Karajan.

El **Agentic Development Environment** nativo para macOS que dirige agentes de IA como una orquesta: del requerimiento en lenguaje natural al **pull request aprobado**, con usted al mando de cada compás.

- macOS nativo · Swift 6
- Multi-agente en paralelo
- GitHub · Azure DevOps · Local
- Doble aprobación humana
- Versionado y changelog automáticos

EL PRODUCTO

Su equipo de desarrollo, dirigido — no reemplazado

Karajan convierte un requerimiento escrito en español en software fusionado a `master`. Un analista IA lo descompone en tickets y tareas; ejecutores IA programan en paralelo, cada uno aislado en su propio worktree git; revisores IA auditan cada diff; y usted aprueba en las dos compuertas que importan: **la propuesta** y **el diff final antes del PR**.

Orquestación real

Tickets y tareas forman DAGs con precedencia declarada por el analista: lo independiente corre en paralelo, lo dependiente espera a que su base esté fusionada.

Git como sustrato

Cada tarea trabaja en su propio worktree y rama. La "comunicación" entre agentes es el merge; el artefacto que usted revisa es un diff consolidado, no una promesa.

Humano en el circuito

Doble aprobación: nada se ejecuta sin su visto bueno a la propuesta, y ningún PR se abre sin su aprobación del diff. La IA propone y verifica; usted decide.

Gates de calidad

Comandos por proyecto (build, tests, linters) que deben pasar en verde antes de que el ticket llegue a revisión. Configurables por proyecto, exit 0 = pasa.

Release automático

Al aprobar, Karajan sube la versión y antepone la entrada al changelog dentro del mismo PR — con lógica merge-train: N PRs abiertos reciben números únicos y se re-versionan solos si otro se fusiona primero.

Autorreparación

Conflictos de versión se curan solos; conflictos de código se devuelven a un agente resolutor con un clic. Si el daemon muere, la reconciliación de arranque repara todo estado a medias.

Lo que usted ve

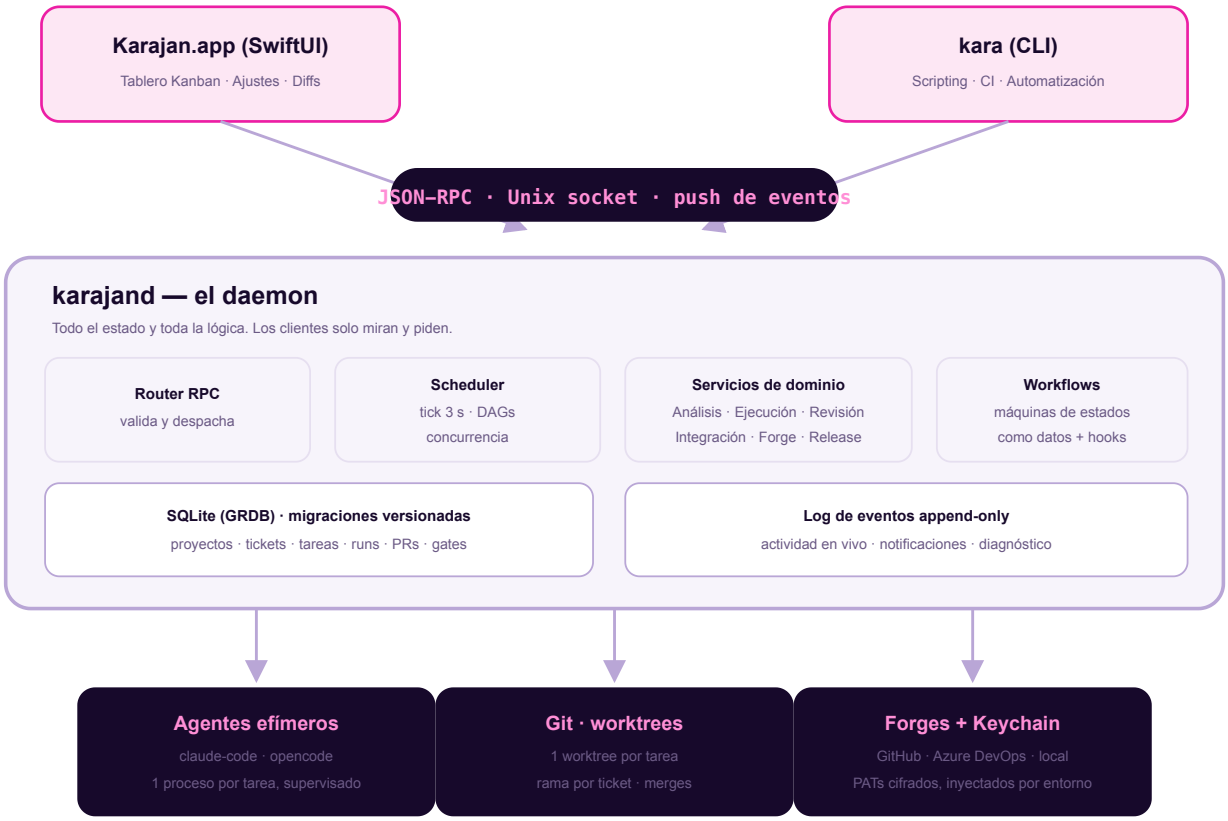
- ✓ **Tablero Kanban en vivo** por proyecto: del requerimiento nuevo al ticket hecho, con costos por tarjeta.
- ✓ **Transparencia total**: transcript de cada agente, diff de cada tarea, veredicto de cada revisión, motivo de cada bloqueo.
- ✓ **Costos en USD** por tarea, ticket y proyecto, con presupuesto por ticket y alertas.

Lo que no tiene que hacer

- ✓ Crear ramas, worktrees ni PRs a mano — ni acordarse del número de versión.
- ✓ Vigilar terminales: notificaciones nativas cuando algo necesita su decisión.
- ✓ Mantener su `master` local al día: se sincroniza solo al fusionarse cada PR.

BAJO EL CAPÓ

Arquitectura: un daemon con todo el estado, clientes sin ninguno



Workers desechables

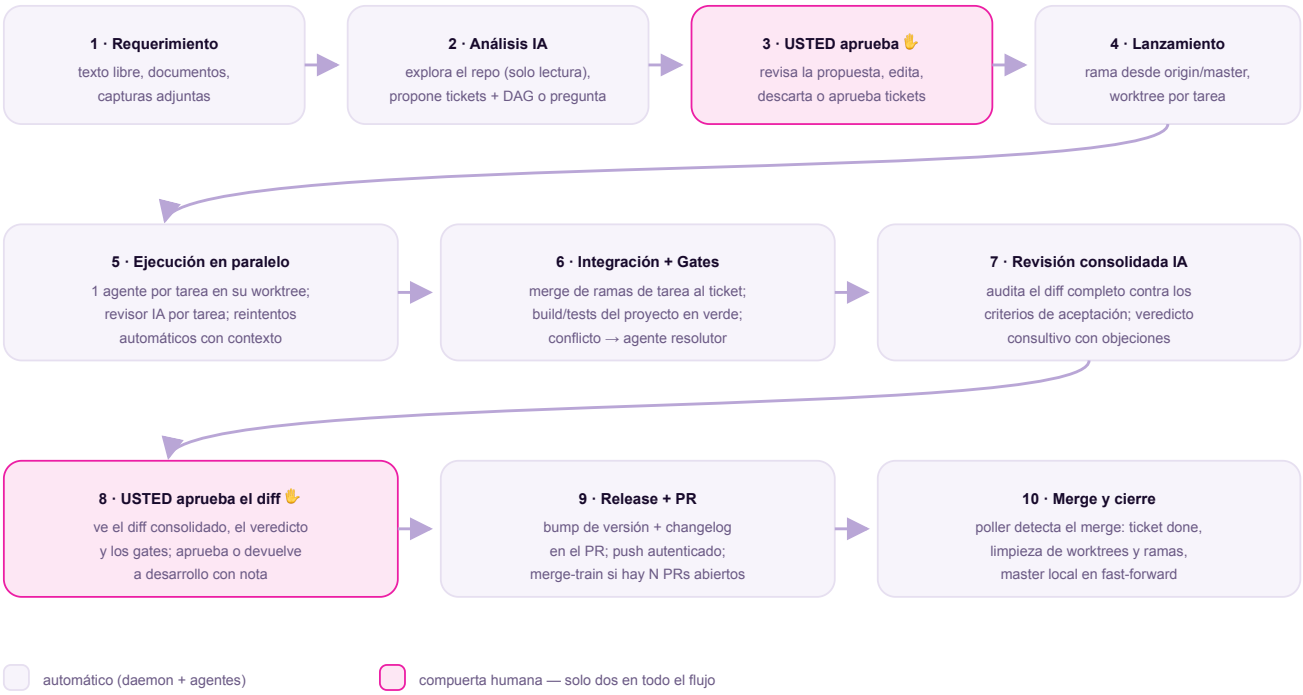
Ningún agente vive esperando: el daemon lanza un proceso por tarea con un prompt autocontenido, lee su transcript JSONL y lo termina. Matar el daemon nunca corrompe estado — al arrancar, la reconciliación repara todo lo que quedó a medias.

Estados como datos

Los workflows de ticket y tarea son datos configurables (estados, transiciones, hooks) en cascada global → proyecto. Toda transición se valida contra la máquina de estados; nada escribe estados directamente.

EL FLUJO

Del requerimiento al merge, sin perder el control



Precedencia entre tickets: si el analista declaró dependencias, los tickets aprobados esperan a que su base esté fusionada; los independientes corren en paralelo dentro del límite de con

Tip

Adjunte planillas, PDFs o capturas al requerimiento: el analista los ingiere y los commitea en `.karajan/adjuntos/` como contexto permanente del proyecto.

Tip

¿La propuesta quedó ambigua? El analista no inventa: devuelve preguntas de clarificación y el requerimiento espera sus respuestas antes de planificar.

LA INTERFAZ

Un tablero que se mueve solo — y le avisa cuando lo necesita

Todo a un clic desde la tarjeta

- ✓ **Detalle del ticket**: tareas con intentos y costo, diff por tarea, reporte del ejecutor, transcript en vivo del agente.
- ✓ **Motivo del bloqueo**: si algo se detuvo, ve la causa exacta y los archivos en conflicto — y el botón "*Resolver conflictos con agente*" al lado.
- ✓ **Veredicto IA + gates** antes de aprobar: resumen, objeciones con archivo:línea, y el resultado de cada gate.
- ✓ **Precedencia visible**: tarjetas con insignia de dependencias; "esperando" cuando el scheduler las retiene a propósito.

Organización y control

- ✓ **Carpetas en el sidebar** para agrupar proyectos (clientes, áreas, personal), con badges de actividad y alerta roja.
- ✓ **Ajustes (⌘,)**: agentes y modelos por rol, límites de concurrencia y reintentos, credenciales PAT (Keychain) con botón *Probar*, políticas de release.
- ✓ **Configuración por proyecto**: overrides de agentes, gates, presupuesto por ticket, credencial y release propios.
- ✓ **Actividad**: feed en vivo de todos los eventos, con push del daemon.

Consejo
Deje la app abierta mientras trabaja en otra cosa: las notificaciones nativas de macOS le avisan exactamente cuando hay una decisión suya pendiente (propuesta lista, revisión lista, ticket bloqueado, presupuesto excedido). El resto del tiempo, Karajan no lo interrumpe.

Consejo
Antes de aprobar un diff, use "*Ver diff del ticket*": es el merge consolidado real de todas las tareas — exactamente lo que llegará al PR, versión y changelog incluidos.

EL MOTOR

karajand: ingeniería para confiar en la autonomía

⚙️ Scheduler determinista

Un tick cada 3 segundos recorre cada proyecto: lanza tickets aprobados cuyas dependencias estén fusionadas, promueve tareas cuyo DAG esté satisfecho, llena la capacidad de agentes hasta su límite configurado y liquida tickets terminados. Sin colas mágicas: la base de datos ES la cola.

👁️ Supervisión de procesos

Cada agente corre bajo un supervisor: timeout de inactividad (sin output = colgado), kill del árbol completo de procesos, registro compartido para que un shutdown alcance a todos. Los transcripts JSONL se escriben en streaming — puede mirarlos en vivo.

🔁 Reconciliación al arrancar

Si el daemon murió con trabajo en vuelo, el arranque lo detecta todo: runs activos → fallidos y relanzables; análisis a medias → propuesta parcial retirada; gates huérfanos → re-integración; PR a medio crear → bloqueado con motivo. Nada queda colgado para siempre.

🔐 Credenciales de verdad

Los PAT de GitHub/Azure DevOps viven cifrados en el Keychain de macOS y se inyectan por variable de entorno a cada proceso (CLI del forge y git push), con credential helper inline. Nunca tocan disco en claro ni viajan de vuelta por RPC.

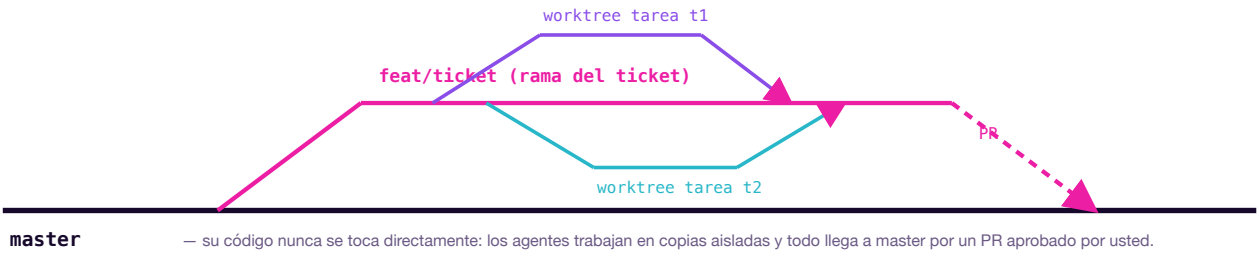
🚂 Merge-train de versiones

La versión se calcula al abrir el PR contando master Y los otros PRs abiertos: N PRs salen con números únicos consecutivos. Si otro se fusiona primero, el poller re-mergea master, re-versiona y actualiza el mismo PR — los conflictos de versión/changelog se curan solos.

🧪 Testeable por diseño

Runner mock, HOME de scratch, ejecutores git/CLI inyectables y backend de credenciales en archivo para CI: 248 tests automatizados cubren del parser de versiones al ciclo completo ticket → PR sin tocar red ni agentes reales.

La seguridad del modelo de aislamiento



— su código nunca se toca directamente: los agentes trabajan en copias aisladas y todo llega a master por un PR aprobado por usted.

EN LA PRÁCTICA

Casos de uso reales

🏢

Mantención de sistemas legados

Reportes, planillas Excel, formatos de ministerio: describa el cambio en español, adjunte la planilla de ejemplo, y reciba un PR con el formato replicado, versión subida y changelog al día. Ideal para carteras de sistemas .NET/PHP/Java con demanda constante de ajustes.

bugs · reportería · cambios normativos

🚀

Productos nuevos desde cero

"Comenzar con descripción": Karajan crea el repo, el README y el commit inicial, y el primer requerimiento levanta la arquitectura base (Docker, API, modelos). Los tickets con precedencia construyen el producto por capas, en orden.

greenfield · MVPs · prototipos serios

🧩

Features grandes, descompuestas

Un requerimiento complejo (multiempresa, auditoría, OCR…) se convierte en varios tickets con DAG: lo independiente avanza en paralelo con varios agentes a la vez, lo dependiente espera su base. Usted aprueba propuesta y diffs; Karajan administra el resto.

épicas · paralelismo · DAG de tickets

🔍

Auditorías y migraciones

"Analiza las diferencias entre estas dos ramas y pórtalas": el analista lee el repo completo y produce un plan verificable; los ejecutores portan módulo a módulo con criterios de aceptación explícitos y gates que compilan en cada paso.

refactors · portes · deuda técnica

Tips de los que ya lo usan a diario

Gates primero

Configure al menos un gate de build por proyecto (`dotnet build`, `swift test`…). Es la diferencia entre "el agente cree que funciona" y "compila en verde antes de que usted lo mire".

Presupuesto por ticket

Defina un tope en USD por ticket. Karajan le avisa al excederlo — los costos reales por tarea son visibles siempre, y un ticket típico de mantención cuesta centavos.

Criterios de aceptación ricos

Mientras más concreto el requerimiento (rutas, nombres de campos, ejemplos), mejores los criterios que genera el analista — y más dura la revisión IA que audita contra ellos.

Pruebe sus PAT

Tras configurar credenciales, use el botón *Probar*: verifica contra el forge real con la misma inyección que usará el PR. Verde ahí = push sin sorpresas.

Devuelva con nota

¿El diff no le convence? "Devolver a desarrollo" con una nota crea una tarea correctiva que hereda las objeciones de la IA más su comentario — la siguiente ronda llega corregida y re-revisada.

Confíe en el bloqueo

Un ticket bloqueado no es un error: es Karajan negándose a avanzar sin usted. El motivo exacto está en la tarjeta, y casi siempre hay un botón que lo resuelve con un agente.

FICHA TÉCNICA

Especificaciones y superficie de control

PLATAFORMA	
Sistema	macOS (Apple Silicon): app SwiftUI + CLI + daemon
Lenguaje	Swift 6.2, SPM, sin dependencias de UI de terceros
Persistencia	SQLite (GRDB) + log de eventos append-only
IPC	JSON-RPC 2.0 sobre Unix socket, push de eventos
Agentes	claude-code y opencode; modelo por rol (análisis / ejecución / revisión)
Forges	GitHub, Azure DevOps y modo local sin remoto
Secretos	PATs en Keychain; inyección por entorno
Aislamiento	1 worktree por tarea, 1 rama por ticket; limpieza automática
Calidad	Gates + revisión IA doble + doble aprobación humana
Release	Bump + changelog en el PR, merge-train, sync de master

La CLI en una mirada

```
# proyectos y requerimientos
kara project add ~/Repos/MiSistema
kara req new MiSistema "Agregar export a PDF…"
kara req answer 1a2b3c "usar wkhtmltopdf"

# tickets: aprobar, seguir, decidir
kara ticket list MiSistema
kara ticket approve 4d5e6f
kara ticket verdict 4d5e6f
kara ticket approve-review 4d5e6f   # crea el PR
kara ticket resolve-conflicts 4d5e6f

# operación
kara watch                # eventos en vivo
kara costs MiSistema
kara credentials set azure --org https://dev.azure.com/mi-org
kara config set-release --bump on --sync on
kara pr refresh
```

Automatizable
Todo lo que hace la app lo hace la CLI: scripting, cron, integración con otras herramientas. Misma fuente de verdad, mismo daemon.

Deje de escribir tickets que nadie ejecuta.

Escríbalos una vez, apruébelos dos veces, y véalos llegar a master — con versión, changelog y PR incluidos. **Karajan dirige; usted decide.**