Agentic Infrastructure

Giving an AI Agent the Keys Without Giving Away the Keys

Adam K. Clifford · Clifftech · A design brief generalized from a working system, with instance details removed.

What this brief is and isn't. This is the pattern and the reasoning, not a deployment guide for a specific environment. Agent names, hostnames, addresses, vault names, and anything tied to a specific tenant or topology are left out on purpose. The same discipline the system applies to its own credentials applies to what is published about it. The technologies are named where naming them is the point. The instance is not.

The Frontier With Teeth

The agentic AI conversation in 2026 points overwhelmingly at code. Every model launch is a coding demo, and the market for agents that write and ship software is crowded. That is the easy frontier, because a coding agent's mistakes mostly land in a pull request a human still reviews before anything runs.

Infrastructure is the frontier with teeth. An agent that operates real systems can restart the wrong service, push the wrong config, rotate the wrong credential, or open the wrong door. And the fact underneath it is blunt: you cannot fully trust what an agent will decide to do, because agents act on natural language and that language can arrive hidden inside the content they are asked to process. Prompt injection is not solved, and the consensus is that it will not be. So the real problem is not whether an agent can run infrastructure. It is whether an agent can run infrastructure such that a compromised agent still cannot cause harm. That constraint shapes the entire design.

Where the Market Actually Is

Two conversations are happening in 2026, and they are happening apart from each other. One is agentic SRE and self-healing operations: agents that detect, diagnose, and remediate incidents. That market mostly waves past the access question and assumes the agent already holds the credentials to act. The other is non-human identity and privileged access management, which one industry body has named the "non-human identity governance vacuum." That work mostly sells identity governance as a product to buy and waves past the operations question. The interesting gap is the seam between them. Almost nobody shows both halves solved together, as one architecture, by a builder who ran it rather than a vendor selling it. The industry agrees the problem is real and open. Working answers are the scarce thing.

The Core Idea: Decide at Use-Time, Keep Nothing Standing

Every weak version of agent access fails the same way. Authorization is decided once, baked into a token, a role, a group membership, or a long-lived secret, and after that it cannot be revoked on your timeline. A credential granted last week is still good the instant it is stolen. This system inverts that. Authorization for a privileged action is decided at the moment of the action, and requires a fresh, strong, human factor bound to that exact action. Nothing privileged sits waiting to be stolen. One rule governs the whole design: no standing, usable, privileged credential exists anywhere an agent, or the broker acting for it, can reach.

The Custom Broker: One Narrow Door

The agents never hold shell access, cloud admin credentials, or a domain account they can simply use. Every privileged action, from restarting a container to running an infrastructure-as-code deployment to reading a protected configuration, passes through a single broker. The broker authenticates the calling agent's identity token, checks the requested action against a tight allowlist of granted capabilities, rate-limits, and logs everything. There is no general "run anything." Each capability is one named, scoped verb, and the privileged work executes as an unprivileged, scoped service account, not as the agent and not as an administrator. The broker is hand-built because the approval binding below is the novel part and it should be auditable. The parts that are not novel, secret storage and rotation, use a proven tool instead.

Verified ID and the Two-Key System

A privileged release requires two keys turned together, and they are deliberately unequal. The first key is the operator's: the request is bound to one exact action, and releasing it requires a fresh approval presented from a hardware-backed credential using Microsoft Entra Verified ID, tied through a single-use value to this agent, this verb, this target, right now. A presentation for one request cannot be replayed against another. The second key is the agent's: it signs its request with a non-exportable key held in hardware, so the request cannot be forged or replayed, and the key cannot be stolen off the host even under full compromise. The most an attacker on that host can do is ask the key to sign while the host is live, which is exactly what the human approval is there to catch.

The reason the keys are unequal is the center of the design. A human approval carries an inherence factor, proof that a specific living person is present, through a biometric. A non-human identity can never have that. The closest a workload can hold is attestation, proof that the request came from genuine, unmodified code, which proves the code is authentic but not that its request is legitimate. A correctly attested, prompt-injected agent will produce a valid signature over an attacker's request. So the two keys are asymmetric on purpose. The agent's key provides independence, non-exportability, and binding to the action. The operator's key provides the one factor no compromise of the agent can reproduce. The security comes from the asymmetry, not from pretending a machine can carry a human's presence.

The Rotation Engine: Solved Problems Get Solved Tools

The credentials are not left standing and static. The account model is tiered, and the two most dangerous accounts are disabled at rest, enabled and rotated only inside an approved, time-boxed, monitored window, then disabled and rotated again when the window closes. Outside that window the account cannot authenticate at all, so it is inert rather than merely secret, and there is nothing to steal or move laterally with. The engine that performs rotation and checkout is HashiCorp Vault, self-hosted on dedicated hardware kept off the main compute cluster so it holds its own failure and compromise domain. Its directory secrets engine natively rotates, leases, checks out, and audits the account passwords, which keeps the sensitive rotation logic in a mature, audited tool rather than a hand-written script. It auto-unseals against a cloud key vault and snapshots to a verified backup chain. The rule is simple: build the part that is genuinely new, and use a proven tool for the part that is not.

Two Planes, One Model: On-Premises and the Cloud Tenant

The same authorization model governs two very different planes. On-premises, the privileged action is a credential release or a configuration change on real servers, network, and storage, gated by the broker. In the cloud, the same shape governs read and write to the management and directory APIs of a Microsoft 365 and Azure tenant: agents hold governed, directory-native identities under Microsoft's emerging agent-identity model, gated by certificate and policy for standing reads, with write access split into narrow, scoped grants routed through the same use-time approval instead of standing administrative roles. One model, two planes, whether the action touches a domain controller in a rack or a Conditional Access policy in the cloud.

What It Is For

None of this is security for its own sake. Governed, real access lets agents do useful infrastructure work that would otherwise sit on a human's plate:

Automate deployments. Every service ships from a git repository through an infrastructure-as-code pipeline, run through the broker, so a deployment is a governed, logged, reversible action rather than a hand-edit at midnight.

Prevent configuration drift. With git as the single source of truth and direct undeclared changes banned, an agent continuously compares what is declared against what is running and surfaces or corrects the difference before it becomes an outage.

Aid troubleshooting. An always-on agent with standing read access triages across the whole estate, correlates logs and state, and hands a human a real starting point. Read is low-risk and always available; anything that changes state escalates through the approval gate.

Manage both planes together. The same agents reason over on-premises infrastructure and the cloud tenant at once, which is where the leverage is, because most incidents and changes cross that boundary.

Assume the Compromise, Instrument the Effect

Because prompt injection cannot be reliably prevented, prevention is only half the design. Detecting the malicious instruction itself is a losing game, because it lives in the model's reasoning and never lands in a log. The durable signal is the effect, as a correlation rather than a signature: the release requires two authorizations, so a privileged action that happened without a matching approval is by definition out of band. That single correlation is the highest-fidelity compromise signal in the system, and it is safe to wire to automated response precisely because it is a correlation and not a guess. Around it sits the rest of a real program: native directory and cloud-platform threat detections for attacks that never make the privileged account itself act, credential-store access logs shipped where the agent host cannot edit them, and a scheduled attestation that deliberately attempts a forbidden action and confirms it is refused, because controls decay silently and a control that fails open is worse than none.

Decisions, Unedited

Architecture is a series of calls made under constraint. A running decision log keeps the reasoning attached to the decision. A few from this build.

ACCEPTED · CORE PRINCIPLE

Build the novel part, buy the solved part

Decision. Hand-build the broker and the use-time approval binding. Use HashiCorp Vault and a cloud key vault for storage and rotation. Do not write custom secret rotation.

Rationale. The approval binding is genuinely new and should be auditable. Secret rotation is solved by mature, audited tooling, and rolling a worse version only adds attack surface.

Consequence. Less custom security-critical code, not more. The one novel component gets full attention; the boring, dangerous parts are handled by specialists' tools.

ACCEPTED (REVISED)

Abandoning the human access model the moment the tenant proved it wrong

Decision. Grant cloud access as narrow, per-identity application permissions. Abandon the group, package, and just-in-time elevation scaffolding built for humans.

Rationale. Tested against a live tenant, the human pattern was structurally wrong for a non-human identity. Agents cannot join the groups it depends on, and the elevation tooling does not see them. Agents are shaped like applications; their runtime gate is a certificate and a policy, not a person's step-up.

Consequence. The read tier went live cleanly on the corrected model. The lesson worth more than the result: abandon a half-built design the moment the evidence says it is wrong.

ACCEPTED · DESIGN FLOOR

The human approval is the last line, on purpose

Decision. Do not pretend the agent's key can carry the security. Make the two keys asymmetric and let the human approval carry the factor a compromised host cannot reproduce.

Rationale. Attestation proves the code is genuine, not that its request is legitimate. The only factor an on-host compromise cannot forge is a live human presence bound to the specific action.

Consequence. A successful injection does not get the credential. It gets the agent to request one, and still needs a human to approve an action rendered in plain language. The residual risk is named honestly: approval fatigue, which the design fights by making the approval show exactly what it authorizes.

Why This Matters

The good part was never a specific vault or hostname. It is the posture, and postures outlast products. Authorize at use-time so a stolen grant is worthless. Keep the privileged identity inert until an approved, watched window. Make the two keys asymmetric. Build the novel part and buy the solved part. And design from the assumption that the agent will eventually be compromised, so the architecture, not the agent's good behavior, is what contains it. The industry named the non-human identity governance vacuum in 2026 without showing many working ways out of it, and it is building autonomous infrastructure agents in a separate conversation that assumes the access problem away. This is what it looks like when one person solves both halves together and keeps the decision log.