GITHUB COPILOT AGENTIC FRAMEWORK

Hooks: interceptando el ciclo de vida del agente.

Intercepta, valida y controla el comportamiento de agentes en cada punto del ciclo de vida. La capa de gobernanza que el agente no puede eludir: del pre-hook que bloquea al post-hook que encadena el siguiente agente.

AUTORA
Paula Silva

FUNCIÓN
Software Global Black Belt

DURACIÓN
45 a 60 minutos

FECHA
2026-06-11

AGENDA

Cinco partes, un modelo mental.

PARTE I	Concepto, qué son los hooks, y las dos analogías que destraban todo: middleware y línea de montaje
PARTE II	Tipos, pre-generation, post-generation, validación y hooks personalizados
PARTE III	Ciclo de vida, el flujo completo, exit codes, los 3 handlers y el chaining multi-agente
PARTE IV	Implementación, la configuración real, el security scanning, la mecánica de ejecución y dónde corren los hooks
PARTE V	Producción, tres escenarios reales, el impacto, la economía de tokens con AI Credits, y las best practices

PARTE

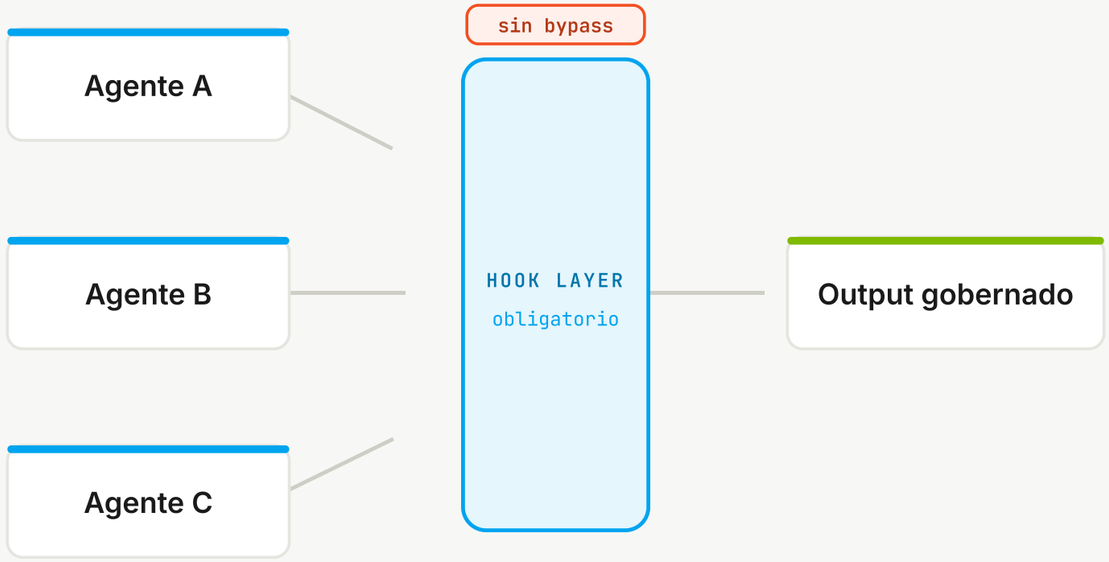
I

El concepto.

Los sensores silenciosos de la automatización agéntica. Duermen hasta ser activados, y el agente no puede eludirlos.

CONCEPTO · QUÉ SON LOS HOOKS

Mecanismos pasivos que responden a eventos del ciclo de vida del agente.



INTERCEPTORES DE EVENTO

Responden a eventos específicos en el ciclo de vida de un agente. Duermen hasta ser activados automáticamente.

BARRERAS OBLIGATORIAS

Los agentes NO pueden eludirlos. Los hooks son impuestos por el sistema, asegurando que toda acción siga estándares de gobernanza.

ZERO OVERHEAD

Los hooks solo consumen ciclos cuando realmente necesitan ejecutarse. Cuando duermen, el overhead es prácticamente cero.

Sin hooks, cada agente gestionaría su propia validación y manejo de errores. Los hooks centralizan estas responsabilidades, garantizando consistencia en toda la flota.

CONCEPTO · ANALOGÍA 01

Si conoces Express.js, ya conoces hooks.

MIDDLEWARE HTTP

```
app.use(authMiddleware);    // PRE
app.get("/api", handler);   // EXECUTION
app.use(loggerMiddleware);  // POST
```

Intercepta requests HTTP antes y después del handler principal. Valida auth, registra, transforma datos.

HOOKS DE AGENTE

```
pre_hook:  validate_permissions  // PRE
execution: agent_task            // EXECUTION
post_hook: quality_check         // POST
err_hook:  fallback_handler      // ON ERROR
```

Intercepta acciones de agentes antes y después de la ejecución. Valida permisos, escanea seguridad, garantiza calidad.

La misma intuición de middleware se aplica: interceptar, validar, transformar. Pero para agentes de IA en vez de HTTP requests.

CONCEPTO · ANALOGÍA 02

Hooks como línea de montaje: checkpoints de calidad en cada etapa.

SENSOR DE ENTRADA · PRE

Verifica si la materia prima cumple los criterios antes de entrar a producción. Si falla, bloquea.

→

ESTACIÓN DE TRABAJO · EXEC

El agente ejecuta su tarea principal usando habilidades y herramientas disponibles.

→

SENSOR DE SALIDA · POST

Verifica la calidad del producto final. Si aprueba, sigue. Si reprueba, reprocesa.

→

ALARMA · ERROR

Si algo sale mal, detiene la línea, registra el incidente y alerta a los supervisores.

Así como los sensores hacen una fábrica eficiente y confiable, los hooks hacen agentes gobernados, auditables y seguros.

PARTE

II

Los tipos.

Pre-generation, post-generation, validación y hooks personalizados. Cada tipo guarda un momento distinto del ciclo.

TIPOS · PRE-GENERATION

Los pre-generation hooks validan contexto y permisos ANTES de que el agente actúe.

hooks.json · pre-generation

```
{
  "hooks": {
    "pre-generation": [{
      "handler": "command",
      "script": "check_permissions.sh",
      "on_failure": "block",
      "timeout": 5000
    }]
  }
}
```

Valida permisos del usuario antes de cualquier generación.

Carga directrices del repo, como los archivos .instructions.md.

Verifica recursos y precondiciones del ambiente.

Puede BLOQUEAR la acción si falla. Es el único momento de decir no.

TIPOS · POST-GENERATION

Los post-generation hooks validan resultados DESPUÉS de que el agente completa la tarea.

hooks.json · post-generation

```
{
  "hooks": {
    "post-generation": [{
      "handler": "prompt",
      "action": "Review output quality",
      "min_coverage": 80,
      "next_agent": "SecurityAgent"
    }]
  }
}
```

- Valida la calidad de los resultados generados.
- Registra métricas y logs de cada ejecución.
- Notifica a sistemas interesados al concluir.
- **Dispara el siguiente agente vía chaining, creando pipelines multi-agente.**

TIPOS · VALIDACIÓN

Validation hooks: calidad, seguridad y compliance como portones.

Lint & Format	Security Scan	Compliance	Test Coverage
warn	block	block	block

LINT & FORMAT

Valida estilo de código, formato y estándares de nomenclatura automáticamente.

SECURITY SCAN

Escanea código generado en busca de vulnerabilidades, secrets expuestos y patrones inseguros.

COMPLIANCE CHECK

Verifica adherencia a políticas corporativas, LGPD, HIPAA y estándares regulatorios.

TEST COVERAGE

Garantiza que el código generado tenga cobertura mínima de tests antes de aceptar.

```
hooks:
  validation:
    - name: security-scan   handler: command   script: "semgrep --config auto ."          on_failure: block
    - name: lint-check      handler: command   script: "eslint --fix ."                   on_failure: warn
    - name: coverage-gate   handler: command   script: "jest --coverageThreshold=80"     on_failure: block
```

TIPOS · PERSONALIZADOS

Custom hooks: créalos a medida para necesidades específicas.

NOTIFICATION HOOK

Envía alertas a Slack, Teams o email cuando los agentes completan tareas críticas.

METRICS HOOK

Recolecta métricas de performance, costo y calidad para dashboards de observabilidad.

TRANSFORM HOOK

Transforma el output del agente antes de entregarlo al usuario: sanitiza, formatea, enriquece.

Extensibilidad: los custom hooks permiten que las organizaciones adapten el framework a sus necesidades únicas sin modificar el core del sistema.

PARTE

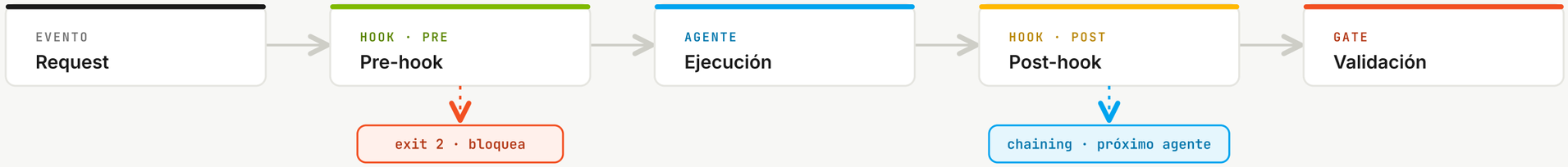
III

El ciclo de vida.

De la petición al output, todos los puntos de interceptación. Nada pasa sin verificación.

CICLO DE VIDA · EL FLUJO COMPLETO

De la petición al output: cinco puntos, cinco oportunidades de interceptación.



1 · Request

El evento llega al sistema: push, PR, comando del usuario o agendamiento cron.

2 · Pre-hook

Valida permisos, carga contexto, verifica precondiciones. Puede bloquear la ejecución (exit code 2).

3 · Ejecución

El agente ejecuta su tarea principal usando skills y tools provistos por el framework.

4 · Post-hook

Valida resultados, registra métricas, notifica sistemas. Puede disparar el siguiente agente vía chaining.

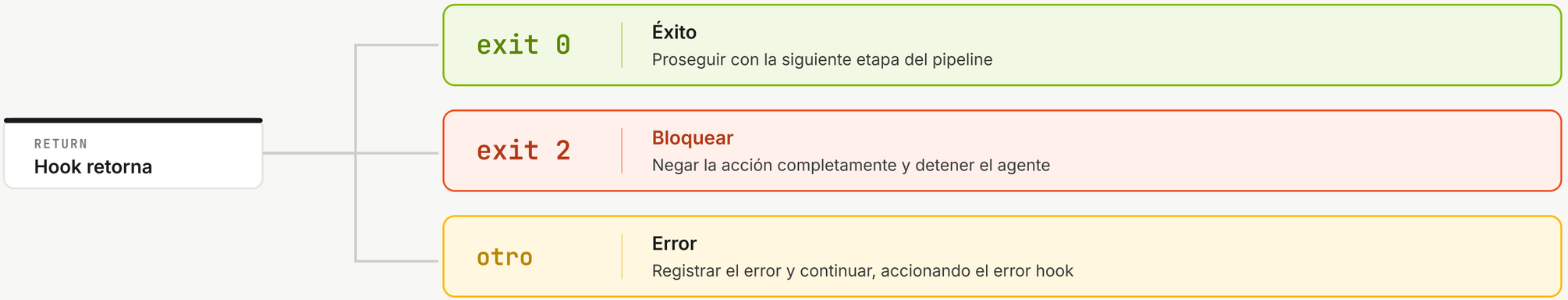
5 · Validación

Quality gates de seguridad y compliance antes de entregar. El output sale auditado y registrado.

Cada punto del ciclo de vida es una oportunidad de interceptación. Los hooks garantizan que nada pase sin verificación.

CICLO DE VIDA · EXIT CODES

Tres números lo controlan todo: 0 prosigue, 2 bloquea, otro registra.



La timeline: disparo, ejecución, validación. Éxito continúa, bloqueo detiene, error acciona el error hook. Es la semántica más importante del sistema.

CICLO DE VIDA · LOS 3 HANDLERS

Tres tipos de manipuladores, del script directo al subagente completo.

COMMAND

Scripts de shell directos que ejecutan acciones rápidas. Ideales para validaciones simples y ligeras.

```
BRANCH=$(git branch --show-current)
if [[ "$BRANCH" == "main" ]]; then
  exit 2  # Block
fi
exit 0    # Continue
```

PROMPT

LLMs que analizan y toman decisiones inteligentes. Usan razonamiento para evaluar contexto complejo.

```
"Analyze the diff for:
 1. Security vulnerabilities
 2. Breaking changes
 If critical: exit 2 (block)
 Otherwise: exit 0 (approve)"
```

AGENT

Subagentes completos con herramientas y habilidades. Ejecutan tareas complejas como revisión de código.

```
Agent: SecurityReviewAgent
 Skills: [code-review, vuln-scan]
 Tools:  [semgrep, codeql, gitleaks]
 Output: structured_report.json
```

Empieza con command para validaciones ligeras. Gradúa a prompt cuando necesites razonamiento, y a agent cuando la verificación sea una tarea completa.

CICLO DE VIDA · CHAINING

Hook chaining: el pipeline multi-agente de un pull request.

```
on_pull_request · chain

[event] pull request abierto por el dev o por el Coding Agent

[lint-check] eslint + prettier validan formato, bloquean si falla
[security-scan] semgrep y CodeQL escanean, exit 2 bloquea el merge
[test-generation] el agente genera y ejecuta tests, cobertura sobre 80%
[code-review] el agente de revisión analiza la lógica y comenta en el PR

todos los hooks pasaron: PR aprobado y merged
pipeline total: ~28s, en secuencia, parada inmediata si un blocking falla
```

CADA ETAPA AGREGA

Los post-hooks disparan el siguiente agente. El pipeline se vuelve una cadena donde cada eslabón agrega valor.

BLOCKING PRIMERO

Lint y security son blocking y van primero: falla rápido, antes de gastar en los agentes caros.

HUMANO FUERA DEL LOOP

28 segundos de pipeline automatizado contra 45 minutos de revisión manual por PR.

PARTE

IV

La implementación.

La configuración real, el hook de security scanning línea por línea, y dónde corren hoy los hooks de GitHub Copilot.

IMPLEMENTACIÓN · LA CONFIGURACIÓN

Una única configuración JSON define todos los hooks del repositorio.

.github/hooks/hooks.json

```
{
  "hooks": {
    "pre-commit": [{
      "name": "validate-syntax",
      "handler": "command",
      "script": "npx eslint .",
      "on_failure": "block",
      "timeout": 10000
    }, {
      "name": "check-secrets",
      "handler": "command",
      "script": "gitleaks detect --source .",
      "on_failure": "block"
    }],
    "post-commit": [{
      "name": "notify-team",
      "handler": "command",
      "script": "curl -X POST $SLACK_WEBHOOK",
      "on_failure": "warn"
    }]
  }
}
```

ANATOMÍA DE UN HOOK

Cada hook tiene nombre, handler, script, comportamiento en falla y timeout. Cinco campos, contrato completo.

BLOCK VS WARN

on_failure block para lo crítico (sintaxis, secrets). warn para lo informativo (notificación). Sepáralos siempre.

VERSIONADO EN GIT

En .github/hooks/*.json en el branch por defecto. Vale para todo agente del repositorio, heredado por el equipo.

IMPLEMENTACIÓN · SECURITY SCANNING

El hook de seguridad, línea por línea: tres escaneos, un veredicto.

security-scan.sh

```
#!/bin/bash
# Security scanning para código gerado por agente

# 1. Secrets expostos
gitleaks detect --source . --no-git
[ $? -ne 0 ] && exit 2  # BLOCK

# 2. Análise estática OWASP
semgrep --config "p/owasp-top-ten" --error .
[ $? -ne 0 ] && exit 2  # BLOCK

# 3. Auditoria de dependências
npm audit --audit-level=critical
[ $? -ne 0 ] && exit 2  # BLOCK

exit 0  # CONTINUE: todas as varreduras passaram
```

GITLEAKS

Detecta secrets y credenciales expuestas antes de que lleguen al commit.

SEMGREP

Escanea el código generado contra el OWASP top ten. Una vulnerabilidad crítica bloquea.

NPM AUDIT

Audita dependencias y frena vulnerabilidades de nivel crítico en la cadena.

El script puede ejecutar múltiples verificaciones en secuencia. Exit code 2 bloquea, exit code 0 continúa. Simple de auditar, imposible de saltar.

IMPLEMENTACIÓN · MECÁNICA DE EJECUCIÓN

La mecánica fina: secuencial, con timeout, y fail-secure donde importa.

`Secuencial`	Los hooks del mismo evento corren en el orden del array. El primer deny salta los demás.
`Bajo 5s`	La recomendación oficial de ejecución. El timeout por defecto es 30 segundos por hook.
`El timeout no derriba`	Si expira: el hook se termina, el evento se registra y el agente continúa, en la mayoría de los eventos.
`preToolUse es fail-secure`	La excepción: error, crash o timeout NIEGAN la tool en vez de dejarla pasar. Un hook roto nunca se vuelve puerta abierta.
`Nunca registres secrets`	Los prompts y argumentos de tool pueden contener datos sensibles. Redacta antes de persistir o reenviar.

En el CLI, disableAllHooks pausa la configuración sin borrar nada: útil en debug, en releases sensibles, o para que un contribuidor haga opt-out local.

IMPLEMENTACIÓN · DÓNDE CORREN

Los hooks de GitHub Copilot corren en todas las superficies del agente.

Admin del enterprise — **Organización** — **Branch por defecto** — **Todas las superficies** siempre activa

SUPERFICIE	ESTADO	OBSERVACIÓN
GitHub GitHub Copilot coding agent	GA	Agente en nube vía Issues y PRs, hooks en el branch por defecto
GitHub Copilot CLI	GA	En el terminal, con hooks personales extra en ~/.copilot/hooks
VS Code	Preview	En preview desde marzo de 2026, con 8 eventos
JetBrains IDEs	Preview	Agent hooks en preview desde marzo de 2026

Desde junio de 2026, los admins distribuyen hooks gestionados por todo el enterprise: la misma configuración, siempre activa, en cada cliente de GitHub Copilot de la organización.

PARTE

V

La producción.

Tres escenarios reales, el impacto medido con versus sin hooks, y las prácticas que separan a quien opera de quien apaga incendios.

PRODUCCIÓN · TRES ESCENARIOS

Tres equipos, tres problemas, el mismo mecanismo.

MARIA · DEVOPS, FINTECH

CI/CD con hooks

Branch protection en el pre-hook, semgrep y gitleaks como blocking, post-hooks disparando el pipeline completo de CI.

`Cero vulnerabilidades en producción en 6 meses`

ANA · TECH LEAD, E-COMMERCE

Code quality con hooks

ESLint y Prettier como blocking, un hook agent genera tests y valida cobertura sobre 80%, JSDoc obligatorio.

`Quality score de 72% a 96%, onboarding 60% menor`

SOFIA · SECURITY, HEALTHCARE

Compliance HIPAA

Un hook con LLM analiza cada diff por patrones HIPAA, cadena SAST + SCA + secrets + DAST, auditoría con hash y timestamp.

`100% compliance en 3 auditorías, review de 2h a 30s`

El patrón es el mismo: el pre-hook valida, el blocking escanea, el post-hook encadena, el custom hook mide. Solo cambia el dominio.

PRODUCCIÓN · EL IMPACTO

Con versus sin hooks: los números que cierran el caso.

SIN HOOKS

- Revisión manual de código: 45 minutos por PR.
- Vulnerabilidades detectadas tarde, ya en producción.
- Inconsistencia de estilo entre equipos y agentes.

→

CON HOOKS

- ✓ Pipeline automatizado: 28 segundos por PR.
- ✓ Vulnerabilidades bloqueadas antes del merge.
- ✓ 100% de consistencia de estilo, compliance continuo.

80x

MÁS RÁPIDO: DE 45MIN A 28S POR PR

0

VULNERABILIDADES EN PRODUCCIÓN EN LOS ÚLTIMOS 6 MESES

100%

COBERTURA DE AUDITORÍA: CADA ACCIÓN REGISTRADA

PRODUCCIÓN · ECONOMÍA DE TOKENS

Desde el 1º de junio, los tokens son dinero. Los hooks son el control de costo.

REGLA FUERA DEL CONTEXTO

Una regla en instructions ocupa el contexto y se cobra como input en cada request. La misma regla en un hook corre como shell: costo cero de token.

BLOQUEAR TEMPRANO ES BARATO

El deny del preToolUse corta el ciclo ejecutar, fallar, analizar, reintentar. Cada loop evitado es input y output que no llega a la factura.

RESPUESTA SIN LLM

En el CLI, un hook de userPromptSubmitted puede devolver la respuesta directo en stdout y saltarse el modelo entero. Cero tokens para respuestas deterministas.

OUTPUT MAGRO

El stdout del hook entra al contexto del agente. Un hook verboso agrega tokens cobrados. Silencioso en el éxito, específico en la falla.

Desde el 1º de junio de 2026, GitHub Copilot cobra por AI Credits medidos en tokens de input, output y caché, con 1 crédito valiendo un centavo de dólar.
Cada loop desperdiciado del agente ahora aparece en la factura.

PRODUCCIÓN · BEST PRACTICES

Cinco prácticas y un anti-pattern de quien corre hooks en producción.

Falla rápido	Pon los hooks blocking primero en la cadena.	`Lint → Security → Tests`
Timeout explícito	Define siempre timeout para evitar hooks colgados.	`"timeout": 10000`
Idempotencia	Los hooks deben producir el mismo resultado si se ejecutan múltiples veces.	`re-ejecutar = mismo output`
Registra todo	Cada ejecución de hook debe generar log auditable.	`JSON structured logs`
Blocking vs warning	on_failure block para críticos, warn para informativos.	`block | warn`
Anti-pattern	Nunca pongas hooks lentos (más de 30s) como blocking en flujos interactivos. Usa async hooks o muévelos a post-commit.	

RESUMEN

Seis conceptos clave para llevarse a casa.

HOOKS

Interceptores pasivos en el lifecycle del agente. Gobernanza centralizada.

PRE-HOOKS

Validan antes de la acción. Pueden bloquear la ejecución con exit 2.

POST-HOOKS

Validan resultados. Disparan chaining hacia el siguiente agente.

ERROR HOOKS

Capturan fallas, intentan fallback, notifican al equipo.

HANDLERS

Command (shell), Prompt (LLM), Agent (subagente completo).

CHAINING

Los hooks crean pipelines multi-agente. Cada etapa agrega valor.

CIERRE

El agente no puede eludirlo. Ese es el punto.

CONTACTO

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

PRÓXIMO PASO

Un hook blocking, hoy
un check-secrets con gitleaks, en tu repo, esta semana

.GITHUB/HOOKS/HOOKS.JSON · EMPIEZA AQUÍ

```
"pre-commit": [{
  "name": "check-secrets",
  "handler": "command",
  "script": "gitleaks detect --source .",
  "on_failure": "block"
}]
# y ningún secret vuelve a pasar
```

Publicado el 2026-06-11 · v2.1