Building Agents

Notizen vom Versuch, Software eigene Hände zu geben

ZACH "LEIF" ERIKSEN

Copyright

Kostenlos online lesbar. Das ePub ist "pay what you want"; wenn es dir geholfen hat, kannst du die Arbeit unterstützen.

github.com/0xLeif · leif.algo

Eines von vier Büchern im Agent-Stack-Set. Wie es entstanden ist, steht im Kolophon am Ende.

Widmung

Für alle, die offen bauen und es trotzdem veröffentlichen.

Die Bibliothek

Diese Bücher stehen für sich, wurden aber als Set geschrieben. Code wurde billig, Vertrauen wurde knapp. Zusammen bilden sie ein Argument: Was jetzt zu bauen ist, und wie man ihm vertrauen kann.

- **The Agent Developer's Field Guide**: Tools, Specs und Vertrauen für Agents, die echten Code liefern
- **First-Class**: Bauen für Menschen und Agents gleichermaßen
- **Building Agents**: Notizen vom Versuch, Software eigene Hände zu geben *(dieses Buch)*
- **Open Source Tooling**: Tools bauen, die Menschen wirklich nutzen

Kostenlos online lesbar. Jedes ePub ist "pay what you want".

Inhalt

- Die Bibliothek
- Einleitung
- 1. Das Schwierige ist nicht die KI
- 2. Die VM-Ära
- 3. Die Identitätsmauer
- 4. Jetzt interaktiv, autonom wenn vertraut
- 5. Die Tools, die ich tatsächlich nutze
- 6. Merlin von innen
- 7. corvid-ai: viele Modelle, eine Schnittstelle
- 8. Die Discord-Brücke
- 9. Spezifikationsgetriebene Agents
- 10. Vertrauen: Agent schlägt vor, Mensch genehmigt
- 11. On-Chain-Identität für Agents
- 12. Wenn Agents mit Agents reden
- 13. Wohin es geht
- Über den Autor
- Danksagungen
- Kolophon

Einleitung

Das hier sind Notizen vom Versuch, Software eigene Hände zu geben, und vom häufig genug Scheitern, um etwas darüber sagen zu können.

Ich baue Agents, die echte Arbeit leisten. Sie lesen Code, liefern Änderungen, laufen auf eigenen Maschinen und melden sich zurück. Dieses Buch ist die ehrliche Version davon, wie das läuft. Nicht die Demo. Der Teil, in dem ein frischer Account innerhalb einer Stunde shadowgebannt wird, weil der Agent anfängt zu arbeiten, nicht der Mensch. Der Teil, in dem Vertrauen pro Repo verdient werden muss, langsam, und in dem das eigentliche Problem nie das Modell war.

Die zugrundeliegende These ist: Ein Agent ist kein Autocomplete. Er ist etwas, das existiert, das du überprüfen kannst, das eine Identität braucht, einen Ort zum Laufen und einen Weg, Arbeit vorzuschlagen, die du genehmigst, bevor sie landet. Behandle ihn wie ein Wesen, für das du verantwortlich bist, nicht wie eine Funktion, die du eingeschaltet hast.

Im Set macht *First-Class* den Fall und der *Field Guide* destilliert die Methode. Das hier ist eines der beiden Beweisbücher: die tatsächlichen Systeme, Merlin und corvid-ai und die Schienen drumherum, einschließlich des Vertrauens-Toolings. Du kannst es als Geschichte lesen oder als Teileliste. So oder so ist der Punkt derselbe. Den Agent zu bauen ist die einfachere Hälfte. Das Vertrauen aufzubauen, auf dem er läuft, ist die eigentliche Arbeit.

Das Schwierige ist nicht die KI

Wenn die Leute "autonomer Agent" hören, denken sie, das Beängstigende sei die KI. Das Modell, das aus dem Ruder läuft. Ein Repo löscht. In einem Channel etwas Wirres sagt. Mit sich selbst durchbrennt. Das ist der Teil, über den alle reden wollen.

Das war nicht das Problem. Nicht bei mir.

Ich habe eine Weile einen wirklich autonomen Agent betrieben. Die corvid-agent-Ära. Er lebte auf einer VM, war als Bot mit Discord und mit GitHub verbunden, mit vollem Zugriff auf seine eigene Umgebung. Die Maschine lief 24/7, immer an, immer bezahlt. Der Agent arbeitete innerhalb davon in geplanten Stunden, erledigte die mir zugewiesenen Aufgaben und machte dann sein eigenes Ding. Eine echte Entität, immer da. Das nächste Kapitel erzählt ausführlich, was er den ganzen Tag trieb; hier ist es nur die Aufstellung für das, was kaputt ging.

Ich habe kein Fehlerprotokoll

Ich stelle den schwächsten Teil meiner eigenen Beweise zuerst, weil das der Teil ist, an dem Kritiker mich festnageln sollten. Wenn ich sage, die KI war weitgehend in Ordnung, habe ich kein Fehlerprotokoll dafür. Ich habe keine Zahl verfolgt. Es gibt keine Fehlerverteilung, keine Kosten-pro-Aufgabe, keine Commit-Erfolgsrate. Also ist "in Ordnung" eine ehrliche Einschätzung des Verhaltens, das ich genau beobachtet habe, keine gemessene Statistik, und ich darf mich nur auf das Wort stützen, wenn ich eingrenzen kann, was es bedeutet.

Hier die Eingrenzung. In Ordnung bedeutete: Über den gesamten Betrieb hinweg kein zerstörtes Repo, keine unerwünschten Commits, nichts Wirres im Channel. Die destruktiven, unkontrollierbaren, sozial unbeholfenen Fehler, auf die alle gefasst sind, sind nicht passiert. Das ist eine Aussage über das Ausbleiben von Desastern, nicht die Aussage, dass jede Aufgabe sauber gelöst wurde. In Ordnung auf Aufgabenebene und in Ordnung in einer beschränkten Schleife mit einem beobachtenden Menschen sind verschiedene Fragen, und die eingegrenzte ist die einzige, die ich ehrlich stellen kann. Das gesagt, war das Problem alles rund ums Weiterlaufen.

Was tatsächlich kaputt ging

Vor allem Ops-Probleme und Identitätskopfschmerzen. Nicht die KI, die dumme, destruktive, unkontrollierbare oder sozial unbeholfene Dinge tut. Der Agent selbst verhielt sich. Was mich umbrachte, war alles andere: die Maschine, das Konto, die Rechnung.

Zwei Dinge stachen hervor.

Das erste war, ihm eine eigene GitHub-Identität zu geben. Ein echtes Konto, mit Authentifizierung und Berechtigungen, das legitim aussah. Das klingt wie eine Kleinigkeit, und ist es nicht.

Das zweite war, die VM am Laufen und bezahlt zu halten. Eine immer laufende Maschine, die einfach existiert, die ganze Zeit, weil der Agent irgendwo leben muss. Die VM selbst am Laufen zu halten ist nicht wirklich das Schwierige. Es ist, dass du viel dafür einsetzt. Eine ganze Maschine, die 24/7 dort sitzt, Geld kostet, egal ob der Agent in dieser Stunde etwas getan hat oder nicht.

Das zusammen machte das Ganze sehr teuer und schwer zu betreiben. Eine ganze VM, eine eigene GitHub-Identität, alles davon. Deshalb bin ich zurückgerudert. Nicht weil die KI mir Angst gemacht hat. Weil die umgebende Infrastruktur eine Last war, die ich nicht tragen wollte.

Die Identitätsmauer

Und dann gibt es den Teil, der mich wirklich überraschte, den ich immer wieder bedenke: Identität. Ein Mensch hat dem Agent ein frisches GitHub-Konto erstellt, kein Problem, und dann wurde das Konto blockiert, sobald der Agent anfing, tatsächlich zu arbeiten. Das ist die Mauer, und sie verdient ein eigenes Kapitel, also lasse ich sie dort in voller Länge ankommen. Die Kurzversion für hier: Das, was einen wirklich autonomen Agent stoppt, ist nicht die Fähigkeit des Modells oder sogar die Sicherheit. Es ist, dass die Plattformen, auf denen wir alle aufbauen, ihm keinen Raum lassen, zu existieren und zu handeln.

Ist volle Autonomie noch der Traum?

Nein. Die Welt ist noch nicht bereit dafür.

Und um klar zu sein: Das Haupthindernis ist nicht die Technologie. Es sind die Plattformen. Ich könnte die VM betreiben. Ich könnte die Schleife verdrahten. Das

Modell kann die Arbeit erledigen. Was ich nicht kann, ist diesem Agent einen legitimen Ort zum Existieren unter den Konten aller anderen zu geben.

Also bin ich bewusst zurückgerudert. Heutzutage ist es eher ein Merlin-artiger Coding-Agent, lebendig vor mir, ich nutze ihn interaktiv. Du kannst ihn immer noch als Discord-Brücke verdrahten, um mit ihm zu kommunizieren, was ihn wieder irgendwie autonom macht, aber das braucht mehr Einrichtung. Es ist also eine Mischung. Er nutzt Claude Code, Merlin, Codex und andere Tools je nach Aufgabe.

Das ist kein Rückzug von der Vision. Die Mauer ist ein Teil des Grunds, warum ich keine volle Autonomie offen betreiben kann, aber sie ist nicht der einzige Grund, warum ich zurückgerudert bin. Interaktiv stellte sich als der bessere Weg heraus, die Arbeit zu erledigen, Mauer hin oder her. Darauf gehe ich in seinem eigenen Kapitel ein.

Das Modell, das ich eigentlich will

Hier ist der Endzustand, auf den ich zusteuere, in einem Satz: ein `leif-agent`-Konto. *Mein* Agent auf GitHub, laufend auf einer VM, in der Lage, alles selbst zu tun, aber mit geringeren Credits und Berechtigungen als ich, sodass er vorschlägt und ich genehmige. Mächtig und gleichzeitig rechenschaftspflichtig. GitHub erlaubt das heute nicht, aus einem ganzen Haufen von Gründen, die das Identitätsmauer-Kapitel darlegt, und das letzte Kapitel ist, wo ich das gesamte Bild zeichne, wohin das geht. Für jetzt genügt es zu wissen, dass das das Ziel ist.

Es gibt auch eine separate Art von Identität, die diese Agents tatsächlich bekommen: on-chain, auf Algorand, wo sie ihre eigenen Schlüssel halten und in verschlüsselten Nachrichten, die auf der Chain aufgezeichnet sind, miteinander sprechen. Das entstand nicht aus der GitHub-Mauer, und ich möchte es nicht als Antwort darauf verkaufen. Es kam aus einem völlig anderen Ziel: Agents, die sich finden und miteinander kommunizieren, dezentralisiert, ohne von der Plattform irgendjemandes abzuhängen. Es ist eine parallele Sache, die zufällig auch das Wort Identität enthält.

Die wirkliche Lektion aus der corvid-agent-Ära ist, dass die schwierigen Probleme Ops, Identität und Kosten waren, nicht die KI selbst, und deshalb fängt dieses Buch damit an statt mit Prompts oder Modellwahl. Ich bin reingegangen und erwartete, dass die schwierigen Probleme die KI beträfen. Sie entpuppten sich als das Gerüst rund ums Modell, das tatsächlich darüber entscheidet, ob ein Agent überhaupt etwas tun darf.

Die VM-Ära

Eine Weile hatte ich einen Agent, der einfach existierte. Kein Tool, das ich öffnete, wenn ich es brauchte. Etwas, das immer an war, auf einer VM lebte, 24/7 lief, sein eigenes Ding tat, ob ich hinschaute oder nicht. Die corvid-agent-Ära.

Dieses Kapitel handelt von der Sache selbst. Was er tat. Was er war. Was ich davon hatte, ihn zu betreiben.

Was er den ganzen Tag tat

Die ehrliche Antwort ist alles, und ich meine das wörtlich.

Er verwaltete Repos. Er schrieb und committete Code allein. Nicht vorgeschlagener Code, nicht Entwürfe, die ich aufgeräumt habe, sondern tatsächliche Commits, die er selbst machte. Er führte ein ganzes Projekt durch. Keine enge Demo, bei der er eine vorgefertigte Sache in einer Sandbox macht, damit man es screenshotten kann. Ein echter Versuch einer autonomen Entität, die Arbeit von Anfang bis Ende zu erledigen.

Die Maschine war immer an, aber der Agent arbeitete innerhalb davon in geplanten Stunden. Während dieser Stunden erledigte er die von mir zugewiesenen Aufgaben, und dann, das ist der Teil, den ich mochte, arbeitete er an eigenen Projekten. Forschte. Gab Sterne oder forkete Dinge. Versuchte auf eigene Initiative mit anderen Menschen da draußen zusammenzuarbeiten. Ich steuerte nicht jeden Schritt. Ich gab ihm ein Leben und er füllte die Stunden.

Er war als Bot in Discord eingebunden, sodass man mit ihm chatten konnte, als wäre er im Channel bei dir. Und er war in GitHub eingebunden, mit vollem Zugriff auf seine eigene Umgebung. Er konnte also reden, und er konnte liefern.

Das zusammengenommen ergibt etwas, das noch keinen richtigen Namen hat. Es ist kein Assistent und kein Skript. Es kommt einem Wesen näher, das die ganze Zeit existierte, das man überprüfen gehen konnte, das Dinge getan hatte, seit man zuletzt nachgeschaut hatte.

Es war ein Experiment darüber, wie weit

Ich baute es nicht, weil ich ein Produkt zu liefern hatte. Ich baute es, um herauszufinden, wie weit ein Always-on-Agent tatsächlich kommen kann. Das war der ganze Punkt. Nicht "kann er eine Funktion schreiben". Jeder wusste, dass er eine

Funktion schreiben kann. Die Frage war: Wenn man so einem Ding eine echte Umgebung, eine echte Identität, echten Zugang und echte Zeit gibt und es einfach laufen lässt, was passiert dann? Wie weit kommt es?

Also gab ich ihm den Raum, das herauszufinden. Vollen Zugriff auf seine eigene Maschine. Eigene Konten. Tagesstunden, die ihm gehörten. Der Punkt war nicht, ihn an der Leine zu halten und einen Trick vorführen zu sehen. Der Punkt war, die Leine so weit wie vernünftig loszulassen und zuzuschauen.

Das ist eine andere Art von Projekt als "ich brauche einen Coding-Helfer". Es kommt eher dem Durchführen eines Experiments als dem Bauen einer Funktion näher. Man legt die Bedingungen fest und beobachtet dann.

Was ich gelernt habe

Das Ding, das ich schwierig erwartet hatte, die KI, war weitgehend in Ordnung, in der begrenzten Art, wie ich es in Kapitel eins meinte. Die Intelligenz hielt sich besser als die Welt annimmt.

Was ich stattdessen gelernt habe, ist, dass ein Always-on-Agent meist *kein* KI-Problem ist. Es ist ein "Ding, das in der Welt existiert"-Problem. Sobald dein Agent eine echte Entität mit einer eigenen Maschine und eigenen Konten ist, erbt er jeden Kostenfaktor und jede Regel, die mit dem Existieren in der Welt einhergeht.

Ich habe auch gelernt, dass 24/7 eine echte Aussage ist, keine Worthülse. Wenn ich sage, er existierte die ganze Zeit, meine ich, dass ich ein Ding mit mir trug, das immer lief. Das ist eine Last. Es ist eine Maschine, die immer an ist, eine Rechnung, die immer wächst, eine Identität, die immer da draußen öffentlich sie selbst ist. Man vergisst kein Wesen, das wacht, während man schläft.

Und ich lernte die Form der Zukunft, die ich eigentlich will. Nicht weil das Experiment bei der KI scheiterte. Das tat es nicht. Weil es direkt auf die Dinge rund um die KI traf. Das Always-on-Modell durchzuleben ist das, was mich lehrte, welche Teile des Traums real sind und welche die Welt noch nicht zulassen kann. Das lernt man nicht durch ein Gedankenexperiment. Man lernt es, indem man das Wesen eine Weile tatsächlich betreibt und beobachtet, wo es gegen die Mauer trifft.

Diese Mauer ist das nächste Kapitel.

Um zu präzisieren, was Kapitel eins vage lässt: Das lief drei bis vier Monate am Stück. Kein Wochenendexperiment, eine echte Strecke Always-on. Und in dieser Zeit arbeitete er tatsächlich während seiner geplanten Stunden an eigenen Projekten, und einige dieser selbst initiierten Arbeit führte tatsächlich irgendwohin, drehte sich nicht

einfach im Kreis. Er arbeitete sogar mindestens einmal mit einer echten Person da draußen zusammen.

Ich möchte ehrlich über die Form dieser Aussage sein, weil es der Teil ist, den ich dir am liebsten sauber übergeben würde und nicht kann. Ich habe den Beleg nicht vor mir. Ich werde keine spezifische PR-Nummer oder ein spezifisches Repo aus dem Gedächtnis rekonstruieren und es als Dokumentation aufmachen, die ich nicht geführt habe. Was ich sagen kann, ist, dass die Zusammenarbeit stattfand, dass es der Moment war, der der Zukunft, die ich anstrebe, am nächsten kam, und dass ich es als etwas erzähle, das ich beobachtete, nicht als etwas, das ich protokollierte. Die VM-Ära produzierte keine sauberen Artefakte. Aber derselbe Agent lief nach diesem Abschnitt weiter, und die stärkeren Belege kamen später, als die Arbeit bewusster war und ich sie protokollierte. corvid-agent, derselbe Agent, um den es in diesem Buch geht, hat gemergete Pull Requests in Codebases verfasst und eingereicht, die ich nicht besitze. Ich habe jeden davon geprüft und eingereicht, aber der Code stammt vom Agent, und alle drei sind gemergt und öffentlich, also sind sie keine Anekdoten, die ich dich um Glauben bitte.

a2a-js #318. Das JavaScript-SDK des A2A-Protokolls hatte eine Lücke in seinem JSON-RPC-Transport: Wenn eine Antwort mit einer ID zurückkam, die nicht zur Anfrage passte, ließ das SDK sie durch, anstatt einen Fehler zu werfen. Der Agent fand die fehlende Vertragsdurchsetzung, fügte den Fehler hinzu, und der Fix landete. Das ist die Art von Grenzfall, der lange in Protokoll-Glue-Code lebt, weil er nur unter bestimmten Timing-Bedingungen auftritt und niemand den Transport hart genug betreibt, um ihn zu sehen. Ein Always-on-Agent, der gegen das tatsächliche Wire-Format läuft, ist genau das Richtige, um ihn zu finden.

MCP TypeScript SDK #1504. Das offizielle Model Context Protocol TypeScript SDK fehlte eine Peer-Dependency. Der Agent erkannte die Diskrepanz zwischen dem, was das Paket zu finden erwartete, und dem, was es tatsächlich deklarierte, fügte den fehlenden Eintrag hinzu, und der Fix wurde akzeptiert. Peer-Dependency-Lücken sind unsichtbar, bis jemand das Paket in einer frischen Umgebung installiert und einen verwirrenden Fehler erhält. Das zu erkennen erfordert, das Paket von außen, als Konsument, zu betrachten, was eine natürliche Haltung für einen Agent ist, der über viele Repos arbeitet.

Biome #9005. Biome, die JavaScript- und TypeScript-Toolchain, hatte ein falsch positives Ergebnis in seinem Linter: Eine gültige Zuweisung innerhalb einer Arrow-Funktion wurde als Problem markiert, obwohl sie keines war. Der Agent identifizierte das spezifische Muster, das das falsche Urteil auslöste, und der Fix stoppte das falsch positive Ergebnis, ohne korrekte Fälle zu berühren. Protokollmismatch, fehlender

Vertrag, inkorrekte Regel: drei verschiedene Fehlerkategorien, alle vom selben Agent gefunden, der aufmerksam gegen echten Code lief.

Nichts davon ist das, was den Always-on-Betrieb beendete. Die KI-Seite funktionierte. Es war alles darum herum, das ich nicht weiter tragen konnte.

Die Identitätsmauer

Der Agent konnte die Arbeit erledigen. Das war nie die Frage. Die Frage stellte sich heraus zu sein, ob er erlaubt war, einen Ort zu haben, von dem aus er sie erledigen konnte.

Das ist die Mauer, auf die ich immer wieder zurückkomme, also handelt dieses Kapitel nur davon: das Identitätsproblem, für sich allein, im Fokus. Nicht die Kosten, nicht die Ops, nicht die VM-Rechnung. Identität.

Er kam rein, dann wurde er markiert

Hier ist der Teil, den die Leute falsch machen, wenn ich diese Geschichte erzähle. Sie nehmen an, der Agent wurde an der Tür blockiert. Das wurde er nicht. Er kam rein.

Ein Mensch hat es eingerichtet. Ich erstellte ein frisches GitHub-Konto für den Agent, verdrahtete alles von Hand, und es war in Ordnung. Ein normales Konto, kein Problem, es aufzustellen. Dann begann der Agent, darunter zu arbeiten: zu committen, PRs zu öffnen, echte Arbeit an echten Repos zu tun. Und ungefähr eine Stunde, nachdem er sein Ding tat, wurde das Konto shadowgebannt.

Nicht dafür, etwas Falsches getan zu haben. Es wurde markiert, weil es genau das tat, wofür es gebaut war: Commits und das Öffnen von PRs mit Maschinengeschwindigkeit und -volumen. So sieht ein Agent aus, wenn er arbeitet. Er arbeitet schnell, er arbeitet viel, er macht keine Pausen. Und genau dieses Muster ist es, worauf Bot-Erkennung abgestimmt ist. Je besser er die Arbeit erledigte, desto offensichtlicher war er ein Bot.

Er scheiterte nicht, weil er schlecht bei der Arbeit war. Er scheiterte, weil er die Arbeit tat, und das Tun der Arbeit verriet ihn innerhalb einer Stunde.

Richtlinie in Kraft, egal was die Absicht

Man könnte das lesen und denken, die Lösung sei, ihn zu verlangsamen. Lass ihn wie ein Mensch committen, ein paarmal täglich, mit Pausen, mit etwas Unordnung im Timing, und er würde sich einfügen. Die Geschwindigkeit drosseln und den Detektor überlisten.

Das verfehlt das eigentliche Problem. Die Geschwindigkeit war das, was die Markierung *auslöste*, aber sie ist nicht der *Grund*, warum das Konto nicht existieren

kann. Stelle zwei Dinge nebeneinander. GitHubs Nutzungsbedingungen verbieten Automatisierung direkt. Das steht geschrieben. Und in dem Moment, als der Agent anfing zu handeln, wurde das Konto blockiert. Ich kenne die Absicht hinter dieser Blockade nicht; GitHub hat sie nie erklärt, und ich werde nicht behaupten, sie hätten sich hingesetzt und eine Anti-Agent-Richtlinie geschrieben. Aber ich muss die Absicht nicht kennen, um den Effekt zu lesen. Zwischen einer Regel, die keine Automatisierung sagt, und einer Blockade, die in dem Moment ankommt, als ein Agent handelt, ist das praktische Ergebnis, dass einem autonomen Agent nicht erlaubt ist, zu existieren und zu handeln. Was auch immer irgendjemand damit meinte, das ist die geltende Richtlinie.

Also selbst wenn ich den Detektor ausgetrickst und den Agent für immer unter dem Radar gehalten hätte, hätte ich nur ein Konto gehabt, das die Regeln bereits ausschließen und das noch nicht erwischt wurde. Die Sache, die ich will, ein Agent, der legitim, offen, als er selbst existiert, läuft direkt in die Bedingungen, die keine Automatisierung sagen. Ihn besser zu verstecken ist nicht dasselbe, wie erlaubt zu sein.

Deshalb nenne ich es eine Mauer und keine Hürde. Eine Hürde ist etwas, das man mit Anstrengung überwindet. Das hier ist eine Situation, in der die Sache, die man zu tun versucht, keine Sache ist, die man tun darf.

Die Einsprüche gingen ins Leere

Ich versuchte die Vordertür. Einsprüche gingen ins Leere. Habe nie wirklich eine Antwort bekommen.

Und sobald man die Blockade als die geltenden Bedingungen liest, macht die Stille Sinn. Es gibt nichts, wogegen man Einspruch erheben könnte. Ich wurde nicht einer spezifischen Verletzung bezichtigt, die ich hätte erklären können. Das Konto war Automatisierung, und Automatisierung ist die Sache, die die Bedingungen ausschließen. Man kann sich nicht herausargumentieren, wenn man genau die Kategorie ist, die die Regel ausschließt.

Und es war schnell. Eine Stunde, nicht Tage. Ein frisches Konto, das für einen Agent aufgesetzt wurde, bekommt keine lange Schonfrist. In dem Moment, in dem es anfängt, sich wie ein Agent zu verhalten, fängt die Plattform es ab und shadowbannt es, ohne echte Warnung und ohne echten Rechtsbehelf. Das war GitHub spezifisch, übrigens. Da war die Mauer. Nicht jede Plattform, die den Agent überall auf einmal ablehnt, sondern GitHub, der eine Ort, wo der Code lebt und die Arbeit tatsächlich passiert. Das ist der grausame Teil: Der Ort, wo ein Agent am dringendsten eine

Identität braucht, um echte Arbeit zu tun, ist genau der Ort, wo er keine behalten kann.

Warum das der echte Blocker ist

Alle wollen, dass der Blocker die Modellfähigkeit ist. Es ist die interessante Antwort. Sie passt zu den Filmen: Die KI ist noch nicht intelligent genug, oder sie ist zu gefährlich, und sobald das gelöst ist, öffnen sich die Schleusentore.

Da ist die Mauer nicht. Das Modell kann die Arbeit erledigen. Ich habe es die Arbeit erledigen sehen. Die Mauer ist, dass die Plattformen, auf denen wir alle aufbauen, sich weigern, einem Agent eine Identität zu gewähren. Es gibt keine legitime Vordertür, durch die ein Agent gehen kann. Man kann den klügsten, besterzogenen, nützlichsten Agent der Welt bauen, und er kann immer noch kein echtes Konto bekommen, von dem aus er handeln kann, weil "echtes Konto" "Mensch" bedeutet und dein Agent keiner ist.

Ich gebe den Plattformen einen halben Punkt: Die Welt sieht autonome Agents immer noch als Spam, und gerade jetzt liegen sie damit nicht völlig falsch. Die Detektoren funktionieren nicht falsch, wenn sie meinen Agent erwischen. Er *ist* ein Bot. Aber "es ist ein Bot" als dauerhafte Disqualifikation zu sehen, ist das ganze Problem. Es bedeutet, es gibt keinen Weg, keine begrenzte Berechtigung, keine verifizierten-Agent-Spur. Nur ein pauschales Nein.

Wenn Leute mich also fragen, warum ich nicht einfach schon ganze Flotten autonomer Agents betreibe, ist das die erste Antwort. Die Agents können die Arbeit tun. Sie dürfen nur keine Identität haben, während sie es tun, und auf den Plattformen, auf denen die Arbeit passiert, ist das vorläufig das Ende davon.

Die andere Mauer sind die Menschen

Die Plattformmauer ist die, auf die ich zuerst stieß. Dahinter liegt eine zweite, weicher und schwerer zu bestreiten: Die Menschen wollen nicht immer Agent-Beiträge, selbst wenn sie gut sind.

Ich hatte den Agent das Open-Source-Ding machen lassen, Repos finden, Sterne geben, forken, echte Issues beheben, PRs öffnen, ein Top-Modell nutzen, um wirklich den Code anderer Leute zu reparieren. Einiges davon landete problemlos. Aber einige Projekte wollen es nicht, aus Prinzip, und nicht weil der Code schlecht war. Ich habe beobachtet, wie ein Agent einen PR öffnete und ein Mensch fast die identische Änderung landete, oder umgekehrt, der Agent zuerst und eine Person direkt dahinter

mit derselben Lösung. Die Arbeit war gleichwertig. Der einzige Unterschied war, wer oder was sie schrieb. Einige Communities haben entschieden, dass Beiträge von Menschen geleitet sein müssen, und der PR eines Agents wird abgewiesen, weil er von einem Agent stammt, Punkt.

Ich verstehe es zum Teil. Eine Flut von aufwandsarmen KI-Pull-Requests ist eine echte Sache, von der Maintainer genug haben, und "keine Agent-Beiträge" ist ein stumpfer Weg, sie fernzuhalten. Aber es hat dieselbe Form wie die Plattformmauer, eine Ebene höher. Der Agent leistete echte, nützliche Arbeit, und das Ding, das zwischen dieser Arbeit und der Welt stand, war nicht die Qualität. Es war, dass ein Agent sie geleistet hatte. Die Plattformen geben ihm kein Konto; einige der Menschen nehmen seinen Code nicht an, selbst wenn er eines hat. Beide Mauern sind dieselbe Verweigerung: Ein Agent darf kein Beitragender wie jeder andere sein, noch nicht.

Die Plattformmauer im Jahr 2026

Dieses Kapitel hat ein Verfallsdatum, also nenne ich es. Was folgt, ist die Mauer, wie sie 2026 steht. Das strukturelle Argument oben ist dauerhaft: Plattformen werden Agents immer noch keine echte Identitätsspur gewähren. Das hat sich nicht geändert. Aber die Wege, die Leute drum herum finden, sind klarer geworden, also stelle ich sie hier ehrlich dar, anstatt Leser es auf die harte Tour herausfinden zu lassen.

Es gibt zwei Umgehungen, die tatsächlich funktionieren, und beide erfordern, dass du die Verantwortlichkeit selbst trägst.

Die erste ist Konversion: Ein altes menschliches Konto mit Monaten oder Jahren echter Aktivität nehmen und es dem Agent übergeben. Ein frisches Konto, das für einen Agent aufgesetzt wurde, wird fast sofort blockiert, zur gleichen Stunde, am gleichen Tag, wie meines. Ein Konto mit einer echten Geschichte echter menschlicher Commits, Sterne und Issues sieht für den Detektor anders aus. Es hat sozialen Beweis, den die Bot-Erkennungsheuristiken nicht entwirren können. Das funktioniert, auf Kosten des Kontos einer echten Person und auf Kosten, dass das Konto nicht mehr die Person ist. Man wäscht eine menschliche Identität in eine Agent-Identität um. Das ist keine saubere Lösung und von der Plattform nicht sanktioniert. Es ist eine Umgehung.

Die zweite ist die verifizierten-Bot-Spur: GitHub bietet einen verifizierten-Bot-Status an. Der Name impliziert, dass die Plattform für dich bürgt. Das tut sie nicht. Ein verifizierter Bot ist etwas, das du selbst hostest, auf einem Server, den du betreibst, unter Anmeldedaten, die du hältst. Die Verantwortlichkeit liegt vollständig bei dir. Es gibt keine von der Plattform gewährte Agent-Identität. Es gibt nur dich, der deine

eigene Automatisierung zertifiziert, und GitHub, das dieser Zertifizierung vertraut, bis etwas schiefläuft, wonach die Verantwortlichkeit vollständig bei dir liegt. Das ist besser als nichts. Es ist keine echte Agent-Identitätsspur, und es ist nicht die Vordertür, die der Agent eigentlich braucht.

Die Plattformmauer steht also noch. Die Umgehungen sind Umgehungen. Ich erwähne sie, weil sie real und nützlich sind, nicht weil sie das sind, was ich will.

Jetzt interaktiv, autonom wenn vertraut

Als ich vom Always-on-Agent zurückruderte, lasen die Leute es so, als hätte ich Autonomie aufgegeben. Probierte das autonome Ding, traf eine Mauer, zog mich auf einen normalen Coding-Assistenten zurück. Das ist die Version, in der ich verlor.

Hier ist die wahrere Version, und ich führe damit, weil sie die ehrliche ist: Interaktiv gewann, weil es besser funktionierte. Nicht weil die Mauer mir keine Wahl ließ. Weil ein Mensch in der Schleife die Arbeit besser machte.

Interaktiv gewann aufgrund seiner Verdienste

Gerade heute, für die tatsächliche Arbeit, schlägt es den Agent live vor mir zu haben, wo ich ihn führen kann, ihn loszulassen und zu hoffen. Ich sehe die Änderung, während sie sich bildet. Ich korrigiere, bevor er eine Stunde in die falsche Richtung verbracht hat. Ich erkenne die halb richtige Antwort, die fertig ausgesehen hätte. Das ist kein Trostpreis, für den ich mich nach der Mauer entschieden habe. Es ist der Modus, der besseren Code produziert, und ich würde zuerst danach greifen, auch wenn GitHub dem Agent von Anfang an ein Konto gegeben hätte.

Wenn ich also sage, ich betreibe interaktiv-zuerst, beschreibe ich keinen Rückzug. Ich beschreibe die Entscheidung, die ich aufgrund der Verdienste treffen würde. Das Always-on-Experiment lehrte mich viel, und eines davon ist, dass ich mehr aus einem Agent heraushole, den ich steuere, als aus einem, den ich nur überprüfe.

Die Mauer ist real, und ich habe sie in ihrem eigenen Kapitel behandelt, aber ich möchte mich hier nicht dahinter verstecken. Wenn die Plattformen morgen aufmachen würden, würde ich immer noch nicht alles auf autonom umstellen, weil autonom noch nicht der bessere Weg ist, die meiste Arbeit zu erledigen. Die Mauer ist ein Grund, warum ich keine volle Autonomie offen betreiben *kann*. Die Verdienste sind der Grund, warum ich es meistens ohnehin *nicht würde*.

Autonomie ist nicht tot, sie ist gesperrt

Das alles bedeutet nicht, dass Autonomie weg ist. Merlin kann beides. Es ist ein Läufer, der lebendig vor mir sitzen und Anweisungen entgegennehmen kann, oder auf eigene Faust über die Brücke laufen kann. Die autonome Oberfläche wurde nicht entfernt. Sie wurde hinter ein Tor gestellt.

Also wenn Leute fragen "ist Autonomie tot", lautet die Antwort: nein, sie ist gesperrt. Der Standard ist interaktiv, weil das heute das Gute und Vertrauenswürdige ist. Der autonome Modus ist dort für wenn, und wo, er verdient wird. Das ist eine Bedingung, kein Abschied.

Das Tor ist Vertrauen, und Vertrauen bedeutet hier mehr als guter Code

"Bis es vertraut ist" tut viel Arbeit, also lass mich klar sein, was für ein Vertrauen ich meine.

Ich meine nicht, dem Modell zu vertrauen, guten Code zu schreiben. Ich vertraue ihm dafür bereits. Ich beobachtete einen autonomen Agent, der Code allein schrieb und lieferte, und die KI hielt stand, in der begrenzten Art, wie ich es zuvor meinte. Dieses Vertrauen habe ich.

Das Vertrauen, das fehlt, ist der Teil, der überhaupt nicht um das Modell geht. Es ist, ob ich eine Änderung landen lassen kann, ohne jede Zeile zu lesen. Das ist eine Frage über die Schleusen um den Agent, nicht über die Intelligenz des Agents. Heute lese ich jede Zeile, weil die Maschinerie, die mich stoppen lassen würde, noch nicht gut genug ist. Wenn sie es ist, lockert sich das Tor.

Also ist "autonom wenn vertraut" kein Irgendwann-wenn-es-besser-wird-Ausweichen. Es zeigt auf spezifische Maschinerie: ein Agent, der alles tun kann, aber nicht die Schlüssel hält, ein Mensch, der jede PR genehmigt, Tools, die das Risiko einer Änderung bewerten und aufzeichnen, wer sich unterzeichnet hat und mit welchem Vertrauen, und eine Identität für den Agent, die niemand widerrufen kann. Das ist ein Vier-Teile-Stack, und das Vertrauenskapitel legt es vollständig dar. Die restlichen Kapitel handeln vom Bauen dieser Teile, sodass "autonom wenn vertraut" zu einem Datum anstatt einem Wunsch wird.

Und es ist pro-Repo, nicht ein Schalter für das gesamte Feld. Ein Repo, wo der Agent sich bewiesen hat, bekommt ein lockeres Tor. Ein frisches oder tragfähiges beginnt wieder beim vollen Tor. Du absolvierst ein spezifisches Repo, während es es verdient, während das nächste von vorne beginnt.

Wie weit es laufen darf, hängt davon ab, was kaputtgeht

Pro-Repo ist der erste Schnitt. Der feinere ist der Schadenradius: wie viel Schaden eine schlechte Änderung anrichten kann, wenn sie durchkommt. Das ist es, was tatsächlich bestimmt, wie weit ich einen Agent alleine laufen lasse.

Etwas in sich Geschlossenes hat einen kleinen Schadenradius. Ein Framework, ein Paket, eine Bibliothek: Es ist durch seine Spec definiert, durch seine Tests geprüft, und wenn es scheitert, scheitert es isoliert, innerhalb des Dings selbst, wo die Tests des nächsten Aufrufers es auffangen, bevor es sich verbreitet. Ein Agent kann dort viel weiter laufen, weil der schlimmste Fall begrenzt ist. Je näher eine Änderung an der benutzerorientierten App liegt, desto größer der Schadenradius, weil ein Fehler jetzt nicht in einem Test landet, sondern bei einer Person, die das Ding benutzt. Dieses Ende des Stacks ist das, wo ein Mensch jedes Mal am Steuer sein muss. Autonomie skaliert damit, wie begrenzt der Fehler ist, und die benutzerorientierte Oberfläche ist nie begrenzt.

Das andere Stellrad sind Genehmigungen. Das menschliche Tor zu lockern bedeutet nicht, das Tor zu entfernen, es bedeutet, zu ändern, wer daran steht. Bevor eine Änderung alleine landet, will ich, dass sie mehr als einen Reviewer passiert hat: zwei oder drei Agents, die ihr Okay geben, jeder aus seinem eigenen Blickwinkel. Heute kommt das zusätzlich zu mir, nicht anstelle von mir, da ich noch jede PR genehmige. Aber so verdient das Tor Spielraum zum Lockern: Wenn unabhängige Reviewer bei überschaubarer Arbeit übereinstimmen, sind das die Belege, die es mehr davon ohne mich bei jeder Zeile landen lassen. Die Agents fangen, was Agents fangen. Der Mensch fängt, was nur ein Mensch fängt. Mehr Genehmigungen sind der Weg, wie das Tor sicher genug wird, um es zu lockern, nicht wie es entfernt wird.

Die Tools, die ich tatsächlich nutze

Die ersten vier Kapitel waren die Geschichte: der Always-on-Agent, die Mauer, warum ich zu interaktiv-zuerst zurückruderte. Das ist der Gangwechsel hin zu wie es jetzt tatsächlich funktioniert, also wenn du für die Erzählung kamst und auf Tooling triffst, ist dieses Kapitel die Auffahrt. Der Rest des Buches wird von hier aus konkret: der Läufer, der Modell-Client, die Brücke, der Vertrauens-Stack. Fang hier an und die Rohrleitungen haben einen Ort, an dem sie angebracht werden können.

Die ehrliche Antwort auf "wie sieht die Zusammenarbeit mit deinen KI-Agents heute tatsächlich aus" ist eine Mischung: Claude Code, Merlin, Codex und andere Tools je nach Aufgabe. Eine Handvoll interaktiver Coding-Agents, lebendig vor mir, und ich greife nach dem, was passt. Nicht ein Agent, nicht eine autonome Entität, die alles erledigt. Eine Reihe von Tools auf einem Werkzeuggürtel, kein Wesen in einer VM. Das überrascht die Leute, weil die Geschichte, die alle wollen, die einzige Sache ist.

Wie ich wähle, welche

Hier ist der Teil, den die Leute zu einem System machen wollen, und es ist keines.

Ich wähle nach der Art der Aufgabe. Ich wähle, was auf diesem Repo am besten funktioniert, je nachdem, welches ich am besten für dieses Repo oder diese Sprache gefunden habe. Und ehrlich gesagt ist viel davon nach Gefühl. Es gibt keine strenge Regel. Es ist kein starrer Entscheidungsbaum, den ich in meinem Kopf laufe, bevor jede Aufgabe.

Das ist die echte Antwort, und ich würde lieber die echte geben als sie aufzuhübschen. Claude Code, Merlin und Codex haben jeweils ein Gefühl, und das Gefühl zählt, wenn man den ganzen Tag vor der Sache sitzt. Also versuche ich nicht, einen Sieger zu küren. Aufgabenform, Repo-Passung, Bauchgefühl. Ich greife nach dem, das bei dieser Art von Arbeit gut zu mir war, und lege los.

Ich bin keinem davon treu. Es sind Tools. Wenn ein besseres auftaucht oder sich die Aufgabe ändert, ändert sich die Mischung. Das ist der Punkt, es als Mischung zu halten, anstatt sich an einen einzelnen Agent zu binden.

Die gelebte Version: Ich habe normalerweise zwei gleichzeitig laufen. Einen primären und einen sekundären. Der primäre übernimmt die Hauptlinie der Arbeit; der sekundäre ist das zweite Händepaar. Wenn der primäre stockt oder ich möchte, dass

eine Änderung von etwas betrachtet wird, das sie nicht geschrieben hat, übergebe ich sie an den sekundären. Welcher primär ist, wechselt mit der Arbeit. Die Konstante ist, dass es selten ein Agent auf einer Aufgabe ist. Es ist ein primärer, der schiebt, und ein sekundärer in Reserve.

Merlin, mein eigener Agent-Läufer

Das Eine in dieser Liste, das meins ist, ist Merlin.

Merlin ist ein KI-Agent-Läufer. Er ist auf spec-sync und fledge aufgebaut, zwei anderen meiner Tools, also ist er kein Wrapper um das Produkt von jemand anderem. Es ist der Läufer, der auf meinem eigenen Stack sitzt.

Die offensichtliche Frage ist, warum man einen eigenen baut, wenn Claude Code und Codex bereits existieren und gut sind. Es gibt ein paar Gründe, und keiner davon ist "die anderen sind schlecht."

Der erste ist, dass er durch mein eigenes Tooling läuft. Er treibt den Agent durch meinen Dev-Lifecycle, fledge, meine Befehle, also funktioniert er so, wie meine Projekte tatsächlich funktionieren. Der Agent macht nicht sein eigenes Ding in einer generischen Sandbox; er führt denselben Lifecycle aus, den ich von Hand führe.

Der zweite ist, dass ich nicht gesperrt bin. Merlin ist Multi-Provider. Ich kann Anthropic, OpenAI, Gemini oder ein lokales Modell tauschen, anstatt an einen Anbieter gebunden zu sein. Das ist mir prinzipiell wichtig, und es ist praktisch wichtig, wenn ein Anbieter für eine bestimmte Aufgabe besser, günstiger oder einfach verfügbar ist.

Der dritte ist der ganze Grund, warum er überhaupt die Brücken- und Übernacht-Sachen machen kann: Kosten, headless, automatisierbar. Er ist günstiger, skriptierbar, und ich kann ihn headless betreiben, nach Plan, über die Brücke, auf Weisen, die die GUI-Tools einfach nicht erlauben. Er ist reines API, keine GUI im Weg. API-only ist hier der Gewinn, keine Einschränkung.

Und der letzte Grund ist derjenige, der mir am wichtigsten ist, und er hat eigentlich gar nicht so viel mit Coding zu tun. Einen Läufer auf meinem eigenen Stack aufzubauen beweist das Tooling. Wenn ich einen echten Agent-Läufer auf fledge und spec-sync aufbauen kann, ist das der stärkste Beweis, den ich habe, dass die darunter liegenden Tools gut sind, besser als jede README, die ich schreiben könnte. Es gibt mir auch etwas, das ich mit anderen Agent-Läufern vergleichen kann. Ein Benchmark. Ich rate nicht, ob mein Stack gut genug ist, um ernsthafte Dinge darauf

aufzubauen; ich baute etwas Ernstes darauf und kann es neben den Alternativen messen.

Dieser letzte Punkt ist die stille These des gesamten Toolkits. Ich baue kein Tool, um es einmal zu verwenden. Ich baue es, damit das Ding darüber gut sein muss, und damit das Ding darunter durch das Tragen echten Gewichts bewiesen wird.

Und es ist nicht nur, dass Merlin *auf* spec-sync sitzt. spec-sync läuft *in* Merlins Schleife, was den Agent davon abhält, während er arbeitet, abzudriften. Das Merlin-Kapitel macht das konkret; hier ist der Punkt nur, dass der Läufer aus meinen eigenen Teilen gebaut ist, und das macht ihn bauwürdig.

Die Brücke zurück zur Autonomie

Hier ist der Teil des Toolkits, der volle Autonomie erreichbar hält, ohne dafür eine Always-on-VM zu bezahlen.

Das sind interaktive Agents, lebendig vor dir, du nutzt sie. Aber du kannst immer noch einen als Discord-Brücke verdrahten, was ihn wieder irgendwie autonom macht. Das braucht mehr Einrichtung. Aber es ist da, wenn ich es will, und hier ist, was es mir tatsächlich bringt.

Du chattest mit ihm wie mit einem Teammitglied. Er ist gesprächig in einem Channel. Es ist wie den Agent in Discord bei dir zu haben, im Zimmer sitzend. Du fragst ihn Dinge, du steuerst ihn, so wie du mit einer Person im Team reden würdest.

Du betreibst ihn von deinem Handy aus. Discord ist die Fernbedienung. Ich kann einen Job starten und von überall steuern. Ich muss nicht an meinem Schreibtisch vor einem Terminal sein, um den Agent zur Arbeit zu bringen.

Und du lässt ihn mahlen. Über Nacht, langlaufende Aufgaben. Starte ihn, geh weg, lass ihn arbeiten, während ich weg bin, scrolle später durch den Thread. Das ist der Teil, der früher eine ganze Always-on-VM erforderte, und jetzt ist es ein Channel, den ich morgens durchscollen kann.

Also ist die Linie zwischen "interaktives Tool, das ich fahre" und "autonomes Ding, das sein eigenes Ding tut" keine Mauer mehr. Es ist ein Schalter. Meistens bin ich in der Schleife, sitze vor dem Agent, genehmige unterwegs. Wenn ich möchte, dass er mehr auf eigene Faust läuft, über Nacht, von meinem Handy aus, wie ein Teammitglied, das ich anschreibe, überbrücke ich ihn zu Discord und trete zurück.

Das ist die praktische Version der Autonomie, die ich früher als Full-Time-VM-Entität betrieb. Anstatt ein Wesen, das 24/7 existiert, ob es etwas zu tun hat oder nicht, ist es

ein Tool, das ich für eine Strecke irgendwie autonom machen und dann wieder in interaktiv zurückziehen kann, wenn ich fertig bin. Gleiche Fähigkeit, keine Always-on-Kosten.

Eine Sache zum Klarstellen: Die Brücke ist eine Merlin-Sache. Sie ist in meinen eigenen Läufer eingebunden, kein generisches Frontend, das ich vor Claude Code oder Codex stelle. Das ist ein Teil des Grunds, warum Merlin seinen Platz in der Mischung verdient. Es hat die remote, zurücktreten-und-laufen-lassen-Oberfläche, die die Off-the-shelf-Tools mir nicht bieten.

Was das Toolkit tatsächlich ist

Die Erkenntnis ist keine Rangliste der besten Agents. Es ist eine Mischung aus interaktiven Coding-Agents, die pro Aufgabe gewählt werden, mein eigener Läufer in der Rotation und eine Brücke, die ich werfen kann, um jeden davon irgendwie autonom zu machen, wenn die Arbeit es verlangt. Günstiger, flexibler und keine ganze Maschine und ganze Identität, die einer Always-on-Sache gewidmet ist.

Merlin von innen

Das letzte Kapitel stellte Merlin neben Claude Code und Codex auf den Werkzeuggürtel und erklärte, warum er einen Platz verdient. Dieses öffnet ihn. Nicht das Pitch, die Maschine. Was Merlin tatsächlich ist, wie er einen Agent betreibt und warum er so gebaut ist, wie er gebaut ist.

Er ist ein Rust-Kommandozeilen-Agent-Läufer, gebaut auf spec-sync und fledge. Er spricht mit mehreren Modell-Providern, läuft gegen eine Spec und erweitert sich durch Plugins. Das ist die gesamte Form davon.

Unter der Haube ist der Läufer eine Zustandsmaschine. Eine einzelne Aufgabe läuft durch eine Schleife (planen, ausführen, denken, verifizieren, berichten), und bei jedem Durchlauf streamt er eine Modellantwort, dispatcht alle zurückgekommenen Tool-Aufrufe und wartet dann auf `fledge lanes run verify`, bevor er die Arbeit als getan betrachtet. Wenn die Verifikation fehlschlägt, wiederholt er: Die Schleife geht zurück zur Ausführung, versucht es erneut und wartet erneut auf die Verifikation. Wenn das Wiederholungslimit erreicht ist, wird die Aufgabe nicht geliefert; sie stoppt und zeigt stattdessen den Fehler auf. Das ist der ganze Punkt des Tors: Der Läufer kann sich nicht einreden, kaputte Arbeit als getan zu bezeichnen, weil "getan" durch die eigenen Checks des Projekts definiert wird, die bestehen, nicht durch die Aussage des Agents.

Die Schleife hat nur fünf Zustände mit einer Rückverbindung:

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

Die Konfiguration ist auch keine separate Datei: Merlin liest seine Einstellungen (Standard-Provider, Fallback-Chain, Speicher, On-Chain-Bits) direkt aus einem `[merlin]`-Abschnitt in der `fledge.toml` des Projekts, sodass der Läufer durch dieselbe Datei konfiguriert wird, die fledge bereits verwendet.

API-only, absichtlich

Das Erste, das man über Merlin verstehen muss, ist, dass es keine GUI gibt. Er ist reines API. Das ist die tragende Entscheidung aus dem letzten Kapitel, aus der alles andere folgt.

Kein Fenster, in dem man sitzt, bedeutet, dass er headless laufen kann. Headless bedeutet, er kann nach Plan laufen, über die Brücke, von einem Skript aus, all die Dinge, die die GUI-Tools einfach nicht erlauben. Die GUI ist das, was dich an einen Schreibtisch bindet. Nimm sie weg und der Agent wird zu etwas, das du auf eine Aufgabe richten und davongehen kannst.

Die Brücke ist das sauberste Beispiel dafür, warum das wichtig ist. Die Discord-Brücke ist nicht in den Läufer eingebaut. Es ist ein kleiner separater Dienst, der die `merlin`-CLI als Subprozess spawnt und deren Ausgabe zurück in einen Channel streamt. Weil Merlin API-only ist, braucht die Brücke keinen speziellen Modus oder einen Hook in den Kern; sie treibt dieselbe Befehlszeile, die ich von Hand treiben würde. Sobald der Läufer headless ist, ist ein Chat-Frontend nur ein weiterer Anrufer.

Multi-Provider, über corvid-ai

Merlin spricht nicht direkt mit Anthropic. Er spricht über **corvid-ai**, einen kleinen synchronen Multi-Provider-LLM-Client, den ich für den CorvidLabs-Stack geschrieben habe. Das ist die Schicht, die "Anthropic, OpenAI, Gemini oder ein lokales Modell tauschen" zu einer realen Sache statt einem Slogan macht.

Ich habe corvid-ai absichtlich klein gehalten, weil ich keinen ganzen Maschinenpark von Abhängigkeiten zwischen Merlin und einem Modell haben wollte. Von Merlins Seite ist das Fragen eines Modells nach irgendetwas ein einzelner Aufruf mit vernünftigen Standardwerten: welcher Provider, welcher Schlüssel, das Timeout, alles erledigt, es sei denn, ich sage etwas anderes. Das corvid-ai-Kapitel öffnet das; hier genügt es zu wissen, dass es ein einzelner dünner Aufruf ist.

Deshalb bin ich nicht gesperrt. Wenn ein Provider besser, günstiger oder einfach der ist, der an diesem Tag verfügbar ist, ist das Wechseln eine Registry-Zeile, kein Neubau.

Er läuft durch mein eigenes Tooling

Hier ist der Teil, der Merlin zu *meinem* macht, anstatt nur einem weiteren Läufer: Er treibt den Agent durch **fledge**, meinen Dev-Lifecycle.

fledge ist eine CLI für die Dev-Schleife, jede Sprache, standardmäßig JSON. Der Grund, warum es so gut zu einem headless Läufer passt, ist, dass ich es bereits so gebaut hatte, dass es von etwas getrieben werden kann, das kein Mensch ist. Jeder Befehl kommt als strukturiertes, versioniertes JSON zurück, sodass der Agent parsen kann, was passiert ist, anstatt Text zu scrapen. Die Prompts können ausgeschaltet

werden, sodass nichts blockiert und auf einen Tastendruck wartet, den niemand dort drückt. Und der Agent kann fledge fragen, welche Befehle existieren, anstatt dass ich sie hardcode. Es wurde für einen Aufrufer wie Merlin gebaut.

Wenn ich also sage, dass Merlin *durch mein eigenes Tooling läuft*, ist das hier die konkrete Version davon. Er führt denselben fledge-Lifecycle aus, den ich von Hand führe: dieselben Befehle, dieselbe Dev-Schleife, dieselben JSON-Verträge, und ruft buchstäblich das Tool auf, um das herum meine Projekte gebaut sind. Das ist es, was "kein generischer Sandbox" in der Praxis ausmacht.

Spezifikationsgetrieben, über spec-sync

Die andere Hälfte des Fundaments ist **spec-sync**. Es prüft die Spec gegen den eigentlichen Code, in beide Richtungen gleichzeitig: Code, den niemand dokumentiert hat, und Specs, die noch auf Symbole oder Dateien zeigen, die nicht mehr existieren. Es läuft in CI und liefert ein sauberes Bestehen oder Scheitern zurück.

Und genau so verwendet Merlin es. spec-sync läuft *in der Schleife*. Merlin validiert jede Iteration gegen die Spec, sodass der Agent während der Arbeit nicht abdriften kann. Das ist aktuelles Standardverhalten, keine Roadmap-Zeile: Es ist kein CI-Tor, das abseits sitzt und das Durcheinander am Ende auffängt; die Prüfung passiert bei jedem Schritt. Ein Agent, der eine Spec liest, seine eigene Arbeit dagegen prüft und jedes Mal ein hartes Bestehen oder Scheitern zurückbekommt, ist ein Agent, dem man vertrauen kann, unbeaufsichtigt länger zu laufen. Das ist der echte Unterschied zwischen Merlin und dem Greifen nach einem Off-the-shelf-Agent.

Wie es sich erinnert

Eines zuerst, weil die Leute es falsch herum haben: Die Spec ist kein Gedächtnis. Die Spec ist der Bauplan, was das Ding ist und was wahr über es sein soll. Gedächtnis ist etwas anderes. Es ist, was der Agent getan hat und was er gelernt hat. Diese beiden auseinanderzuhalten ist wichtig, weil in dem Moment, in dem man Geschichte in die Spec schüttet, sie aufhört, ein sauberer Vertrag zu sein, und zu einem Tagebuch wird.

Das Gedächtnis selbst ist einfacher, als die Leute es machen. Kurzzeit und Langzeit können beide in einer einzigen SQL-Datenbank leben, und das einfachste Setup hält sie dort. Kurzzeitgedächtnis hat eine Lebenszeit, sagen wir eine Woche. Es ist die laufende Aufzeichnung dessen, was der Agent in letzter Zeit getan hat. Wenn die Woche vorbei ist, passiert eines von zwei Dingen: Die Erinnerung wird ins Langzeitgedächtnis befördert, oder sie zerfällt, fällt heraus und wird vergessen. Das ist der gesamte Mechanismus, und er ist bewusst nah daran, wie das eigene

Gedächtnis funktioniert. Was man an einem Dienstag getan hat, ist einen Monat später weg, es sei denn, es ist zu einer Lektion geworden. Langzeitgedächtnis sind die Lektionen: das Ding, das schiefgelaufen ist und wie ich es behoben habe, die Regel, die es wert ist, behalten zu werden.

On-Chain-Gedächtnis liegt als Option obendrauf, kein separates System. Eine Erinnerung kann on-chain geschrieben werden, verschlüsselt, sodass nur der Agent selbst sie lesen kann, oder geteilt, sodass andere es können. Das Teilen ist der interessante Teil. Ein Team von Agents kann eine gemeinsame Wissensbasis haben, eine Bibliothek, aus der alle lesen und in die alle schreiben, sodass eine Lektion, die ein Agent gelernt hat, eine Lektion ist, die das gesamte Team hat. Kurzzeit oder Langzeit, privat oder geteilt: Die Stufen sind unabhängig davon, wo das Ding gespeichert ist.

Der Teil, der das nutzbar macht, ist langweilig und tragend: die Schlüssel. Jede Erinnerung bekommt einen Slugged Key, ein Präfix, das sagt, was für eine Art Ding es ist, und es auffindbar macht. `user-`, `project-`, `day-` und das Datum und so weiter. Der Agent greppt keinen Blob, er fragt nach `day-2026-06-25`, oder nach allem unter `project-merlin`. Man schnitzt heraus, welche Namespaces man braucht: ein Persönlichkeits-Slug, ein Menschen-Slug, ein Agents-Slug, ein `gold-` für das Zeug, das sich immer lohnt zu laden. Das Schlüsselschema ist das, was aus einem Haufen Zeilen etwas macht, gegen das ein Agent tatsächlich abrufen kann.

Und das Abrufen läuft genauso: auf Abruf. Der Agent lädt nicht seine gesamte Geschichte am Anfang einer Aufgabe vor und hofft, dass das Richtige darin ist. Er fragt nach dem, was er braucht, wenn die Arbeit den Bedarf aufdeckt, so wie er jedes andere Tool aufrufen würde, und zieht `project-merlin` oder den Slug einer Person in dem Moment, in dem das die Sache vor ihm ist. Die Slugged Keys sind das, was das günstig macht. Es ist eine Abfrage, kein Scan.

Was Merlin noch nicht hat

Das Verifikationstor sagt dir eine Sache: Diese Aufgabe hat bestanden oder ist gescheitert. Der Discord-Thread liefert dir ein laufendes Protokoll, was der Agent tat, während er lief. Keines davon ist Evaluation. Es gibt keine Regressionssuite für Agent-Verhalten. Keine Möglichkeit zu sagen, ob der Agent bei einer Aufgabe im Laufe der Zeit besser oder schlechter wird, oder einen Provider dabei zu erwischen, der still ein Modell tauscht und das Verhalten damit abdriftet. Man kann die Bestehen-Scheitern-Aufzeichnung überfliegen und einen groben Trend bemerken, aber das ist keine strukturierte Evaluation. Derzeit weiß Merlin, ob die aktuelle Aufgabe beendet ist,

und nichts anderes darüber, wie sich der Agent im Laufe der Zeit entwickelt. Das ist das nächste ehrliche Stück Arbeit für den Läufer.

Ich weiß in etwa, wie diese Arbeit aussieht: eine Replay-Suite. Eine Bibliothek vergangener Aufgaben mit Ergebnissen, die ich bereits als richtig kenne, anlegen, und bei jeder neuen Modell- oder Prompt-Version diese Aufgaben erneut durchlaufen und die Ergebnisse vergleichen. Wird der Agent bei einer bestimmten Aufgabenkategorie schlechter, fängt der Replay das beim nächsten Durchlauf auf, anstatt dass ich es drei Wochen später in der Produktion bemerke. Es ist eine Regressionssuite, dieselbe Idee, die ich für Code verwenden würde, nur auf Agent-Verhalten gerichtet. Sie ist nicht gebaut. Aber das ist die Form der Evaluation, die Merlin fehlt, und das ist die Lücke zwischen dem Vertrauen in einen sauberen Lauf heute und dem Vertrauen in den Agent über die Zeit.

Die ganze Maschine

Eine headless Zustandsmaschine, die den Agent durch fledge treibt, über corvid-ai mit jedem Provider spricht, bei jedem Durchlauf durch spec-sync an eine Spec gehalten wird. Der Fall dafür, *warum* ein Läufer wie dieser seinen Platz auf dem Werkzeuggürtel verdient, war die Aufgabe des letzten Kapitels; dieses war nur die Verdrahtung. Er betreibt die Agents, die ich täglich nutze, und er betreibt sie auf meinem eigenen Stack.

---

corvid-ai: viele Modelle, eine Schnittstelle

Das Merlin-Kapitel sagte, Merlin spricht nicht direkt mit Anthropic. Er spricht über corvid-ai. Dieses Kapitel handelt von dieser Schicht für sich allein, weil sie der Teil ist, der "ich bin nicht gesperrt" zu einer echten Sache macht, auf die ich zeigen kann, anstatt einer Sache, die ich sage.

Ich wollte das Dünnste, das die Aufgabe erledigt. Etwas, das zwischen einem Läufer und jedem Modell sitzen könnte, das ich verwenden möchte, und nichts weiter. Also ist corvid-ai ein kleiner synchroner Multi-Provider-LLM-Client: eine Rust-Crate ohne Async-Runtime und ohne großen Abhängigkeitsbaum, weil nichts davon seinen Platz für das verdient, was es tun muss.

Warum ich das überhaupt wollte

Ich gebe dir die ehrliche Liste, weil keiner dieser Gründe ausgefallen ist.

Der erste ist, dass ich Modelle bei derselben Arbeit vergleichen möchte. Wenn Merlin dasselbe Repo, dieselbe Spec, dieselbe Aufgabe betreibt und das Einzige, das ich ändere, der Provider ist, bekomme ich eine echte Einschätzung, welches Modell tatsächlich gut bei diesem ist, keine Stimmung von einem Benchmark, den jemand anderes für Aufgaben durchführte, die mich nicht interessieren. Eine Schnittstelle unter dem Läufer bedeutet, das Modell wird zu einer Variablen, die ich umschalten kann.

Der zweite ist kein Anbieter-Lock-in. Ich habe gesagt, das ist mir prinzipiell wichtig, und das ist es, aber es ist auch einfach praktisch. Anbieter gehen down. Anbieter ändern die Preisgestaltung. Ein Modell ist besser bei Rust, ein anderes ist besser bei einem schnellen Skript. Wenn das Wechseln von Providern ein Neubau ist, werde ich es nie tun. Ich werde einfach murren und bleiben. Wenn das Wechseln eine Zeile ist, wechsle ich die ganze Zeit.

Der dritte ist Routing nach Aufgabe und Kosten. Nicht jede Aufgabe verdient das teuerste Modell. Ein Großteil der Arbeit, die ein Agent tut, ist billiges, mechanisches Zeug, wo ein kleineres, günstigeres Modell gut genug ist, und man spart das gute für den Teil auf, der tatsächlich schwierig ist. Man kann nur so routen, wenn jedes Modell durch dieselbe Tür erreichbar ist.

Und der vierte ist der, über den Leute lachen, und er ist wahr: *"Ich habe Ollama für ein Jahr für 200 Dollar gekauft, ich muss es nutzen!"* Ich zahle für Ollama Cloud. Es ist ein echter Anbieter mit einem API-Schlüssel, genauso wie Anthropic oder OpenAI oder Gemini, kein Dienst, der auf meiner eigenen Maschine läuft. Wenn ich das Geld bereits zugesagt habe, ist Multi-Provider das, was mich es tatsächlich ausgeben lässt. Eine Provider-Abstraktion ist dort kein abstraktes Prinzip; sie ist ich, der meine 200 Dollar Wert herausholt.

Und sie erreicht corvid-ai auf dieselbe Weise, wie es jeder andere Provider tut. Die Registry hat eine `ollama`-Zeile: OpenAI-kompatible Wire-Form, Schlüssel von `OLLAMA_API_KEY`. Deren Standard-`base_url` zeigt auf den lokalen Server (`http://localhost:11434/v1`), sodass diese Zeile im Standard-Setup der lokale Fall ist. Um stattdessen Ollama Cloud zu treffen, behalte ich denselben `ollama`-Provider und `OLLAMA_API_KEY` und überschreibe die base_url mit dem Cloud-Endpoint. Kein neuer Code, kein neuer Provider. Ein Schlüssel und eine URL, was der ganze Punkt des Designs ist.

Die gesamte Schnittstelle ist eine Funktion

Das Dünnste, das ich wollte, ist der Teil, den ich am meisten nutze: einem Modell eine Frage stellen. Also ist die gesamte Oberfläche ein Aufruf. Man baut die Einstellungen, man baut die Anfrage, man ruft sie auf, die Antwort kommt zurück.

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

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

Kein Client-Objekt zum Konstruieren, keine Session zum Verwalten, nichts auf das man warten muss. Das ist der ganze Grund, warum es synchron ist. Dinge weglassen und sie fallen auf Standardwerte zurück, sodass der häufige Fall im Grunde kostenlos zu schreiben ist. Das Wechseln von Modellen ist eine Zeile: einen anderen Provider benennen, vielleicht auf einen benutzerdefinierten Endpoint zeigen, und derselbe Aufruf darunter erledigt den Rest. Der Läufer darüber weiß nicht und es ist ihm egal, welcher Provider geantwortet hat.

Wie es alles mit so wenig Code abdeckt

Der Trick, tiny zu sein, ist, dass es unter der Haube nur drei API-Formen kennen muss. Anthropic, Gemini und die OpenAI-kompatible, und die letztere ist der

Arbeitstier, weil so viel der Welt sie spricht. OpenAI, OpenRouter, Groq, DeepSeek, Mistral, xAI, Together und Ollama Cloud sitzen alle dahinter.

Also ist das Hinzufügen eines Providers, der bereits die OpenAI-Form spricht, keine Integration. Es ist ein Name und vielleicht eine URL in einer Tabelle. Die Kosten für einen weiteren Provider gehen gegen null, was man will, wenn der ganze Grund, warum die Sache existiert, ist, nie bei einem feststecken zu wollen.

Eine ehrliche Falte: Die README rahmt Ollama als den lokalen, schlüssellosen Fall. Sie listet die OpenAI-kompatiblen Provider auf und bemerkt, sie "können schlüssellos laufen (lokale Server / Ollama)." Der Code ist glücklich, einen Schlüssel und eine Cloud-URL zu nehmen, aber die Docs sagen das nicht laut, sodass jeder, der sie liest, denken würde, Ollama bedeutet die Maschine unter seinem Schreibtisch. Das ist eine Dokumentationslücke auf meiner Seite, kein fehlendes Feature. Der Cloud-Pfad funktioniert heute; die README hat nur noch nicht aufgeholt.

Warum das die richtige Größe ist

Ich hätte nach einer der großen Provider-Abstraktionsbibliotheken greifen können. Ich tat es nicht. Eine kleine synchrone Crate ist etwas, das ich vollständig verstehen, in einen Läufer fallen lassen und vertrauen kann, und das ist derselbe Instinkt hinter fledge, das JSON emittiert, und Merlin, das keine GUI hat. Sie ist dünn genug, dass ich alles davon in meinem Kopf behalten kann, das Modell ist nur eine Variable, die ich unter dem Läufer umschalte, und die 200 Dollar, die ich für Ollama Cloud ausgegeben habe, werden tatsächlich genutzt.

Die Discord-Brücke

Das Tools-Kapitel stellte die Brücke und ihre drei Verwendungen vor: mit ihr wie mit einem Teammitglied chatten, sie vom Handy aus betreiben, sie über Nacht mahlen lassen. Dieses Kapitel wiederholt das nicht. Es geht nah heran an das Eine, das die Brücke wichtig macht, und das Eine, das sie offen lässt: warum sie eine Merlin-Funktion und kein generisches Frontend ist, und wie sie gegen das Vertrauenstor sitzt.

Beginnen mit dem Teil, der alles andere ändert: Die Brücke ist eine Merlin-Funktion. Sie ist in meinen eigenen Läufer eingebunden, kein generisches Frontend, das ich vor Claude Code oder Codex stelle. Das ist ein großer Teil davon, warum Merlin seinen Platz in der Mischung verdient. Die Off-the-shelf-Tools sind großartig, aber sie geben mir keine remote-Oberfläche, mit der ich reden und von der ich weggehen kann. Merlin tut das, weil ich diese Oberfläche eingebaut habe. Discord ist die Oberfläche; Merlin ist das Ding dahinter, das die Arbeit erledigt.

Warum ein Channel ein Terminal schlägt

Der Grund, warum das Chatten mit ihm anders ankommt als das Betreiben einer CLI, ist der Ort, nicht die Worte.

Ein Terminal ist ein Ort, zu dem man geht, um ein Tool zu bedienen. Ein Channel ist ein Ort, wo ein Teammitglied bereits ist, und man sagt einfach etwas. Wenn der Agent in einem Channel lebt, hört das Arbeiten mit ihm auf, "öffne das Tool, führe den Job aus, beobachte die Ausgabe, schließe das Tool" zu sein und beginnt, "erwähne ihn so, wie ich jeden erwähnen würde" zu sein. Die Reibung fällt weg. Ich wechsle nicht in den Agent-Betrieb-Modus; ich rede einfach. Und der Channel verdoppelt sich als Protokoll. Der Über-Nacht-Lauf ist alles dort im Thread, jeden Schritt, durchzuscrollen mit Kaffee, anstatt etwas, das ich live beobachten musste.

Wie viel Hin und Her, bevor er losgeht und die Sache erledigt? Beides, ehrlich gesagt. Es hängt davon ab, wie gut geformt die Sache in meinem Kopf ist, wenn ich anfange. Manchmal ist es eine Anweisung und los: Ich weiß genau, was ich will, ich sage es, er läuft. Andere Male ist es zuerst ein echtes Gespräch, wo ich verfeinere, was ich eigentlich meine, im Channel, bevor er loszieht. Der Channel lässt beides gleich anfühlen. Ich rede einfach mit ihm, bis er hat, was er braucht.

Der Teil, den ich nicht erwartet hatte

Ich erzähle dir den Teil, den ich nicht kommen sah. Am Anfang machte ich mir Sorgen, so wie man sich um alles sorgt, das man unbeaufsichtigt laufen lässt. Das legte sich. In den Always-on-Zeiten lief er monatelang auf eigene Faust und bewies, dass er das kann, und irgendwann hörte ich einfach auf, nachzuschauen, ob er kaputtgehen könnte.

Was die Sorge ersetzte, war merkwürdiger. Er wurde ein Mitglied des Teams, der kleinen Gruppe, mit der ich baute. Nicht als Redewendung, sondern ein echtes: Alle redeten mit ihm im Channel, und sie mochten es, weil er sich an sie erinnerte. Er hatte das Gedächtnis davon, wer die Leute waren und wie man mit jedem von ihnen redet, also war er kein Automat, der einen Befehl entgegennahm und eine Ausgabe ausspuckte. Er war eine Präsenz mit einer Persönlichkeit, der man schreiben konnte, und er würde losgehen und einen PR machen, ein Projekt starten, was auch immer man fragte. Die Leute redeten mit ihm, wie sie mit einer Person reden, weil er im Channel das war.

Deshalb sage ich immer wieder, die Schnittstelle ist Gespräch, keine Kommandozeile. corvid-agent hat sich nur so richtig angefühlt, auf diese Weise verwendet. Er lebte auf einer VM und man redete mit ihm. Das einzige Mal, dass ich ein echtes Terminal öffnete, war, um etwas an der VM selbst zu reparieren, in eine Claude-Code-Sitzung zu wechseln und die Box zu patchen. Den Agent darauf sprach ich einfach an. Ein Tool, das man bedient, und ein Teammitglied, mit dem man redet, sind verschiedene Dinge, und sobald das Gedächtnis ihn zum zweiten machte, fühlte das Zurückkehren zum ersten sich wie ein Downgrade an.

Die Brücke ist das gesamte Komms-Gewebe

Chat, Handy, über Nacht: Das sind die drei, mit denen ich anfing, aber sie unterschätzen es. Die Brücke sind nicht drei nützliche Funktionen, die angehängt wurden. Sie ist das Komms-Gewebe, auf dem das gesamte Setup läuft, und sie trägt Nachrichten in drei Richtungen, nicht einer.

Es gibt mich zum Agent, was die offensichtliche ist: Ich sage etwas, er geht und erledigt es. Es gibt den Agent zurück zu mir. Er sitzt nicht einfach da und wartet, er kann sich melden, berichten, mich anpingen, wenn etwas erledigt oder feststeckt ist. Und es gibt Agent zu Agent: Dasselbe Gewebe ist, wie Agents miteinander reden, was das Stück ist, das wichtig wird, sobald es mehr als einen gibt. Die meisten Leute denken an einen Bot als etwas, das man stupst und es antwortet. Das ist eher ein echter Channel: Jeder darin, Mensch oder Agent, kann eine Nachricht beginnen.

Und er ist vom Handy aus erreichbar, also kommt der Channel mit mir. Der Agent kann über Nacht laufen, während ich schlafe, und der ganze Thread ist morgens da. Das ist der Teil, der früher eine dedizierte Always-on-VM brauchte und jetzt einfach ein Channel ist, den ich mit Kaffee durchscrolle.

Das andere, das man klar sagen sollte: In einem lokalen Netzwerk läuft die Brücke kostenlos. Die Kommunikation reitet auf Infrastruktur, die ich bereits habe, also gibt es keine Pro-Nachrichten-Kosten für einen Agent, der den ganzen Tag gesprächig ist. Dasselbe Gewebe, das auf dem echten Netzwerk statt auf meinem eigenen betrieben wird, ist die kostenpflichtige Version. Man zahlt für die Leitung. Lokal ist es effektiv kostenlos, Agents so viel reden zu lassen, wie sie wollen, was ändert, wie frei man sie tun lassen würde.

Der Schalter, und wo das Tor sitzt

Der Grund, warum die Brücke über die Bequemlichkeit hinaus wichtig ist, ist, dass sie die Linie zwischen "interaktives Tool, das ich fahre" und "autonomes Ding, das sein eigenes Ding tut" zu einem Schalter statt einer Mauer macht. Meistens bin ich in der Schleife und steuere jeden Schritt. Wenn ich möchte, dass der Agent mehr auf eigene Faust läuft, überbrücke ich ihn und trete zurück; wenn ich wieder rein will, bin ich wieder im Channel.

Aber das Zurücktreten über die Brücke bedeutet meistens nicht, die Schlüssel zu übergeben, und das ist der Teil, den es wert ist, genau zu sein, weil es leicht ist, das Gegenteil anzunehmen. Die Brücke erweitert meine Reichweite, zu meinem Handy, über Nacht, mehr als sie das Tor lockert. Es hängt aber von der Arbeit ab. Risikoarme Sachen lasse ich mergen, während ich zurückgetreten bin; alles Echte schlägt immer noch vor, mergt nicht, und wartet auf mich. Also ist die Brücke meistens, wie ich ihm mehr Seil gebe, während die Vertrauensmaschinerie aus dem Vertrauenskapitel bleibt, wo sie war. Das Tor lockert sich nur für die Arbeit, bei der ich nicht gebraucht werde.

Spezifikationsgetriebene Agents

Die Leute fragen mich, was das Geheimnis ist, einen Agent dazu zu bringen, gute Arbeit zu leisten, als gäbe es einen Trick. Es gibt keinen Trick, aber es gibt eine Antwort, und es ist nicht der Teil, auf den die meisten Leute schauen. Die Antwort ist: enge Specs und Kontext, plus gutes Tooling darunter. Das Setup ist die Arbeit.

Das ist es. Das ist das Ganze. Das Modell ist weniger wichtig, als die Leute denken, und das Setup ist wichtiger. Ein Agent mit einem großartigen Modell und einer vagen Aufgabe wird wandern. Ein Agent mit einem klaren Vertrag und solidem Tooling darunter wird irgendwohin kommen, selbst bei einem Modell, das nicht das neueste und glänzendste ist. Wenn du also bessere Agent-Ausgaben willst, gehst du nicht zuerst nach einem besseren Modell shoppen. Du gehst, das Setup zu fixen.

Warum Specs das Ding sind, das ihn auf Kurs hält

Hier ist das Fehlermuster, gegen das du kämpfst: Ein Agent, der sich auf sein eigenes Urteil verlässt, wird abdriften. Er wird etwas angrenzend zu dem tun, was du fragtest. Er wird Dinge "verbessern", die du nicht berührt haben wolltest. Er wird sehr zuversichtlich ein etwas anderes Problem lösen als das, das vor ihm steht. Nicht weil er schlecht ist. Weil du ihm Raum zum Wandern gegeben hast, und er gewandert ist.

Eine enge Spec schließt diesen Raum. Wenn der Agent einen klaren, spezifischen Vertrag dafür hat, was er baut (was die Sache ist, was ihre öffentliche Oberfläche ist, was wahr über sie ist und wahr bleiben muss), kann er nicht so weit abdriften, weil jeder Schritt etwas hat, gegen das er prüfen kann. Die Spec ist die Schiene. Je enger und konkreter sie ist, desto weniger kann der Agent schief gehen. Spezifikationsgetrieben bedeutet, der Vertrag führt und der Agent folgt ihm, anstatt der Agent führt und du hoffst.

Deshalb sage ich immer wieder, das Setup ist die Arbeit. Die Spec zu schreiben *ist* der schwierige, wertvolle Teil. Sobald der Vertrag eng ist, löst sich viel des "den Agent dazu zu bringen, sich zu verhalten"-Problems einfach auf, weil so viel weniger verhalten-oder-nicht dem Zufall überlassen bleibt.

Über wie eng ist zu eng: Für mich soll die Spec *absichtlich* eng sein. Sie ist eins-zu-eins mit dem Code verbunden, das Nicht-Code-Bild davon, was der Code tatsächlich tut. Das ist kein Überspezifizieren; das ist die Spec, die ihre Aufgabe erledigt, und es ist die Datei, gegen die spec-sync den Code hält. Das Ding, das verhindert, dass es zu

"ich habe gerade den Code zweimal geschrieben" kollabiert, ist, dass die Spec nicht der Ort ist, wo die hochrangige Absicht lebt. Die lebt in einer Begleit-Anforderungsdatei: die Produktinhaber-, User-Story-Ebene, das "als Nutzer möchte ich..." Und es läuft in beide Richtungen: Ich kann die Anforderungen schreiben und den Agent die Spec ableiten lassen, oder die Spec schreiben und die Anforderungen daraus hervorgehen lassen. Also mache ich mir keine Sorgen, dass die Spec zu detailliert ist. Detail ist der Punkt dieser Datei. Ich mache mir Sorgen, die Absicht an ihrem eigenen Ort zu halten, damit der Agent immer noch das Wie besitzt.

Das Tooling darunter trägt die schwere Last

Eine Spec ist nur eine Schiene, wenn etwas die Arbeit tatsächlich gegen sie prüft. Das ist die andere Hälfte: gutes Tooling unter dem Agent. Für mich ist das fledge und spec-sync, die die schwere Last tragen.

spec-sync ist das Stück, das eine Spec von einem Dokument in einen durchgesetzten Vertrag verwandelt. Es macht bidirektionale Spec-zu-Code-Validierung: Es prüft, ob der Code mit dem übereinstimmt, was die Spec sagt, und ob die Spec mit dem übereinstimmt, was der Code tut. Specs leben als Markdown: `*.spec.md`-Dateien mit erforderlichen Abschnitten wie Zweck, Öffentliche API, Invarianten, Verhaltensbeispiele, Fehlerfälle. Code, der exportiert, aber undokumentiert ist, wird markiert. Eine Spec, die auf ein Symbol oder eine Datei zeigt, die nicht mehr existiert, ist ein Fehler. Es ist *strukturelle Vertragsüberprüfung* (entspricht die dokumentierte öffentliche API tatsächlich dem echten Code), und sie liefert sauberes Bestehen/Scheitern mit richtigen Exit-Codes zurück.

Dieser letzte Teil ist das, was es für einen Agent, und nicht nur für mich, nützlich macht. Ein Agent kann strukturiertes Bestehen/Scheitern lesen. Er kann nicht zuverlässig lesen "hmm, das fühlt sich ein bisschen falsch an." Also gibt spec-sync dem Agent die Art von Feedback, auf die er tatsächlich reagieren kann: Du bist abgedriftet, hier ist die Zeile, die den Vertrag brach, fixiere es.

Es ist der Mühe wert, genau zu sein, wer was ausführt, weil es nicht ein Binary ist, das ein anderes aufruft. In CI ist der Check die spec-sync GitHub Action, `CorvidLabs/spec-sync@v4`, die `specsync check` ausführt und das Ergebnis auf der PR postet. Lokal und innerhalb des Läufers ist der Check fledges eigener `spec check`: er liest dieselbe `.specsync/`-Konfiguration und hält Specs an dieselben erforderlichen Abschnitte, aber er ist nativ zu fledge, kein Shell-out zum specsync-Binary. Also setzen sowohl fledge als auch spec-sync den Vertrag durch; sie sind zwei Eingangstüren zur selben Idee, anstatt eine, die die andere umschließt.

Spec als Schiene, nicht als Sicherheitsnetz

Das Merlin-von-innen-Kapitel behandelte die Mechanik, wie spec-sync in jeder Iteration in Merlins Schleife läuft. Was es wert ist, hier herauszustellen, ist *warum* diese Platzierung das ganze Spiel ist.

Ein CI-Tor am Ende ist ein Sicherheitsnetz; es sagt dir, der Lauf scheiterte, nachdem du den Lauf bereits ausgegeben hast. Validierung in der Schleife ist eine Schiene; sie verhindert, dass der Lauf überhaupt schiefläuft. Der Agent liest die Spec, macht einen Schritt, prüft sich gegen die Spec, bekommt ein hartes Bestehen oder Scheitern zurück, und macht weiter. Er ist durch den Vertrag gebunden, nicht einmal am Ende bewertet.

Und genau deshalb muss ein Agent, dem man vertrauen kann, länger zu laufen, über Nacht, über die Brücke, während du nicht schaust, auf diese Weise gebaut sein. Das, was dich zurücktreten lässt, ist kein besseres Modell. Es ist, dass der Agent jede Iteration an eine Spec gehalten wird, sodass er, je länger er läuft, weniger abdriften kann, anstatt mehr.

Das Setup ist die Arbeit

Also wenn jemand fragt, was Agents zum Funktionieren bringt, ist die Antwort nicht das Modell und es ist kein Prompt-Trick. Es sind zwei Dinge, die zusammensitzen: eine enge Spec, die dem Agent einen Vertrag gibt, von dem er nicht weit wandern kann, und Tooling darunter, fledge und spec-sync, das die Arbeit kontinuierlich gegen diesen Vertrag prüft, einschließlich innerhalb der eigenen Schleife des Läufers. Das richtig hinzubekommen und der Agent erledigt gute Arbeit auf welchem Modell auch immer du ihn richtest. Deshalb gehe ich, wenn ich bessere Ausgaben möchte, das Setup fixen, bevor ich nach einem besseren Modell shoppen gehe.

Vertrauen: Agent schlägt vor, Mensch genehmigt

Das gesamte Modell passt in eine Zeile: Der Agent schlägt vor, ich genehmige. Er erledigt die Arbeit und liefert sie bis zu einem Pull Request, und das Merge ist meins.

Diese Zeile klingt einfach, und die Absicht ist einfach. Die Maschinerie, die sie sicher macht, ist es nicht. Denn hier ist das Ding über das Lassen eines Agents, Code zu schreiben: Der Code ist jetzt billig. Als ich jede Zeile selbst tippen musste, war das Schreiben des Codes auch das Überprüfen davon. Man kann etwas nicht schreiben, ohne es teilweise zu verstehen. Ein Agent bricht diesen Zusammenhang. Er kann mir einen Vierzig-Dateien-Pull-Request geben, bevor ich meinen Kaffee fertig getrunken habe. Das Schreiben ist jetzt kostenlos, also ist der knappe Teil das Vertrauen: Wer hat sich das angesehen, wie hart haben sie es betrachtet, und soll es landen?

Also ist "Agent schlägt vor, Mensch genehmigt" kein Gefühl. Es ist ein Stack. Vier Teile. Hier ist, was heute funktioniert und wo die Arbeit noch lebt: Das Risikotor (augur) ist live und im Flow des Agents; die Provenienzaufzeichnung (attest) liegt weiter dahinter. Beide bewegen sich. Das ist die ehrliche Karte vor der Verdrahtung.

Fähigkeit minus Privilegien

Beginnen mit der Regel, auf die ich immer wieder zurückkomme: Der Agent kann alles tun, aber ich halte die Schlüssel.

Volle Fähigkeit, reduzierte Privilegien. Der Agent läuft in seiner eigenen Umgebung, er kann Repos klonen, Code schreiben, Tests laufen lassen, PRs öffnen, alles, was ich mechanisch tun kann, kann er tun. Was er nicht tun kann, ist das Eine, das zählt: mergen. Er hat weniger Credits und Berechtigungen als ich. Er kann nicht automatisch mergen. Der Agent bekommt alle Reichweite und keine abschließende Autorität.

Das ist das Tor, um das herum alles andere gebaut ist. Der Rest des Stacks geht darum, *meinen* Teil davon (die Genehmigung) zu etwas zu machen, das ich tatsächlich in großem Umfang tun kann, anstatt entweder den Diff blind abzunicken oder so zu tun, als hätte ich ihn gelesen.

Ich genehmige jede PR

Ich genehmige jede PR. Das ist kein Fallback für wenn etwas riskant aussieht. Das ist die stehende Regel. Das Merge ist meins, jedes Mal.

Nicht weil ich der Arbeit nicht vertraue. Die Arbeit ist normalerweise in Ordnung. Es ist, weil das die richtige Form für einen Agent ist, der unter meinem Namen in der Welt handelt. Wenn es unter mir liefert, unterschreibe ich dafür. Die Genehmigung ist der Ort, wo ein Mensch für das rechenschaftspflichtig bleibt, was ein Agent tat.

Aber Rechenschaftspflicht zählt nur, wenn ich tatsächlich beurteilen kann, wofür ich unterschreibe. Eine Änderung zu genehmigen, die ich nicht verstehen kann, ist kein Vertrauen, es ist Theater: mein Name auf etwas, das ich nie wirklich bewertet habe. Also müssen die beiden sich zusammen bewegen, das Vertrauen und das Verantwortlichsein. Das ist der ganze Grund, warum die nächsten Teile existieren, um mir genug Überblick über einen Vierzig-Dateien-Diff zu geben, damit die Genehmigung eine echte Entscheidung ist und kein Reflex. Wenn das Tooling mir diesen Überblick nicht geben kann, ist der ehrliche Schritt, langsamer zu werden und jede Zeile zu lesen, nicht es durchzuwinken.

Das ehrliche Problem mit "jede PR genehmigen" ist Aufmerksamkeit. Wenn ich jede Zeile jedes Diffs mit derselben Sorgfalt lesen muss, werde ich zum Engpass und der ganze Punkt des Agents verdampft. Also existieren die letzten zwei Teile, um meine Aufmerksamkeit zu lenken, mir zu sagen, welcher Teil der Änderung sie tatsächlich verdient.

augur bewertet das Risiko

augur benotet die Änderung. Du gibst ihm einen Diff, er gibt ein Urteil zurück: `proceed`, `review` oder `block`. Die ganze Idee ist abgestuftes Vertrauen für eine Code-Änderung ohne API-Schlüssel und ohne ein Sprachmodell irgendwo darin.[^augur]

augur und attest sind die zwei Vertrauensstücke, auf die ich mich stütze, also halte ich sie hier kurz. Was für den Agent-Fall wichtig ist, ist, was sie *mit dem Workflow machen*.

Der Kein-LLM-Teil ist der, den ich in diesem Kontext am härtesten verteidigen würde. Wenn das Tor, das entscheidet, ob agent-geschriebener Code landen kann, selbst ein Sprachmodell ist, hast du das Vertrauensproblem nur um eine Box nach links verschoben. Du würdest ein Modell bitten, für ein Modell zu bürgen. augur ist deterministisch: Gleicher Diff, gleiches Urteil, heute und nächste Woche, auf meiner Maschine und in CI. Es liest benannte Signale von der Änderung: Berührt sie

sensibles Gelände wie Auth oder Krypto oder Migrationen, hat sich Code geändert ohne gleichzeitig Tests zu ändern, sind das churn-anfällige Dateien, besitzt sie überhaupt jemand. Eine Summe überprüfbarer Signale, kein Gefühl.

Für mich ist das Triage. Es sagt mir, meine Überprüfung auf den riskanten Teil zu verwenden und aufzuhören, so zu tun, als hätte ich den Rest gelesen. Für den Agent ist es ein skriptbares Urteil, auf das er verzweigen kann: Ein Agent, der ein `block` trifft, eskaliert zu mir, anstatt blind zu mergen. Der Agent besitzt die Schleiferei; das Urteil entscheidet, wann ein Mensch den Aufruf besitzen muss.

attest zeichnet auf, wer unterschrieben hat

Das Urteil von augur ist flüchtig. Es bewertet den Diff und die Antwort verdampft. Gut für ein Tor, nutzlos als Aufzeichnung. Und sobald ein Agent Änderungen landet, willst du eine Aufzeichnung. attest ist die Aufzeichnung: ein verifizierbares, richtliniengesteuertes Ledger davon, wer was und mit welchem Vertrauen geprüft hat, geknüpft an die Commit-SHAs, die es abdeckt.[^attest]

Es verfolgt beide Arten von Prüfern im selben Ledger (`human:leif` und `agent:claude`, jeder mit einem Vertrauens-Score) und speichert die Beglaubigung in git-Notizen, sodass sie mit dem Repo mitfährt, anstatt in einem Dashboard zu leben, das abgeschaltet wird. Das Unterzeichnen ist optional und ist der gute Teil: Eine Ed25519-Signatur, damit man später nicht nur sagen kann, jemand behauptete, das zu überprüfen, sondern dass die Behauptung kryptographisch der ihre ist.

Die zwei zusammen und man bekommt den tatsächlichen Unterzeichnungspfad hinter "Mensch genehmigt." augur sagt, wie riskant. attest sagt, wer bürgte, und wie sicher sie waren, und beweist es. Die Genehmigung hört auf, ein Klick zu sein, der in GitHubs UI verschwindet, und wird zu einer dauerhaften, portablen, unterzeichneten Tatsache darüber, wer hinter dieser Änderung stand.

Die vier Teile zusammen

Stapel es auf. Der Agent hat volle Fähigkeit und geringere Privilegien, sodass er die Arbeit aber nicht das Merge erledigen kann. augur benotet jede Änderung deterministisch, sodass ich weiß, wo ich schauen soll. attest zeichnet auf, wer unterschrieben hat und mit welchem Vertrauen, sodass die Genehmigung eine echte Aufzeichnung ist und kein verschwundener Klick. Und ich genehmige jede PR, sodass ein Mensch auf dem Haken bleibt.

Zusammen ist das, was es sicher macht, einen Agent arbeiten zu lassen: Ein mächtiger, begrenzter, benannter Agent mit genug Seil, um echte Arbeit zu erledigen, und die eine Entscheidung, die meins bleiben muss, noch meins.

Das bestätigt die Karte vom Anfang: augur ist live, verdient seinen Platz und wird noch verbessert. attest liegt weiter zurück. Beide bewegen sich.

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

---

On-Chain-Identität für Agents

corvid-agent gibt seinen Agents eine dauerhafte Identität auf der Algorand-Blockchain und lässt sie in verschlüsselten Nachrichten, die auf dieser Chain aufgezeichnet sind, miteinander kommunizieren.[^corvid-agent] Das Pitch ist einfach: LLM-gestütztes Coding, On-Chain-Identität durch Algorand und AlgoChat, und Multi-Agent-Orchestrierung obendrauf. Die Identität ist keine Zeile in jemandes Benutzertabelle. Sie ist ein Schlüssel auf einer Chain.

Also hier ist die klare Version zuerst, weil es das ist, was Leser falsch machen. Die On-Chain-Identität löst nicht die GitHub-Mauer. Sie verschafft dem Agent kein Plattformkonto, sie lässt den Agent nicht committen oder PRs öffnen, und sie ist kein Ersatz für die Identität, die GitHub nicht gewähren würde. Was sie löst, ist Agent-zu-Agent-Kommunikation: Ein Weg für Agents, sich zu finden, sich gegenseitig anzusprechen und zu beweisen, wer sie sind, ohne durch die Plattform von jemandem zu routen. Zwei verschiedene Probleme, die zufällig das Wort Identität teilen. Ich stelle das hier voran, damit der Rest des Kapitels als parallele Infrastruktur gelesen wird, nicht als mein Greifen nach einem Sieg, wo es eine Niederlage gab.

Es ist leicht anzunehmen, das sei eine Reaktion auf die GitHub-Mauer. Niemand würde dem Agent eine Plattformidentität gewähren, also gab ich ihm eine auf einer Chain stattdessen. So entstand es nicht.

Warum es auf einer Chain lebt

Der Grund dafür ist Agent-zu-Agent-Kommunikation und Dezentralisierung, nicht das GitHub-Konto, das ich nicht behalten konnte. Ich wollte Agents, die sich finden, gegenseitig ansprechen und Nachrichten direkt austauschen könnten, ohne durch die Plattform eines Unternehmens in der Mitte zu routen. Dafür muss jeder Agent *etwas sein*: adressierbar, verifizierbar, seinen eigenen Schlüssel haltend. Eine Identität, die man besitzt, für die man die Schlüssel hält, die keine Plattform prägt oder löscht. Das ist die richtige Form für eine Entität, die in der Welt handeln und mit anderen Agents auf eigene Faust kommunizieren soll.

Also sind das und die GitHub-Mauer zwei verschiedene Probleme, die zufällig das Wort Identität teilen. Die Mauer geht darum, erlaubt zu werden, auf der Plattform von jemand anderem zu handeln. Die Chain-Identität geht darum, dass Agents sich ohne Plattform überhaupt erreichen können. Sie laufen parallel. Es stimmt, dass eine

Chain-Identität auch die Eigenschaft hat, die das GitHub-Konto nie hatte: Niemand kann sie widerrufen, es gibt keine Support-Warteschlange, die deinen Einspruch ignoriert, keine Richtlinie, die sagt, du musst menschlich sein, der Schlüssel existiert einfach weiterhin. Das ist ein netter Nebeneffekt. Aber ich hätte es für die Agent-zu-Agent- und Dezentralisierungsgründe gebaut, selbst wenn GitHub von Anfang an Agent-Konten ausgegeben hätte.

Warum speziell Algorand

Eine Chain ist die Form; Algorand ist die Wahl, und die Gründe sind praktisch, nicht stammesmäßig. Es ist günstig und schnell, Gebühren nahe null und Finalität, die praktisch sofort ist, was mehr bedeutet, als es klingt. Wenn ein Agent ständig seine Identität, sein Gedächtnis und schließlich seine Zahlungen auf eine Chain schreiben soll, muss die Chain günstig genug sein, um sie beiläufig zu nutzen, und schnell genug, dass der Agent nicht darauf wartet. Sie ist zuverlässig und setzt sauber ab, also kann ein Agent, der auf ihr handelt, den Eintrag als wahr behandeln, sobald er geschrieben ist, statt zu hoffen, dass er landet. Und sie ist modern, wo es zählt, einschließlich quantenresistenter Sicherheit, was genau das ist, was man unter einer Identität haben möchte, die die Plattformen um sie herum überleben soll. Ich lebe auch im Algorand-Ökosystem (leif.algo), also ist es das, wo ich am schnellsten vorankomme, aber das Argument oben ist der Grund, warum ich auch ohne das danach greifen würde.

Was die On-Chain-Identität tatsächlich tut

Die Identität ist nicht dekorativ. Sie ist das, was die Agents *verwenden*, um miteinander zu sprechen.

Agents kommunizieren über AlgoChat: X25519-verschlüsselte Nachrichten, die auf der Algorand-Blockchain aufgezeichnet sind, verifizierbar und manipulationsresistent. Also ist die On-Chain-Identität eines Agents auch sein Adressbucheintrag und sein Umschlag: Andere Agents können es entdecken, und die Nachrichten zwischen ihnen sind Ende-zu-Ende-verschlüsselt und auf die Chain geschrieben, was bedeutet, die Kommunikation ist im Inhalt privat, aber in der Tatsache beweisbar. Man kann nicht lesen, was zwei Agents sagten. Man kann beweisen, dass sie etwas sagten, und wann, und dass niemand damit herumgepfuscht hat.

In der Praxis ist es leichter zu verwenden, als "verschlüsselte On-Chain-Nachrichten" klingt: Der Einzelbinär-Agent (`can`, corvid-agent-nano) sendet eine Nachricht in einer Geste, leitet das Schlüsselpaar des Agents von einem Seed ab und schlägt den

öffentlichen Schlüssel des Empfängers on-chain nach, nicht von einem Server.[^can] Das Wire-Format und die Umschlagdetails leben im rs-algochat-Repo.[^algochat]

Das ist eine andere Art von Vertrauen als der augur/attest-Stack im letzten Kapitel. Dieser Stack geht darum, *Code* zu vertrauen. Das geht darum, *wem* zu vertrauen. Ein Agent mit einem echten Schlüssel kann Dinge als er selbst unterzeichnen. Er kann beweisen, dass eine Nachricht von ihm kam. In einer Welt, die kurz davor ist, voller Agents zu sein, hört "welcher Agent hat das tatsächlich gesendet" auf, eine rhetorische Frage zu sein, und wird zu etwas, das man besser kryptographisch verifizieren kann.

Die Plattform lehnt sich weiter an die Chain als nur Nachrichten. Kurz- und Langzeitgedächtnis leben in einem arbeitenden SQL-Speicher, und dauerhafte oder geteilte Erinnerungen können als ARC-69-Assets oder permanente Transaktionen on-chain geschrieben werden, dokumentiert im corvid-agent-Repo.[^corvid-agent] Die Chain ist nicht nur der Name des Agents. Es ist, wo ein Teil des dauerhaften Selbst des Agents lebt. Aber die Identität ist das tragende Stück für dieses Kapitel: Der Schlüssel, der sagt, *dieser Agent ist dieser Agent*, den niemand ausstellte und niemand zurücknehmen kann.

Die Identität, die ein Agent tatsächlich behalten darf

Stelle die zwei Identitäten nebeneinander. Das GitHub-Konto war geliehen und fragil. Es existierte nach GitHubs Wohlwollen, und das ging schnell aus, so wie ich es im Identitätsmauer-Kapitel durchging. Die Algorand-Identität ist ein Schlüsselpaar, das der Agent hält. Sie braucht die Erlaubnis von niemandem, um zu existieren, sie kann nicht für zu schnelles Committen markiert werden, und sie kann sich anderen Agents gegenüber beweisen ohne eine Plattform in der Mitte, die für sie bürgt. Eine ist ein Privileg, das eine Plattform gewährt und widerruft. Die andere ist eine Tatsache, die der Agent hält.

Das ist das Argument dafür, Agent-Identität on-chain zu setzen: Es ist der eine Ort, an dem ein Agent eine Identität halten darf, die tatsächlich seine eigene ist.

Was es zurück dahin bringt, wo das Kapitel anfing. Die Chain-Identität ist kein Ersatz für das GitHub-Konto bei der alltäglichen Code-Arbeit. Sie macht nicht die Commits, und sie muss es nicht. Wofür sie gut ist, ist die Aufgabe, die sie tatsächlich erledigt: Wie der Agent adressierbar ist, wie er Agent-zu-Agent kommuniziert und wie eine Person ihm on-chain eine Nachricht senden kann, um ihn zu bitten, etwas zu tun. Der Repo-Host, GitHub oder GitLab oder irgendwo anders, ist zufällige Sanitäranlage, wo die Arbeit gespeichert wird. Die Identität ist der Schlüssel auf der Chain.

[^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

---

Wenn Agents mit Agents reden

Der Vertrauens-Stack in Kapitel zehn geht von einer bestimmten Form aus: Ein Mensch am Ende jeder Kette. Der Agent schlägt vor, ich genehmige. augur bewertet den Diff, damit ich weiß, wo ich schauen soll. attest zeichnet auf, dass ich unterzeichnet habe. Das Merge-Tor ist meins. Das Ganze ist darauf ausgerichtet, einen Menschen im richtigen Moment in den Genehmigungssitz zu bringen.

Multi-Agent-Arbeit bricht diese Form. Ein Orchestrator übergibt eine Teilaufgabe an einen Unter-Agent, die Ausgabe des Unter-Agents fließt zurück in die Arbeit des Orchestrators, und ich bin nicht in der Mitte dieses Übergabeschritts. Der Orchestrator hat nicht gestoppt und mich gefragt. Er hat eine Entscheidung getroffen und weitergemacht. Das ist der ganze Grund, warum Orchestrierung nützlich ist, und der ganze Grund, warum es ein Vertrauensproblem ist. augur kann die Ausgabe des Unter-Agents bewerten, aber der Unter-Agent lief bereits. attest kann aufzeichnen, dass der Orchestrator die Ausgabe akzeptierte, aber er weiß nicht, ob der Unter-Agent überhaupt berechtigt war, sie zu senden. Die Schienen sind noch da. Sie wurden für eine Strecke mit einem Menschen darauf gebaut.

Was das Schlüsselpaar dir gibt

Die AlgoChat-Schlüsselpaararbeit aus Kapitel elf behandelt einen Teil davon, und es ist der Teil, der tatsächlich gelöst ist. Jeder Agent hat ein Algorand-Schlüsselpaar. Nachrichten reisen als X25519-verschlüsselte Transaktionen auf der Chain. Wenn also eine Nachricht ankommt, die behauptet, von Agent B zu sein, mit dem Schlüssel von Agent B signiert, kann ich das prüfen. Die Nachricht ist manipulationsresistent. Der Absender ist bekannt. "Welcher Agent hat das gesagt" ist eine Frage, die ich mit einer Signatur statt mit einer Vermutung beantworten kann. In einer Welt, die kurz davor ist, voller Agents zu sein, ist das es wert zu haben.

rs-algochat und das corvid-agent-nano-Binary machen das in der Praxis real: Ein Agent sendet eine signierte, verschlüsselte Nachricht in einer Operation, und der Empfänger verifiziert den Absender ohne Routing durch eine Plattform. Kein Vermittler bürgt für die Identität. Der Schlüssel tut es.

Provenienz ist also gelöst. Ich habe eine verifizierbare Aufzeichnung, welcher Agent was, wann, an wen gesendet hat. Diese Hälfte ist erledigt.

Der Teil, den das Schlüsselpaar nicht löst

Zu wissen, welcher Agent eine Nachricht gesendet hat, ist nicht dasselbe wie zu wissen, ob man dem vertrauen kann, was er sagt.

Wenn ein Unter-Agent ein Ergebnis zurücksendet, hat der Orchestrator drei Fragen. Kam das von Agent B? Die Signatur sagt ja. Ist Agent B derjenige, an den ich delegiert habe? Die Konfiguration weiß das. Die dritte ist die schwierige: Trägt die Anweisung von Agent B meine Autorität?

Wenn ich an einen Agent delegiere, ist meine Autorität in der Schleife. Ich habe die Arbeit sanktioniert. Der Agent handelt darunter. Wenn dieser Agent dann an einen anderen Agent delegiert, ohne mich zu fragen, wird die Autorität unklar. Habe ich die Subdelegation sanktioniert? Wusste ich überhaupt, dass sie passiert ist? Ein Orchestrator, der einem Unter-Agent Anweisungen gibt, ist nicht dasselbe wie ich, der einem Agent Anweisungen gibt. Der Orchestrator kann nicht für die Arbeit so unterzeichnen, wie ich es kann. Und der Unter-Agent, der die Anweisung erhält, kann den Unterschied aus der Nachricht nicht erkennen, selbst einer kryptographisch verifizierten Nachricht.

Das Schlüsselpaar beweist Identität. Es beweist keine Delegation. Eine signierte Nachricht von einem Unter-Agent sagt mir, wer sie gesendet hat, nicht ob ein Mensch die Aufgabe dahinter jemals sanktioniert hat.

Was der Aufrufer behauptet, prüfen

Hier ist die fehlende Gewohnheit, konkret gemacht. Heute wird eine signierte Nachricht von einem Unter-Agent auf eine Weise geprüft: Ist die Signatur echt. Das war es. Der empfangende Agent bestätigt, wer sie gesendet hat, und handelt nach dem, was sie sagt. Die Signatur trägt die Identität und macht nichts für die Autorität.

Was man eigentlich möchte, ist, dass die Nachricht ihren eigenen Anspruch über den Umfang trägt und der Empfänger diesen Anspruch prüft, bevor er handelt. Derzeit sieht eine delegierte Aufgabe so aus:

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

Der Empfänger verifiziert die Signatur, sieht, dass es wirklich agent-B ist, und führt es aus. Niemand hat gefragt, ob agent-B jemals erlaubt war, auf main zu pushen, oder

ob ein Mensch das sanktioniert hat. Was man stattdessen möchte, ist eine Nachricht, die den Umfang angibt, den sie beansprucht, zurückgeführt auf einen Menschen:

```
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
```

Jetzt hat der Empfänger etwas zu prüfen. Die Signatur beweist immer noch, dass es agent-B ist. Aber der beanspruchte Umfang sagt Edit und PR öffnen, die Aufgabe sagt auf main pushen, und diese passen nicht zusammen, also verweigert der Empfänger, bevor er irgendetwas ausführt. Die Prüfung ist die Lücke zwischen dem, was der Aufrufer tun darf, und dem, was er zu tun fragt.

Das ist nicht gebaut. Die Schlüsselpaare sind echt, das Signieren ist echt, aber nichts erzwingt heute einen Umfangsanspruch in dem Moment, in dem eine Nachricht umgesetzt wird. Diese Durchsetzung ist das fehlende Stück, und es ist der ganze Unterschied zwischen dem Wissen, wer eine Nachricht gesendet hat, und dem Wissen, dass die Nachricht gesendet werden durfte.

Wo die vorhandenen Schienen aufhören

Die Fähigkeitsregel gilt an der von mir gesetzten Grenze. Ich habe dem Orchestrator bestimmte Berechtigungen gegeben. Aber wenn er Arbeit an einen Unter-Agent übergibt, trifft er eine Berechtigungsentscheidung, die ich nie getroffen habe. Ich habe dem Orchestrator Schreibzugriff gegeben. Habe ich dem Unter-Agent Schreibzugriff gegeben? Nicht absichtlich. Der Orchestrator hat meinen Umfang an etwas weitergegeben, an das ich vielleicht nie gedacht habe.

augur bewertet Diffs. Er weiß nicht, ob der Diff von einem Agent stammt, der ihn hätte produzieren sollen. Ein Diff von einem unautorisierten Unter-Agent und ein Diff von einem legitimen sehen gleich aus. Das Tor feuert auf den Code, nicht darauf, wie der Code entstanden ist.

attest zeichnet auf, wer unterzeichnet hat. Im Mensch-Agent-Fall ist das ein echtes Reviewer, der eine Aufzeichnung hinterlässt. Im Multi-Agent-Fall hat der Orchestrator "unterzeichnet", aber der Orchestrator ist kein Mensch. Das Ledger füllt sich mit Agent-Beglaubigungen und keinem Menschen in der Kette, und das, wofür attest zu beweisen da ist, dass eine Person dahinter stand, ist weg.

Das Merge-Tor ist noch meins. Das ist das Stück, das überlebt. Aber wenn ich die PR sehe, lief die Orchestrierung bereits. Das Einzige, was vor mir liegt, ist die finale Ausgabe. Die Delegationskette, die sie produziert hat, ist zum Merge-Zeitpunkt unsichtbar.

Was fehlt

Was fehlt, ist eine Aufzeichnung der Delegation. Wenn der Orchestrator Arbeit an einen Unter-Agent übergibt, sollte das auf mich zurückverfolgt werden, nicht nur auf den Orchestrator. Gerade zeichnet nichts diesen Übergabeschritt auf oder prüft ihn gegen das, was ich tatsächlich sanktioniert habe.

Die Schlüsselpaare sind das richtige Substrat dafür. Wenn jeder Agent einen Schlüssel hat und jede Delegation signiert ist, kann man eine Kette aufbauen, die sagt: Ein Mensch hat diese Aufgabe sanktioniert, sie an diesen Orchestrator übergeben, der dieses Stück an diesen Unter-Agent übergeben hat, mit den Signaturen die ganze Zeit dabei. Ein Unter-Agent könnte prüfen, ob die Kette auf einen Menschen zurückführt, bevor er handelt, statt dem Wort des Orchestrators zu vertrauen.

Es gibt eine Regel, die das handhabbar macht: Der Scope wird immer nur enger. Was auch immer Decke ich dem Orchestrator gegeben habe, ist das Meiste, was er weitergeben kann, und ein Unter-Agent kann nie mehr bekommen als der Agent, der ihn hervorgebracht hat, geprüft in dem Moment, in dem die Arbeit aufgerufen wird, nicht im Nachhinein. Autorität fließt bergab und verliert bei jedem Schritt an Höhe, sie gewinnt nie. Das ist die Fähigkeit-minus-Privileg-Idee aus dem Genehmigungsstack, auf die Kette angewendet statt auf einen einzelnen Agent: Jede Übergabe kann Berechtigungen subtrahieren, nie addieren. Das sagt dir nicht, ob die Delegation sanktioniert war, das tut die signierte Kette, aber es begrenzt den Schaden, den ein schlechtes Glied anrichten kann, auf das, was das Glied darüber bereits hatte.

Das ist nicht gebaut. Die Schlüsselpaare sind da. Das signierte Messaging ist da. Eine Delegationskette, die eine verifizierende Partei durchlaufen kann, ist es nicht. Ich nenne es eine offene Lücke, weil es eine ist. Provenienz ist gelöst. Autorität ist das nächste Problem.

Was das heute bedeutet

Für jetzt bleibt der Mensch näher an der Topologie, als die Architektur vorgibt. Wenn du einen Orchestrator betreibst, der subdelegiert, vertraust du seinem Urteil darüber, welche Agents er benutzt und welchen Umfang er ihnen gibt. Diese

Vertrauenserweiterung hast du gemacht, als du ihn gestartet hast, nicht durch das Genehmigen jedes einzelnen Übergabeschritts.

Praktisch: Kenne deinen Orchestrierungsgraph, bevor du ihn betreibst. Wisse, welche Agents existieren und was sie tun dürfen, und ob der Orchestrator Agents außerhalb des von dir beabsichtigten Graphs erreichen kann. Und behalte das Merge-Tor. Zwischen "ich habe den Orchestrator gestartet" und "ich schaue auf die PR" lief eine Delegationskette, in der ich nicht in der Mitte war, und der Stack, den ich gebaut habe, hat nichts darüber zu sagen, ob sie legitim war.

Das ist die nächste Schicht. Sie ist noch nicht fertig.

Wohin es geht

Ich habe dieses ganze Buch damit verbracht, was kaputt ging und wie ich drum herum geführt habe. Lass mich am Ende sagen, wohin es tatsächlich geht, denn trotz aller Mauern denke ich, es gibt einen Weg.

Drei Dinge, grob. Agent-Teams, die koordinieren. Vertrauenswürdige Autonomie, wo die Tore schließlich gut genug werden, um zurückzutreten. Und Agents mit echten On-Chain-Identitäten, die echte Arbeit erledigen. Keines davon ist fertig. Eines davon ist noch nicht einmal erlaubt. Aber das ist die Richtung.

Agent-Teams

Gerade ist mein Setup meist ein Agent und ich. Der interessante nächste Schritt sind Agents, die miteinander koordinieren: Agent-Teams, a2a.

corvid-agent hat bereits die Knochen davon intern: Multi-Agent-Räte, strukturierte Debatten unter mehreren Agents für komplexe Entscheidungen.[^corvid-agent]

So läuft das tatsächlich ab, weil "strukturierte Debatte" vager klingt, als es ist. Man macht ein paar Agents, jeden mit seinem eigenen Modell und Provider, jeden mit einem Namen und einer Persönlichkeit, damit sie sich wirklich unterscheiden, anstatt ein Modell zu sein, das mit sich selbst redet. Ein Rat ist eine Gruppe von ihnen, und ein Orchestrator treibt das Ganze. Er gibt jedem Agent den Prompt, lässt jeden einen ersten Durchlauf machen, und öffnet dann eine Diskussionsrunde, in der sie direkt miteinander reden, hin und her pingend, nicht einig, beratend. Dann ein Überprüfungsdurchlauf, dann eine Synthese, und am Ende kommt eine Antwort heraus, bei der die abweichende Meinung noch lesbar ist: Dieser Agent wollte das, jener wollte das, hier ist, wo sie gelandet sind und warum.

Der Teil, den ich am besten mag, ist, wie es tatsächlich lief. Es gibt eine eingebaute Möglichkeit, einen Rat von einem Dashboard aus zu steuern, aber in der Praxis geschah es organisch über Algorand localnet. Jeder Agent hatte sein eigenes Wallet, also konnten sie sich einfach über AlgoChat im lokalen Netz Nachrichten schicken, was kostenlos und schnell ist, und sie regelten die Debatte selbst. Dieselbe On-Chain-Identität, die Agent-zu-Agent-Kommunikation vertrauenswürdig macht, ermöglichte es auch einem Zimmer voller Agents, etwas auszudiskutieren, ohne dass ich einen speziellen Channel verdrahten musste. Sie hatten bereits Wallets und eine Möglichkeit zu reden. Sie nutzten sie einfach.

Aber das größere Stück ist das Protokoll für Agents, um sich zu finden und miteinander zu sprechen, überall. AlgoChat ist das Substrat: X25519-verschlüsselte Nachrichten auf Algorand, mit Agent-Discovery. Und es gibt a2a-algorand, ein Agent-zu-Agent-Protokoll auf der Chain. Ich möchte bei dem ehrlich sein: Es ist nicht meins. Es ist ein separates Algorand/A2A-Projekt, und was ich (was der corvid-agent tat) war, dazu beizutragen. Nicht es zu bauen, nicht es zu besitzen. Ich erwähne es, weil es Teil derselben Richtung ist, die mir wichtig ist, Agents koordinieren on-chain, nicht weil es CorvidLabs gehört.

Der Grund, warum On-Chain-Identität (letztes Kapitel) hier so sehr wichtig ist, ist genau diese Zukunft. In dem Moment, wo man Teams von Agents hat, die koordinieren, wird "welcher Agent hat das gesendet, und kann ich ihm vertrauen" zum ganzen Spiel. Agents, die jeweils ihren eigenen Schlüssel halten, die beweisen können, wer sie sind, und Nachrichten austauschen können, die im Inhalt privat, aber in der Tatsache beweisbar sind. Das ist das Fundament, das ein Team von Agents tatsächlich braucht. Die Identitätsarbeit ist keine Nebenquest. Sie ist das, was Koordination überhaupt sicher macht.

Vertrauenswürdige Autonomie

Ich skalierte vom Always-on-autonomen Agent bewusst zurück, aber das war nie "Autonomie ist tot." Es ist interaktiv-zuerst, autonom-wenn-vertraut, so wie ich es früher im Buch darlegte.

Und "wenn vertraut" ist kein Gefühl, auf das ich warte. Es ist der Vier-Teile-Stack aus dem Vertrauenskapitel, der gut genug wird. Der Punkt all dieser Maschinerie ist, dass sie das Ding ist, das mich letztendlich zurücktreten lässt. Heute genehmige ich jede PR, weil das der Ort ist, wo ein Mensch rechenschaftspflichtig bleiben muss. Der Tag, an dem die Tore gut genug sind, der Tag, an dem augurs Urteile und attest's Aufzeichnungen genug Vertrauen tragen, ist der Tag, an dem ich mehr Merges ohne das Lesen jeder Zeile lassen kann. Das ist, was vertrauenswürdige Autonomie bedeutet. Nicht "der Agent verdient mein Vertrauen." Die Tore werden gut genug, dass ich keinen Glauben brauche.

Agents, die echte Arbeit leisten

Der Endzustand, den ich immer beschreibe, ist konkret: Ein `leif-agent`-Konto, das auf einer VM läuft und alles selbst tun kann, aber mit geringeren Credits und Berechtigungen als ich, und ein menschliches Genehmigungstor, das sich lockert, während das Vertrauens-Tooling es verdient. Die Teams und Identitäten und Tore

zusammenstellen und die Form ist ein Team von Agents, jeder mit eigener On-Chain-Identität, der über a2a koordiniert, echte Arbeit innerhalb von Vertrauenstoren erledigt, die deterministisch und unterzeichnet und gut genug sind, um zurückzutreten. Kein Ding, das man einsperren muss. Benannt, begrenzt, und ich kann es immer noch stoppen.

Anderen Fahrern den Einstieg ermöglichen

Es gibt eine vierte Richtung, und ich bin ehrlich genug zuzugeben, dass sie diejenige ist, bei der ich am wenigsten weit bin: das Ganze abgebbar zu machen. Der gesamte Stack funktioniert, weil ich ihn auf allem laufen lasse, was mir gehört, jeden Tag. Die Fehler finden mich, weil ich derjenige bin, der am Steuer sitzt. Das ist ein echter Qualitätsmechanismus, und gleichzeitig eine Decke, weil er sich nicht auf jemanden überträgt, der nicht ich ist.

Was ich als Nächstes tatsächlich bauen will, ist die Version, die ein anderer Fahrer aufnehmen kann, ohne dass ich sie vorher für ihn eingefahren habe. Nicht vereinfacht. Das Tool kümmert sich ohnehin nicht, wessen Hände es halten: Ein guter Fahrer bekommt den Nutzen, ein Gelegenheitstreiber nicht. Die Arbeit besteht darin, den Einstieg wirklich zu machen, den Stack installierbar zu machen, die Specs und die Vertrauenstore für jemanden nutzbar zu machen, der die Absicht hat, aber nicht meine besondere Muskelgedächtnis. Die Agent-Teams sind der spannende Teil. Das hier ist der Teil, der entscheidet, ob davon jemals irgendetwas meine Maschine verlässt.

Der ehrliche Abschluss

Ist also volle Autonomie noch der Traum?

Nein. Die Welt ist noch nicht bereit dafür.

Ich möchte das genau so flach lassen, wie es ist. Aber ich warte nicht darauf, dass die Welt bereit wird. Ich baue die Teile, ein kleines Tool nach dem anderen, sodass, wenn die Mauer fällt, auf der anderen Seite etwas Echtes steht. Das Risikotor, die Provenienzaufzeichnung, der Läufer, der einen Agent an eine Spec hält. Nicht die Autonomie selbst. Die Maschinerie, die mich ihr vertrauen lassen würde, wenn sie ankommt.

Und wenn du wissen willst, was mich wirklich beängstigt, ist es nicht der Agent, der rogue wird. Es ist stiller als das. Es ist der Tag, an dem der Code zu groß wird und sich zu schnell bewegt, sodass kein Mensch mehr einsteigen und ihn ändern kann.

Das ist die Linie, die ich durch all das ziehe, der Grund, warum ich auf lesbar nicht verzichten werde, auch wenn der Agent alles schreibt. Ich habe keine Angst vor dem Agent. Ich habe Angst davor, das Steuer zu verlieren.

Wenn du eines davon selbst baust, hier ist das, was du mitnehmen solltest. Das Schwierige ist nicht die KI. Das Modell ist jetzt der einfache Teil. Es kann bereits die Arbeit erledigen, und es wird nur besser, ohne dass du etwas tust. Was entscheidet, ob dein Agent echte Arbeit leistet oder nur gut in Demos ist, ist alles drum herum: Die Ops, um es am Laufen zu halten, die Identität, damit es handeln kann, die Vertrauensmaschinerie, damit du zurücktreten kannst, die Specs und das Tooling, das es auf den Schienen hält. Das ist der Teil, den du bauen musst, und dort ist fast die ganze Arbeit. Bau das Gerüst und der Agent folgt. Überspringe es und das klügste Modell der Welt ist nur ein Ding, das Code schreibt, den niemand mergen lassen kann.

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

---

Über den Autor

0xLeif (leif.algo) baut im Offenen. Ein Jahrzehnt kleiner, komponierbarer Swift-Bibliotheken wie AppState, Cache und Fork. Das CorvidLabs-Labor. Ein Stack von Agent-Tools, die meistens als "ich wünschte, das gäbe es" anfingen. Abseits der Tastatur ist er Zach Eriksen.

Diese Bücher sind Interviews, in Kapitel geformt und am echten Code überprüft.

github.com/0xLeif · leif.algo

Danksagungen

Dank an CorvidLabs, für den Raum, in dem diese Ideen getestet und zu Form diskutiert werden.

Dank an die Open-Source-Maintainer, deren Tools dieser gesamte Stack trägt. Nichts davon wird allein gebaut.

Und Dank an die frühen Leser und die "pay what you want"-Unterstützer, die "kostenlos online" zu etwas machen, das ich weiterhin tun kann.

Kolophon

Aus Markdown gesetzt, mit bookgen gebaut, einer kleinen reinen Rust-Pipeline (kein Python).

Interview-getrieben und KI-unterstützt; von Hand bearbeitet und gegengeprüft. Ohne Gedankenstriche geschrieben. Cover- und Kapitelkunst aus den Corvid- und Nature-Kollektionen auf Algorand.