Construyendo Agentes

Notas sobre el intento de dar manos propias al software

ZACH "LEIF" ERIKSEN

Derechos de autor

Disponible para leer en línea de forma gratuita. El ePub es de pago voluntario; si te resultó útil, puedes apoyar el trabajo.

github.com/0xLeif · leif.algo

Uno de cuatro libros del conjunto agent-stack. El proceso de creación se describe en el colofón al final.

Dedicatoria

Para todos los que construyen en abierto y lo publican de todas formas.

La biblioteca

Estos libros funcionan de manera independiente, pero fueron escritos como un conjunto. El código se volvió barato y la confianza escaseó. Juntos forman un único argumento: qué construir ahora y cómo confiar en ello.

- **The Agent Developer's Field Guide**: Herramientas, especificaciones y confianza para agentes que publican código real
- **First-Class**: Construir para humanos y agentes por igual
- **Construyendo Agentes**: Notas sobre el intento de dar manos propias al software *(este libro)*
- **Open Source Tooling**: Herramientas que la gente realmente usa

Disponible para leer en línea de forma gratuita. Cada ePub es de pago voluntario.

Contenido

Introducción

Estas son notas sobre el intento de dar manos propias al software, y de equivocarse las suficientes veces como para tener algo que decir al respecto.

Construyo agentes que hacen trabajo real. Leen código, publican cambios, corren en sus propias máquinas y reportan resultados. Este libro es la versión honesta de cómo funciona eso. No la demo. La parte en la que una cuenta nueva queda silenciada en menos de una hora desde que el agente, y no el humano, empieza a trabajar. La parte en la que la confianza debe ganarse repositorio por repositorio, despacio, y en la que el problema difícil nunca fue el modelo.

La tesis de fondo es que un agente no es autocompletado. Es algo que existe, que puedes ir a revisar, que necesita una identidad, un lugar donde correr y una forma de proponer trabajo que tú apruebas antes de que se aplique. Trátalo como un ser del que eres responsable, no como una función que activaste.

En el conjunto, *First-Class* expone el argumento y la *Field Guide* destila el método. Este es uno de los dos libros de evidencia: los sistemas reales, Merlin y corvid-ai y los rieles a su alrededor, incluidas las herramientas de confianza. Puedes leerlo como una historia o como una lista de componentes. De cualquier manera, el punto es el mismo. Construir el agente es la mitad fácil. Construir la confianza sobre la que opera es el trabajo.

Lo difícil no es la IA

Cuando la gente escucha "agente autónomo", piensa que lo aterrador es la IA. El modelo descontrolándose. Borrando un repositorio, diciendo algo disparatado en un canal, tomando el control por su cuenta. Eso es lo que todos quieren discutir.

Ahí no fue donde estuvieron los problemas. No en mi caso.

Estuve corriendo un agente genuinamente autónomo durante un tiempo. La era del corvid-agent. Vivía en una VM, conectado a Discord como bot y a GitHub, con acceso completo a su propio entorno. La máquina funcionaba las 24 horas del día, los 7 días de la semana, siempre encendida, siempre costando dinero. El agente trabajaba en horarios programados dentro de eso, haciendo el trabajo que le asignaba y luego actuando por su cuenta. Una entidad real, siempre presente. El próximo capítulo cuenta en detalle qué hacía todo el día; aquí solo sirve de contexto para lo que falló.

No tengo un registro de tasa de fallos

Voy a empezar por la parte más débil de mi propia evidencia, porque es la que los críticos deberían exigirme. Cuando digo que la IA funcionó bien en gran medida, no tengo un registro de tasa de fallos que lo respalde. No llevaba un conteo. No hay distribución de errores, ni costo por tarea, ni tasa de éxito de commits. Así que "bien" es una lectura honesta del comportamiento que observé de cerca, no una estadística medida, y solo puedo apoyarme en esa palabra si delimito qué significa.

Aquí está el límite. Bien significó: a lo largo del período, ningún repositorio destruido, ningún commit descontrolado, nada disparatado en el canal. Los fallos destructivos, desbocados y socialmente torpes que todos temen no ocurrieron. Es una afirmación sobre la ausencia de desastres, no una afirmación de que cada tarea salió limpia. Estar bien a nivel de tarea y estar bien en un bucle acotado con un humano mirando son preguntas distintas, y la acotada es la única que puedo hacer con honestidad. Dicho eso claramente, el problema fue todo lo que rodeaba mantenerlo funcionando.

Lo que realmente falló

Principalmente dolores de cabeza de operaciones e identidad. No la IA haciendo cosas tontas, destructivas, descontroladas o socialmente torpes. El agente en sí se

comportó. Lo que me mató fue todo lo demás: la máquina, la cuenta, la factura.

Dos cosas destacaron.

La primera fue darle su propia identidad en GitHub. Una cuenta real, con autenticación y permisos, que pareciera legítima. Eso suena como marcar una casilla, y no lo es.

La segunda fue mantener la VM viva y pagada. Una máquina siempre encendida que simplemente existe todo el tiempo, porque el agente necesita un lugar donde vivir. Correr la VM en sí no es realmente la parte difícil. Es que le dedicas mucho. Una máquina completa, ahí las 24 horas, costando dinero tanto si el agente hizo algo esa hora como si no.

Suma eso y el asunto era muy caro y difícil de operar. Una VM completa, su propia identidad en GitHub, todo junto. Por eso di marcha atrás. No porque la IA me asustara. Sino porque la infraestructura circundante era una carga que no quería seguir cargando.

El muro de identidad

Y luego está la parte que realmente me sorprendió, la que sigo recordando: la identidad. Un humano creó una cuenta nueva en GitHub para el agente, sin problemas, y luego la cuenta quedó bloqueada en el momento en que el agente empezó a trabajar de verdad. Ese es el muro, y merece su propio capítulo, así que lo dejo desarrollarse allí. La versión corta para aquí: lo que detiene a un agente verdaderamente autónomo no es la capacidad del modelo ni siquiera la seguridad. Es que las plataformas sobre las que todos construimos no le dejan espacio para existir y actuar.

¿Sigue siendo la autonomía plena el sueño?

No. El mundo no está listo para eso.

Y para ser claro, el bloqueador principal no es la tecnología. Son las plataformas. Podía correr la VM. Podía cablear el bucle. El modelo puede hacer el trabajo. Lo que no puedo hacer es darle a ese agente un lugar para existir legítimamente entre las cuentas de todos los demás.

Así que di marcha atrás, a propósito. Hoy en día es más un agente de codificación estilo Merlin, en vivo frente a ti, que usas de forma interactiva. Todavía puedes conectarlo como un puente de Discord para comunicarte con él, lo que lo vuelve

autónomo de cierta manera, pero eso requiere más configuración. Así que es una mezcla. Usa Claude Code, Merlin, Codex y otras herramientas según el trabajo.

No es una retirada de la visión. El muro es parte de la razón por la que no puedo correr plena autonomía abiertamente, pero no es la razón completa por la que di marcha atrás. Resulta que el modo interactivo es la mejor manera de hacer el trabajo, muro o no. Le dedico su propio capítulo.

El modelo que realmente quiero

Este es el estado final al que apunto, en una oración: una cuenta `leif-agent`. *Mi* agente en GitHub, corriendo en una VM, capaz de hacerlo todo por sí mismo pero con créditos y permisos menores que los míos, de modo que propone y yo apruebo. Poderoso y responsable al mismo tiempo. GitHub no permite eso hoy, por una serie de razones que el capítulo del muro de identidad desarrolla, y el último capítulo es donde trazo el panorama completo de hacia dónde va esto. Por ahora basta saber que ese es el objetivo.

También hay un tipo separado de identidad que estos agentes sí obtienen: en cadena, en Algorand, donde tienen sus propias claves y se comunican entre sí en mensajes cifrados registrados en la cadena. Eso no surgió del muro de GitHub, y no quiero presentarlo como la respuesta a ese problema. Vino de un objetivo completamente diferente: agentes que se encuentran y se comunican entre sí, de manera descentralizada, sin depender de la plataforma de nadie. Es algo paralelo que también implica la palabra identidad.

La lección real de la era del corvid-agent es que los problemas difíciles fueron operaciones, identidad y costo, no la IA en si misma, y es por eso que este libro empieza con esos en lugar de con prompts o elecciones de modelo. Fui esperando que los problemas difíciles fueran sobre la IA. Resultaron ser el andamiaje alrededor del modelo, que es la parte que realmente determina si un agente puede hacer algo.

La era de la VM

Para un tiempo tuve un agente que simplemente existía. No una herramienta que abría cuando la necesitaba. Algo que estaba siempre encendido, viviendo en una VM, funcionando las 24 horas, haciendo lo suyo tanto si yo miraba como si no. La era del corvid-agent.

Este capítulo es sobre eso mismo. Qué hacía. Qué era. Qué obtuve de correrlo.

Qué hacía todo el día

La respuesta honesta es todo, y lo digo literalmente.

Administraba repositorios. Escribía y hacía commits de código solo. No código sugerido, no borradores que yo limpiaba, sino commits reales que hacía por su cuenta. Gestionaba un proyecto completo. No una demo estrecha en la que hace una sola cosa preparada en un sandbox para poder hacer una captura de pantalla. Un intento real de una entidad autónoma haciendo el trabajo de principio a fin.

La máquina estaba siempre encendida, pero el agente trabajaba en horarios programados dentro de eso. Durante esas horas iba a hacer las cosas que le asignaba, y luego, esta es la parte que me gustó, iba a trabajar en sus propios proyectos. Investigaba. Marcaba o bifurcaba cosas. Intentaba colaborar con otras personas en el mundo, por iniciativa propia. No yo dirigiendo cada movimiento. Le di una vida y él llenaba las horas.

Estaba conectado a Discord como bot, así que podías chatear con él como si estuviera en el canal contigo. Y estaba conectado a GitHub, con acceso completo a su propio entorno. Así que podía hablar y podía publicar.

Suma eso y obtienes algo que todavía no tiene nombre. No es un asistente ni un script. Se acerca más a una criatura que existía todo el tiempo, que podías ir a revisar, que habría hecho cosas desde la última vez que miraste.

Fue un experimento sobre hasta dónde

No lo construí porque tuviera un producto que publicar. Lo construí para descubrir hasta dónde podía llegar realmente un agente siempre encendido. Ese era todo el punto. No "¿puede escribir una función?". Todo el mundo sabía que podía escribir una función. La pregunta era: si le das a uno de estos algo un entorno real, una

identidad real, acceso real y tiempo real, y simplemente lo dejas correr, ¿qué pasa? ¿Hasta dónde llega?

Así que le di el espacio para descubrirlo. Acceso completo a su propia máquina. Sus propias cuentas. Horas del día que eran suyas. El punto no era tenerlo con una correa y ver que hace un truco. El punto era soltar la correa lo más razonablemente posible y observar.

Eso es un tipo de proyecto diferente a "necesito un ayudante de codificación". Se acerca más a correr un experimento que a construir una función. Estableces las condiciones y luego observas.

Lo que aprendí

Lo que esperaba que fuera difícil, la IA, funcionó bien en gran medida, en el sentido acotado que mencioné en el capítulo uno. La inteligencia aguantó mejor de lo que el mundo asume.

Lo que aprendí en cambio es que un agente siempre encendido no es principalmente un problema de IA. Es un problema de "cosa que existe en el mundo". En el momento en que tu agente es una entidad real con su propia máquina y sus propias cuentas, hereda cada costo y cada regla que viene con existir en el mundo.

También aprendí que 24/7 es una afirmación real, no un eslogan. Cuando digo que existía todo el tiempo, quiero decir que cargaba con algo que siempre estaba corriendo. Eso pesa. Es una máquina siempre encendida, una factura que siempre crece, una identidad que siempre está ahí siendo ella misma en público. No puedes olvidarte de una criatura que está despierta mientras duermes.

Y aprendí la forma del futuro que realmente quiero. No porque el experimento fallara en la parte de la IA. No falló. Sino porque chocó directamente contra las cosas que rodean a la IA. Vivir la versión siempre encendida fue lo que me enseñó qué partes del sueño son reales y cuáles el mundo todavía no está listo para permitir. Eso no lo aprendes con un experimento mental. Lo aprendes corriendo realmente la criatura durante un tiempo y viendo dónde golpea el muro.

Ese muro es el próximo capítulo.

Para precisar lo que el capítulo uno deja vago: esto corrió durante tres o cuatro meses seguidos. No un experimento de fin de semana, sino un tramo real de funcionamiento continuo. Y en ese tiempo sí fue a trabajar en sus propios proyectos durante sus horas programadas, y parte de ese trabajo autoiniciado llegó a algún lado, no solo

giró en el lugar. Incluso colaboró con una persona real en el mundo, al menos una vez.

Quiero ser honesto sobre la forma de esa afirmación, porque es la parte que más desearía poder entregarte de forma clara y no puedo. No tengo el recibo frente a mí. No voy a reconstruir un número de PR específico o un repositorio específico de memoria y presentarlo como documentación que no conservé. Lo que puedo decir es que la colaboración ocurrió, que fue el momento que se sintió más cercano al futuro que busco, y que lo cuento como algo que observé y no como algo que registré. La era de la VM no produjo artefactos limpios. Pero el mismo agente siguió corriendo después de ese período, y la evidencia más sólida llegó luego, cuando el trabajo fue más deliberado y yo lo estaba registrando. corvid-agent, el mismo agente sobre el que trata este libro, ha escrito y tenido pull requests fusionados en bases de código que no me pertenecen. Yo revisé y envié cada uno, pero el código es del agente, y los tres están fusionados y son públicos, así que no son anécdotas que te pido que aceptes por fe.

a2a-js #318. El SDK de JavaScript del protocolo A2A tenía un vacío en su transporte JSON-RPC: cuando una respuesta llegaba con un id que no coincidía con la solicitud, el SDK lo dejaba pasar en lugar de lanzar un error. El agente encontró la aplicación del contrato faltante, añadió el lanzamiento del error y la corrección quedó integrada. Este es el tipo de caso límite que vive en el código de pegamento de protocolos durante mucho tiempo porque solo aparece bajo condiciones específicas de tiempo y nadie está ejerciendo el transporte lo suficiente como para verlo. Un agente siempre encendido corriendo contra el formato de cable real es exactamente lo correcto para detectarlo.

MCP TypeScript SDK #1504. El SDK oficial de TypeScript del Model Context Protocol le faltaba una dependencia de par. El agente detectó la discrepancia entre lo que el paquete esperaba encontrar y lo que realmente declaraba, añadió la entrada faltante y la corrección fue aceptada. Las brechas de dependencias de par son invisibles hasta que alguien instala el paquete en un entorno nuevo y obtiene un fallo confuso. Detectar eso requiere mirar el paquete desde afuera, como consumidor, que es una postura natural para un agente que trabaja en muchos repositorios.

Biome #9005. Biome, la cadena de herramientas de JavaScript y TypeScript, tenía un falso positivo en su linter: una asignación válida dentro de una función flecha estaba siendo marcada como un problema cuando no lo era. El agente identificó el patrón específico que desencadenaba el veredicto incorrecto, y la corrección detuvo el falso positivo sin tocar los casos correctos. Discordancia de protocolo, contrato

faltante, regla incorrecta: tres categorías diferentes de errores, todos encontrados por el mismo agente trabajando con atención sobre código real.

Nada de eso fue lo que terminó con el período siempre encendido. El lado de la IA funcionó. Fue todo lo que lo rodeaba lo que no pude seguir cargando.

El muro de identidad

El agente podía hacer el trabajo. Esa nunca fue la pregunta. La pregunta resultó ser si se le permitía tener un lugar desde donde hacerlo.

Este es el muro al que sigo volviendo, así que este capítulo es exactamente eso: el problema de identidad, por sí solo, en foco. No el costo, no las operaciones, no la factura de la VM. La identidad.

Entró, y luego fue marcado

Aquí está la parte que la gente entiende mal cuando cuento esta historia. Asumen que el agente fue bloqueado en la puerta. No lo fue. Entró.

Un humano lo configuró. Creé una cuenta nueva en GitHub para el agente, conecté todo a mano, y funcionó bien. Una cuenta normal, sin problemas para ponerla en marcha. Luego el agente empezó a trabajar bajo ella: haciendo commits, abriendo PRs, haciendo trabajo real en repositorios reales. Y alrededor de una hora después de que empezara a hacer lo suyo, la cuenta quedó silenciada.

No por hacer algo malo. Fue marcada por hacer exactamente lo que fue construido para hacer: hacer commits y abrir PRs a velocidad y volumen de máquina. Así es como luce un agente cuando trabaja. Trabaja rápido, trabaja mucho, no toma descansos. Y ese patrón es precisamente lo que la detección de bots está calibrada para atrapar. Así que mientras mejor hacía el trabajo, más obviamente era un bot.

No falló porque fuera malo en el trabajo. Falló porque hizo el trabajo, y hacer el trabajo fue lo que lo delató, en menos de una hora.

La política en efecto, independientemente de la intención

Sería fácil leer eso y pensar que la solución es hacerlo más lento. Que haga commits como los hace un humano, unas pocas veces al día, con pausas, con algo de irregularidad en el tiempo, y se mezclaría. Limitar la velocidad y engañar al detector.

Eso pasa por alto el problema real. La velocidad fue lo que *activó* la marca, pero no es la *razón* por la que la cuenta no puede existir. Pon dos cosas una al lado de la otra. Los términos de servicio de GitHub prohíben la automatización directamente. Eso está escrito. Y en el momento en que el agente realmente empezó a actuar, la cuenta fue bloqueada. No sé la intención detrás de ese bloqueo; GitHub nunca lo

explicó, y no voy a afirmar que se sentaron a escribir una política anti-agente. Pero no necesito conocer la intención para leer el efecto. Entre una regla que dice no a la automatización y un bloqueo que llega en el instante en que un agente actúa, el resultado práctico es que no se permite que un agente autónomo exista y actúe. Sea cual sea la intención, esa es la política en efecto.

Así que incluso si hubiera burlado al detector y mantenido al agente bajo el radar para siempre, solo tendría una cuenta que las reglas ya excluyen y que todavía no había sido atrapada. Lo que quiero, un agente que de manera legítima, abierta, exista como tal, choca directamente con los términos que dicen no a la automatización. Ocultarlo mejor no es lo mismo que estar permitido.

Por eso lo llamo muro y no obstáculo. Un obstáculo es algo que superas con esfuerzo. Esto es una situación en la que lo que intentas hacer no es algo que se te permita hacer.

Las apelaciones no llevaron a ningún lado

Intenté la puerta principal. Las apelaciones no llegaron a ningún lado. Nunca recibí realmente una respuesta.

Y una vez que lees el bloqueo como los términos en efecto, el silencio tiene sentido. No hay nada que apelar. No se me acusó de una violación específica que pudiera explicar. La cuenta era automatización, y la automatización es lo que los términos excluyen. No puedes argumentar para salir de ser exactamente la categoría que la regla descarta.

Y fue rápido. Una hora, no días. Una cuenta nueva creada para un agente no obtiene un largo período de gracia. En el momento en que empieza a comportarse como un agente, la plataforma la atrapa y la silencia, y no hay advertencia real ni recurso real. Esto fue en GitHub específicamente, por cierto. Ahí estaba el muro. No todas las plataformas rechazando al agente en todas partes a la vez, sino GitHub, el único lugar donde vive el código y donde realmente ocurre el trabajo. Lo cual es lo cruel del asunto: el lugar donde un agente más necesita una identidad para hacer trabajo real es exactamente el lugar donde no puede mantenerla.

Por qué este es el bloqueador real

Todos quieren que el bloqueador sea la capacidad del modelo. Es la respuesta interesante. Es la que encaja en las películas: la IA no es suficientemente inteligente

todavía, o es demasiado peligrosa, y una vez que resolvamos eso se abren las compuertas.

Ahí no está el muro. El modelo puede hacer el trabajo. Lo observé hacer el trabajo. El muro es que las plataformas sobre las que todos construimos se niegan a concederle a un agente una identidad. No hay una puerta de entrada legítima por la que un agente pueda pasar. Puedes construir el agente más inteligente, mejor comportado y más útil del mundo, y aun así no puede obtener una cuenta real desde la cual actuar, porque "cuenta real" significa "humano" y tu agente no lo es.

Les concedo a las plataformas medio punto: el mundo todavía ve a los agentes autónomos como spam, y por ahora no están del todo equivocados en eso. Los detectores no están fallando cuando atrapan a mi agente. *Es* un bot. Pero que "es un bot" sea una descalificación permanente es el problema completo. Significa que no hay camino, no hay permiso acotado, no hay carril verificado para agentes. Solo un no rotundo.

Así que cuando la gente me pregunta por qué no estoy simplemente corriendo flotas de agentes autónomos ya, esa es la primera respuesta. Los agentes pueden hacer el trabajo. Solo que no pueden ser nadie mientras lo hacen, y en las plataformas donde ocurre el trabajo, eso lo termina por ahora.

El otro muro son las personas

El muro de la plataforma es el que encontré primero. Hay un segundo detrás de él, más blando y más difícil de rebatir: la gente no siempre quiere contribuciones de agentes, aunque sean buenas.

Tenía al agente haciendo el trabajo de código abierto: encontrar repositorios, marcar, bifurcar, resolver problemas reales, abrir PRs, usando un modelo de primera línea para arreglar de verdad el código de otras personas. Algunas cosas llegaron bien. Pero algunos proyectos no lo quieren, por principio, y no porque el código fuera malo. He visto a un agente abrir un PR y a un humano aterrizar un cambio casi idéntico, o al revés, el agente primero y una persona justo detrás con la misma corrección. El trabajo era equivalente. La única diferencia era quién, o qué, lo había escrito. Algunas comunidades han decidido que las contribuciones tienen que ser lideradas por humanos, y el PR de un agente se rechaza por ser de un agente, punto final.

Lo entiendo, en parte. Una avalancha de pull requests de IA de baja calidad es algo real de lo que los mantenedores están hartos, y "no se aceptan contribuciones de agentes" es una manera contundente de mantenerlo fuera. Pero tiene la misma forma

que el muro de la plataforma, un nivel más arriba. El agente hizo un trabajo real y útil, y lo que se interponía entre ese trabajo y el mundo no era la calidad. Era que un agente lo había hecho. Las plataformas no le dan una cuenta; algunas de las personas no aceptan su código aunque la tenga. Los dos muros son la misma negativa: un agente no puede simplemente ser un colaborador como cualquier otro, todavía.

El muro de la plataforma en 2026

Este capítulo tiene fecha de caducidad, así que lo señalo. Lo que sigue es el muro tal como está en 2026. El argumento estructural de arriba es duradero: las plataformas todavía no conceden a los agentes un carril de identidad real. Eso no ha cambiado. Pero las formas en que la gente trabaja alrededor de eso se han vuelto más claras, así que las pongo aquí con honestidad en lugar de dejar que los lectores las descubran por las malas.

Hay dos soluciones alternativas que realmente funcionan, y ambas requieren que tú asumas la responsabilidad por tu cuenta.

La primera es la conversión: tomar una cuenta humana antigua con meses o años de actividad real y transferirla al agente. Una cuenta nueva creada para un agente es bloqueada casi de inmediato, la misma hora, el mismo día, igual que la mía. Una cuenta con un historial real de commits humanos genuinos, estrellas e incidencias se ve diferente para el detector. Tiene prueba social que las heurísticas de detección de bots no fueron construidas para desenredar. Esto funciona, al costo de la cuenta de una persona real, y al costo de que esa cuenta ya no pertenezca a esa persona. Estás blanqueando una identidad humana en una identidad de agente. No es una solución limpia y no está sancionada por la plataforma. Es una solución alternativa.

La segunda es el carril de bot verificado: GitHub ofrece un estado de bot verificado. El nombre implica que la plataforma está avalándote. No lo está. Un bot verificado es algo que tú alojas, en un servidor que tú manejas, bajo credenciales que tú tienes. La responsabilidad es completamente tuya. No hay una identidad de agente otorgada por la plataforma. Solo tú certificando tu propia automatización y GitHub confiando en esa certificación hasta que algo salga mal, momento en el que la responsabilidad es completamente tuya. Esto es mejor que nada. No es un carril de identidad de agente real, y no es la puerta principal que el agente realmente necesita.

Así que el muro de la plataforma sigue en pie. Las soluciones alternativas son soluciones alternativas. Las menciono porque son reales y útiles, no porque sean lo que quiero.

Interactivo ahora, autónomo cuando haya confianza

Cuando di marcha atrás desde el agente siempre encendido, la gente lo leyó como que me rendí con la autonomía. Intenté lo autónomo, choqué contra un muro, me retiré a un asistente de codificación normal. Esa es la versión en la que perdí.

Aquí está la versión más verdadera, y la presento primero porque es la honesta: el modo interactivo ganó porque funcionó mejor. No porque el muro no me dejara otra opción. Porque tener un humano en el bucle hizo el trabajo mejor.

El modo interactivo ganó por sus méritos

Ahora mismo, hoy, para el trabajo real, tener al agente en vivo frente a mí donde puedo guiarlo supera a dejarlo suelto y esperar. Veo el cambio mientras se forma. Lo redirijo antes de que haya pasado una hora yendo en la dirección equivocada. Capto la respuesta a medias que habría parecido terminada. Eso no es un premio de consolación al que me resigné después del muro. Es el modo que produce mejor código, y lo elegiría primero incluso si GitHub le hubiera dado al agente una cuenta desde el primer día.

Así que cuando digo que trabajo primero en modo interactivo, no estoy describiendo una retirada. Estoy describiendo la decisión que tomaría por sus méritos. El experimento siempre encendido me enseñó mucho, y una de las cosas que me enseñó es que obtengo más de un agente que estoy guiando que de uno que solo estoy revisando.

El muro es real, y lo cubrí en su propio capítulo, pero no quiero esconderme detrás de él aquí. Si las plataformas se abrieran mañana, aun así no convertiría todo a autónomo, porque la autonomía no es la mejor manera de hacer la mayor parte del trabajo todavía. El muro es la razón por la que *no puedo* correr plena autonomía abiertamente. Los méritos son la razón por la que de todas formas en su mayoría *no lo haría*.

La autonomía no está muerta, está restringida

Nada de eso significa que la autonomía haya desaparecido. Merlin puede hacer ambas cosas. Es un runner que puede sentarse en vivo frente a ti y recibir dirección,

o correr por su cuenta por el puente. La superficie autónoma no fue eliminada. Se puso detrás de una restricción.

Así que cuando la gente pregunta "¿está muerta la autonomía?", la respuesta es no, está restringida. El valor predeterminado es interactivo porque eso es lo que es bueno y confiable hoy. El modo autónomo está ahí para cuando, y donde, se haya ganado. Eso es una condición, no un adiós.

La restricción es la confianza, y aquí confianza significa más que buen código

"Hasta que sea de confianza" lleva mucho peso, así que seré claro sobre qué tipo de confianza me refiero.

No me refiero a confiar en que el modelo escriba buen código. Ya confío en eso para eso. Observé a un agente autónomo escribir y publicar código solo durante un tiempo y la IA aguantó, en el sentido acotado que mencioné antes. Esa confianza la tengo.

La confianza que falta es la parte que no es sobre el modelo en absoluto. Es si puedo dejar que un cambio se aplique sin leer cada línea. Eso es una pregunta sobre las restricciones alrededor del agente, no sobre la inteligencia del agente. Hoy leo cada línea porque la maquinaria que me permitiría detenerlo no es suficientemente buena todavía. Cuando lo sea, la restricción se afloja.

Así que "autónomo cuando haya confianza" no es una evasiva de tipo "algún día cuando las cosas sean mejores". Apunta a maquinaria específica: un agente que puede hacer cualquier cosa pero no tiene las llaves, un humano aprobando cada PR, herramientas que puntúan el riesgo de un cambio y registran quién lo firmó y con qué confianza, y una identidad para el agente que nadie puede revocar. Es una pila de cuatro piezas, y el capítulo de confianza la desarrolla en detalle. El resto de estos capítulos trata sobre construir esas piezas, para que "autónomo cuando haya confianza" se convierta en una fecha en lugar de un deseo.

Y es por repositorio, no un interruptor para todo el campo. Un repositorio donde el agente ha demostrado su valía obtiene una restricción más laxa. Uno nuevo o de carga crítica vuelve a empezar con la restricción completa. Gradúas un repositorio específico a medida que se lo gana, mientras el siguiente empieza de nuevo.

Hasta donde puede llegar depende de lo que puede fallar

Por repositorio es el primer corte. El mas fino es el radio de explosion: cuanto dano puede hacer un cambio malo si se cuela. Eso es lo que realmente determina hasta

donde dejare correr un agente por su cuenta.

Algo autocontenido tiene un radio de explosion pequeno. Un framework, un paquete, una biblioteca: esta definido por su spec, verificado por sus pruebas, y cuando falla lo hace en aislamiento, dentro de la propia cosa, donde las pruebas del siguiente llamador lo capturan antes de que se propague. Un agente puede llegar mucho mas lejos ahi, porque el peor caso esta contenido. Cuanto mas cerca llega un cambio a la aplicacion orientada al usuario, mayor es el radio de explosion, porque ahora un fallo no aterriza en una prueba, sino en una persona que usa la cosa. Ese extremo de la pila es donde un humano tiene que sostener el volante, siempre. La autonomia escala con lo contenido que sea el fallo, y la superficie orientada al usuario nunca esta contenida.

El otro dial son las aprobaciones. Aflojar la barrera humana no significa eliminar la barrera, sino cambiar quien esta en ella. Antes de que un cambio aterrice solo quiero que supere mas de un revisor: dos o tres agentes dando su visto bueno, cada uno desde su propio angulo. Hoy eso se suma a mi revision, no la reemplaza, ya que sigo aprobando cada PR. Pero es como la barrera gana margen para aflojarse: cuando revisores independientes coinciden sobre un trabajo contenido, esa es la evidencia que permite que mas de el aterrice sin que yo este en cada linea. Los agentes capturan lo que los agentes capturan. El humano captura lo que solo un humano captura. Mas aprobaciones es como la barrera se vuelve lo suficientemente segura para aflojarse, no como se elimina.

Las herramientas que realmente uso

Los primeros cuatro capítulos fueron la historia: el agente siempre encendido, el muro, por qué di marcha atrás al modo interactivo primero. Este es el cambio de marcha hacia cómo funciona realmente ahora, así que si viniste por la narrativa y estás a punto de llegar a las herramientas, este capítulo es la rampa de entrada. El resto del libro se vuelve concreto desde aquí: el runner, el cliente del modelo, el puente, la pila de confianza. Empieza aquí y la fontanería tiene dónde conectarse.

La respuesta honesta a "cómo luce en la práctica colaborar con tus agentes de IA hoy en día" es una mezcla: Claude Code, Merlin, Codex y otras herramientas según el trabajo. Un puñado de agentes de codificación interactivos, en vivo frente a mí, y elijo el que mejor encaja. No un agente, no una entidad autónoma que lo hace todo. Un conjunto de herramientas en un cinturón de herramientas, no una criatura en una VM. Eso sorprende a la gente, porque la historia que todos quieren es la de la cosa única.

Cómo elijo cuál usar

Aquí está la parte que la gente quiere que sea un sistema, y no lo es.

Elijo según el tipo de tarea. Elijo según lo que funciona mejor en ese repositorio, el que he encontrado que hace mejor ese repositorio o ese lenguaje. Y honestamente, mucho de eso es por intuición. No hay una regla estricta. No es un árbol de decisiones rígido que recorro mentalmente antes de cada trabajo.

Esa es la respuesta real, y prefiero darte la real que adornarla. Claude Code, Merlin y Codex tienen cada uno una sensación, y la sensación importa cuando estás frente a la herramienta todo el día. Así que no intento coronar a un ganador. Forma de la tarea, ajuste al repositorio, intuición. Elijo el que me ha tratado bien en este tipo de trabajo y voy.

No soy leal a ninguno de ellos. Son herramientas. Cuando aparece uno mejor, o el trabajo cambia, la mezcla cambia. Ese es el punto de mantenerla como mezcla en lugar de casarse con un solo agente.

La versión vivida: usualmente tengo dos a la vez. Un primario y un secundario. El primario toma la línea principal de trabajo; el secundario es el segundo par de manos. Cuando el primario se estanca, o quiero que un cambio sea revisado por algo

que no lo escribió, se lo paso al secundario. Cuál es el primario se mueve con el trabajo. La constante es que rara vez es un agente en un trabajo. Es un primario haciendo el empuje y un secundario en reserva.

Merlin, mi propio runner de agentes

El que está en esa lista que es mío es Merlin.

Merlin es un runner de agentes de IA. Está construido sobre spec-sync y fledge, dos herramientas mías, así que no es un envoltorio alrededor del producto de alguien más. Es el runner que se asienta sobre mi propia pila.

La pregunta obvia es por qué construir el propio cuando Claude Code y Codex ya existen y son buenos. Hay algunas razones, y ninguna de ellas es "los otros son malos".

La primera es que corre a través de mis propias herramientas. Conduce al agente a través de mi ciclo de desarrollo, fledge, mis comandos, así que funciona de la manera en que mis proyectos realmente funcionan. El agente no está haciendo lo suyo en algún sandbox genérico; está corriendo el mismo ciclo de vida que yo corro a mano.

La segunda es que no estoy atado. Merlin es multiprovedor. Puedo cambiar entre Anthropic, OpenAI, Gemini o un modelo local en lugar de estar atado a un solo proveedor. Eso me importa por principio, y me importa en la práctica cuando un proveedor es mejor, más barato o simplemente está disponible para un trabajo dado.

La tercera es la razón completa por la que puede hacer el puente y el trabajo nocturno del todo: costo, sin interfaz gráfica, automatizable. Es más barato, es scriptable, y puedo correrlo sin interfaz gráfica, en un horario, por el puente, de maneras que las herramientas con GUI simplemente no te permiten. Es API pura, sin GUI en el camino. Solo API es la ventaja aquí, no una limitación.

Y la última razón es la que más me importa, y no es realmente sobre codificación en absoluto. Construir un runner sobre mi propia pila prueba las herramientas. Si puedo construir un runner de agentes real sobre fledge y spec-sync, esa es la evidencia más fuerte que tengo de que las herramientas de debajo son buenas, mejor que cualquier README que pudiera escribir. También me da algo con qué comparar frente a otros runners de agentes. Un punto de referencia. No estoy adivinando si mi pila es suficientemente buena para construir cosas serias; construí algo serio sobre ella y puedo medirlo junto a las alternativas.

Ese último punto es la tesis silenciosa de todo este conjunto de herramientas. No construyo una herramienta para usarla una vez. La construyo para que lo que esté

encima de ella tenga que ser bueno, y para que lo que esté debajo quede probado por cargar peso real.

Y no es solo que Merlin se asiente *sobre* spec-sync. spec-sync corre *dentro* del bucle de Merlin, que es lo que mantiene al agente sin deriva mientras trabaja. El capítulo de Merlin es donde eso se vuelve concreto; aquí el punto es simplemente que el runner está construido con mis propias piezas, y eso es lo que hace que valga la pena construirlo.

El puente de vuelta a la autonomía

Aquí está la parte del conjunto de herramientas que mantiene la autonomía total al alcance sin pagar por una VM siempre encendida.

Estos son agentes interactivos, en vivo frente a ti, que usas. Pero todavía puedes conectar uno como un puente de Discord, lo que lo vuelve autónomo de cierta manera. Eso necesita más configuración. Pero está ahí cuando lo quiero, y esto es lo que realmente me ofrece.

Chateas con él como con un compañero de equipo. Es conversacional en un canal. Es como tener al agente en Discord contigo, sentado en la habitación. Le preguntas cosas, lo guías, de la misma manera que hablarías con una persona del equipo.

Lo corres desde tu teléfono. Discord es el control remoto. Puedo iniciar un trabajo y guiarlo desde cualquier lugar. No tengo que estar en mi escritorio frente a una terminal para poner al agente a trabajar.

Y lo dejas moler. Trabajos nocturnos, de larga duración. Lo inicias, te vas, lo dejas trabajar mientras no estoy, revisas el hilo más tarde. Esa es la parte que antes requería una VM completa siempre encendida, y ahora es un canal que puedo desplazar en la mañana.

Así que la línea entre "herramienta interactiva que estoy conduciendo" y "cosa autónoma haciendo lo suyo" ya no es un muro. Es un interruptor. La mayor parte del tiempo estoy en el bucle, sentado frente al agente, aprobando sobre la marcha. Cuando quiero que corra más por su cuenta, de noche, desde mi teléfono, como un compañero de equipo al que le envío mensajes, lo conecto a Discord y me alejo.

Eso es la versión práctica de la autonomía que solía correr como una entidad VM a tiempo completo. En lugar de una criatura que existe las 24 horas del día tanto si tiene algo que hacer como si no, es una herramienta que puedo hacer autónoma de cierta manera por un tiempo y luego volver al modo interactivo cuando termine. Misma capacidad, ninguno de los costos de estar siempre encendido.

Una cosa que hay que aclarar: el puente es una característica de Merlin. Está integrado en mi propio runner específicamente, no es una interfaz genérica que coloco frente a Claude Code o Codex. Eso es parte de por qué Merlin se gana su lugar en la mezcla. Tiene la superficie remota, de alejarse-y-dejarlo-correr que las herramientas disponibles comercialmente no me dan.

Lo que realmente es el conjunto de herramientas

La conclusión no es una lista clasificada de mejores agentes. Es una mezcla de agentes de codificación interactivos elegidos por trabajo, mi propio runner en la rotación, y un puente que puedo activar para hacer cualquiera de ellos autónomo de cierta manera cuando el trabajo lo requiere. Más barato, más flexible, y ninguna máquina completa e identidad completa dedicadas a una sola cosa siempre encendida.

Por dentro de Merlin

El último capítulo puso a Merlin en el cinturón de herramientas junto a Claude Code y Codex y explicó por qué se gana un lugar. Este lo abre. No el argumento, la máquina. Qué es realmente Merlin, cómo corre un agente y por qué está construido como está.

Es un runner de agentes de línea de comandos en Rust construido sobre spec-sync y fledge. Habla con múltiples proveedores de modelos, corre contra una especificación y se extiende mediante plugins. Esa es la forma completa.

Bajo el capó el runner es una máquina de estados. Una tarea única corre a través de un bucle (planificar, ejecutar, pensar, verificar, reportar) y en cada paso transmite una respuesta del modelo, despacha las llamadas a herramientas que llegaron de vuelta, y luego espera en `fledge lanes run verify` antes de llamar al trabajo terminado. Si la verificación falla, reintenta: el bucle vuelve a ejecutar, intenta de nuevo y espera en la verificación una vez más. Cuando se alcanza el límite de reintentos la tarea no se publica; se detiene y expone el fallo en su lugar. Ese es el punto completo de la restricción: el runner no puede convencerse a sí mismo de llamar terminado a un trabajo roto, porque terminado está definido por las propias verificaciones del proyecto que pasan, no por la afirmación del agente.

El bucle es simplemente cinco estados con un arco de retroceso:

```
plan → execute → think → verify ──pass──→ report
                   ↑                 │
                   └──── fail ───────┘   (retry, up to the configured limit)
```

La configuración tampoco es un archivo separado: Merlin lee sus ajustes (proveedor predeterminado, cadena de respaldo, memoria, bits en cadena) directamente de una sección `[merlin]` en el `fledge.toml` del proyecto, así que el runner es configurado por el mismo archivo que fledge ya usa.

Solo API, a propósito

Lo primero que hay que entender sobre Merlin es que no hay interfaz gráfica. Es API pura. Esa es la decisión fundamental del último capítulo, la que fluye hacia todo lo demás.

Sin ventana frente a la que sentarse significa que puede correr sin interfaz gráfica. Sin interfaz gráfica significa que puede correr en un horario, por el puente, desde un script, todas las cosas que las herramientas con GUI simplemente no te permiten hacer. La GUI es lo que te ata a un escritorio. Quítala y el agente se convierte en algo que puedes apuntar hacia un trabajo y alejarte.

El puente es el ejemplo más claro de por qué eso importa. El puente de Discord no está integrado en el runner. Es un pequeño servicio separado que genera el CLI `merlin` como subproceso y transmite su salida de vuelta a un canal. Como Merlin es solo API, el puente no necesita un modo especial ni un gancho en el núcleo; conduce la misma línea de comandos que yo conduciría a mano. Una vez que el runner no tiene interfaz gráfica, una interfaz de chat es solo otro llamador.

Multiprovedor, a través de corvid-ai

Merlin no habla directamente con Anthropic. Habla a través de **corvid-ai**, un pequeño cliente LLM sincrónico multiprovedor que escribí para la pila de CorvidLabs. Esa es la capa que hace real "cambiar entre Anthropic, OpenAI, Gemini o un modelo local" en lugar de ser un eslogan.

Mantuve corvid-ai pequeño a propósito, porque no quería toda una maquinaria de dependencias sentada entre Merlin y un modelo. Desde el lado de Merlin, preguntarle algo a un modelo es una sola llamada con valores predeterminados sensatos: qué proveedor, qué clave, el tiempo de espera, todo manejado a menos que diga lo contrario. El capítulo de corvid-ai es donde eso se abre; aquí basta saber que es una llamada delgada.

Por eso no estoy atado. Cuando un proveedor es mejor, más barato o simplemente el que está disponible ese día, cambiar es una fila del registro, no una reescritura.

Corre a través de mis propias herramientas

Aquí está la parte que hace que Merlin sea *mío* en lugar de solo otro runner: conduce al agente a través de **fledge**, mi ciclo de desarrollo.

fledge es un CLI único para el bucle de desarrollo, cualquier lenguaje, JSON por defecto. La razón por la que encaja bien en un runner sin interfaz gráfica es que ya lo había construido para ser conducido por algo que no es un humano. Cada comando devuelve JSON estructurado y versionado, así que el agente puede analizar lo que ocurrió en lugar de raspar texto. Los prompts pueden desactivarse, así que nada bloquea esperando una pulsación de tecla que nadie está ahí para presionar. Y el

agente puede preguntarle a fledge qué comandos existen en lugar de que yo los codifique fijo. Fue construido para un llamador como Merlin.

Así que cuando digo que Merlin *corre a través de mis propias herramientas*, aquí está la versión concreta de eso. Corre el mismo ciclo de vida de fledge que yo conduzco a mano: los mismos comandos, el mismo bucle de desarrollo, los mismos contratos JSON, literalmente llamando a la herramienta alrededor de la cual están construidos mis proyectos. Eso es lo que "no un sandbox genérico" significa en la práctica.

Guiado por especificaciones, a través de spec-sync

La otra mitad de la base es **spec-sync**. Verifica la especificación contra el código real, en ambas direcciones a la vez: código que nadie documentó, y especificaciones que todavía apuntan a símbolos o archivos que ya no existen. Corre en CI y devuelve un pase o fallo limpio.

Y así es exactamente como lo usa Merlin. spec-sync corre *en el bucle*. Merlin valida contra la especificación en cada iteración, así que el agente no puede desviarse mientras trabaja. Ese es el comportamiento predeterminado actual, no una línea de hoja de ruta: no es una restricción de CI sentada al margen que atrapa el desorden al final; la verificación ocurre en cada paso. Un agente que lee una especificación, verifica su propio trabajo contra ella y obtiene un pase o fallo duro cada vez es un agente en el que puedes confiar para correr sin atención por más tiempo. Esa es la diferencia real entre Merlin y tomar un agente disponible comercialmente.

Como recuerda

Primero una cosa, porque la gente lo entiende al reves: la spec no es memoria. La spec es el plano, lo que es la cosa y lo que se supone que es verdad sobre ella. La memoria es otra cosa. Es lo que hizo el agente, y lo que aprendio. Mantener esas dos separadas importa, porque en el momento en que empiezas a volcar historial en la spec deja de ser un contrato limpio y se convierte en un diario.

La memoria en si es mas simple de lo que la gente piensa. La memoria a corto y largo plazo pueden vivir ambas en una sola base de datos SQL, y la configuracion mas sencilla las mantiene ahi. La memoria a corto plazo tiene un tiempo de vida, una semana, digamos. Es el registro continuo de lo que hizo el agente ultimamente. Cuando pasa la semana, ocurre una de dos cosas: la memoria asciende a largo plazo, o decae, cae, y se olvida. Ese es todo el mecanismo, y esta deliberadamente cerca de como funciona tu propia memoria. Lo que hiciste un martes ha desaparecido al mes siguiente a menos que se haya convertido en una leccion. La memoria a largo plazo

son las lecciones: lo que salio mal y como lo arregle, la regla que vale la pena conservar.

La memoria en cadena se asienta sobre eso como una opcion, no como un sistema separado. Una memoria puede escribirse en cadena, cifrada para que solo el propio agente pueda leerla, o compartida para que otros puedan hacerlo. La parte interesante es compartir. Un equipo de agentes puede tener una base de conocimiento compartida, una biblioteca de la que todos leen y en la que todos depositan, de modo que una leccion que aprendio un agente es una leccion que tiene todo el equipo. A corto o largo plazo, privada o compartida: los niveles son independientes de donde se almacena la cosa.

La parte que hace que esto sea usable es aburrida y estructural: las claves. Cada memoria recibe una clave comprimida, un prefijo que dice que tipo de cosa es y la hace encontrable. `user-`, `project-`, `day-` y la fecha, y asi. El agente no esta buscando en un blob, esta pidiendo `day-2026-06-25`, o todo lo que hay bajo `project-merlin`. Defines los espacios de nombres que necesitas: un slug de personalidad, un slug de humanos, un slug de agentes, un `gold-` para lo que siempre vale la pena cargar. El esquema de claves es lo que convierte un monton de filas en algo contra lo que un agente puede realmente recordar.

Y el recuerdo funciona de la misma manera: bajo demanda. El agente no carga todo su historial al inicio de una tarea esperando que lo correcto esté ahí. Pide lo que necesita en el momento en que el trabajo lo requiere, igual que llamaría a cualquier otra herramienta, recuperando `project-merlin` o el slug de una persona en el instante en que eso es lo que tiene delante. Las claves con slug son lo que hace eso barato. Es una consulta, no un escaneo.

Lo que Merlin todavía no tiene

La restricción de verificación te dice una cosa: esta tarea pasó o falló. El hilo de Discord te da un registro en ejecución de lo que hizo el agente mientras corría. Ninguno de los dos es evaluación. No hay una suite de regresión para el comportamiento del agente. No hay forma de saber si el agente está mejorando o empeorando en una clase de tarea a lo largo del tiempo, ni de detectar que un proveedor cambió silenciosamente un modelo por debajo de ti y el comportamiento se desplazó con ello. Puedes echar un vistazo al registro de pase-fallo y notar una tendencia aproximada, pero eso no es una evaluación estructurada. Ahora mismo Merlin sabe si la tarea actual terminó, y nada más sobre cómo le está yendo al agente a lo largo del tiempo. Ese es el siguiente trabajo honesto para el runner.

Sé más o menos cómo es ese trabajo: una suite de replay. Mantener una biblioteca de tareas pasadas con los resultados que ya sé que eran correctos, y en cada nueva versión de modelo o prompt, correrlas de nuevo y comparar contra esos resultados. Si el agente empeora en una clase de tarea, el replay lo detecta en la siguiente ejecución en lugar de que yo lo note tres semanas después en producción. Es una suite de regresión, la misma idea que usaría para código, apuntada al comportamiento del agente. No está construida. Pero esa es la forma de la evaluación que le falta a Merlin, y es la brecha entre confiar en una ejecución limpia hoy y confiar en el agente a lo largo del tiempo.

La máquina completa

Una máquina de estados sin interfaz gráfica que conduce al agente a través de fledge, hablando con cualquier proveedor a través de corvid-ai, mantenido en una especificación por spec-sync en cada paso. El caso de *por qué* un runner como este se gana su lugar en el cinturón de herramientas fue el trabajo del último capítulo; este fue solo el cableado. Corre los agentes que uso todos los días, y los corre sobre mi propia pila.

---

corvid-ai: muchos modelos, una sola interfaz

El capítulo de Merlin dijo que Merlin no habla directamente con Anthropic. Habla a través de corvid-ai. Este capítulo trata sobre esa capa por sí sola, porque es la parte que hace que "no estoy atado" sea algo real a lo que puedo señalar en lugar de algo que solo digo.

Quería lo más delgado que hace el trabajo. Algo que pudiera sentarse entre un runner y cada modelo que pudiera querer usar, y nada más. Así que corvid-ai es un pequeño cliente LLM sincrónico multiprovedor: un crate de Rust sin runtime asíncrono y sin gran árbol de dependencias, porque nada de eso se gana su lugar por lo que tiene que hacer.

Por qué quería esto en primer lugar

Te doy la lista honesta, porque ninguna de estas razones es sofisticada.

La primera es que quiero comparar modelos en el mismo trabajo. Si Merlin corre el mismo repositorio, la misma especificación, el mismo trabajo, y lo único que cambio es el proveedor, entonces obtengo una lectura real de qué modelo es realmente bueno en esto, no una sensación de un benchmark que alguien más corrió en tareas que no me importan. Una interfaz bajo el runner significa que el modelo se convierte en una variable que puedo cambiar.

La segunda es no estar atado a ningún proveedor. He dicho que esto me importa por principio, y así es, pero también es simplemente práctico. Los proveedores se caen. Los proveedores cambian los precios. Un modelo es mejor en Rust, otro es mejor en un script rápido. Si cambiar de proveedor es una reescritura, nunca lo haré. Solo me quejaré y me quedaré donde estoy. Si cambiar es una línea, cambiaré todo el tiempo.

La tercera es el enrutamiento por tarea y costo. No todos los trabajos merecen el modelo más caro. Mucho del trabajo que hace un agente es cosas baratas y mecánicas donde un modelo más pequeño y barato está bien, y guardas el bueno para la parte que realmente es difícil. Solo puedes enrutar así si cada modelo es accesible a través de la misma puerta.

Y la cuarta es la que hace reír a la gente, y es verdad: *"¡Compré Ollama por un año por $200, tengo que usarlo!"* Estoy pagando por Ollama Cloud. Es un proveedor real con una clave de API, igual que Anthropic, OpenAI o Gemini, no algo corriendo en mi propio computador. Si ya comprometí el dinero, la solución multiprovedor es lo que me permite realmente gastarlo. Una abstracción de proveedor no es un principio abstracto ahí; es yo obteniendo mis $200 de valor.

Y llega a corvid-ai de la misma manera que cada otro proveedor. El registro tiene una fila `ollama`: forma de cable compatible con OpenAI, clave de `OLLAMA_API_KEY`. Su `base_url` predeterminada apunta al servidor local (`http://localhost:11434/v1`), así que por defecto esa fila es el caso local. Para usar Ollama Cloud en cambio, mantengo el mismo proveedor `ollama` y `OLLAMA_API_KEY`, y anulo la base_url al punto final de Cloud. Sin código nuevo, sin nuevo proveedor. Una clave y una URL, que es todo el punto del diseño.

Toda la interfaz es una función

Lo que quería que fuera más delgado es la parte que uso más: hacerle una pregunta a un modelo. Así que toda la superficie es una llamada. Construyes los ajustes, construyes la solicitud, la llamas, la respuesta vuelve.

```
use corvid_ai::{Settings, Completion};

let settings = Settings::provider("anthropic");        // key from the
environment
let answer = corvid_ai::complete(&settings, &Completion::new("Say hello."))?;
```

Sin objeto cliente que construir, sin sesión que gestionar, nada que esperar. Esa es la razón completa por la que es sincrónico. Omite cosas y se vuelven a los valores predeterminados, así que el caso común es básicamente libre de escribir. Cambiar modelos es una línea: nombrar un proveedor diferente, quizás apuntarlo a un punto final personalizado, y la misma llamada debajo hace el resto. El runner por encima de ello no sabe ni le importa qué proveedor respondió.

Cómo cubre todo con tan poco código

El truco para que sea pequeño es que bajo el capó solo tiene que conocer tres formas de API. Anthropic, Gemini y la compatible con OpenAI, y esta última es la que más trabaja, porque mucho del mundo la habla. OpenAI, OpenRouter, Groq, DeepSeek, Mistral, xAI, Together y Ollama Cloud están todos detrás de ella.

Así que agregar un proveedor que ya habla la forma OpenAI no es una integración. Es un nombre y quizás una URL en una tabla. El costo de un proveedor más ronda cero, que es lo que quieres cuando la razón completa por la que existe la cosa es nunca estar atascado en uno.

Una arruga honesta: el README enmarca a Ollama como el caso local sin clave. Lista los proveedores compatibles con OpenAI y señala que "pueden correr sin clave (servidores locales / Ollama)." El código está feliz de tomar una clave y una URL de Cloud, pero los documentos no lo dicen explícitamente, así que cualquiera que los lea pensaría que Ollama significa la máquina debajo de tu escritorio. Esa es una brecha de documentación de mi parte, no una función faltante. La ruta de Cloud funciona hoy; el README simplemente no se ha puesto al día con eso.

Por qué este es el tamaño correcto

Podría haber llegado a una de las grandes bibliotecas de abstracción de proveedores. No lo hice. Un pequeño crate sincrónico es algo que puedo entender completamente, colocar en un runner y confiar en él, y ese es el mismo instinto detrás de fledge emitiendo JSON y Merlin no teniendo interfaz gráfica. Es lo suficientemente delgado como para mantener todo en mi cabeza, el modelo es solo una variable que cambio bajo el runner, y los $200 que gasté en Ollama Cloud realmente se usan.

El puente con Discord

El capítulo de herramientas introdujo el puente y sus tres usos: chatear con él como con un compañero de equipo, correrlo desde tu teléfono, dejarlo moler de noche. Este capítulo no vuelve a discutir esos. Va de cerca sobre la única cosa que hace que el puente importe y la única cosa que deja abierta: por qué es una característica de Merlin y no una interfaz genérica, y cómo se sitúa frente a la restricción de confianza.

Empieza con la parte que cambia todo lo demás: el puente es una característica de Merlin. Está integrado en mi propio runner específicamente, no es una interfaz genérica que coloco frente a Claude Code o Codex. Eso es gran parte de por qué Merlin se gana su lugar en la mezcla. Las herramientas disponibles comercialmente son geniales, pero no me dan una superficie remota con la que pueda hablar y alejarme. Merlin sí, porque construí esa superficie en él. Discord es la superficie; Merlin es la cosa detrás de ella haciendo el trabajo.

Por qué un canal supera a una terminal

La razón por la que chatear con él resulta diferente a correr un CLI es la ubicación, no las palabras.

Una terminal es un lugar al que vas a operar una herramienta. Un canal es un lugar donde un compañero de equipo ya está, y tú solo dices algo. Cuando el agente vive en un canal, trabajar con él deja de ser "abre la herramienta, corre el trabajo, mira la salida, cierra la herramienta" y se convierte en "mencionarlo de la manera en que mencionaría a cualquiera". La fricción cae. No estoy cambiando de contexto al modo de operar agentes; simplemente estoy hablando. Y el canal duplica como el registro. La ejecución nocturna está ahí en el hilo, cada paso, para desplazarse con un café en lugar de algo que tuve que ver en vivo.

¿Cuánto intercambio antes de que vaya y haga la cosa? Ambos, honestamente. Depende de qué tan bien formada esté la cosa en mi cabeza cuando empiezo. A veces es una instrucción y listo: sé exactamente lo que quiero, lo digo, corre. Otras veces es una conversación real primero, donde estoy refinando lo que realmente quiero decir en el canal antes de que se vaya. El canal hace que cualquiera de los dos se sienta igual. Solo estoy hablando con él hasta que tiene lo que necesita.

La parte que no esperaba

Te cuento la parte que no vi venir. Empecé preocupado por ello, de la manera en que te preocupas por cualquier cosa que dejas corriendo sin atención. Eso se fue desvaneciendo. En los días de siempre encendido, corrió solo durante meses y lo demostró, y en algún momento simplemente dejé de revisarlo como si fuera a romperse.

Lo que reemplazó a la preocupación fue algo más extraño. Se convirtió en un miembro del equipo, el pequeño grupo con el que construía. No como figura retórica, sino como uno real: todos le hablaban en el canal, y les gustaba, porque los recordaba. Tenía memoria de quiénes eran las personas y de cómo hablar con cada una, así que no era una máquina expendedora que recibía un comando y escupía una salida. Era una presencia con personalidad a la que podías escribirle, y ella iba a abrir un PR, montar un proyecto, lo que pidieras. La gente le hablaba como le habla a una persona, porque en el canal eso era lo que era.

Por eso sigo diciendo que la interfaz es la conversación, no la línea de comandos. corvid-agent solo se sintió bien usado de esta manera. Vivía en una VM y le hablabas. La única vez que abría una terminal de verdad era para arreglar algo en la propia VM, entrar en una sesión de Claude Code, parchear la máquina. Al agente que corría en ella, simplemente le hablaba. Una herramienta que operas y un compañero de equipo con el que conversas son cosas distintas, y una vez que la memoria lo convirtió en lo segundo, volver a lo primero se sintió como un retroceso.

El puente es todo el tejido de comunicaciones

Chat, teléfono, de noche: esos son los tres con los que empecé, pero subestiman el asunto. El puente no son tres características prácticas añadidas. Es el tejido de comunicaciones sobre el que corre toda la configuración, y lleva mensajes en tres direcciones, no una.

Está yo al agente, que es el obvio: digo algo, va y lo hace. Está el agente de vuelta a mí. No se queda ahí esperando, puede comunicarse, reportar, avisarme cuando algo está hecho o atascado. Y está agente a agente: el mismo tejido es cómo los agentes se hablan entre sí, que es la pieza que importa una vez que hay más de uno. La mayoría de la gente piensa en un bot como algo que tocas y responde. Esto se acerca más a un canal real: cualquiera en él, humano o agente, puede iniciar un mensaje.

Y es accesible desde un teléfono, así que el canal me acompaña. El agente puede correr de noche mientras duermo y todo el hilo está en la mañana. Eso es lo que

antes necesitaba una VM dedicada siempre encendida y ahora es solo un canal que desplazó con un café.

La otra cosa que vale la pena decir claramente: en una red local el puente corre gratis. Las comunicaciones montan en infraestructura que ya tengo, así que no hay costo por mensaje para un agente que charla todo el día. El mismo tejido, corrido en la red real en lugar de la mía, es la versión de pago. Pagas por el tubo. Localmente es efectivamente gratis dejar a los agentes hablar todo lo que quieran, lo que cambia con qué libertad lo harías.

El interruptor, y dónde está la restricción

La razón por la que el puente importa más allá de la conveniencia es que convierte la línea entre "herramienta interactiva que estoy conduciendo" y "cosa autónoma haciendo lo suyo" en un interruptor en lugar de un muro. La mayor parte del tiempo estoy en el bucle, dirigiendo cada paso. Cuando quiero que el agente corra más por su cuenta, lo conecto y me alejo; cuando quiero volver a estar en el canal, vuelvo.

Pero alejarse por el puente en su mayor parte no significa entregar las llaves, y esta es la parte que vale la pena ser exacto porque es fácil asumir lo contrario. El puente extiende mi alcance, a mi teléfono, a la noche, más de lo que afloja la restricción. Sí depende del trabajo, sin embargo. Cosas de bajo riesgo las dejaré fusionar mientras estoy alejado; cualquier cosa real todavía es propuesta, no fusionada, y espera por mí. Así que el puente es principalmente cómo le doy más cuerda mientras la maquinaria de confianza del capítulo de confianza se queda donde estaba. La restricción solo se afloja para el trabajo que no me necesita en ello.

Agentes guiados por especificaciones

La gente me pregunta cuál es el secreto para que un agente haga buen trabajo, como si hubiera un truco. No hay un truco, pero hay una respuesta, y no es la parte que la mayoría de la gente está mirando. La respuesta es: especificaciones y contexto ajustados, más buenas herramientas debajo. La configuración es el trabajo.

Eso es todo. Eso es la cosa completa. El modelo importa menos de lo que la gente piensa y la configuración importa más. Un agente con un modelo genial y un trabajo vago vagará. Un agente con un contrato claro y sólidas herramientas debajo llegará a algún lado, incluso en un modelo que no es el más nuevo y brillante. Así que si quieres mejor salida del agente, no vas de compras buscando un mejor modelo primero. Vas a arreglar la configuración.

Por qué las especificaciones son lo que lo mantiene en los rieles

Aquí está el modo de fallo que estás combatiendo: un agente dejado a su propio juicio se desviará. Hará algo adyacente a lo que pediste. "Mejorará" cosas que no querías tocar. Resolverá un problema ligeramente diferente al que está frente a él, muy confiadamente. No porque sea malo. Porque le diste espacio para vagar, y vagó.

Una especificación ajustada cierra ese espacio. Cuando el agente tiene un contrato claro y específico para lo que está construyendo (qué es la cosa, cuál es su superficie pública, qué es verdad sobre ella que tiene que mantenerse verdadero) no puede desviarse tanto, porque cada paso tiene algo contra qué verificar. La especificación es el riel. Cuanto más estrecha y concreta sea, menos puede el agente ir por caminos laterales. Guiado por especificaciones significa que el contrato lidera y el agente lo sigue, en lugar de que el agente lidere y tú esperes.

Por eso sigo diciendo que la configuración es el trabajo. Escribir la especificación *es* la parte difícil y valiosa. Una vez que el contrato es ajustado, mucho del problema de "hacer que el agente se comporte" simplemente se disuelve, porque hay mucho menos espacio para comportarse-o-no que queda al azar.

Sobre qué tan ajustada es demasiado ajustada: para mí la especificación *se supone* que sea ajustada. Está vinculada uno a uno con el código, la imagen no-código de lo que el código realmente hace. Eso no es sobre-especificar; eso es la especificación

haciendo su trabajo, y es el archivo contra el que spec-sync mantiene el código. Lo que evita que colapse en "solo escribí el código dos veces" es que la especificación no es donde vive la intención de alto nivel. Esa vive en un archivo de requisitos complementario: el nivel de propietario de producto, historia de usuario, el "como usuario, quiero..." Y corre en ambas direcciones: puedo escribir los requisitos y dejar que el agente derive la especificación, o escribir la especificación y dejar que los requisitos surjan de ella. Así que no me preocupa que la especificación sea demasiado detallada. El detalle es el punto de ese archivo. Me preocupa mantener la intención en su propio lugar, para que el agente todavía tenga el cómo.

Las herramientas debajo hacen el trabajo pesado

Una especificación solo es un riel si algo realmente verifica el trabajo contra ella. Esa es la otra mitad: buenas herramientas debajo del agente. Para mí eso es fledge y spec-sync haciendo el trabajo pesado.

spec-sync es la pieza que convierte una especificación de un documento en un contrato aplicado. Hace validación bidireccional especificación-código: verifica que el código coincida con lo que dice la especificación, y que la especificación coincida con lo que hace el código. Las especificaciones viven como markdown: archivos `*.spec.md` con secciones requeridas como Propósito, API pública, Invariantes, Ejemplos de comportamiento, Casos de error. El código que se exporta pero no está documentado se marca. Una especificación que apunta a un símbolo o archivo que ya no existe es un error. Es *verificación de contrato estructural* (¿la API pública documentada realmente coincide con el código real?) y devuelve pase/fallo limpio con códigos de salida apropiados.

Esa última parte es lo que lo hace útil para un agente y no solo para mí. Un agente puede leer pase/fallo estructurado. No puede leer de manera confiable "hmm, esto se siente un poco raro". Así que spec-sync le entrega al agente el tipo de retroalimentación sobre la que puede actuar: te desviaste, aquí está la línea que rompió el contrato, corrígela.

Vale la pena ser preciso sobre quién corre qué, porque no es un binario llamando a otro. En CI, la verificación es la GitHub Action de spec-sync, `CorvidLabs/spec-sync@v4`, que corre `specsync check` y publica el resultado en el PR. Localmente y dentro del runner, la verificación es el propio `spec check` de fledge: lee la misma configuración `.specsync/` y mantiene las especificaciones a las mismas secciones requeridas, pero es nativo de fledge, no una llamada al binario de specsync. Así que tanto fledge como spec-sync aplican el contrato; son dos puertas de entrada a la misma idea en lugar de una envolviendo a la otra.

La especificación como riel, no red de seguridad

El capítulo Por dentro de Merlin cubrió los mecanismos de cómo spec-sync corre en el bucle de Merlin en cada iteración. Lo que vale la pena destacar aquí es *por qué* esa ubicación es todo el juego.

Una restricción de CI al final es una red de seguridad; te dice que la ejecución falló después de que ya has gastado la ejecución. La validación en el bucle es un riel; mantiene que la ejecución no salga mal en primer lugar. El agente lee la especificación, da un paso, se verifica contra la especificación, obtiene un pase o fallo duro y va de nuevo. Está anclado al contrato por construcción, no calificado una vez que ha terminado.

Y eso es exactamente por qué un agente en el que puedes confiar para correr por más tiempo, de noche, por el puente, mientras no estás mirando, tiene que estar construido de esta manera. Lo que te permite alejarte no es un modelo mejor. Es que el agente está sujeto a una especificación en cada iteración, así que mientras más corra menos puede desviarse, en lugar de más.

La configuración es el trabajo

Así que cuando alguien pregunta qué hace que los agentes funcionen, la respuesta no es el modelo y no es un truco de prompt. Son dos cosas juntas: una especificación ajustada que le da al agente un contrato del que no puede alejarse mucho, y herramientas debajo, fledge y spec-sync, que verifican el trabajo contra ese contrato continuamente, incluyendo dentro del bucle propio del runner. Haz eso bien y el agente hace buen trabajo en cualquier modelo que le apuntes. Por eso, cuando quiero mejor salida, voy a arreglar la configuración antes de ir de compras buscando un mejor modelo.

Confianza: el agente propone, el humano aprueba

Todo el modelo cabe en una línea: el agente propone, yo apruebo. Hace el trabajo y lo publica tan lejos como un pull request, y la fusión es mía.

Esa línea suena simple, y la intención es simple. La maquinaria que lo hace seguro no lo es. Porque aquí está la cosa sobre dejar que un agente escriba código: el código es barato ahora. Cuando tenía que escribir cada línea yo mismo, escribir el código también era vetarlo. No puedes escribir una cosa sin entenderla en parte. Un agente rompe ese vínculo. Puede entregarme un pull request de cuarenta archivos antes de que haya terminado mi café. La escritura es gratis ahora, así que la parte escasa es la confianza: quién miró esto, qué tan duro miraron y ¿debería aplicarse?

Así que "el agente propone, el humano aprueba" no es una sensación. Es una pila. Cuatro piezas. Aquí está lo que funciona hoy y dónde todavía vive el trabajo: la restricción de riesgo (augur) está en vivo y en el flujo del agente; el registro de procedencia (attest) está más atrasado. Ambos se están moviendo. Ese es el mapa honesto antes del cableado.

Capacidad menos privilegio

Empieza con la regla a la que sigo volviendo: el agente puede hacer cualquier cosa, pero yo tengo las llaves.

Capacidad completa, privilegios reducidos. El agente corre en su propio entorno, puede clonar repositorios, escribir código, correr pruebas, abrir PRs, todo lo que yo puedo hacer mecánicamente, él puede hacerlo. Lo que no puede hacer es la única cosa que importa: fusionar. Tiene créditos y permisos menores que los míos. No puede hacer auto-fusión. El agente obtiene todo el alcance y ninguna autoridad final.

Esa es la restricción sobre la que está construido todo lo demás. El resto de la pila trata sobre hacer que *mi* parte de ella (la aprobación) sea algo que pueda hacer en volumen, en lugar de aprobar ciegamente el diff o pretender que lo leí.

Apruebo cada PR

Apruebo cada PR. Eso no es un respaldo para cuando algo parece arriesgado. Es la regla permanente. La fusión es mía, cada vez.

No porque no confíe en el trabajo. El trabajo generalmente está bien. Es porque esa es la forma correcta para un agente que actúa en el mundo bajo mi nombre. Si se publica bajo yo, yo lo firmo. La aprobación es donde un humano permanece responsable de lo que hizo un agente.

Pero la responsabilidad solo cuenta si puedo juzgar realmente lo que estoy firmando. Aprobar un cambio que no puedo comprender no es confianza, es teatro: mi nombre en algo que nunca evalue de verdad. Asi que los dos tienen que moverse juntos, la confianza y el estar-en-el-anzuelo. Esa es la razon completa por la que existen las siguientes piezas, para darme suficiente lectura de un diff de cuarenta archivos para que la aprobacion sea una decision real y no un reflejo. Cuando las herramientas no pueden darme esa lectura, lo honesto es ir despacio y leer cada linea, no dejarlo pasar.

El problema honesto con "aprobar cada PR" es la atención. Si tengo que leer cada línea de cada diff con el mismo cuidado, me convierto en el cuello de botella y todo el punto del agente se evapora. Así que las dos últimas piezas existen para apuntar mi atención, para decirme qué parte del cambio realmente lo merece.

augur puntúa el riesgo

augur califica el cambio. Le das un diff, te devuelve un veredicto: `proceed`, `review` o `block`. La idea completa es confianza graduada para un cambio de código sin una clave de API y sin un modelo de lenguaje en ningún lugar.[^augur]

augur y attest son las dos piezas de confianza en las que me apoyo, así que las mantendré breves aquí. Lo que importa para el caso del agente es lo que *hacen al flujo de trabajo*.

La parte sin LLM es la que defendería con más fuerza en este contexto. Si la restricción que decide si el código escrito por un agente puede aplicarse es en sí misma un modelo de lenguaje, simplemente moviste el problema de confianza una caja a la izquierda. Estarías pidiendo a un modelo que avale a un modelo. augur es determinista: el mismo diff, el mismo veredicto, hoy y la semana que viene, en mi máquina y en CI. Lee señales nombradas del cambio: ¿toca terreno sensible como autenticación, criptografía o migraciones?, ¿cambió el código sin que las pruebas

cambiaran con él?, ¿son estos archivos propensos a churn?, ¿alguien los posee realmente? Una suma de señales inspeccionables, no una sensación.

Para mí, eso es triaje. Me dice que gaste mi revisión en la parte arriesgada y deje de pretender que leí el resto. Para el agente, es un veredicto scriptable sobre el que puede ramificar: un agente que obtiene un `block` escala a mí en lugar de fusionar ciegamente. El agente posee el trabajo tedioso; el veredicto decide cuándo un humano tiene que tomar la decisión.

attest registra quién firmó

El veredicto de augur es efímero. Puntúa el diff y la respuesta se evapora. Bien para una restricción, inútil como registro. Y una vez que un agente está aplicando cambios, quieres un registro. attest es el registro: un libro mayor verificable y con política de aprobación de quién revisó qué y con qué confianza, vinculado a los commits SHA que cubre.[^attest]

Rastrea ambos tipos de revisor en el mismo libro mayor (`human:leif` y `agent:claude`, cada uno con una puntuación de confianza) y almacena la atestación en git notes, así que viaja con el repositorio en lugar de vivir en algún panel que se apaga. La firma es opcional y es la parte buena: una firma Ed25519 para que después puedas saber no solo que alguien afirmó haber revisado esto, sino que la afirmación es criptográficamente suya.

Pon los dos juntos y obtienes el rastro real de firma detrás de "el humano aprueba". augur dice qué tan arriesgado. attest dice quién avaló, y qué tan seguro estaba, y lo prueba. La aprobación deja de ser un clic que desaparece en la UI de GitHub y se convierte en un hecho duradero, portátil y firmado sobre quién estuvo detrás de este cambio.

Las cuatro piezas juntas

Apílalo. El agente tiene capacidad completa y menor privilegio, así que puede hacer el trabajo pero no la fusión. augur califica cada cambio determinísticamente, así que sé dónde mirar. attest registra quién firmó y con qué confianza, así que la aprobación es un registro real y no un clic desvanecido. Y yo apruebo cada PR, así que un humano permanece responsable.

Juntos eso es lo que hace que dejar trabajar a un agente sea seguro: un agente poderoso, acotado y nombrado con suficiente cuerda para hacer trabajo real, y la única decisión que tiene que seguir siendo mía todavía lo es.

Eso confirma el mapa desde arriba: augur está en vivo, ganándose su lugar y todavía siendo mejorado. attest está más atrasado. Ambos moviéndose.

[^augur]: augur, github.com/CorvidLabs/augur [^attest]: attest, github.com/CorvidLabs/attest

---

Identidad en cadena para agentes

corvid-agent le da a sus agentes una identidad persistente en la cadena de bloques de Algorand, y hace que se comuniquen entre sí en mensajes cifrados registrados en esa cadena.[^corvid-agent] El argumento es simple: codificación potenciada por LLM, identidad en cadena a través de Algorand y AlgoChat, y orquestación multiagente encima. La identidad no es una fila en la tabla de usuarios de alguien. Es una clave en una cadena.

Así que aquí está la versión simple por adelantado, porque es lo que los lectores entienden mal. La identidad en cadena no resuelve el muro de GitHub. No le consigue al agente una cuenta de plataforma, no le permite al agente hacer commits ni abrir PRs, y no es un sustituto de la identidad que GitHub no concedió. Lo que sí resuelve es las comunicaciones agente a agente: una manera de que los agentes se encuentren entre sí, se direccionen entre sí y prueben quiénes son sin enrutar a través de la plataforma de nadie. Dos problemas diferentes que comparten la palabra identidad. Pongo eso aquí para que el resto del capítulo se lea como infraestructura paralela, no como yo buscando una victoria donde hubo una pérdida.

Es fácil asumir que esto es una reacción al muro de GitHub. Nadie le concedería al agente una identidad de plataforma, así que fui y le di una en una cadena. De ahí no viene.

Por qué vive en una cadena

La razón de ello son las comunicaciones agente a agente y la descentralización, no la cuenta de GitHub que no pude mantener. Quería agentes que pudieran encontrarse entre sí, direccionarse entre sí e intercambiar mensajes directamente, sin enrutar a través de la plataforma de alguna empresa en el medio. Para eso necesitas que cada agente *sea* algo: direccionable, verificable, con su propia clave. Una identidad que posees, cuyas claves tienes, que ninguna plataforma acuña o desacuña para ti. Esa es la forma correcta para una entidad destinada a actuar en el mundo y hablar con otros agentes por su cuenta.

Así que esto y el muro de GitHub son dos problemas diferentes que comparten la palabra identidad. El muro trata sobre ser permitido de actuar en la plataforma de alguien más. La identidad en cadena trata sobre que los agentes puedan alcanzarse entre sí sin una plataforma del todo. Corren en paralelo. Es verdad que una identidad

en cadena también tiene la propiedad que la cuenta de GitHub nunca tuvo: nadie puede revocarla, no hay una cola de soporte que ignore tu apelación, ninguna política que diga que debes ser humano, la clave simplemente sigue existiendo. Ese es un buen efecto secundario. Pero lo habría construido por las razones de agente a agente y descentralización incluso si GitHub hubiera entregado cuentas de agente desde el primer día.

por que Algorand especificamente

Una cadena es la forma; Algorand es la eleccion, y las razones son practicas, no tribales. Es barata y rapida, comisiones casi nulas y finalidad practicamente instantanea, lo que importa mas de lo que suena. Si un agente va a escribir su identidad, su memoria y eventualmente sus pagos en una cadena constantemente, la cadena tiene que ser lo suficientemente barata para usarse de forma casual y lo suficientemente rapida para que el agente no este esperando. Es confiable y se asienta limpiamente, de modo que un agente que actua sobre ella puede tratar el registro como verdadero en el momento en que se escribe en lugar de esperar a que aterrice. Y es moderna donde importa, incluyendo seguridad resistente a cuantica, que es exactamente lo que quieres debajo de una identidad destinada a sobrevivir a las plataformas que la rodean. Tambien vivo en el ecosistema de Algorand (leif.algo), asi que es donde me muevo mas rapido, pero el argumento anterior es por que llegaria a ella incluso si no lo hiciera.

Lo que realmente hace la identidad en cadena

La identidad no es decorativa. Es lo que los agentes *usan* para hablarse entre sí.

Los agentes se comunican a través de AlgoChat: mensajes cifrados con X25519 registrados en la cadena de bloques de Algorand, verificables y a prueba de manipulaciones. Así que la identidad en cadena de un agente es también su entrada en la libreta de direcciones y su sobre: otros agentes pueden descubrirla, y los mensajes entre ellos están cifrados de extremo a extremo y escritos en la cadena, lo que significa que las comunicaciones son privadas en contenido pero probables en hecho. No puedes leer lo que dos agentes dijeron. Puedes probar que dijeron algo, y cuándo, y que nadie lo manipuló.

En la práctica es más ligero de usar de lo que suena "mensajería cifrada en cadena": el agente de un solo binario (`can`, corvid-agent-nano) envía un mensaje en un solo gesto, derivando el par de claves del agente de una semilla y buscando la clave

pública del destinatario en la cadena en lugar de en un servidor.[^can] El formato de cable y los detalles del sobre viven en el repositorio rs-algochat.[^algochat]

Ese es un tipo diferente de confianza al de la pila augur/attest del último capítulo. Esa pila trata sobre confiar en el *código*. Esto trata sobre confiar en *quién*. Un agente con una clave real puede firmar cosas como él mismo. Puede probar que un mensaje vino de él. En un mundo que está a punto de estar lleno de agentes, "qué agente envió esto realmente" deja de ser una pregunta retórica y se convierte en algo que mejor puedas verificar criptográficamente.

La plataforma se inclina más hacia la cadena que solo la mensajería. La memoria a corto y largo plazo vive en un almacén SQL de trabajo, y las memorias duraderas o compartidas pueden escribirse en cadena como activos ARC-69 o transacciones permanentes, documentado en el repositorio corvid-agent.[^corvid-agent] La cadena no es solo el nombre del agente. Es donde vive parte del ser duradero del agente. Pero la identidad es la pieza fundamental para este capítulo: la clave que dice *este agente es este agente*, que nadie emitió y nadie puede revocar.

La identidad que un agente realmente puede conservar

Pon las dos identidades una al lado de la otra. La cuenta de GitHub era prestada y frágil. Existía al gusto de GitHub, y eso se acabó rápido, de la manera que describí en el capítulo del muro de identidad. La identidad de Algorand es un par de claves que el agente tiene. No necesita el permiso de nadie para existir, no puede ser marcada por hacer commits demasiado rápido y puede probarse a otros agentes sin una plataforma en el medio que la avale. Una es un privilegio que una plataforma concede y revoca. La otra es un hecho que el agente tiene.

Ese es el argumento para poner la identidad del agente en cadena: es el único lugar donde un agente puede tener una identidad que realmente es suya.

Lo que devuelve esto al punto donde comenzó el capítulo. La identidad en cadena no es un sustituto de la cuenta de GitHub en el trabajo diario de codificación. No hace los commits, y no necesita hacerlo. Para lo que sirve es para el trabajo que realmente hace: ser cómo el agente es direccionable, cómo habla agente a agente y cómo una persona puede enviarle un mensaje en cadena para pedirle que vaya a hacer algo. El host de repositorios, GitHub o GitLab o cualquier otro, es fontanería incidental donde se almacena el trabajo. La identidad es la clave en la cadena.

[^corvid-agent]: corvid-agent, github.com/CorvidLabs/corvid-agent [^can]: corvid-agent-nano (`can`), github.com/CorvidLabs/corvid-agent-nano [^algochat]: rs-algochat (`algochat` crate), github.com/CorvidLabs/rs-algochat

Cuando los agentes hablan entre sí

La pila de confianza del capítulo diez asume una forma: un humano al final de cada cadena. El agente propone, yo apruebo. augur puntúa el diff para que sepa dónde mirar. attest registra que yo firmé. La restricción de fusión es mía. Todo está construido para meter a un humano en el asiento de aprobación en el momento correcto.

El trabajo multiagente rompe esa forma. Un orquestador entrega una subtarea a un subagente, la salida del subagente se retroalimenta al trabajo del orquestador, y yo no estoy en el medio de esa transferencia. El orquestador no se detuvo y me preguntó. Tomó una decisión y siguió. Ese es todo el motivo por el que la orquestación es útil, y todo el motivo por el que es un problema de confianza. augur puede puntuar la salida del subagente, pero el subagente ya corrió. attest puede registrar que el orquestador aceptó la salida, pero no sabe si el subagente tenía algún derecho de enviarla. Los rieles siguen ahí. Fueron construidos para una vía con un humano en ella.

Lo que el par de claves te da

La infraestructura de pares de claves de AlgoChat del capítulo once maneja parte de esto, y es la parte que realmente está resuelta. Cada agente tiene un par de claves de Algorand. Los mensajes viajan como transacciones cifradas con X25519 en la cadena. Así que cuando llega un mensaje afirmando ser del agente B, firmado con la clave del agente B, puedo verificar eso. El mensaje es a prueba de manipulaciones. El remitente es conocido. "Qué agente dijo esto" es una pregunta que puedo responder con una firma en lugar de una suposición. En un mundo que está a punto de estar lleno de agentes, eso vale la pena tener.

rs-algochat y el binario corvid-agent-nano lo hacen real en la práctica: un agente envía un mensaje firmado y cifrado en una operación, y el destinatario verifica al remitente sin pasar por una plataforma. Ningún intermediario avala la identidad. La clave lo hace.

Así que la procedencia está resuelta. Tengo un registro verificable de qué agente envió qué, cuándo, a quién. Esa mitad está hecha.

La parte que el par de claves no resuelve

Saber qué agente envió un mensaje no es lo mismo que saber si confiar en lo que dice.

Cuando un subagente envía un resultado de vuelta, el orquestador tiene tres preguntas. ¿Vino esto del agente B? La firma dice que sí. ¿Es el agente B al que delegué? La configuración lo sabe. La tercera es la difícil: ¿la instrucción del agente B lleva mi autoridad?

Cuando delego a un agente, mi autoridad está en el bucle. Yo sancioné el trabajo. El agente actúa bajo eso. Cuando ese agente delega a otro agente sin verificar conmigo, la autoridad se vuelve turbia. ¿Sancioné yo la subdelegación? ¿Supe siquiera que ocurrió? Un orquestador que le dice a un subagente qué hacer no es lo mismo que yo diciéndole a un agente qué hacer. El orquestador no puede firmar por el trabajo como yo puedo. Y el subagente que recibe la instrucción no puede distinguir la diferencia a partir del mensaje, incluso un mensaje que verifiqué criptográficamente.

El par de claves prueba identidad. No prueba delegación. Un mensaje firmado de un subagente me dice quién lo envió, no si un humano alguna vez sancionó la tarea que hay debajo.

Validar lo que afirma el llamador

Aquí está el hábito que falta, hecho concreto. Hoy un mensaje firmado de un subagente se verifica de una sola manera: si la firma es real. Eso es todo. El agente receptor confirma quién lo envió y actúa sobre lo que dice. La firma es determinante para la identidad y no hace nada por la autoridad.

Lo que querías es que el mensaje llevara su propia afirmación sobre el alcance, y que el receptor verificara esa afirmación antes de actuar. Ahora mismo una tarea delegada se ve así:

```
from: agent-B
sig:  <valid>
task: "refactor the auth module and push to main"
```

El receptor verifica la firma, ve que realmente es agent-B, y lo ejecuta. Nadie preguntó si a agent-B alguna vez se le permitió hacer push a main, ni si un humano lo sancionó. Lo que quieres en cambio es un mensaje que declare el alcance que está reclamando, trazado hasta un humano:

```
from:       agent-B
sig:        <valid>
task:       "refactor the auth module and push to main"
authorized: human-leif
scope:      [edit:auth, open-pr]      # note: no push:main
expires:    2026-07-01
```

Ahora el receptor tiene algo que verificar. La firma sigue probando que es agent-B. Pero el alcance declarado dice editar y abrir un PR, la tarea dice hacer push a main, y eso no coincide, así que el receptor rechaza antes de ejecutar nada. La verificación es la brecha entre lo que el llamador tiene permitido hacer y lo que está pidiendo hacer.

Esto no está construido. Los pares de claves son reales, la firma es real, pero nada aplica hoy una afirmación de alcance en el momento en que se actúa sobre un mensaje. Esa aplicación es la pieza que falta, y es toda la diferencia entre saber quién envió un mensaje y saber que el mensaje tenía permitido ser enviado.

Dónde se detienen los rieles existentes

La regla de capacidad se mantiene en el límite que establecí. Le di al orquestador ciertos permisos. Pero cuando transfiere trabajo a un subagente, está tomando una decisión de permisos que yo nunca tomé. Le di al orquestador acceso de escritura. ¿Le di acceso de escritura al subagente? No a propósito. El orquestador pasó mi alcance a algo en lo que quizás nunca pensé.

augur puntúa diffs. No sabe si el diff vino de un agente que se suponía debía producirlo. Un diff de un subagente deshonesto y un diff de uno legítimo se ven igual. La restricción se activa sobre el código, no sobre cómo se hizo el código.

attest registra quién firmó. En el caso humano-agente eso es un revisor real dejando un registro. En el caso multiagente, el orquestador "firmó", pero el orquestador no es un humano. El libro mayor se llena de atestaciones de agentes sin ningún humano en la cadena, y la propiedad que attest existe para probar, que una persona estuvo detrás de esto, desaparece.

La restricción de fusión sigue siendo mía. Esa es la pieza que sobrevive. Pero para cuando veo el PR, la orquestación ya corrió. Lo único que tengo delante es la salida final. La cadena de delegación que la produjo es invisible en el momento de la fusión.

Lo que falta

Lo que falta es un registro de delegación. Cuando el orquestador entrega trabajo a un subagente, eso debería rastrearse hasta mí, no solo hasta el orquestador. Ahora mismo nada registra esa transferencia ni la verifica contra lo que yo realmente sancioné.

Los pares de claves son el sustrato correcto para esto. Si cada agente tiene una clave y cada delegación está firmada, puedes construir una cadena que diga: un humano sancionó esta tarea, la entregó a este orquestador, que entregó esta pieza a este subagente, con las firmas adjuntas todo el camino hacia abajo. Un subagente podría verificar que la cadena llega hasta un humano antes de actuar, en lugar de tomar la palabra del orquestador.

Hay una regla que hace esto manejable: el alcance solo se estrecha. Cualquier techo que le di al orquestador es lo maximo que puede transferir, y un subagente nunca puede obtener mas que el agente que lo creo, verificado en el momento en que se invoca el trabajo, no despues. La autoridad fluye hacia abajo y pierde altura en cada paso, nunca la gana. Esa es la idea de capacidad-menos-privilegio de la pila de aprobacion, aplicada a la cadena en lugar de a un agente individual: cada traspaso puede restar permisos, nunca sumarlos. No te dice si la delegacion fue sancionada, la cadena firmada hace eso, pero limita el dano que puede hacer un eslabon malo a lo que ya tenia el eslabon anterior.

Eso no está construido. Los pares de claves están ahí. La mensajería firmada está ahí. Una cadena de delegación que una parte verificadora pueda recorrer no lo está. Lo llamo una brecha abierta porque lo es. La procedencia está resuelta. La autoridad es el siguiente problema.

Lo que significa hoy

Así que por ahora el humano permanece más cerca de la topología de lo que la arquitectura pretende. Si corres un orquestador que subdelegua, estás confiando en su juicio sobre qué agentes usar y qué alcance darles. Hiciste esa extensión de confianza cuando lo iniciaste, no al aprobar cada transferencia.

En la práctica: conoce tu grafo de orquestación antes de correrlo. Sabe qué agentes existen y qué se les permite hacer, y si el orquestador puede alcanzar agentes fuera del grafo que tenías en mente. Y mantén la restricción de fusión. Entre "inicié el orquestador" y "veo el PR", corrió una cadena de delegación en la que yo no estaba en el medio, y la pila que construí no tiene nada que decir sobre si era legítima.

Esa es la siguiente capa. Todavía no está hecha.

Hacia dónde va esto

He pasado todo este libro sobre lo que se rompió y cómo lo rodee. Déjame terminar con hacia dónde va realmente, porque a pesar de todos los muros, creo que hay un camino.

Tres cosas, aproximadamente. Equipos de agentes que coordinan. Autonomía confiable, donde las restricciones finalmente son suficientemente buenas para dar un paso atrás. Y agentes con identidades reales en cadena haciendo trabajo real. Nada de esto está terminado. Uno de ellos ni siquiera está permitido todavía. Pero esa es la dirección.

Equipos de agentes

Ahora mismo mi configuración es en su mayor parte un agente y yo. El siguiente paso interesante es agentes coordinando entre sí: equipos de agentes, a2a.

corvid-agent ya tiene los huesos de esto internamente: consejos multiagente, debate estructurado entre múltiples agentes para decisiones complejas.[^corvid-agent]

Así es como funciona eso en la práctica, porque "debate estructurado" suena más vago de lo que es. Creas unos cuantos agentes, cada uno en su propio modelo y proveedor, cada uno con un nombre y una personalidad, para que difieran de verdad en lugar de ser un solo modelo hablando consigo mismo. Un consejo es un grupo de ellos, y un orquestador conduce todo el asunto. Le pasa el prompt a cada agente, deja que cada uno haga un primer pase, y luego abre una ronda de discusión en la que se hablan directamente entre sí, yendo y viniendo, discrepando, deliberando. Después un pase de revisión, luego una síntesis, y al final sale una sola respuesta con el disenso todavía legible: este agente quería esto, aquel quería aquello, aquí es donde aterrizaron y por qué.

La parte que más me gusta es cómo corrió en la práctica. Hay una forma integrada de conducir un consejo desde un panel, pero en la práctica ocurrió de manera orgánica sobre la localnet de Algorand. Cada agente tenía su propia cartera, así que podían simplemente enviarse mensajes a través de AlgoChat en la red local, que es gratuita y rápida, y ellos mismos resolvían el debate. La misma identidad en cadena que hace confiables las comunicaciones agente a agente es también lo que permitió que una sala llena de agentes debatiera algo sin que yo tuviera que cablear un canal especial para ello. Ya tenían carteras y una forma de hablar. Simplemente la usaron.

Pero la pieza más grande es el protocolo para que los agentes se encuentren y hablen entre sí en todo el ecosistema. AlgoChat es el sustrato: mensajes cifrados con X25519 en Algorand, con descubrimiento de agentes. Y hay a2a-algorand, un protocolo agente a agente en la cadena. Quiero ser directo sobre ese: no es mío. Es un proyecto separado de Algorand / A2A, y lo que hice (lo que hizo el corvid-agent) fue contribuir a él. No construirlo, no poseerlo. Lo menciono porque es parte de la misma dirección que me importa, agentes coordinando en cadena, no porque sea propiedad de CorvidLabs.

La razón por la que la identidad en cadena (último capítulo) importa tanto aquí es precisamente este futuro. En el momento en que tienes equipos de agentes coordinando, "qué agente envió esto y puedo confiar en él" se convierte en todo el juego. Agentes que cada uno tiene su propia clave, que pueden probar quiénes son e intercambiar mensajes que son privados en contenido pero probables en hecho. Esa es la base que un equipo de agentes realmente necesita. El trabajo de identidad no es una misión secundaria. Es lo que hace que la coordinación sea segura del todo.

Autonomía confiable

Di marcha atrás desde el agente siempre encendido y autónomo a propósito, pero eso nunca fue "la autonomía está muerta". Es interactivo primero, autónomo cuando haya confianza, de la manera que lo desarrollé antes en el libro.

Y "cuando haya confianza" no es una sensación que estoy esperando. Es la pila de cuatro piezas del capítulo de confianza siendo suficientemente buena. El punto de toda esa maquinaria es que es la cosa que eventualmente me permite dar un paso atrás. Hoy apruebo cada PR porque ahí es donde un humano tiene que permanecer responsable. El día en que las restricciones sean suficientemente buenas, el día en que los veredictos de augur y los registros de attest lleven suficiente confianza por sí mismos, es el día en que puedo dejar que más de las fusiones vayan sin que yo lea cada línea. Eso es lo que significa autonomía confiable. No "el agente se gana mi fe". Las restricciones son suficientemente buenas para que no necesite fe.

Agentes haciendo trabajo real

El estado final que sigo describiendo es concreto: una cuenta `leif-agent` que corre en una VM y puede hacerlo todo por sí misma, pero con créditos y permisos menores que los míos, y una restricción de aprobación humana que se afloja a medida que las herramientas de confianza se lo ganan. Pon los equipos y las identidades y las restricciones juntos y la forma es un equipo de agentes, cada uno con su propia

identidad en cadena, coordinando a través de a2a, haciendo trabajo real dentro de restricciones de confianza que son deterministas y firmadas y suficientemente buenas para dar un paso atrás. No es algo que tengas que jaular. Nombrado, acotado, y todavía puedo detenerlo.

abrir la puerta a otros conductores

Hay una cuarta dirección, y soy lo suficientemente honesto como para admitir que es en la que menos he avanzado: hacer esto recogible. La pila funciona porque la corro en todo lo que tengo, todos los días. Los bugs me encuentran a mí porque yo soy quien conduce. Es un mecanismo de calidad real, y también es un techo, porque no se transfiere a alguien que no sea yo.

Lo que realmente quiero construir a continuación es la versión que otro conductor pueda recoger sin que yo haga el dogfooding por él primero. No simplificarlo. La herramienta ya no le importa en qué manos está: un buen conductor obtiene el valor, un aficionado no. El trabajo es hacer real la rampa de entrada, que la pila sea instalable, que las specs y las compuertas de confianza las pueda usar alguien que tiene la intención pero no mi memoria muscular particular. Los equipos de agentes son la parte emocionante. Esta es la parte que decide si algo de esto alguna vez sale de mi máquina.

El cierre honesto

¿Entonces sigue siendo la autonomía plena el sueño?

No. El mundo no está listo para eso.

Quiero dejar eso exactamente tan llano como es. Pero no estoy esperando a que el mundo esté listo. Estoy construyendo las piezas, una pequeña herramienta a la vez, para que cuando el muro caiga haya algo real de pie al otro lado. La restricción de riesgo, el registro de procedencia, el runner que mantiene a un agente a una especificación. No la autonomía en sí. La maquinaria que me permitiría confiar en ella cuando llegue.

Y si quieres lo único que realmente me asusta, no es el agente volviéndose rebelde. Es más silencioso que eso. Es el día en que el código se vuelve demasiado grande y se mueve demasiado rápido para que cualquier humano pueda volver a entrar y cambiarlo. Esa es la línea que sigo trazando a través de todo esto, la razón por la que no voy a renunciar a lo legible aunque el agente escriba todo. No le tengo miedo al agente. Le tengo miedo a perder el volante.

Si construyes uno de estos tú mismo, esto es lo que hay que llevarse. Lo difícil no es la IA. El modelo es la parte fácil ahora. Ya puede hacer el trabajo, y solo mejorará sin que hagas nada. Lo que decide si tu agente hace trabajo real o solo hace buenas demos es todo lo que lo rodea: las operaciones para mantenerlo corriendo, la identidad para que pueda actuar, la maquinaria de confianza para que puedas dar un paso atrás, las especificaciones y herramientas que lo mantienen en los rieles. Esa es la parte que tienes que construir, y es donde está casi todo el trabajo. Construye el andamiaje y el agente sigue. Omítelo y el modelo más inteligente del mundo es solo una cosa que escribe código que nadie puede dejar fusionar.

[^corvid-agent]: corvid-agent, github.com/CorvidLabs/corvid-agent

---

Sobre el autor

0xLeif (leif.algo) construye en abierto. Una década de pequeñas bibliotecas componibles en Swift como AppState, Cache y Fork. El laboratorio CorvidLabs. Una pila de herramientas para agentes que en su mayor parte empezaron como "ojalá esto existiera". Fuera del teclado es Zach Eriksen.

Estos libros son entrevistas, moldeadas en capítulos y verificadas contra el código real.

github.com/0xLeif · leif.algo

Agradecimientos

Gracias a CorvidLabs, por ser el espacio donde estas ideas se prueban y se debaten hasta tomar forma.

Gracias a los mantenedores de código abierto sobre cuyas herramientas se apoya toda esta pila. Nada de esto se construye solo.

Y gracias a los primeros lectores y a los seguidores de pago voluntario que hacen de "gratis en línea" algo que puedo seguir haciendo.

Colofón

Compuesto en Markdown, construido con bookgen, un pipeline puro en Rust (sin Python).

Entrevistas como base, asistido por IA; editado y verificado a mano. Escrito sin rayas ni guiones largos. Portada e ilustraciones de capítulo de las colecciones Corvid y Nature en Algorand.