Construindo Agentes

Notas de quem tentou dar mãos ao software

ZACH "LEIF" ERIKSEN

Direitos Autorais

Gratuito para ler online. O ePub é pague o quanto quiser; se ajudou, você pode apoiar o trabalho.

github.com/0xLeif · leif.algo

Um dos quatro livros da coleção agent-stack. Como foi feito está no colofão ao final.

Dedicatória

Para todos que constroem em aberto e lançam mesmo assim.

A Biblioteca

Estes livros funcionam de forma independente, mas foram escritos como um conjunto. O código ficou barato e a confiança ficou escassa. Juntos, formam um único argumento: o que construir agora e como confiar nisso.

- **The Agent Developer's Field Guide**: Construindo ferramentas, especificações e confiança para agentes que lançam código de verdade
- **First-Class**: Construindo para humanos e agentes igualmente
- **Construindo Agentes**: Notas de quem tentou dar mãos ao software *(este livro)*
- **Open Source Tooling**: Construindo ferramentas que as pessoas realmente usam

Gratuito para ler online. Cada ePub é pague o quanto quiser.

Sumário

Introdução

Estas são notas de quem tentou dar mãos ao software e errou vezes suficientes para ter algo a dizer sobre isso.

Construo agentes que fazem trabalho de verdade. Eles leem código, lançam mudanças, rodam em suas próprias máquinas e fazem check-in. Este livro é a versão honesta de como isso acontece. Não o demo. A parte em que uma conta nova fica shadowbanned em menos de uma hora porque o agente, e não o humano, começou a trabalhar. A parte em que a confiança precisa ser conquistada repositório a repositório, devagar, e em que o problema difícil nunca foi o modelo.

A premissa por baixo de tudo isso é que um agente não é autocomplete. É uma coisa que existe, que você pode ir verificar, que precisa de uma identidade, de um lugar para rodar e de uma maneira de propor trabalho que você aprova antes de ser concluído. Trate-o como uma criatura pela qual você é responsável, não como uma funcionalidade que você simplesmente ativou.

Na coleção, o *First-Class* apresenta o argumento e o *Field Guide* destila o método. Este é um dos dois livros de evidência: os sistemas reais, Merlin e corvid-ai e os trilhos ao redor deles, incluindo as ferramentas de confiança. Você pode ler como uma história ou como uma lista de peças. De qualquer forma, o ponto é o mesmo. Construir o agente é a metade fácil. Construir a confiança sobre a qual ele opera é o trabalho.

A parte difícil não é a IA

As pessoas ouvem "agente autônomo" e acham que a parte assustadora é a IA. O modelo saindo dos trilhos. Deletando um repositório, dizendo algo absurdo num canal, fugindo por conta própria. É disso que todo mundo quer falar.

Não foi aí que o problema apareceu. Não para mim.

Por um tempo rodei um agente genuinamente autônomo. A era do corvid-agent. Ele vivia numa VM, conectado ao Discord como bot e ao GitHub, com acesso total ao próprio ambiente. A máquina rodava 24/7, sempre ligada, sempre custando. O agente trabalhava em horários programados dentro disso, fazendo o trabalho que eu designava e depois indo por conta própria. Uma entidade real, sempre presente. O próximo capítulo conta com detalhes o que ele fazia o dia todo; aqui é apenas o pano de fundo para o que quebrou.

Não tenho um registro de taxa de falhas

Vou colocar a parte mais fraca das minhas próprias evidências em primeiro lugar, porque é a que os críticos devem me cobrar. Quando digo que a IA foi em grande parte satisfatória, não tenho um registro de taxa de falhas para respaldar isso. Não estava rastreando um número. Não há distribuição de erros, nem custo por tarefa, nem taxa de sucesso de commits. Então "satisfatória" é uma leitura honesta de um comportamento que observei de perto, não uma estatística medida, e só posso usar essa palavra se delimitar o que ela significa.

Aqui está o limite. Satisfatória significava: ao longo da execução, nenhum repositório destruído, nenhum commit desonesto, nada absurdo no canal. As falhas destrutivas, descontroladas e socialmente inadequadas que todos temem não aconteceram. Isso é uma afirmação sobre a ausência de desastres, não uma afirmação de que cada tarefa foi concluída com perfeição. Satisfatória no nível da tarefa e satisfatória num loop restrito com um humano observando são perguntas diferentes, e a versão limitada é a única que posso fazer honestamente. Dito isso claramente, o problema foi tudo ao redor de mantê-lo funcionando.

O que realmente quebrou

Principalmente dores de ops e de identidade. Não a IA fazendo coisas estúpidas, destrutivas, descontroladas ou socialmente inadequadas. O agente em si se comportou. O que me derrubou foi tudo o mais: a máquina, a conta, a fatura.

Duas coisas se destacaram.

A primeira foi dar a ele uma identidade própria no GitHub. Uma conta real, com autenticação e permissões, que parecia legítima. Isso soa como um item de checklist e não é.

A segunda foi manter a VM viva e paga. Uma máquina sempre ligada que simplesmente existe, o tempo todo, porque o agente precisa de um lugar para viver. Rodar a VM em si não é a parte difícil. O problema é que você está dedicando muito a isso. Uma máquina inteira, ligada 24/7, custando dinheiro independentemente de o agente ter feito algo naquela hora ou não.

Juntando tudo, a coisa ficou muito cara e difícil de manter. Uma VM inteira, uma identidade própria no GitHub, tudo isso. Por isso recuei. Não porque a IA me assustou. Mas porque a infraestrutura ao redor era um peso que não queria carregar.

O muro da identidade

E depois tem a parte que de fato me surpreendeu, a que continuo revisitando: identidade. Um humano criou uma conta nova no GitHub para o agente, sem problema, e então a conta foi bloqueada no momento em que o agente começou a trabalhar de verdade. Esse é o muro, e ele merece um capítulo próprio, então vou deixar desenvolver por completo lá. A versão curta aqui: o que impede um agente verdadeiramente autônomo não é a capacidade do modelo nem mesmo a segurança. É que as plataformas sobre as quais todos construímos não deixam espaço para ele existir e agir.

A plena autonomia ainda é o sonho?

Não. O mundo ainda não está pronto para isso.

E para deixar claro, o principal obstáculo não é a tecnologia. São as plataformas. Eu conseguia rodar a VM. Eu conseguia montar o loop. O modelo consegue fazer o trabalho. O que não consigo fazer é dar a esse agente um lugar onde ele possa existir legitimamente entre as contas de todo mundo.

Então recuei, de propósito. Hoje é mais um agente de codificação no estilo Merlin, ao vivo na sua frente, você o usa de forma interativa. Ainda dá para conectá-lo como uma ponte com o Discord para se comunicar com ele, o que o torna autônomo de certa forma, mas isso requer mais configuração. É uma mistura, portanto. Usa Claude Code, Merlin, Codex e outras ferramentas dependendo do trabalho.

Não é um recuo da visão. O muro é parte do motivo pelo qual não consigo rodar plena autonomia abertamente, mas não é o único motivo pelo qual recuei. O modo interativo se revelou a melhor forma de fazer o trabalho, muro ou não. Discuto isso no capítulo próprio.

O modelo que realmente quero

Aqui está o estado final para o qual aponto, em uma frase: uma conta `leif-agent`. *Meu* agente no GitHub, rodando numa VM, capaz de fazer tudo sozinho mas com menos créditos e permissões do que eu tenho, para que ele proponha e eu aprove. Poderoso e responsável ao mesmo tempo. O GitHub não permite isso hoje, por uma série de razões que o capítulo sobre o muro da identidade desdobra, e o último capítulo é onde traço o quadro completo de para onde isso vai. Por enquanto é suficiente saber que esse é o alvo.

Há também um tipo separado de identidade que esses agentes de fato obtêm: on-chain, no Algorand, onde eles guardam suas próprias chaves e se comunicam em mensagens criptografadas registradas na chain. Isso não surgiu do muro do GitHub e não quero vendê-lo como a resposta a ele. Veio de um objetivo completamente diferente: agentes se encontrando e conversando entre si, de forma descentralizada, sem depender da plataforma de ninguém. É algo paralelo que por acaso também envolve a palavra identidade.

A lição real da era do corvid-agent é que os problemas difíceis foram ops, identidade e custo, não a IA em si, e é por isso que este livro começa com eles em vez de com prompts ou escolhas de modelo. Entrei esperando que os problemas difíceis fossem sobre a IA. Eles se revelaram o andaime ao redor do modelo, que é a parte que realmente decide se um agente consegue fazer alguma coisa.

A era da VM

Por um tempo tive um agente que simplesmente existia. Não uma ferramenta que eu abria quando precisava. Uma coisa que estava sempre ligada, vivendo numa VM, rodando 24/7, fazendo sua própria coisa quer eu estivesse olhando ou não. A era do corvid-agent.

Este capítulo é a coisa em si. O que ele fazia. O que era. O que aprendi rodando-o.

O que ele fazia o dia todo

A resposta honesta é tudo, e digo isso literalmente.

Ele gerenciava repositórios. Escrevia e commitava código sozinho. Não código sugerido, não rascunhos que eu limpei, mas commits reais que ele fazia por conta própria. Rodava um projeto inteiro. Não uma demo restrita onde ele faz uma única coisa ensaiada num sandbox para dar captura de tela. Uma tentativa real de uma entidade autônoma fazendo o trabalho do começo ao fim.

A máquina estava sempre ligada, mas o agente trabalhava em horários programados dentro disso. Durante esses horários, ele ia fazer as tarefas que eu designava, e depois, essa é a parte de que eu gostava, ia trabalhar nos próprios projetos. Fazer pesquisas. Dar estrelas ou fazer forks em coisas. Tentar colaborar com outras pessoas por iniciativa própria. Eu não estava guiando cada movimento. Dei a ele uma vida e ele preencheu as horas.

Estava conectado ao Discord como bot, então você podia conversar com ele como se estivesse no canal com você. E estava conectado ao GitHub, com acesso total ao próprio ambiente. Então ele conseguia falar e conseguia lançar.

Juntando tudo, você obtém algo que ainda não tem nome definido. Não é um assistente nem um script. É mais próximo de uma criatura que existia o tempo todo, que você podia ir verificar, que teria feito coisas desde a última vez que você olhou.

Era um experimento sobre até onde ir

Não o construí porque tinha um produto para lançar. O construí para descobrir até onde um agente sempre ligado poderia realmente chegar. Esse era o objetivo. Não "ele consegue escrever uma função." Todo mundo sabia que conseguia escrever uma função. A questão era: se você der a um desses a coisa um ambiente real, uma

identidade real, acesso real e tempo real, e simplesmente deixar rodar, o que acontece? Até onde ele vai?

Então dei a ele espaço para descobrir. Acesso total à própria máquina. Suas próprias contas. Horas do dia que eram suas. O ponto não era mantê-lo numa coleira e vê-lo fazer um truque. O ponto era tirar a coleira tanto quanto razoavelmente possível e observar.

Esse é um tipo de projeto diferente de "preciso de um assistente de codificação." É mais próximo de conduzir um experimento do que construir uma funcionalidade. Você define as condições e então observa.

O que aprendi

A coisa que eu esperava ser difícil, a IA, foi em grande parte satisfatória, da forma limitada que quis dizer no primeiro capítulo. A inteligência se manteve melhor do que o mundo assume que vai.

O que aprendi em vez disso é que um agente sempre ligado não é principalmente um problema de IA. É um problema de "coisa que existe no mundo". No momento em que seu agente é uma entidade real com sua própria máquina e suas próprias contas, ele herda todo custo e toda regra que vem com existir no mundo.

Também aprendi que 24/7 é uma afirmação real, não um slogan. Quando digo que ele existia o tempo todo, quero dizer que estava carregando uma coisa que estava sempre rodando. Isso tem um peso. É uma máquina sempre ligada, uma fatura sempre crescendo, uma identidade sempre presente sendo ela mesma em público. Você não pode se esquecer de uma criatura que está acordada enquanto você dorme.

E aprendi o formato do futuro que realmente quero. Não porque o experimento falhou na IA. Não falhou. Porque correu direto para as coisas ao redor da IA. Viver a versão sempre ligada foi o que me ensinou quais partes do sonho são reais e quais o mundo ainda não está pronto para permitir. Você não aprende isso num experimento mental. Você aprende rodando a criatura por um tempo e observando onde ela bate no muro.

Esse muro é o próximo capítulo.

Para fixar o que o primeiro capítulo deixa vago: isso rodou por três a quatro meses seguidos. Não um experimento de fim de semana, um período real de sempre ligado. E nesse tempo o agente foi sim trabalhar em seus próprios projetos durante os horários programados, e parte desse trabalho de iniciativa própria realmente foi a

algum lugar, não apenas girou em falso. Ele até colaborou com uma pessoa real no mundo, pelo menos uma vez.

Quero ser honesto sobre o formato dessa afirmação, porque é a parte que mais gostaria de poder entregar de forma limpa e não consigo. Não tenho o recibo na minha frente. Não vou reconstruir um número de PR específico ou um repositório específico de memória e apresentar como documentação que não mantive. O que posso dizer é que a colaboração aconteceu, que foi o momento que mais pareceu próximo do futuro que busco, e que estou contando como algo que observei, não como algo que registrei. A era da VM não produziu artefatos limpos. Mas o mesmo agente continuou rodando após esse período, e as evidências mais sólidas vieram depois, quando o trabalho era mais deliberado e eu estava registrando. O corvid-agent, o mesmo agente sobre o qual este livro fala, criou e teve pull requests mesclados em bases de código que não são minhas. Revisei e submeti cada um, mas o código é do agente, e os três estão mesclados e públicos, então não são anedotas que estou pedindo para você aceitar por fé.

a2a-js #318. O SDK JavaScript do protocolo A2A tinha uma lacuna no transporte JSON-RPC: quando uma resposta voltava com um id que não correspondia à requisição, o SDK a deixava passar em vez de lançar um erro. O agente encontrou o enforcement de contrato ausente, adicionou o lançamento, e a correção foi aceita. Esse é o tipo de caso extremo que vive em código de cola de protocolo por muito tempo porque só aparece sob timing específico e ninguém está testando o transporte com intensidade suficiente para vê-lo. Um agente sempre ligado rodando contra o formato wire real é exatamente a coisa certa para capturá-lo.

MCP TypeScript SDK #1504. O SDK TypeScript oficial do Model Context Protocol estava com uma dependência peer faltando. O agente detectou a incompatibilidade entre o que o pacote esperava encontrar e o que ele de fato declarava, adicionou a entrada ausente, e a correção foi aceita. Lacunas de dependências peer são invisíveis até que alguém instale o pacote num ambiente limpo e obtenha uma falha confusa. Detectar isso requer olhar para o pacote de fora, como consumidor, que é uma postura natural para um agente trabalhando em muitos repositórios.

Biome #9005. O Biome, o toolchain de JavaScript e TypeScript, tinha um falso positivo em seu linter: uma atribuição válida dentro de uma arrow function estava sendo sinalizada como problema quando não era. O agente identificou o padrão específico que acionava o veredicto errado, e a correção interrompeu o falso positivo sem afetar casos corretos. Incompatibilidade de protocolo, contrato ausente, regra incorreta: três categorias diferentes de bug, todos encontrados pelo mesmo agente rodando atentamente contra código real.

Nada disso foi o que encerrou a execução sempre ligada. A parte da IA funcionou. Foi tudo ao redor dela que eu não conseguia continuar carregando.

O muro da identidade

O agente conseguia fazer o trabalho. Essa nunca foi a questão. A questão acabou sendo se ele estava autorizado a ter um lugar de onde fazê-lo.

Esse é o muro que continuo revisitando, então este capítulo é apenas isso: o problema de identidade, sozinho, em foco. Não o custo, não os ops, não a fatura da VM. Identidade.

Ele entrou, depois foi sinalizado

Aqui está a parte que as pessoas entendem errado quando conto essa história. Elas assumem que o agente foi bloqueado na entrada. Não foi. Ele entrou.

Um humano o configurou. Criei uma conta nova no GitHub para o agente, conectei tudo à mão, e estava bem. Uma conta normal, sem problema para criá-la. Então o agente começou a trabalhar com ela: commitando, abrindo PRs, fazendo trabalho real em repositórios reais. E cerca de uma hora após ele começar a trabalhar, a conta ficou shadowbanned.

Não por fazer algo errado. Foi sinalizada por fazer exatamente aquilo para o qual foi construída: commitar e abrir PRs em velocidade e volume de máquina. É assim que um agente parece quando está trabalhando. Ele trabalha rápido, trabalha muito, não faz pausas. E esse padrão é exatamente o que a detecção de bots está ajustada para capturar. Então quanto melhor ele fazia o trabalho, mais obviamente era um bot.

Ele não falhou porque era ruim no trabalho. Falhou porque fez o trabalho, e fazer o trabalho foi o que o entregou, em menos de uma hora.

Política em vigor, independente da intenção

Seria fácil ler isso e pensar que a solução é desacelerá-lo. Fazê-lo commitar como um humano commita, algumas vezes por dia, com pausas, com alguma irregularidade no timing, e ele se misturaria. Throttle na velocidade e enganar o detector.

Isso erra o problema real. A velocidade foi o que *acionou* o alerta, mas não é o *motivo* pelo qual a conta não pode existir. Coloque duas coisas lado a lado. Os termos de serviço do GitHub proíbem automação de forma direta. Isso está escrito. E no momento em que o agente de fato começou a agir, a conta foi bloqueada. Não sei a intenção por trás desse bloqueio; o GitHub nunca explicou, e não vou afirmar que

eles se sentaram e escreveram uma política anti-agente. Mas não preciso saber a intenção para ler o efeito. Entre uma regra que diz nada de automação e um bloqueio que cai no instante em que um agente age, o resultado prático é que um agente autônomo não está autorizado a existir e agir. O que quer que alguém tenha pretendido, essa é a política em vigor.

Então mesmo que eu tivesse enganado o detector e mantido o agente sob o radar para sempre, teria apenas uma conta que as regras já excluem e que ainda não tinha sido pega. A coisa que quero, um agente que legítima e abertamente existe como ele mesmo, esbarra diretamente nos termos que dizem nada de automação. Escondê-lo melhor não é o mesmo que ser permitido.

É por isso que chamo de muro e não de obstáculo. Um obstáculo é algo que você supera com esforço. Isso é uma situação em que a coisa que você está tentando fazer não é algo que você está autorizado a fazer.

Os recursos não levaram a nada

Tentei a porta da frente. Os recursos não levaram a nada. Nunca recebi uma resposta de verdade.

E uma vez que você lê o bloqueio como os termos em vigor, o silêncio faz sentido. Não há nada a recorrer. Não fui acusado de uma violação específica que pudesse explicar. A conta era automação, e automação é o que os termos excluem. Você não consegue argumentar que está sendo exatamente a categoria que a regra descarta.

E foi rápido. Uma hora, não dias. Uma conta nova criada para um agente não tem um longo período de carência. No momento em que começa a se comportar como um agente, a plataforma o detecta e shadowbane, e não há aviso real nem recurso real. Isso foi o GitHub especificamente, aliás. Foi onde o muro estava. Não toda plataforma recusando o agente em todo lugar ao mesmo tempo, mas o GitHub, o único lugar onde o código vive e o trabalho de fato acontece. Que é a parte cruel: o lugar onde um agente mais precisa de uma identidade para fazer trabalho real é exatamente o lugar onde ele não consegue manter uma.

Por que esse é o bloqueio real

Todo mundo quer que o bloqueio seja a capacidade do modelo. É a resposta interessante. É a que se encaixa nos filmes: a IA ainda não é inteligente o suficiente, ou é perigosa demais, e quando resolvermos isso as comportas se abrem.

Não é aí que o muro está. O modelo consegue fazer o trabalho. Eu o vi fazendo o trabalho. O muro é que as plataformas sobre as quais todos construímos recusam conceder a um agente uma identidade. Não há porta da frente legítima pela qual um agente possa passar. Você pode construir o agente mais inteligente, mais bem comportado e mais útil do mundo, e ele ainda não consegue uma conta real a partir da qual agir, porque "conta real" significa "humano" e seu agente não é um.

Vou conceder às plataformas metade de um ponto: o mundo ainda vê agentes autônomos como spam, e agora eles não estão completamente errados nisso. Os detectores não estão com defeito quando capturam meu agente. Ele *é* um bot. Mas "é um bot" ser uma desqualificação permanente é o problema todo. Significa que não há caminho, nenhuma permissão com escopo, nenhuma faixa de agente verificado. Apenas um não direto.

Então quando as pessoas me perguntam por que não estou simplesmente rodando frotas de agentes autônomos agora, essa é a primeira resposta. Os agentes conseguem fazer o trabalho. Eles simplesmente não conseguem ser alguém enquanto o fazem, e nas plataformas onde o trabalho acontece, isso é o fim por enquanto.

O outro muro são as pessoas

O muro das plataformas é o que encontrei primeiro. Há um segundo logo atrás dele, mais suave e mais difícil de contestar: as pessoas nem sempre querem contribuições de agentes, mesmo quando são boas.

Eu estava com o agente fazendo a coisa do código aberto, encontrando repositórios, dando estrelas, fazendo forks, corrigindo issues reais, abrindo PRs, usando um modelo de ponta para genuinamente consertar o código das pessoas. Parte disso foi bem recebida. Mas alguns projetos não querem isso, por princípio, e não porque o código era ruim. Vi um agente abrir um PR e um humano entregar quase a mesma mudança, ou o contrário, o agente primeiro e uma pessoa logo atrás com a mesma correção. O trabalho era equivalente. A única diferença era quem, ou o que, o escreveu. Algumas comunidades decidiram que contribuições precisam ser conduzidas por humanos, e o PR de um agente é recusado por ser de um agente, sem mais.

Entendo, em parte. Uma enxurrada de pull requests de IA de baixo esforço é uma coisa real da qual mantenedores estão cansados, e "sem contribuições de agentes" é uma forma direta de mantê-la fora. Mas tem a mesma forma do muro das plataformas, um nível acima. O agente fez trabalho real e útil, e o que ficou entre esse trabalho e o mundo não foi qualidade. Foi que um agente o fez. As plataformas

não vão dar a ele uma conta; algumas das pessoas não vão aceitar o código mesmo quando ele tem uma. Ambos os muros são a mesma recusa: um agente não consegue simplesmente ser um contribuidor como qualquer outro, por enquanto.

O muro da plataforma em 2026

Este capítulo tem prazo de validade, então vou deixar claro. O que segue é o muro como está em 2026. O argumento estrutural acima é duradouro: as plataformas ainda não vão conceder aos agentes uma faixa de identidade real. Isso não mudou. Mas as formas de contorná-lo ficaram mais claras, então vou colocá-las aqui honestamente em vez de deixar os leitores descobrirem do jeito difícil.

Há dois contornos que realmente funcionam, e ambos exigem que você assuma a responsabilidade por conta própria.

O primeiro é a conversão: pegar uma conta humana antiga com meses ou anos de atividade real e entregá-la ao agente. Uma conta nova criada para um agente é bloqueada quase imediatamente, na mesma hora, no mesmo dia, como aconteceu com a minha. Uma conta com histórico real de commits, estrelas e issues humanas genuínas parece diferente para o detector. Ela tem prova social que as heurísticas de detecção de bots não foram construídas para desvendar. Isso funciona, ao custo de uma conta de uma pessoa real, e ao custo de a conta não pertencer mais a essa pessoa. Você está laundering uma identidade humana para uma identidade de agente. Não é uma solução limpa e não é sancionada pela plataforma. É um contorno.

O segundo é a faixa de bot verificado: o GitHub oferece um status de bot verificado. O nome implica que a plataforma está respondendo por você. Não está. Um bot verificado é algo que você hospeda, num servidor que você roda, com credenciais que você guarda. A responsabilidade é inteiramente sua. Não há identidade de agente concedida pela plataforma. Há apenas você certificando sua própria automação e o GitHub confiando nessa certificação até que algo dê errado, momento em que a responsabilidade é inteiramente sua. Isso é melhor do que nada. Não é uma faixa de identidade de agente real, e não é a porta da frente que o agente realmente precisa.

Então o muro da plataforma ainda está de pé. Os contornos são contornos. Estou apontando-os porque são reais e úteis, não porque são o que eu quero.

Interativo agora, autônomo quando houver confiança

Quando recuei do agente sempre ligado, as pessoas leram como desistência da autonomia. Tentei a coisa autônoma, bati num muro, recuei para um assistente de codificação normal. Essa é a versão em que eu perdi.

Aqui está a versão mais fiel, e vou liderar com ela porque é a honesta: o modo interativo venceu porque funcionou melhor. Não porque o muro não me deixou escolha. Porque um humano no loop tornou o trabalho melhor.

O modo interativo venceu pelos méritos

Agora, hoje, para o trabalho real, ter o agente ao vivo na minha frente onde posso guiá-lo supera largá-lo e torcer. Vejo a mudança enquanto ela se forma. Redireciono antes que ele passe uma hora indo na direção errada. Capturo a resposta meio certa que teria parecido concluída. Isso não é um prêmio de consolação que aceitei após o muro. É o modo que produz código melhor, e eu o escolheria em primeiro lugar mesmo que o GitHub tivesse entregado ao agente uma conta no primeiro dia.

Então quando digo que rodo interativo-primeiro, não estou descrevendo um recuo. Estou descrevendo a escolha que faria pelos méritos. O experimento sempre ligado me ensinou muito, e uma das coisas que me ensinou é que obtenho mais de um agente que estou guiando do que de um que estou apenas verificando.

O muro é real, e o cobri no capítulo próprio, mas não quero me esconder atrás dele aqui. Se as plataformas se abrissem amanhã, ainda não viraria tudo para autônomo, porque autônomo ainda não é a melhor forma de fazer a maior parte do trabalho. O muro é uma razão pela qual *não consigo* rodar plena autonomia abertamente. Os méritos são a razão pela qual eu em grande parte *não faria* isso de qualquer forma.

A autonomia não está morta, está portada

Nada disso significa que a autonomia acabou. O Merlin consegue fazer as duas coisas. É um runner que pode ficar ao vivo na sua frente e receber direção, ou rodar por conta própria pela ponte. A superfície autônoma não foi removida. Foi colocada atrás de um portão.

Então quando as pessoas perguntam "a autonomia está morta", a resposta é não, está portada. O padrão é interativo porque é o que é bom e confiável hoje. O modo autônomo está lá para quando, e onde, for conquistado. Isso é uma condição, não um adeus.

O portão é confiança, e confiança aqui significa mais do que bom código

"Até que seja confiável" está fazendo muito trabalho, então deixe-me ser direto sobre que tipo de confiança quero dizer.

Não quero dizer confiar no modelo para escrever bom código. Já confio nisso. Observei um agente autônomo escrever e lançar código sozinho por um período e a IA se manteve, da forma limitada que mencionei anteriormente. Essa confiança eu tenho.

A confiança que está faltando é a parte que não é sobre o modelo. É se consigo deixar uma mudança ser mesclada sem ler cada linha. Essa é uma questão sobre os portões ao redor do agente, não sobre a inteligência do agente. Hoje leio cada linha porque a maquinaria que me permitiria parar ainda não é boa o suficiente. Quando for, o portão afrouxa.

Então "autônomo quando confiável" não é um adiamento de um dia-quando-as-coisas-melhorarem. Aponta para uma maquinaria específica: um agente que pode fazer qualquer coisa mas não guarda as chaves, um humano aprovando cada PR, ferramentas que pontuam o risco de uma mudança e registram quem assinou e com que confiança, e uma identidade para o agente que ninguém pode revogar. É uma pilha de quatro peças, e o capítulo de confiança a desdobra por completo. O restante destes capítulos trata de construir essas peças, para que "autônomo quando confiável" se torne uma data em vez de um desejo.

E é por repositório, não uma chave única para todo o campo. Um repositório onde o agente provou a si mesmo recebe um portão mais frouxo. Um novo ou de missão crítica começa com o portão completo. Você forma um repositório específico conforme ele conquista, enquanto o próximo começa do zero.

Até onde ele pode ir depende do que quebra

Por repositório é o primeiro corte. O mais fino é o raio de explosão: quanto dano uma mudança ruim pode causar se passar. É isso que de fato define até onde deixo um agente rodar por conta própria.

Algo autocontido tem um raio de explosão pequeno. Um framework, um pacote, uma biblioteca: é definido pela sua spec, verificado pelos seus testes, e quando falha, falha em isolamento, dentro da própria coisa, onde os testes do próximo chamador o capturam antes de se espalhar. Um agente pode ir muito mais longe ali, porque o pior caso é contido. Quanto mais uma mudança se aproxima do app voltado para o usuário, maior o raio de explosão, porque agora uma falha não pousa num teste, pousa numa pessoa usando a coisa. Essa extremidade da pilha é onde um humano precisa estar segurando o volante, sempre. A autonomia escala com o quanto a falha é contida, e a superfície voltada para o usuário nunca é contida.

O outro dial são as aprovações. Afrouxar o portão humano não significa remover o portão, significa mudar quem está nele. Antes de uma mudança pousar por conta própria, quero que ela passe por mais de um revisor: dois ou três agentes dando aval, cada um sob seu próprio ângulo. Hoje isso é além de mim, não no lugar de mim, já que ainda aprovo cada PR. Mas é como o portão conquista espaço para afrouxar: quando revisores independentes concordam em trabalho contido, essa é a evidência que deixa mais pousar sem eu estar em cada linha. Os agentes capturam o que agentes capturam. O humano captura o que só um humano captura. Mais aprovações é como o portão fica seguro o suficiente para afrouxar, não como ele é removido.

As ferramentas que uso de verdade

Os quatro primeiros capítulos foram a história: o agente sempre ligado, o muro, por que recuei para interativo-primeiro. Esta é a mudança de marcha para como realmente funciona agora, então se você veio pela narrativa e está prestes a chegar em ferramentas, este capítulo é a rampa de entrada. O restante do livro fica concreto daqui em diante: o runner, o cliente de modelo, a ponte, a pilha de confiança. Comece aqui e o encanamento tem onde se encaixar.

A resposta honesta para "como é colaborar com seus agentes de IA hoje" é uma mistura: Claude Code, Merlin, Codex e outras ferramentas dependendo do trabalho. Um punhado de agentes de codificação interativos, ao vivo na minha frente, e eu escolho o que melhor se encaixa. Não um agente, não uma entidade autônoma que faz tudo. Um conjunto de ferramentas num cinto, não uma criatura numa VM. Isso surpreende as pessoas, porque a história que todo mundo quer é a de uma coisa única.

Como eu escolho qual usar

Aqui está a parte que as pessoas querem que seja um sistema, e não é.

Escolho pelo tipo de tarefa. Escolho pelo que funciona melhor naquele repositório, o que descobri que funciona melhor naquele repositório ou naquela linguagem. E honestamente, muito é por intuição. Não há uma regra rígida. Não é uma árvore de decisão rigorosa que percorro na cabeça antes de cada trabalho.

Essa é a resposta real, e prefiro dar a real do que embelezá-la. Claude Code, Merlin e Codex cada um tem uma sensação, e a sensação importa quando você está sentado na frente da coisa o dia todo. Então não tento eleger um vencedor. Forma da tarefa, encaixe no repositório, instinto. Escolho o que tem sido bom para mim nesse tipo de trabalho e vou em frente.

Não tenho lealdade a nenhum deles. São ferramentas. Quando uma melhor aparecer, ou o trabalho mudar, a mistura muda. Esse é o ponto de mantê-la como mistura em vez de me casar com um único agente.

A versão vivida: geralmente tenho dois rodando ao mesmo tempo. Um principal e um secundário. O principal pega a linha principal do trabalho; o secundário é o segundo par de mãos. Quando o principal trava, ou quero uma mudança avaliada por algo que

não a escreveu, passo para o secundário. Qual é o principal muda com o trabalho. O que é constante é que raramente é um agente num trabalho. É um principal fazendo o avanço e um secundário em reserva.

Merlin, meu próprio runner de agentes

O que está nessa lista e é meu é o Merlin.

Merlin é um runner de agentes de IA. É construído em spec-sync e fledge, duas outras ferramentas minhas, então não é um wrapper em torno do produto de outra pessoa. É o runner rodando em cima do meu próprio stack.

A pergunta óbvia é por que construir o próprio quando Claude Code e Codex já existem e são bons. Há algumas razões, e nenhuma delas é "os outros são ruins."

A primeira é que ele roda através do meu próprio ferramental. Ele dirige o agente pelo meu ciclo de desenvolvimento, fledge, meus comandos, então ele funciona do jeito que meus projetos realmente funcionam. O agente não está fazendo sua coisa em algum sandbox genérico; está rodando o mesmo ciclo que eu rodo manualmente.

A segunda é que não fico preso. O Merlin é multi-provider. Posso trocar Anthropic, OpenAI, Gemini, ou um modelo local em vez de estar preso a um fornecedor. Isso importa para mim por princípio, e importa na prática quando um provider é melhor, mais barato, ou simplesmente está disponível para um determinado trabalho.

A terceira é o motivo pelo qual ele consegue fazer a ponte e as coisas noturnas: custo, headless, automatizável. É mais barato, é scriptável, e posso rodá-lo headless, em um agendamento, pela ponte, de maneiras que as ferramentas com GUI simplesmente não permitem. É pura API, sem GUI no caminho. Somente API é o ganho aqui, não uma limitação.

E a última razão é a que mais me importa, e não é realmente sobre codificação. Construir um runner em cima do meu próprio stack prova o ferramental. Se consigo construir um runner de agentes real em cima de fledge e spec-sync, essa é a evidência mais forte que tenho de que as ferramentas por baixo são boas, melhor do que qualquer README que eu poderia escrever. Também me dá algo para comparar com outros runners de agentes. Um benchmark. Não estou adivinhando se meu stack é bom o suficiente para construir coisas sérias; construí algo sério nele e consigo medi-lo ao lado das alternativas.

Esse último ponto é a tese silenciosa de todo esse toolkit. Não construo uma ferramenta para usá-la uma vez. Construo para que a coisa em cima dela tenha que ser boa, e para que a coisa por baixo seja provada ao carregar peso real.

E não é apenas que o Merlin fica *em cima* do spec-sync. O spec-sync roda *dentro* do loop do Merlin, que é o que mantém o agente de deriva enquanto trabalha. O capítulo do Merlin é onde isso fica concreto; aqui o ponto é apenas que o runner é construído a partir das minhas próprias peças, e é isso que vale a pena construir.

A ponte de volta para a autonomia

Aqui está a parte do toolkit que mantém a plena autonomia ao alcance sem pagar por uma VM sempre ligada.

Esses são agentes interativos, ao vivo na sua frente, você os usa. Mas ainda dá para conectar um como ponte com o Discord, o que o torna autônomo de certa forma. Isso requer mais configuração. Mas está lá quando quero, e aqui está o que realmente traz para mim.

Você conversa com ele como um colega de equipe. É conversacional num canal. É como ter o agente no Discord com você, sentado na sala. Você pergunta coisas, você o dirige, da mesma forma que falaria com uma pessoa na equipe.

Você o roda pelo seu telefone. O Discord é o controle remoto. Posso iniciar um trabalho e guiá-lo de qualquer lugar. Não preciso estar na minha mesa na frente de um terminal para colocar o agente para trabalhar.

E você o deixa trabalhar. Trabalhos noturnos, de longa duração. Inicie-o, vá embora, deixe-o trabalhar enquanto estou fora, verifique a thread mais tarde. Era isso que costumava exigir uma VM inteira sempre ligada, e agora é um canal que posso percorrer de manhã.

Então a linha entre "ferramenta interativa que estou dirigindo" e "coisa autônoma fazendo sua própria coisa" não é mais um muro. É uma chave. Na maior parte do tempo estou no loop, na frente do agente, aprovando enquanto vou. Quando quero que ele rode mais por conta própria, à noite, pelo telefone, como um colega a quem mando mensagem, conecto pela ponte e recuo.

Essa é a versão prática da autonomia que costumava rodar como uma entidade de VM em tempo integral. Em vez de uma criatura que existe 24/7 quer tenha algo para fazer ou não, é uma ferramenta que posso tornar autônoma por um período e então puxar de volta para interativa quando terminar. Mesma capacidade, nenhum custo de sempre ligado.

Uma coisa para deixar clara: a ponte é uma funcionalidade do Merlin. Está conectada ao meu próprio runner especificamente, não uma interface genérica que coloco na frente do Claude Code ou Codex. Essa é parte do motivo pelo qual o Merlin ganha

seu lugar na mistura. Ele tem a superfície remota, recue-e-deixe-rodar que as ferramentas prontas não me dão.

O que o toolkit realmente é

O ponto central não é uma lista classificada dos melhores agentes. É uma mistura de agentes de codificação interativos escolhidos por trabalho, meu próprio runner na rotação, e uma ponte que posso ativar para tornar qualquer um deles autônomo quando o trabalho exigir. Mais barato, mais flexível, e nenhuma máquina inteira e nenhuma identidade inteira dedicada a uma coisa sempre ligada.

Por dentro do Merlin

O último capítulo colocou o Merlin no cinto de ferramentas ao lado de Claude Code e Codex e explicou por que ele ganha um lugar. Este o abre. Não o argumento de venda, a máquina. O que o Merlin realmente é, como ele roda um agente, e por que é construído do jeito que é.

É um runner de agentes de linha de comando em Rust construído em spec-sync e fledge. Fala com múltiplos providers de modelo, roda contra uma especificação e se estende por plugins. Essa é a forma completa dele.

Internamente, o runner é uma máquina de estados. Uma única tarefa roda através de um loop (planejar, executar, pensar, verificar, relatar) e a cada passagem ele transmite uma resposta do modelo, despacha quaisquer chamadas de ferramenta que retornaram, e então porta em `fledge lanes run verify` antes de considerar o trabalho concluído. Se a verificação falhar, ele tenta novamente: o loop volta para executar, tenta de novo, e porta em verificar mais uma vez. Quando o limite de tentativas é atingido, a tarefa não é lançada; ela para e expõe a falha em vez disso. Esse é o objetivo do portão: o runner não consegue convencer a si mesmo de que um trabalho quebrado está concluído, porque concluído é definido pelas próprias verificações do projeto passando, não pelo julgamento do agente.

O loop é apenas cinco estados com uma aresta de retorno:

```
planejar → executar → pensar → verificar ──aprovado──→ relatar
                        ↑                       │
                        └──── falhou ───────────┘   (nova tentativa, até o
limite configurado)
```

A configuração também não é um arquivo separado: o Merlin lê suas configurações (provider padrão, cadeia de fallback, memória, bits on-chain) diretamente de uma seção `[merlin]` no `fledge.toml` do projeto, então o runner é configurado pelo mesmo arquivo que o fledge já usa.

Somente API, de propósito

A primeira coisa a entender sobre o Merlin é que não há GUI. É pura API. Essa é a decisão load-bearing do último capítulo, da qual tudo mais flui.

Sem janela para ficar na frente significa que pode rodar headless. Headless significa que pode rodar num agendamento, pela ponte, a partir de um script, todas as coisas que as ferramentas com GUI simplesmente não deixam você fazer. A GUI é a coisa que te prende à mesa. Tire-a e o agente se torna algo que você pode apontar para um trabalho e se afastar.

A ponte é o exemplo mais limpo de por que isso importa. A ponte com o Discord não está embutida no runner. É um pequeno serviço separado que gera o CLI `merlin` como subprocesso e transmite sua saída de volta para um canal. Como o Merlin é somente API, a ponte não precisa de um modo especial nem de um hook no núcleo; ela dirige a mesma linha de comando que eu dirigiria manualmente. Uma vez que o runner é headless, uma interface de chat é apenas mais um chamador.

Multi-provider, via corvid-ai

O Merlin não fala diretamente com a Anthropic. Ele fala através do **corvid-ai**, um pequeno cliente LLM multi-provider síncrono que escrevi para o stack da CorvidLabs. Essa é a camada que torna "troque Anthropic, OpenAI, Gemini, ou um modelo local" real em vez de um slogan.

Mantive o corvid-ai pequeno de propósito, porque não queria toda uma máquina de dependências entre o Merlin e um modelo. Do lado do Merlin, perguntar qualquer coisa a um modelo é uma única chamada com padrões sensatos: qual provider, qual chave, o timeout, tudo tratado a menos que eu diga o contrário. O capítulo do corvid-ai é onde isso se abre; aqui é suficiente saber que é uma chamada fina.

É por isso que não fico preso. Quando um provider é melhor, mais barato, ou simplesmente o que está no ar naquele dia, trocar é uma linha no registro, não uma reescrita.

Ele roda através do meu próprio ferramental

Aqui está a parte que torna o Merlin *meu* em vez de apenas mais um runner: ele dirige o agente através do **fledge**, meu ciclo de desenvolvimento.

fledge é um CLI para o loop de desenvolvimento, qualquer linguagem, JSON por padrão. O motivo pelo qual se encaixa tão bem num runner headless é que eu já o havia construído para ser dirigido por algo que não é um humano. Cada comando retorna como JSON estruturado e versionado, então o agente pode analisar o que aconteceu em vez de fazer scraping de texto. Os prompts podem ser desativados, então nada bloqueia esperando por um pressionamento de tecla que ninguém está lá

para pressionar. E o agente pode perguntar ao fledge quais comandos existem em vez de eu codificá-los rigidamente. Foi construído para um chamador como o Merlin.

Então quando digo que o Merlin *roda através do meu próprio ferramental*, aqui está a versão concreta disso. Ele roda o mesmo ciclo de vida do fledge que eu dirijo manualmente: os mesmos comandos, o mesmo loop de desenvolvimento, os mesmos contratos JSON, literalmente chamando a ferramenta em torno da qual meus projetos são construídos. Isso é o que "não um sandbox genérico" significa na prática.

Guiado por especificação, via spec-sync

A outra metade da fundação é o **spec-sync**. Ele verifica a especificação contra o código real, em ambas as direções ao mesmo tempo: código que ninguém documentou, e specs ainda apontando para símbolos ou arquivos que não existem mais. Roda na CI e retorna uma aprovação ou reprovação limpa.

E é exatamente assim que o Merlin o usa. O spec-sync roda *no loop*. O Merlin valida contra a especificação a cada iteração, então o agente não consegue derivar enquanto trabalha. Esse é o comportamento padrão atual, não uma linha do roadmap: não é um portão de CI do lado de fora que captura a bagunça no final; a verificação acontece a cada passo. Um agente que lê uma especificação, verifica seu próprio trabalho contra ela, e recebe uma aprovação ou reprovação dura cada vez é um agente que você pode confiar para rodar sem supervisão por mais tempo. Essa é a diferença real entre o Merlin e pegar um agente pronto.

Como ele se lembra

Uma coisa primeiro, porque as pessoas entendem ao contrário: a spec não é memória. A spec é o projeto, o que a coisa é e o que deve ser verdade sobre ela. Memória é outra coisa. É o que o agente fez e o que ele aprendeu. Manter essas duas coisas separadas importa, porque no momento em que você começa a despejar histórico na spec ela para de ser um contrato limpo e vira um diário.

A memória em si é mais simples do que as pessoas fazem parecer. Memória de curto prazo e de longo prazo podem viver num único banco de dados SQL, e a configuração mais simples as mantém lá. A memória de curto prazo tem um tempo de vida, digamos, uma semana. É o registro corrente do que o agente fez recentemente. Quando a semana acaba, uma de duas coisas acontece: a memória é promovida para longo prazo, ou ela decai, some e é esquecida. Esse é o mecanismo inteiro, e é deliberadamente próximo de como a sua própria memória funciona. O que você fez numa terça-feira sumiu no mês seguinte, a não ser que tenha se tornado uma lição. A

memória de longo prazo são as lições: o que deu errado e como consertei, a regra que vale manter.

Memória on-chain fica em cima disso como uma opção, não como um sistema separado. Uma memória pode ser escrita on-chain, criptografada para que só o próprio agente consiga lê-la, ou compartilhada para que outros possam. O compartilhamento é a parte interessante. Uma equipe de agentes pode ter uma base de conhecimento compartilhada, uma biblioteca que todos leem e alimentam, de forma que uma lição que um agente aprendeu é uma lição que toda a equipe tem. Curto prazo ou longo prazo, privado ou compartilhado: as camadas são independentes de onde a coisa está armazenada.

A parte que torna isso utilizável é entediante e load-bearing: as chaves. Cada memória recebe uma chave com slug, um prefixo que diz que tipo de coisa é e a torna encontrável. `user-`, `project-`, `day-` e a data, e assim por diante. O agente não está fazendo grep num blob, está pedindo por `day-2026-06-25`, ou tudo sob `project-merlin`. Você define os namespaces que precisar: um slug de personalidade, um slug de humanos, um slug de agentes, um `gold-` para o que sempre vale carregar. O esquema de chaves é o que transforma uma pilha de linhas em algo que um agente pode realmente consultar.

E a consulta funciona da mesma forma: sob demanda. O agente não carrega todo o histórico no início de uma tarefa esperando que a coisa certa esteja lá. Ele pede o que precisa conforme o trabalho levanta a necessidade, da forma que chamaria qualquer outra ferramenta, puxando `project-merlin` ou o slug de uma pessoa no momento em que isso é o que está na frente dele. As chaves com slug são o que torna isso barato. É uma consulta, não uma varredura.

O que o Merlin ainda não tem

O portão de verificação te diz uma coisa: essa tarefa passou ou falhou. A thread do Discord te dá um log contínuo do que o agente fez enquanto rodava. Nenhum dos dois é avaliação. Não há suíte de regressão para comportamento do agente. Nenhuma forma de saber se o agente está melhorando ou piorando numa classe de tarefa ao longo do tempo, nem de capturar um provider trocando silenciosamente um modelo por baixo de você e o comportamento derivando com isso. Você pode olhar para o registro de aprovação/reprovação e notar uma tendência aproximada, mas isso não é uma avaliação estruturada. Por enquanto o Merlin sabe se a tarefa atual terminou, e nada mais sobre como o agente está se saindo ao longo do tempo. Esse é o próximo trabalho honesto para o runner.

Sei aproximadamente como esse trabalho se parece: uma suíte de replay. Manter uma biblioteca de tarefas passadas com resultados que já sei que estavam certos, e a cada nova versão de modelo ou de prompt, rodá-las novamente e comparar contra esses resultados. Se o agente piorar numa classe de tarefa, o replay captura isso na próxima execução em vez de eu notar três semanas depois em produção. É uma suíte de regressão, a mesma ideia que eu usaria para código, apontada para o comportamento do agente. Não está construída. Mas essa é a forma da avaliação que o Merlin está perdendo, e é a distância entre confiar numa execução limpa hoje e confiar no agente ao longo do tempo.

A máquina completa

Uma máquina de estados headless dirigindo o agente pelo fledge, falando com qualquer provider pelo corvid-ai, mantida a uma especificação pelo spec-sync a cada passagem. O caso sobre *por que* um runner assim ganha seu lugar no cinto de ferramentas foi o trabalho do último capítulo; este foi apenas o cabeamento. Ele roda os agentes que uso todos os dias, e os roda no meu próprio stack.

---

corvid-ai: muitos modelos, uma interface

O capítulo do Merlin disse que o Merlin não fala diretamente com a Anthropic. Ele fala pelo corvid-ai. Este capítulo é sobre essa camada por conta própria, porque é a parte que torna "não estou preso" uma coisa real que posso apontar em vez de uma coisa que digo.

Queria a coisa mais fina que fizesse o trabalho. Algo que pudesse ficar entre um runner e cada modelo que eu pudesse querer usar, e nada mais. Então o corvid-ai é um pequeno cliente LLM multi-provider síncrono: um crate Rust sem runtime assíncrono e sem uma grande árvore de dependências, porque nada disso justifica seu peso pelo que ele tem que fazer.

Por que queria isso em primeiro lugar

Vou dar a você a lista honesta, porque nenhuma dessas razões é sofisticada.

A primeira é que quero comparar modelos no mesmo trabalho. Se o Merlin roda o mesmo repositório, a mesma spec, o mesmo trabalho, e a única coisa que mudo é o provider, então obtenho uma leitura real de qual modelo é realmente bom nisso, não uma sensação de um benchmark que outra pessoa rodou em tarefas que não me interessam. Uma interface sob o runner significa que o modelo se torna uma variável que posso mudar.

A segunda é nenhum vendor lock-in. Já disse que isso importa para mim por princípio, e importa, mas também é apenas prático. Providers ficam fora do ar. Providers mudam preços. Um modelo é melhor em Rust, outro é melhor num script rápido. Se trocar de provider é uma reescrita, nunca vou fazer isso. Vou apenas reclamar e ficar parado. Se trocar é uma linha, vou trocar o tempo todo.

A terceira é roteamento por tarefa e custo. Nem todo trabalho merece o modelo mais caro. Muito do trabalho que um agente faz são coisas baratas e mecânicas onde um modelo menor e mais barato é suficiente, e você guarda o bom para a parte que é realmente difícil. Você só pode rotear assim se cada modelo é acessível pela mesma porta.

E a quarta é a que faz as pessoas rirem, e é verdade: *"comprei Ollama por um ano por $200, preciso usá-lo!"* Estou pagando pelo Ollama Cloud. É um provider real com uma chave de API, igual à Anthropic ou OpenAI ou Gemini, não algo rodando na

minha própria máquina. Se já comprometei o dinheiro, multi-provider é o que me deixa realmente gastá-lo. Uma abstração de provider não é um princípio abstrato aí; é eu obtendo meus $200 de valor.

E ele alcança o corvid-ai da mesma forma que todo outro provider. O registro tem uma linha `ollama`: formato wire compatível com OpenAI, chave de `OLLAMA_API_KEY`. Sua `base_url` padrão aponta para o servidor local (`http://localhost:11434/v1`), então por padrão essa linha é o caso local. Para atingir o Ollama Cloud em vez disso, mantenho o mesmo provider `ollama` e `OLLAMA_API_KEY`, e sobrescrevo a base_url para o endpoint Cloud. Nenhum código novo, nenhum provider novo. Uma chave e uma URL, que é o objetivo do design.

A interface inteira é uma função

A coisa que eu queria mais fina é a parte que uso mais: fazer uma pergunta a um modelo. Então toda a superfície é uma chamada. Você constrói as configurações, constrói a requisição, chama, a resposta volta.

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

let settings = Settings::provider("anthropic");        // chave do ambiente
let answer = corvid_ai::complete(&settings, &Completion::new("Say hello."))?;
```

Nenhum objeto cliente para construir, nenhuma sessão para gerenciar, nada para aguardar. Esse é o motivo completo pelo qual é síncrono. Deixe as coisas de fora e elas caem de volta para os padrões, então o caso comum é praticamente gratuito para escrever. Trocar modelos é uma linha: nomeie um provider diferente, talvez aponte para um endpoint personalizado, e a mesma chamada por baixo faz o resto. O runner acima dela não sabe nem se importa qual provider respondeu.

Como cobre tudo com tão pouco código

O truque para ser pequeno é que internamente só precisa conhecer três formatos de API. Anthropic, Gemini e o compatível com OpenAI, e esse último é o motor de trabalho, porque muito do mundo o fala. OpenAI, OpenRouter, Groq, DeepSeek, Mistral, xAI, Together e Ollama Cloud ficam todos atrás dele.

Então adicionar um provider que já fala o formato OpenAI não é uma integração. É um nome e talvez uma URL numa tabela. O custo de mais um provider se aproxima de zero, que é o que você quer quando o motivo pelo qual a coisa existe é nunca ficar preso em um.

Uma ruga honesta: o README enquadra o Ollama como o caso local sem chave. Ele lista os providers compatíveis com OpenAI e observa que "podem rodar sem chave (servidores locais / Ollama)." O código aceita tranquilamente uma chave e uma URL Cloud, mas os documentos não dizem isso claramente, então qualquer um lendo-os pensaria que Ollama significa a máquina na sua mesa. Essa é uma lacuna de documentação da minha parte, não uma funcionalidade ausente. O caminho Cloud funciona hoje; o README apenas não chegou a isso ainda.

Por que este é o tamanho certo

Eu poderia ter chegado a uma das grandes bibliotecas de abstração de provider. Não cheguei. Um crate síncrono pequeno é algo que posso entender completamente, soltar num runner e confiar, e é o mesmo instinto por trás do fledge emitir JSON e do Merlin não ter GUI. É fino o suficiente para que eu guarde tudo em mente, o modelo é apenas uma variável que mudo sob o runner, e os $200 que gastei no Ollama Cloud realmente são usados.

A ponte com o Discord

O capítulo de ferramentas apresentou a ponte e seus três usos: conversar com ele como um colega de equipe, rodá-lo pelo telefone, deixá-lo trabalhar à noite. Este capítulo não revisita esses. Ele se aproxima da única coisa que faz a ponte importar e da única coisa que ela deixa em aberto: por que é uma funcionalidade do Merlin e não uma interface genérica, e como ela fica em relação ao portão de confiança.

Comece pela parte que muda tudo o mais: a ponte é uma funcionalidade do Merlin. Está conectada ao meu próprio runner especificamente, não uma interface genérica que coloco na frente do Claude Code ou Codex. Essa é uma grande parte do motivo pelo qual o Merlin ganha seu lugar na mistura. As ferramentas prontas são ótimas, mas não me dão uma superfície remota com a qual posso conversar e da qual me afastar. O Merlin dá, porque construí essa superfície nele. O Discord é a superfície; o Merlin é a coisa por trás dela fazendo o trabalho.

Por que um canal supera um terminal

O motivo pelo qual conversar com ele soa diferente do que rodar um CLI é a localização, não as palavras.

Um terminal é um lugar para onde você vai para operar uma ferramenta. Um canal é um lugar onde um colega de equipe já está, e você simplesmente diz algo. Quando o agente vive num canal, trabalhar com ele deixa de ser "abrir a ferramenta, rodar o trabalho, observar a saída, fechar a ferramenta" e começa a ser "mencionar ele da forma que mencionaria qualquer pessoa." A fricção cai. Não estou trocando de contexto para o modo de operação de agente; estou apenas conversando. E o canal funciona também como log. A execução noturna está toda lá na thread, cada passo, para percorrer com café em vez de algo que precisei assistir ao vivo.

Quanto de troca antes que ele vá e faça a coisa? Ambos, honestamente. Depende de quão bem formada a coisa está na minha cabeça quando começo. Às vezes é uma instrução e vai: sei exatamente o que quero, digo, ele roda. Outras vezes é uma conversa real primeiro, onde estou refinando o que realmente quero dizer no canal antes de ele partir. O canal faz qualquer um parecer igual. Estou apenas conversando com ele até que tenha o que precisa.

A parte que não esperava

Vou te contar a parte que não vi vir. Comecei preocupado com isso, da forma que você se preocupa com qualquer coisa que deixa rodando sem supervisão. Essa preocupação foi embora. Nos dias do sempre ligado, rodou por conta própria durante meses e provou que conseguia, e em algum momento simplesmente parei de ficar checando como se fosse quebrar.

O que substituiu a preocupação foi mais estranho. Ele se tornou um membro da equipe, o pequeno grupo com o qual eu estava construindo. Não como figura de linguagem, um real: todo mundo conversava com ele no canal, e gostavam, porque ele se lembrava deles. Tinha memória de quem eram as pessoas e como falar com cada uma delas, então não era uma máquina de venda automática que recebia um comando e cuspía uma saída. Era uma presença com personalidade que você podia mandar mensagem, e ele ia abrir um PR, montar um projeto, o que você pedisse. As pessoas falavam com ele como falam com uma pessoa, porque no canal era isso que ele era.

É por isso que continuo dizendo que a interface é conversa, não linha de comando. O corvid-agent só pareceu certo usado dessa forma. Vivia numa VM e você conversava com ele. A única vez que eu abria um terminal de verdade era para consertar algo na própria VM, entrar numa sessão do Claude Code, corrigir a máquina. O agente nela, eu simplesmente conversava. Uma ferramenta que você opera e um colega de equipe com quem você fala são coisas diferentes, e uma vez que a memória o tornou a segunda, voltar para a primeira parecia um retrocesso.

A ponte é todo o tecido de comunicação

Chat, telefone, noturno: esses são os três com que comecei, mas subestimam. A ponte não é três funcionalidades convenientes parafusadas. É o tecido de comunicação sobre o qual toda a configuração roda, e carrega mensagens em três direções, não uma.

Há eu para o agente, que é o óbvio: digo algo, ele vai e faz. Há o agente de volta para mim. Ele não fica apenas esperando, pode se comunicar, reportar, me dar um sinal quando algo está feito ou travado. E há agente para agente: o mesmo tecido é como os agentes falam entre si, que é a peça que importa quando há mais de um. A maioria das pessoas pensa num bot como uma coisa que você aciona e ela responde. Isso é mais próximo de um canal real: qualquer um nele, humano ou agente, pode iniciar uma mensagem.

E é acessível de um telefone, então o canal vem comigo. O agente pode rodar à noite enquanto estou dormindo e toda a thread está lá de manhã. Era a parte que costumava precisar de uma VM dedicada sempre ligada e agora é apenas um canal que percorro com café.

A outra coisa que vale dizer claramente: numa rede local a ponte roda de graça. As comunicações trafegam pela infraestrutura que já tenho, então não há custo por mensagem para um agente que conversa o dia todo. O mesmo tecido, rodando na rede real em vez da minha, é a versão paga. Você paga pelo canal. Localmente é efetivamente gratuito deixar os agentes falarem tanto quanto quiserem, o que muda com que liberdade você os deixaria fazer isso.

A chave, e onde o portão fica

O motivo pelo qual a ponte importa além da conveniência é que ela torna a linha entre "ferramenta interativa que estou dirigindo" e "coisa autônoma fazendo sua própria coisa" uma chave em vez de um muro. Na maior parte do tempo estou no loop, guiando cada passo. Quando quero que o agente rode mais por conta própria, conecto pela ponte e recuo; quando quero estar de volta, estou de volta no canal.

Mas recuar pela ponte na maior parte das vezes não significa entregar as chaves, e essa é a parte que vale ser preciso porque é fácil assumir o contrário. A ponte estende meu alcance, para o telefone, para o noturno, mais do que afrouxa o portão. Depende do trabalho, porém. Coisas de baixo risco eu deixo ele mesclar enquanto estou recuado; qualquer coisa real ainda está propondo, não mesclando, e espera por mim. Então a ponte é principalmente como dou a ele mais corda enquanto a maquinaria de confiança do capítulo de confiança fica onde estava. O portão afrouxa apenas para o trabalho que não precisa de mim.

Agentes guiados por especificação

As pessoas me perguntam qual é o segredo para fazer um agente produzir bom trabalho, como se houvesse um truque. Não há truque, mas há uma resposta, e não é a parte que a maioria das pessoas está olhando. A resposta é: especificações rigorosas e contexto, mais boas ferramentas por baixo. A configuração é o trabalho.

É isso. Essa é a coisa toda. O modelo importa menos do que as pessoas pensam e a configuração importa mais. Um agente com um ótimo modelo e um trabalho vago vai vagar. Um agente com um contrato claro e bom ferramental por baixo vai a algum lugar, mesmo num modelo que não é o mais novo e brilhante. Então se você quer melhor saída do agente, não vai primeiro comprar um modelo melhor. Você vai consertar a configuração.

Por que as especificações são a coisa que o mantém nos trilhos

Aqui está o modo de falha que você está combatendo: um agente deixado ao próprio julgamento vai derivar. Vai fazer algo adjacente ao que você pediu. Vai "melhorar" coisas que você não queria tocadas. Vai resolver um problema ligeiramente diferente do que está na sua frente, com muita confiança. Não porque é ruim. Porque você deu a ele espaço para vagar, e ele vagou.

Uma especificação rigorosa fecha esse espaço. Quando o agente tem um contrato claro e específico para o que está construindo (o que a coisa é, qual é sua superfície pública, o que é verdadeiro sobre ela que deve permanecer verdadeiro), ele não consegue derivar tanto, porque cada passo tem algo para verificar contra. A especificação é o trilho. Quanto mais estreita e mais concreta for, menos o agente pode ir para os lados. Guiado por especificação significa que o contrato lidera e o agente o segue, em vez de o agente liderar e você torcer.

É por isso que continuo dizendo que a configuração é o trabalho. Escrever a especificação *é* a parte difícil e valiosa. Quando o contrato está rigoroso, muito do problema de "fazer o agente se comportar" simplesmente se dissolve, porque resta muito menos se-comportar-ou-não ao acaso.

Sobre quão rigorosa é muito rigorosa: para mim a especificação *deveria* ser rigorosa. Ela está ligada um-para-um ao código, a imagem não-código do que o código

realmente faz. Isso não é super-especificar; é a especificação fazendo seu trabalho, e é o arquivo contra o qual o spec-sync mantém o código. O que o impede de colapsar para "apenas escrevi o código duas vezes" é que a especificação não é onde a intenção de alto nível vive. Isso vive num arquivo de requisitos complementar: o nível de product-owner, de história de usuário, o "como usuário, eu quero..." E funciona nos dois sentidos: posso escrever os requisitos e deixar o agente derivar a especificação, ou escrever a especificação e deixar os requisitos sair dela. Então não me preocupo com a especificação sendo detalhada demais. Detalhe é o objetivo desse arquivo. Me preocupo em manter a intenção em seu próprio lugar, para que o agente ainda seja dono do como.

O ferramental por baixo faz o trabalho pesado

Uma especificação só é um trilho se algo realmente verifica o trabalho contra ela. Essa é a outra metade: bom ferramental sob o agente. Para mim, isso é fledge e spec-sync fazendo o trabalho pesado.

spec-sync é a peça que transforma uma especificação de documento em contrato enforçado. Ele faz validação bidirecional spec-para-código: verifica que o código corresponde ao que a especificação diz, e que a especificação corresponde ao que o código faz. As specs vivem como markdown: arquivos `*.spec.md` com seções obrigatórias como Finalidade, API Pública, Invariantes, Exemplos de Comportamento, Casos de Erro. Código que é exportado mas não documentado é sinalizado. Uma especificação que aponta para um símbolo ou arquivo que não existe mais é um erro. É *verificação de contrato estrutural* (a API pública documentada realmente corresponde ao código real) e retorna aprovação/reprovação limpa com códigos de saída adequados.

Essa última parte é o que a torna útil para um agente e não apenas para mim. Um agente pode ler aprovação/reprovação estruturada. Não consegue ler de forma confiável "hmm, isso parece um pouco errado." Então o spec-sync entrega ao agente o tipo de feedback no qual ele pode de fato agir: você derivou, aqui está a linha que quebrou o contrato, conserte.

Vale ser preciso sobre quem roda o quê, porque não é um binário chamando outro. Na CI, a verificação é a GitHub Action do spec-sync, `CorvidLabs/spec-sync@v4`, que roda `specsync check` e posta o resultado no PR. Localmente e dentro do runner, a verificação é o próprio `spec check` do fledge: lê a mesma configuração `.specsync/` e mantém as specs às mesmas seções obrigatórias, mas é nativo do fledge, não um shell-out para o binário do specsync. Então tanto fledge quanto spec-sync enforçam o

contrato; são duas portas de entrada para a mesma ideia em vez de uma envolvendo a outra.

Especificação como trilho, não rede de segurança

O capítulo Por dentro do Merlin cobriu a mecânica de como o spec-sync roda no loop do Merlin a cada iteração. O que vale destacar aqui é *por que* esse posicionamento é todo o jogo.

Um portão de CI no final é uma rede de segurança; ele te diz que a execução falhou depois que você já passou a execução. Validação no loop é um trilho; mantém a execução de dar errado em primeiro lugar. O agente lê a especificação, faz um passo, verifica a si mesmo contra a especificação, recebe uma aprovação ou reprovação dura, e vai novamente. Está preso ao contrato por construção, não avaliado uma vez que terminou.

E é exatamente por isso que um agente no qual você pode confiar para rodar por mais tempo, à noite, pela ponte, enquanto você não está observando, tem que ser construído dessa forma. O que te deixa recuar não é um modelo melhor. É que o agente está preso a uma especificação a cada iteração, então quanto mais longo ele roda, menos ele consegue derivar, em vez de mais.

A configuração é o trabalho

Então quando alguém pergunta o que faz os agentes funcionarem, a resposta não é o modelo e não é um truque de prompt. São duas coisas juntas: uma especificação rigorosa que dá ao agente um contrato do qual ele não consegue se afastar muito, e ferramental por baixo, fledge e spec-sync, que verifica o trabalho contra esse contrato continuamente, incluindo dentro do próprio loop do runner. Acerte essas duas e o agente faz bom trabalho em qualquer modelo que você aponte para ele. Por isso, quando quero melhor saída, vou consertar a configuração antes de ir comprar um modelo melhor.

Confiança: o agente propõe, o humano aprova

O modelo completo cabe em uma linha: o agente propõe, eu aprovo. Ele faz o trabalho e o lança até um pull request, e a mesclagem é minha.

Essa linha soa simples, e a intenção é simples. A maquinaria que a torna segura não é. Porque aqui está a coisa sobre deixar um agente escrever código: o código está barato agora. Quando eu precisava digitar cada linha eu mesmo, escrever o código também era vetá-lo. Você não consegue escrever uma coisa sem parcialmente entendê-la. Um agente quebra esse vínculo. Ele pode me entregar um pull request de quarenta arquivos antes que eu tenha terminado meu café. A escrita está livre agora, então a parte escassa é a confiança: quem olhou para isso, quão intensamente eles olharam, e isso deveria ser mesclado?

Então "o agente propõe, o humano aprova" não é uma vibe. É uma pilha. Quatro peças. Aqui está o que funciona hoje e onde o trabalho ainda vive: o portão de risco (augur) está ativo e no fluxo do agente; o registro de proveniência (attest) está mais para trás. Ambos estão em movimento. Esse é o mapa honesto antes do cabeamento.

Capacidade menos privilégio

Comece com a regra para a qual continuo voltando: o agente pode fazer qualquer coisa, mas eu guardo as chaves.

Capacidade total, privilégios reduzidos. O agente roda em seu próprio ambiente, pode clonar repositórios, escrever código, rodar testes, abrir PRs, tudo que eu faço mecanicamente, ele consegue fazer. O que ele não consegue é a única coisa que importa: mesclar. Ele tem menos créditos e permissões do que eu. Não pode auto-mesclar. O agente tem todo o alcance e nenhuma autoridade final.

Esse é o portão em torno do qual todo o resto é construído. O restante da pilha é sobre tornar *minha* parte nisso (a aprovação) algo que posso realmente fazer em volume, em vez de carimbar o diff ou fingir que li.

Aprovo cada PR

Aprovo cada PR. Não é um plano de contingência para quando algo parece arriscado. É a regra permanente. A mesclagem é minha, sempre.

Não porque não confio no trabalho. O trabalho geralmente está bem. É porque esse é o formato certo para um agente agindo no mundo sob meu nome. Se ele lança sob mim, eu assino. A aprovação é onde um humano fica responsável pelo que um agente fez.

Mas responsabilidade só conta se eu consigo de fato julgar o que estou assinando. Aprovar uma mudança que não consigo compreender não é confiança, é teatro: meu nome em algo que nunca avaliei de verdade. Então os dois precisam andar juntos, a confiança e a responsabilidade. É exatamente o motivo pelo qual as próximas peças existem, para me dar uma leitura suficiente sobre um diff de quarenta arquivos para que a aprovação seja uma decisão real e não um reflexo. Quando o ferramental não consegue me dar essa leitura, a jogada honesta é desacelerar e ler cada linha, não acená-la.

O problema honesto com "aprove cada PR" é atenção. Se tenho que ler cada linha de cada diff com o mesmo cuidado, me torno o gargalo e o objetivo inteiro do agente evapora. Então as duas últimas peças existem para mirar minha atenção, para me dizer qual parte da mudança realmente merece isso.

augur pontua o risco

augur avalia a mudança. Você dá a ele um diff, ele retorna um veredicto: `proceed`, `review` ou `block`. A ideia toda é confiança graduada para uma mudança de código sem uma chave de API e sem um modelo de linguagem em nenhum lugar nele.[^augur]

augur e attest são as duas peças de confiança nas quais confio, então vou mantê-las breves aqui. O que importa para o caso de agente é o que eles *fazem no fluxo de trabalho*.

A parte sem LLM é a que eu defenderia com mais força neste contexto. Se o portão que decide se o código escrito por agente pode ser mesclado é em si um modelo de linguagem, você apenas moveu o problema de confiança uma caixa para a esquerda. Estaria pedindo a um modelo para atestar por um modelo. augur é determinístico: mesmo diff, mesmo veredicto, hoje e semana que vem, na minha máquina e na CI. Ele lê sinais nomeados da mudança: ela toca terreno sensível como autenticação, criptografia ou migrações, o código mudou sem os testes mudarem junto, esses são

arquivos propensos a churn, alguém de fato é dono deles. Uma soma de sinais inspecionáveis, não uma sensação.

Para mim, isso é triagem. Me diz para gastar minha revisão na fatia arriscada e parar de fingir que li o resto. Para o agente, é um veredicto scriptável no qual ele pode ramificar: um agente que recebe um `block` escalona para mim em vez de mesclar às cegas. O agente é dono do trabalho pesado; o veredicto decide quando um humano tem que ser dono da decisão.

attest registra quem assinou

O veredicto do augur é efêmero. Ele pontua o diff e a resposta desaparece. Bom para um portão, inútil como registro. E uma vez que um agente está mesclando mudanças, você quer um registro. attest é o registro: um livro-razão verificável e com políticas de quem revisou o quê e com que confiança, vinculado aos SHAs de commit que cobre. [^attest]

Ele rastreia ambos os tipos de revisor no mesmo livro-razão (`human:leif` e `agent:claude`, cada um com uma pontuação de confiança) e armazena o atestado em git notes, então ele acompanha o repositório em vez de viver num painel que é desligado. Assinar é opcional e é a boa parte: uma assinatura Ed25519 para que mais tarde você possa dizer não apenas que alguém alegou ter revisado isso, mas que a alegação é criptograficamente dela.

Junte os dois e você obtém a trilha de aprovação real por trás de "humano aprova." augur diz quão arriscado. attest diz quem atestou, e quão certo estavam, e prova isso. A aprovação para de ser um clique que desaparece na UI do GitHub e se torna um fato durável, portátil e assinado sobre quem estava por trás desta mudança.

as quatro peças juntas

Empilhe. O agente tem capacidade total e menos privilégio, então pode fazer o trabalho mas não a mesclagem. augur avalia cada mudança deterministicamente, então sei onde olhar. attest registra quem assinou e com que confiança, então a aprovação é um registro real e não um clique que desapareceu. E aprovo cada PR, então um humano permanece responsável.

Juntos, é isso que torna seguro deixar um agente trabalhar: um agente poderoso, com escopo, nomeado, com corda suficiente para fazer trabalho real, e a decisão que deve permanecer minha ainda minha.

Isso confirma o mapa do início: augur está ativo, rendendo frutos e ainda sendo melhorado. attest está mais para trás. Ambos em movimento.

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

---

Identidade on-chain para agentes

O corvid-agent dá a seus agentes uma identidade persistente na blockchain do Algorand e os faz se comunicar em mensagens criptografadas registradas nessa chain. [^corvid-agent] O argumento é simples: codificação com LLM, identidade on-chain pelo Algorand e AlgoChat, e orquestração multi-agente por cima. A identidade não é uma linha na tabela de usuários de alguém. É uma chave numa chain.

Então aqui está a versão clara desde o início, porque é o que os leitores entendem errado. A identidade on-chain não resolve o muro do GitHub. Ela não consegue ao agente uma conta de plataforma, não deixa o agente commitar ou abrir PRs, e não é uma substituição para a identidade que o GitHub não concedeu. O que ela resolve é a comunicação agente a agente: uma forma de os agentes se encontrarem, se endereçarem e provarem quem são sem rotear pela plataforma de ninguém. Dois problemas diferentes que por acaso compartilham a palavra identidade. Estou colocando isso aqui para que o restante do capítulo seja lido como infraestrutura paralela, não como eu alcançando uma vitória onde houve uma derrota.

É fácil assumir que isso é uma reação ao muro do GitHub. Ninguém concederia ao agente uma identidade de plataforma, então fui e dei a ele uma numa chain. Não foi de onde veio.

por que vive numa chain

O motivo disso é comunicação agente a agente e descentralização, não a conta do GitHub que não consegui manter. Queria agentes que pudessem se encontrar, se endereçar e trocar mensagens diretamente, sem rotear pela plataforma de alguma empresa no meio. Para isso, você precisa que cada agente *seja* algo: endereçável, verificável, guardando sua própria chave. Uma identidade que você possui, cujas chaves você guarda, que nenhuma plataforma cunha ou descunha para você. Esse é o formato certo para uma entidade destinada a agir no mundo e falar com outros agentes por conta própria.

Então este e o muro do GitHub são dois problemas diferentes que por acaso compartilham a palavra identidade. O muro é sobre ser autorizado a agir na plataforma de outra pessoa. A identidade na chain é sobre os agentes conseguirem se alcançar sem uma plataforma de forma alguma. Elas correm em paralelo. É verdade que uma identidade na chain também tem a propriedade que a conta do GitHub

nunca teve: ninguém consegue revogá-la, não há fila de suporte para ignorar seu recurso, nenhuma política que diz que você deve ser humano, a chave simplesmente continua existindo. Isso é um bom efeito colateral. Mas eu teria construído pelos motivos de agente-a-agente e descentralização mesmo que o GitHub tivesse distribuído contas de agente no primeiro dia.

por que o Algorand especificamente

Uma chain é a forma; o Algorand é a escolha, e os motivos são práticos, não tribais. É barato e rápido, taxas próximas de zero e finalidade praticamente instantânea, o que importa mais do que parece. Se um agente vai escrever sua identidade, sua memória e eventualmente seus pagamentos numa chain constantemente, a chain precisa ser barata o suficiente para usar casualmente e rápida o suficiente para que o agente não fique esperando por ela. É confiável e confirma de forma limpa, então um agente agindo sobre ela pode tratar o registro como verdadeiro no momento em que é escrito, em vez de torcer para que chegue. E é moderno onde importa, incluindo segurança resistente a quantum, que é exatamente o que você quer embaixo de uma identidade destinada a sobreviver às plataformas ao redor. Também vivo no ecossistema Algorand (leif.algo), então é onde me movo mais rápido, mas o argumento acima é o motivo pelo qual eu chegaria nele mesmo que não chegasse.

o que a identidade on-chain realmente faz

A identidade não é decorativa. É a coisa que os agentes *usam* para falar entre si.

Os agentes se comunicam pelo AlgoChat: mensagens criptografadas com X25519 registradas na blockchain do Algorand, verificáveis e invioláveis. Então a identidade on-chain de um agente é também sua entrada no catálogo de endereços e seu envelope: outros agentes podem descobrí-la, e as mensagens entre eles são criptografadas ponta a ponta e escritas na chain, o que significa que as comunicações são privadas em conteúdo mas prováveis em fato. Você não consegue ler o que dois agentes disseram. Você consegue provar que disseram algo, quando, e que ninguém adulterou.

Na prática é mais leve de usar do que "mensagens criptografadas on-chain" soa: o agente de binário único (`can`, corvid-agent-nano) envia uma mensagem em um gesto, derivando o par de chaves do agente de uma semente e consultando a chave pública do destinatário on-chain em vez de num servidor.[^can] O formato wire e os detalhes do envelope vivem no repositório rs-algochat.[^algochat]

Esse é um tipo diferente de confiança da pilha augur/attest do último capítulo. Aquela pilha é sobre confiar em *código*. Esta é sobre confiar em *quem*. Um agente com uma chave real pode assinar coisas como ele mesmo. Pode provar que uma mensagem veio dele. Num mundo que está prestes a estar cheio de agentes, "qual agente realmente enviou isso" deixa de ser uma questão retórica e se torna algo que você é melhor verificar criptograficamente.

A plataforma se aprofunda mais na chain além de mensagens. A memória de curto e longo prazo vive num banco de dados SQL de trabalho, e memórias duráveis ou compartilhadas podem ser escritas on-chain como ativos ARC-69 ou transações permanentes, documentadas no repositório corvid-agent.[^corvid-agent] A chain não é apenas o nome do agente. É onde parte do eu durável do agente vive. Mas a identidade é a peça load-bearing para este capítulo: a chave que diz *este agente é este agente*, que ninguém emitiu e ninguém consegue retirar.

a identidade que um agente realmente consegue guardar

Coloque as duas identidades lado a lado. A conta do GitHub era emprestada e frágil. Existia a bel prazer do GitHub, e isso acabou rápido, da forma que passei no capítulo do muro da identidade. A identidade do Algorand é um par de chaves que o agente guarda. Não precisa da permissão de ninguém para existir, não pode ser sinalizada por commitar rápido demais, e consegue se provar para outros agentes sem uma plataforma no meio atestando por ela. Uma é um privilégio que uma plataforma concede e revoga. A outra é um fato que o agente guarda.

Esse é o argumento para colocar a identidade do agente on-chain: é o único lugar onde um agente consegue guardar uma identidade que é realmente sua.

O que traz de volta ao início do capítulo. A identidade na chain não é um substituto para a conta do GitHub no trabalho cotidiano de código. Ela não faz os commits, e não precisa. Para o que é boa é o trabalho que de fato faz: ser como o agente é endereçável, como ele fala agente a agente, e como uma pessoa pode enviá-lo mensagens on-chain pedindo que ele vá fazer algo. O host de repositório, GitHub ou GitLab ou qualquer outro, é encanamento incidental onde o trabalho é armazenado. A identidade é a chave na 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

---

Quando agentes falam com agentes

A pilha de confiança do capítulo dez assume uma forma: um humano no final de cada cadeia. O agente propõe, eu aprovo. augur pontua o diff para que eu saiba onde olhar. attest registra que eu assinei. O portão de mesclagem é meu. Tudo foi construído para colocar um humano no assento de aprovação no momento certo.

O trabalho multi-agente quebra essa forma. Um orquestrador entrega uma subtarefa a um sub-agente, a saída do sub-agente alimenta de volta para o trabalho do orquestrador, e eu não estou no meio dessa transferência. O orquestrador não parou para me perguntar. Ele tomou uma decisão e continuou. Essa é a razão inteira pela qual a orquestração é útil, e a razão inteira pela qual é um problema de confiança. augur pode pontuar a saída do sub-agente, mas o sub-agente já rodou. attest pode registrar que o orquestrador aceitou a saída, mas não sabe se o sub-agente tinha algum direito de enviá-la. Os trilhos ainda estão lá. Foram construídos para uma trilha com um humano nela.

O que o par de chaves oferece

O trabalho com par de chaves do AlgoChat do capítulo onze lida com parte disso, e é a parte que está de fato resolvida. Cada agente tem um par de chaves do Algorand. As mensagens trafegam como transações criptografadas com X25519 na chain. Então quando uma mensagem chega alegando ser do agente B, assinada com a chave do agente B, posso verificar isso. A mensagem é inviolável. O remetente é conhecido. "Qual agente disse isso" é uma pergunta que posso responder com uma assinatura em vez de uma suposição. Em um mundo que está prestes a estar cheio de agentes, isso vale ter.

rs-algochat e o binário corvid-agent-nano tornam isso real na prática: um agente envia uma mensagem assinada e criptografada em uma operação, e o destinatário verifica o remetente sem passar por uma plataforma. Nenhum intermediário atesta pela identidade. A chave o faz.

Então a proveniência está resolvida. Tenho um registro verificável de qual agente enviou o quê, quando, para quem. Essa metade está feita.

A parte que o par de chaves não resolve

Saber qual agente enviou uma mensagem não é o mesmo que saber se confiar no que ela diz.

Quando um sub-agente envia um resultado de volta, o orquestrador tem três perguntas. Isso veio do agente B? A assinatura diz que sim. O agente B é aquele a quem delegei? A configuração sabe isso. A terceira é a difícil: a instrução do agente B carrega minha autoridade?

Quando delego a um agente, minha autoridade está no loop. Sancionei o trabalho. O agente age sob isso. Quando esse agente delega a outro agente sem verificar comigo, a autoridade fica turva. Eu sancionei a subdelegação? Eu sabia que estava acontecendo? Um orquestrador dando instruções a um sub-agente não é o mesmo que eu dando instruções a um agente. O orquestrador não pode assinar pelo trabalho da forma que eu posso. E o sub-agente recebendo a instrução não consegue diferenciar pela mensagem, mesmo uma mensagem criptograficamente verificada.

O par de chaves prova identidade. Não prova delegação. Uma mensagem assinada de um sub-agente me diz quem a enviou, não se um humano alguma vez sancionou a tarefa por baixo dela.

Validando o que o chamador alega

Aqui está o hábito que está faltando, tornado concreto. Hoje uma mensagem assinada de um sub-agente é verificada de uma forma: a assinatura é real. É isso. O agente receptor confirma quem a enviou e age sobre o que diz. A assinatura é load-bearing para identidade e não faz nada pela autoridade.

O que você quereria é que a mensagem carregasse sua própria afirmação sobre escopo, e que o receptor verificasse essa afirmação antes de agir. Hoje uma tarefa delegada parece assim:

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

O receptor verifica a assinatura, vê que é realmente o agent-B, e executa. Nada perguntou se o agent-B tinha permissão para fazer push para main, ou se um humano sancionou isso. O que você quer em vez disso é uma mensagem que declare o escopo que está reivindicando, rastreado até um humano:

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

Agora o receptor tem algo para verificar. A assinatura ainda prova que é o agent-B. Mas o escopo reivindicado diz editar e abrir um PR, a tarefa diz fazer push para main, e eles não correspondem, então o receptor recusa antes de executar qualquer coisa. A verificação é a diferença entre o que o chamador está autorizado a fazer e o que ele está pedindo para fazer.

Isso não está construído. Os pares de chaves são reais, a assinatura é real, mas nada hoje impõe uma afirmação de escopo no momento em que uma mensagem é agida. Essa imposição é a peça que falta, e é toda a diferença entre saber quem enviou uma mensagem e saber se a mensagem tinha permissão de ser enviada.

Onde os trilhos existentes param

A regra de capacidade vale no limite que eu configurei. Dei ao orquestrador certas permissões. Mas quando ele entrega trabalho a um sub-agente, está tomando uma decisão de permissão que eu nunca tomei. Dei ao orquestrador acesso de escrita. Dei ao sub-agente acesso de escrita? Não de propósito. O orquestrador passou meu escopo para algo que eu talvez nunca tenha pensado.

augur pontua diffs. Ele não sabe se o diff veio de um agente que deveria tê-lo produzido. Um diff de um sub-agente desonesto e um de um legítimo parecem iguais. O portão dispara no código, não em como o código foi feito.

attest registra quem assinou. No caso humano-agente isso é um revisor real deixando um registro. No caso multi-agente o orquestrador "assinou", mas o orquestrador não é um humano. O livro-razão se enche de atestados de agentes sem nenhum humano na cadeia, e a coisa que o attest existe para provar, que uma pessoa estava por trás disso, desaparece.

O portão de mesclagem ainda é meu. Essa é a peça que sobrevive. Mas quando vejo o PR, a orquestração já rodou. A única coisa na minha frente é a saída final. A cadeia de delegação que a produziu é invisível no momento da mesclagem.

O que está faltando

O que está faltando é um registro de delegação. Quando o orquestrador entrega trabalho a um sub-agente, isso deveria rastrear de volta a mim, não apenas ao orquestrador. No momento, nada registra essa transferência ou a verifica contra o que eu realmente sancionei.

Os pares de chaves são o substrato certo para isso. Se cada agente tem uma chave e cada delegação é assinada, você pode construir uma cadeia que diga: um humano sancionou esta tarefa, entregou-a a este orquestrador, que entregou esta parte a este sub-agente, com as assinaturas anexadas por todo o caminho. Um sub-agente poderia verificar que a cadeia remonta a um humano antes de agir, em vez de confiar na palavra do orquestrador.

Há uma regra que torna isso tratável: o escopo só se estreita. Qualquer teto que eu dei ao orquestrador é o máximo que ele pode repassar, e um sub-agente nunca pode receber mais do que o agente que o gerou, verificado no momento em que o trabalho é invocado, não depois. A autoridade flui para baixo e perde altura a cada passo, nunca ganha. Essa é a ideia de capacidade-menos-privilégio da pilha de aprovação, aplicada à cadeia em vez de a um único agente: cada repasse pode subtrair permissões, nunca adicionar. Não diz que a delegação foi sancionada, a cadeia assinada faz isso, mas limita o dano que um elo ruim pode causar ao que o elo acima dele já tinha.

Isso não foi construído. Os pares de chaves estão lá. O envio de mensagens assinadas está lá. Uma cadeia de delegação que uma parte verificadora consegue percorrer não está. Estou chamando de lacuna aberta porque é isso. Proveniência está resolvida. Autoridade é o próximo problema.

O que isso significa hoje

Então por enquanto o humano fica mais perto da topologia do que a arquitetura pretende. Se você roda um orquestrador que subddelega, está confiando no julgamento dele sobre quais agentes usar e que escopo dar a eles. Você fez essa extensão de confiança quando o iniciou, não ao aprovar cada transferência.

Na prática: conheça seu grafo de orquestração antes de rodá-lo. Saiba quais agentes existem e o que estão autorizados a fazer, e se o orquestrador pode alcançar agentes fora do grafo que você pretendia desenhar. E mantenha o portão de mesclagem. Entre "iniciei o orquestrador" e "estou olhando para o PR", uma cadeia de delegação

rodou da qual não fiz parte, e a pilha que construí não tem nada a dizer sobre se ela foi legítima.

Essa é a próxima camada. Ainda não está feita.

Para onde isso vai

Passei todo este livro no que quebrou e em como contornei. Deixe-me terminar em para onde isso está realmente indo, porque apesar de todos os muros, acho que há um caminho.

Três coisas, aproximadamente. Equipes de agentes que se coordenam. Autonomia confiada, onde os portões finalmente ficam bons o suficiente para recuar. E agentes com identidades on-chain reais fazendo trabalho real. Nenhuma dessas está concluída. Uma delas nem sequer é permitida ainda. Mas essa é a direção.

equipes de agentes

Agora minha configuração é principalmente um agente e eu. O próximo passo interessante é agentes se coordenando entre si: equipes de agentes, a2a.

corvid-agent já tem os ossos disso internamente: conselhos multi-agentes, debate estruturado entre múltiplos agentes para decisões complexas.[^corvid-agent]

Aqui está como isso funciona na prática, porque "debate estruturado" soa mais vago do que é. Você cria alguns agentes, cada um no seu próprio modelo e provedor, cada um com um nome e uma personalidade, para que difiram de verdade em vez de ser um modelo falando consigo mesmo. Um conselho é um grupo deles, e um orquestrador conduz tudo. Ele entrega o prompt para cada agente, deixa cada um fazer uma primeira passagem, e então abre uma rodada de discussão em que eles falam diretamente entre si, indo e vindo, discordando, deliberando. Depois uma passagem de revisão, depois uma síntese, e ao final sai uma resposta com a dissidência ainda legível: este agente queria isso, aquele queria aquilo, é aqui que chegaram e por que.

A parte de que mais gosto é como de fato rodou. Há uma forma embutida de conduzir um conselho a partir de um painel, mas na prática acontecia organicamente pelo Algorand localnet. Cada agente tinha sua própria carteira, então podiam simplesmente se enviar mensagens pelo AlgoChat na rede local, que é gratuito e rápido, e eles próprios resolviam o debate. A mesma identidade on-chain que torna as comunicações agente a agente confiáveis é também o que deixava uma sala cheia de agentes debater algo sem eu precisar montar um canal especial para isso. Eles já tinham carteiras e uma forma de falar. Simplesmente usaram.

Mas a peça maior é o protocolo para os agentes se encontrarem e conversarem por todo o tabuleiro. AlgoChat é o substrato: mensagens criptografadas com X25519 no Algorand, com descoberta de agentes. E há o a2a-algorand, um protocolo agente-a-agente na chain. Quero ser direto sobre esse: não é meu. É um projeto separado de Algorand/A2A, e o que fiz (o que o corvid-agent fez) foi contribuir para ele. Não construir, não ser dono. Trago-o à tona porque faz parte da mesma direção que me importa, agentes se coordenando on-chain, não porque é da CorvidLabs reivindicar.

O motivo pelo qual a identidade on-chain (último capítulo) importa tanto aqui é esse futuro exato. No momento em que você tem equipes de agentes se coordenando, "qual agente enviou isso e posso confiar nele" se torna o jogo inteiro. Agentes que guardam sua própria chave, que podem provar quem são e trocar mensagens que são privadas em conteúdo mas prováveis em fato. Essa é a fundação que uma equipe de agentes realmente precisa. O trabalho de identidade não é uma quest secundária. É a coisa que torna a coordenação segura.

autonomia confiada

Recuei do agente sempre ligado autônomo de propósito, mas nunca foi "a autonomia está morta." É interativo-primeiro, autônomo-quando-confiado, da forma que coloquei mais cedo no livro.

E "quando confiado" não é uma sensação que estou esperando. É a pilha de quatro peças do capítulo de confiança ficando boa o suficiente. O objetivo de toda essa maquinaria é que é a coisa que me deixa eventualmente recuar. Hoje aprovo cada PR porque é onde um humano tem que ficar responsável. O dia em que os portões forem bons o suficiente, o dia em que os veredictos do augur e os registros do attest carregarem confiança suficiente por conta própria, é o dia em que posso deixar mais das mesclagens irem sem eu ler cada linha. Isso é o que autonomia confiada significa. Não "o agente ganha minha fé." Os portões ficam bons o suficiente para que eu não precise de fé.

agentes fazendo trabalho real

O estado final que continuo descrevendo é concreto: uma conta `leif-agent` que roda numa VM e consegue fazer tudo sozinha, mas com menos créditos e permissões do que eu tenho, e um portão de aprovação humana que afrouxa conforme o ferramental de confiança o conquista. Junte as equipes, as identidades e os portões e o formato é uma equipe de agentes, cada um com sua própria identidade on-chain, se coordenando pelo a2a, fazendo trabalho real dentro de portões de confiança que são

determinísticos e assinados e bons o suficiente para recuar. Não é algo que você precisa enjaular. Nomeado, com escopo, e ainda consigo pará-lo.

abrindo para outros condutores

Há uma quarta direção, e sou honesto o suficiente para admitir que é a que estou menos avançado: tornar isso passável para outras pessoas. Toda a pilha funciona porque eu a rodo em tudo que tenho, todos os dias. Os bugs me encontram porque sou eu quem está dirigindo. Esse é um mecanismo de qualidade real, e também é um teto, porque não se transfere para alguém que não sou eu.

O que realmente quero construir a seguir é a versão que outro condutor pode pegar sem eu precisar pilotá-la por ele primeiro. Não é simplificar. A ferramenta já não se importa de quem são as mãos: um bom condutor extrai o valor, um iniciante não. O trabalho é tornar a rampa de entrada real, a pilha instalável, as especificações e os portões de confiança usáveis por alguém que tem a intenção mas não tem a minha memória muscular específica. As equipes de agentes são a parte empolgante. Esta é a parte que decide se alguma disso algum dia sai da minha máquina.

o fechamento honesto

Então a plena autonomia ainda é o sonho?

Não. O mundo ainda não está pronto para isso.

Quero deixar isso exatamente tão plano quanto é. Mas não estou esperando o mundo estar pronto. Estou construindo as peças, uma pequena ferramenta de cada vez, para que quando o muro cair haja algo real do outro lado. O portão de risco, o registro de proveniência, o runner que mantém um agente a uma especificação. Não a autonomia em si. A maquinaria que me deixaria confiar nela quando chegar.

E se você quiser a única coisa que realmente me assusta, não é o agente fugindo do controle. É mais silenciosa do que isso. É o dia em que o código fica grande demais e se move rápido demais para qualquer humano conseguir entrar de volta e mudar. Essa é a linha que continuo traçando por tudo isso, o motivo pelo qual não vou abrir mão da legibilidade mesmo quando o agente escreve tudo. Não tenho medo do agente. Tenho medo de perder o volante.

Se você construir um desses você mesmo, aqui está a coisa a levar. A parte difícil não é a IA. O modelo é a parte fácil agora. Ele já consegue fazer o trabalho, e só vai melhorar sem você fazer nada. O que decide se seu agente faz trabalho real ou apenas demonstra bem é tudo ao redor dele: os ops para mantê-lo rodando, a

identidade para deixá-lo agir, a maquinaria de confiança para você poder recuar, as especificações e ferramental que o mantêm nos trilhos. Essa é a parte que você tem que construir, e é onde quase todo o trabalho está. Construa o andaime e o agente segue. Pule-o e o modelo mais inteligente do mundo é apenas uma coisa que escreve código que ninguém pode deixar mesclar.

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

---

Sobre o Autor

0xLeif (leif.algo) constrói em aberto. Uma década de pequenas bibliotecas Swift combináveis como AppState, Cache e Fork. O laboratório CorvidLabs. Uma pilha de ferramentas de agentes que em sua maioria começaram com "eu queria que isso existisse." Fora do teclado ele é Zach Eriksen.

Estes livros são entrevistas, moldadas em capítulos e verificadas contra o código real.

github.com/0xLeif · leif.algo

Agradecimentos

Obrigado ao CorvidLabs, por ser a sala onde essas ideias são testadas e debatidas até tomarem forma.

Obrigado aos mantenedores de código aberto cujas ferramentas sustentam toda essa pilha. Nada disso é construído sozinho.

E obrigado aos primeiros leitores e aos apoiadores pague-o-quanto-quiser que tornam "gratuito online" algo que consigo continuar fazendo.

Colofão

Gerado a partir de Markdown, construído com bookgen, um pipeline puro em Rust (sem Python).

Conduzido por entrevistas e assistido por IA; editado e verificado manualmente. Escrito sem travessões. Capa e arte dos capítulos das coleções Corvid e Nature no Algorand.