.

Ocho patrones implementables mañana, los antipatrones que deshacen el ahorro y los árboles de decisión que vuelven reflejo la disciplina de tokens.

Ocho patrones, un objetivo

La teoría sin implementación no ahorra tokens. Esta parte del playbook aterriza los capítulos anteriores en ocho patrones concretos para GitHub Copilot y las cargas agénticas a su alrededor. Cada patrón resuelve un problema de costo específico y se componen entre sí: el enrutamiento decide qué modelo responde, el caché decide cuánto del input se vuelve a cobrar, el diseño de agentes decide cuánto contexto carga cada paso, y el diseño de herramientas decide el overhead fijo de cada turno.

Enrute por complejidad, planifique con el modelo premium

El patrón 1 es el enrutamiento dinámico. Los equipos configuran el premium como default en Copilot y pagan precio premium por preguntas que un modelo incluido en el plan resolvería. La solución es un clasificador en dos etapas: heurísticas estáticas por palabra clave, sin costo, y un modelo barato como fallback dinámico cuando ellas no deciden. Impacto: reducción de 40-70% en el costo promedio sin pérdida percibida de calidad. El patrón 4 es plan y execute: planificar la tarea grande con el modelo premium y ejecutar con el modelo de trabajo. Según la fuente, esto resulta 64% más barato que correr todo en el premium, con la calidad de plan del premium.

Caché en capas, exacto y semántico

El patrón 2 es el caché jerárquico en tres capas: caché exacto indexado por hash, caché semántico por similitud de embeddings y el prompt cache del proveedor por debajo. Los cachés exactos fallan ante variaciones cosméticas; la capa semántica captura preguntas que significan lo mismo. La tasa de acierto combinada llega a 60-90%, recortando 50-80% del costo de input. El patrón 5 es el caché semántico

standalone: en FAQs y preguntas recurrentes agrega 20-40% de acierto sobre un caché exacto.

Aísle el contexto, especifique antes de generar

El patrón 3 es el fan-out de subagentes. Un agente líder que explora una base de código grande acumula 800K+ tokens de contexto. La salida: subagentes con contexto aislado y herramientas de solo lectura, cada uno devolviendo un resumen de máximo 2K tokens que el líder sintetiza. Impacto: reducción de 70-90% en flujos de exploración. El patrón 8 es el agente personalizado con Spec-Kit: un agente de Copilot en .agents/ que lee una CONSTITUTION, una SPECIFICATION en notación EARS y un IMPLEMENTATION_PLAN antes de escribir código. La estructura reemplaza el retrabajo: 40-60% menos iteraciones de refinamiento.

Componga herramientas, mantenga el MCP ligero

El patrón 6 es la composición de herramientas. Diez herramientas granulares con unos 200 tokens de definición imponen 2.000 tokens de overhead en cada turno; tres compuestas con defaults sensatos cubren el mismo espacio y recortan 60-70% del overhead de system prompt. El patrón 7 la extiende a los servidores MCP de Copilot: nombres cortos, descripciones de 5-10 palabras, 2-4 parámetros esenciales, resultados paginados y recursos lazy-load. Un servidor ligero cabe entero en unos 250 tokens de definición, frente a 1000+ en los verbosos.

> DEFINICIÓN
>
> El número a recordar: combinados en cargas agénticas enterprise, estos ocho patrones entregan una reducción de 40-70% del costo total. No hay bala de plata: enrutamiento, caché, aislamiento de contexto y disciplina de herramientas se acumulan en cada solicitud que hace Copilot.

Los antipatrones que deshacen el ahorro

El catálogo lista diez antipatrones recurrentes de pilotos enterprise. El cementerio de agentes: agentes creados como prueba de concepto, nunca retirados, que siguen consumiendo tokens. Todo es un agente: flujos deterministas implementados como

agentes por flexibilidad futura, cuando el workflow gana en el 70% de los casos y convertirlos de vuelta típicamente ahorra 50-80%. Premium como default: el sesgo de que premium siempre es mejor. El system prompt monolítico: 30K tokens con todas las convenciones de la empresa, que además rompe el caché; la corrección: progressive disclosure y un AGENTS.md curado de menos de 500 líneas. Y la explosión de tool results: sesiones que acumulan 600K tokens de lecturas crudas porque las herramientas devuelven archivos enteros.

La segunda mitad: ausencia de criterios de parada, en lugar de combinar máximo de iteraciones, meta de score, delta de mejora y presupuesto de tokens. El caché que nunca cachea, acierto bajo 10% porque un timestamp o un bloque reordenado rompe el prefijo estable. Mal uso del long context, confundiendo lo que cabe con lo que vale la pena, cuando una llamada de 1M de tokens cuesta $10. Multiagente sin coordinación, donde la comunicación entre agentes consume 30% de los tokens. Y falta de instrumentación: el costo real llega como sorpresa a fin de mes. GitHub Copilot suma sus variantes: modo Agent para preguntas simples que el Ask con modelo incluido responde por 5-10x menos, workspace settings sin versionar que hacen variar el costo 3-5x entre equipos, y prompts atrapados en snippets locales en lugar de .github/prompts/.

La matriz de modos en VS Code

El antídoto práctico es una matriz de modos: Ask con modelo incluido para preguntas, Edit para cambios locales como docstrings y renombres, Agent con el modelo de trabajo para features completas y migraciones multiarchivo, Plan con el modelo premium para arquitectura y tradeoffs irreversibles. Planificar hace pocas llamadas y cuesta poco incluso en el premium; ejecutar hace muchas, y por eso merece el modelo de trabajo.

Tres frameworks de decisión y el rollout

Framework 1, qué modelo: modelos incluidos en el plan para autocompletado y preguntas cortas; long context solo cuando el contexto excede 100K tokens; el tier barato para clasificación y extracción; el modelo de trabajo como default para refactorización y debugging; el premium solo cuando el de trabajo falló o el problema es genuinamente complejo. Regla práctica: el modelo de trabajo cubre el

70% de los casos no incluidos en el plan, y el premium queda para el 5-10% genuinamente difícil. Framework 2, workflow o agente: si la secuencia de pasos es conocida y estable, use un workflow; agregue agencia solo cuando la descomposición realmente varía por input.

El framework 3 es el orden de optimización por ROI: enrutamiento por complejidad primero, luego prompt caching, fan-out de subagentes, limpieza de tool results, diseño ligero de herramientas, auditoría de system prompt, criterios de parada y compactación estructurada al final. Tras 8 semanas de esa secuencia, la fuente reporta reducción típica de 40-70% sobre la línea base. El rollout corre en cuatro fases: descubrimiento del estado actual; quick wins como el default del workspace en el modelo de trabajo y topes de gasto con alertas, con reducción esperada de 25-40%; disciplinas técnicas por un 10-20% adicional; y operación FinOps continua con dashboards y detección de anomalías.

Referencias

Fuentes de los patrones, los antipatrones y los frameworks de decisión.

- Building effective agents, Anthropic, la base del Framework 2.
- Effective context engineering, Anthropic, la lógica del fan-out de subagentes.
- Spec-Kit, Spec-Driven Development, GitHub, la estructura del patrón 8.
- Model Context Protocol Specification, el protocolo del servidor MCP ligero del patrón 7.
- GitHub Docs, Best practices for Copilot at scale, la base operacional de las fases del rollout.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps

01 APÉNDICES

Apéndices, el material de referencia.

Cuatro apéndices sostienen este playbook: prompts optimizados, una tarjeta de referencia de precios, plantillas de reportes FinOps y el código completo. Este capítulo condensa cada uno en lo que necesitas en el momento de uso.

Apéndice A, prompts optimizados

El Apéndice A reúne las plantillas de prompt probadas en los pilotos de América Latina en 2025 y 2026. Los principios de diseño son compactos y repetibles: declara el rol en una línea, enumera severidades o pasos en lugar de describirlos, fija el formato de salida con un schema JSON o una instrucción como solo el docstring, y delega las convenciones del proyecto a AGENTS.md en lugar de repetirlas en cada prompt. Un revisor de código escrito así cabe en unos 50 tokens y reemplaza las versiones verbosas de más de 500 tokens que los equipos escriben por instinto, sin pérdida de calidad de revisión.

El mismo apéndice incluye definiciones de herramientas ligeras para read, search y run_tests, además de custom prompts para .github/prompts/ cuyo frontmatter fija el modelo en el model picker de GitHub Copilot por tarea: revisión profunda de código en un modelo workhorse, diseño de arquitectura en un modelo premium con extended thinking, explicación de código en un modelo bundled que no consume AI Credits. Completan el conjunto plantillas de hooks preToolUse y postToolUse, que agregan verificación de caché, un freno de presupuesto, validación de rutas, redacción de secretos y truncado de resultados, y una plantilla de AGENTS.md que dice a los agentes dónde vive cada cosa y qué no tocar.

Apéndice B, tarjeta de referencia de precios

La tarjeta de referencia usa los precios de lista de Anthropic de mediados de 2026, por millón de tokens, los mismos detrás de las opciones del model picker de Copilot.

Claude Fable 5 cuesta $10 de input y $50 de output. Opus 4.5 a 4.8 cuestan $5 de input y $25 de output. Sonnet 5 cuesta $3 y $15, con precio introductorio de $2 y $10 hasta agosto de 2026. Sonnet 4.6 cuesta $3 y $15, y Haiku 4.5 cuesta $1 y $5. Dos reglas de caché completan la tarjeta: la lectura en caché cuesta el 10% del precio de input, y la escritura de caché de 5 minutos cuesta el precio de input más 25%. Para Sonnet 4.6 eso significa $0.30 por millón de tokens leídos del caché y $3.75 por millón escritos.

Del lado del consumo, el conversor del apéndice trata un AI Credit como aproximadamente un centavo de dólar de precio de lista. La capacidad por plan es la siguiente: Copilot Pro, a $10 al mes, incluye 1,000 AI Credits; Pro+, a $39, incluye 3,900; Business incluye 1,900 por usuario y Enterprise, 3,900 por usuario. Los planes anuales todavía medidos en PRUs convierten con multiplicadores de entre 3.6x y 6x, así que revisa siempre la tabla exacta en docs.github.com antes de comprometer un presupuesto.

> DEFINICIÓN
>
> Si una copia del apéndice de precios todavía lista Opus a $15 de input y $75 de output, es anterior al reajuste de 2026: la lista actual de Opus 4.5 a 4.8 es $5 y $25. Los precios cambian; las reglas de caché del 10% para lecturas y más 25% para escrituras de 5 minutos se han mantenido estables. Verifica en docs.github.com y anthropic.com/pricing antes de publicar cualquier estimación.

Apéndice C, plantillas de reportes FinOps

El Apéndice C entrega dos cadencias de reporte. El reporte semanal operativo, para tech leads cada lunes, sigue el gasto total contra la semana anterior, el costo por mil tokens contra la meta por debajo de $0.005, el cache hit rate contra la meta por encima del 60%, los 10 mayores consumidores con costo promedio por sesión, las anomalías detectadas con sus notas de investigación y las alertas de presupuesto disparadas. El reporte mensual ejecutivo, para el CFO y el VP de Ingeniería, agrega costo por desarrollador, utilización del presupuesto, distribución por centro de costo

y por modelo, las victorias de optimización del mes, áreas de atención como usuarios outliers y el ROI del propio playbook.

Las plantillas vienen con las herramientas para alimentarlas: tres dashboards de Power BI (Executive Overview, Tech Lead Operational y FinOps Detail), seis consultas KQL para Application Insights que cubren costo por equipo, sesiones descontroladas por encima de $5, cache hit rate por feature, anomalías diarias más allá de tres desviaciones estándar, distribución de modelos por usuario y señales de alerta en la razón output/input, y un JSON de dashboard de Grafana listo para importar, con costo total, tendencia de consumo, un medidor de cache hit y una tabla de top usuarios.

Apéndice D, dónde vive el código completo

El Apéndice D centraliza la implementación referenciada a lo largo del playbook. El model router de producción clasifica cada solicitud en cinco tiers, bundled, cheap, workhorse, premium y long context, intentando primero una pasada estática por palabras clave, que es gratuita, y recurriendo a un clasificador dinámico en Haiku 4.5, que cuesta una fracción de centavo por llamada; lleva una cadena de fallback y emite cada decisión de ruteo a la telemetría. El caché jerárquico apila tres capas: coincidencia exacta en Redis, coincidencia semántica con umbral de similitud de 0.93 y el caché del propio proveedor.

Dos activos más cierran el apéndice: un setup de OpenTelemetry con contadores de tokens, costo y llamadas dimensionados por modelo, usuario, feature y equipo, que alimenta las consultas KQL y los dashboards del Apéndice C, y una plantilla de servidor MCP ligero cuyas tres definiciones de herramienta suman unos 250 tokens. Una plantilla de agente personalizado muestra los guardrails en forma de archivo: modelo fijado, lista explícita de herramientas, context_files, max_iterations de 25, presupuesto de sesión de 500,000 tokens y hooks pre y post tool. Los cuatro apéndices acompañan este playbook como archivos markdown, de A_appendix_optimized_prompts.md a D_appendix_complete_code.md, junto a los capítulos; cópialos a tu repositorio y adáptalos a tu stack.

Paula Silva

SOFTWARE GLOBAL BLACK BELT

paulasilva@microsoft.com

v1.0.0 · 2026-06-17

Building the future of software development with AI and Agentic DevOps