plural EVALUATION GUIDE | INFRASTRUCTURE AUTONOMY

Evaluating Infrastructure Autonomy

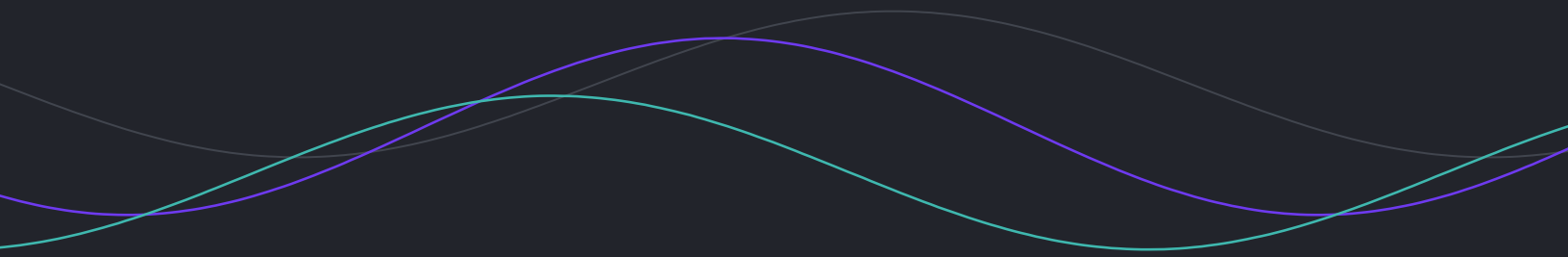
A framework for platform teams putting agents to work on their estate

WRITTEN FOR

CTOs, VPs and Heads of Platform Engineering, DevOps leads, and SRE leads.

LEARN MORE

plural.sh



What's inside

— How to use this guide 3

01 Why autonomy is now a platform decision 4

02 The infrastructure autonomy curve 5

03 What to evaluate 8

Six evaluation criteria 9

04 Questions to take into the room 12

05 Comparing approaches to infrastructure autonomy 14

06 How Plural approaches this 15

AT A GLANCE

5
stages of infrastructure autonomy

6
evaluation criteria that gate each stage

15
questions to take into vendor calls

How to use this guide

Most platform teams are being asked the same question this year, usually by someone from senior leadership in the organization: “what is our plan for AI in infrastructure?” The question is harder than it looks, because the market has converged on a single word, **autonomy**, and different vendors use it to mean very different things.

This guide gives you a way to compare different systems and tools. It sets out:



Five stages of infrastructure autonomy

So you can locate where your team operates today.



Six evaluation criteria

That determine which stage a system can actually reach.



A set of questions

To take into vendor calls.

The criteria are deliberately vendor-neutral. They apply equally to a commercial platform, an assembled open-source stack, and something your team builds itself.

SECTION 01

Why autonomy is now a platform decision

Two things changed over the last eighteen months, and together they are driving changes in how platform teams operate.

CHANGE 1 — VOLUME

The first is volume. Coding agents let one engineer ship what used to take a team. Applications arrive faster, and so does the infrastructure beneath them: more services, more clusters, more pipelines, more Terraform. Platform teams already operate at ratios of a handful of engineers to several hundred developers. That ratio worsens every quarter, and no hiring plan closes the gap.

CHANGE 2 — AGENTS IN THE ESTATE

The second is that agents have started arriving inside the estate itself. A microservice has to clear a defined path before it reaches production. It gets built through a known pipeline, scoped with an identity, authenticated, logged, and owned by someone. An agent clears none of that. An engineer runs one from a laptop holding a long-lived cloud credential, without any policy dictating what it can reach or who is allowed to invoke it.

> Agents will operate inside your estate either way. What you choose is the system they run through.

Agents will operate inside your estate either way. What you choose is the system they run through: where they execute, what credentials they hold, how their changes reach production, and what record exists afterwards. Make that choice deliberately and every agent uses one governed path. Leave it unmade and each team picks its own, so the answer becomes a laptop, a personal API key, a change nobody can reconstruct, and infrastructure operations outside your control.

That is a platform decision rather than a tooling preference, because whatever you pick becomes the path every future agent takes. Making it well means answering two things: **how far you want agents to take the operational work**, and **what has to be true about the system before you let them.** The rest of this guide works through both.

SECTION 02

The infrastructure autonomy curve

Autonomy is not binary. It runs across five stages, and one question defines each of them: **who does the work?** Everything else follows from the answer, including how much of a workflow completes on its own and what starts it.

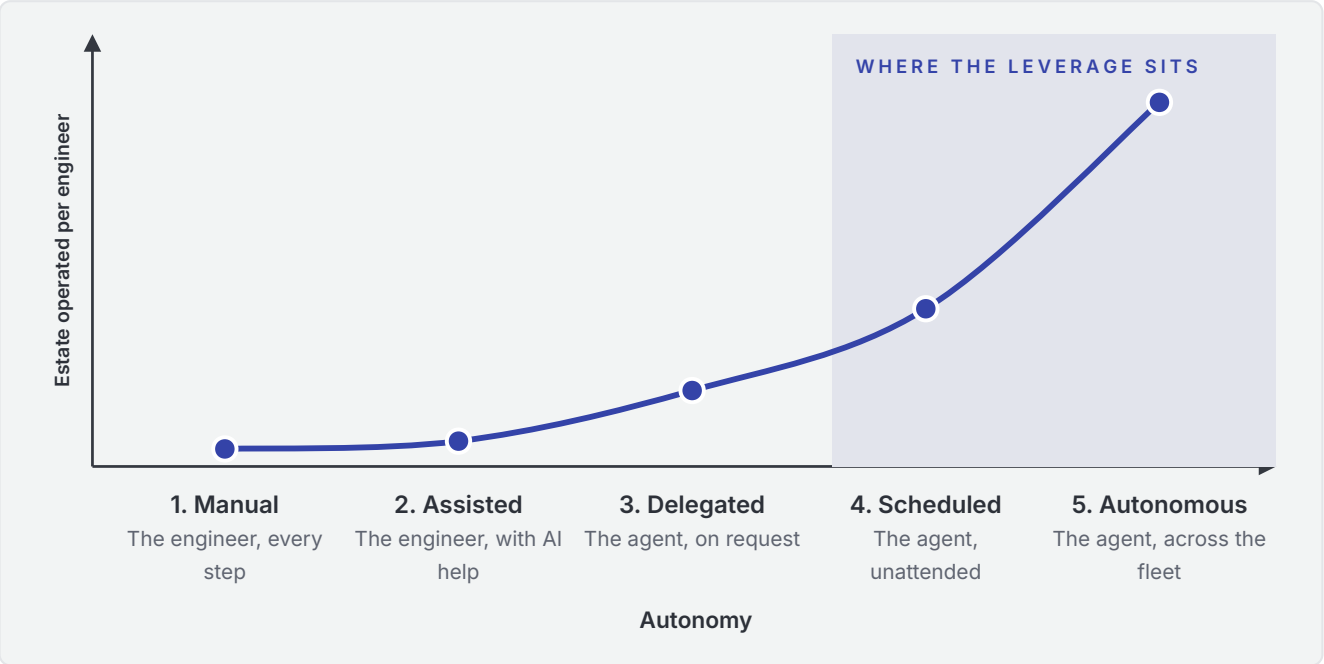


Figure 1: The infrastructure autonomy curve

Where the leverage sits

The curve plots the five stages against the estate one platform engineer can operate, counted in clusters, and it does not rise evenly. Moving from “Manual” to “Assisted” changes almost nothing, because the engineer still does the work with better autocomplete. “Delegated” is the first stage where a person stops producing the change themselves.

The curve then turns sharply between “Scheduled” and “Autonomous”. This is because up to Stage 3, throughput is bounded by how many tasks an engineer can personally start. From Stage 4 onward, work begins without them, and at Stage 5 it begins across the entire fleet at once. **Now a team of six can run the estate that used to need sixty.**

02 · THE INFRASTRUCTURE AUTONOMY CURVE

Five stages, one question: who does the work?

Stage	Who does the work	What that looks like
1. Manual	The engineer, at every step	The system shows state and applies what a human tells it. An engineer plans a cluster upgrade, checks add-on compatibility by hand, and applies it cluster by cluster.
2. Assisted	The engineer, with AI help	A chatbot answers questions and drafts code fragments that a human integrates. An engineer asks why a pod is crash-looping and figures out how to integrate the recommendation into their specific environment.
3. Delegated	The agent, on request	The agent completes a whole task and hands back a change to review. An engineer drives an action, for example “upgrade the staging fleet to V1.31” and gets back pull requests covering every affected cluster.
4. Scheduled	The agent, unattended	The same work runs without a person starting it. A CVE scan runs nightly and opens the remediation pull requests before anyone reads the report.
5. Autonomous	The agent, across the fleet	The workflow runs fleet-wide on its own triggers, and policy decides which changes need a human. An alert fires at 2am, the agent investigates, opens a bounded fix, and any corrections made at review are kept for the next run.

Table 1: One question defines each stage: who does the work?

Locating your own team on the curve

Each stage has signals you can recognize without needing to monitor current working practices:

- **Stage 1** Kubernetes upgrades appear on the quarterly roadmap as a project with a named owner. The CVE backlog is measured in months.
- **Stage 2** A chat assistant is open on someone's second monitor all day, and nothing it produces is recorded anywhere in the estate, or for other team members to reuse.
- **Stage 3** An engineer can hand over a task and walk away, but nothing happens unless a human asks for it.
- **Stage 4** The queue is shorter on Monday morning than it was on Friday afternoon, and no one worked the weekend.
- **Stage 5** More changes ship in a week than any one person could walk through from memory, and the audit record accounts for every one.

02 · THE INFRASTRUCTURE AUTONOMY CURVE

Nobody starts at Stage 5. Teams climb the curve as evidence and experience accumulate, beginning with a workflow where the cost of a mistake is low, watching what the agents produce, and widening scope as the record justifies it. That progression is the normal path and it should be planned for.

What decides whether you can make it is the system you pick. Moving from Stage 3 to Stage 5 means running work unattended, across the whole fleet, with an audit record and a control on every action, and a system that cannot do those things will hold you back from delivering full infrastructure autonomy however much experience your team gains. Section 3 sets out what to look for.

If a human still approves everything, is it really autonomous?

This is where most autonomy models go wrong. They treat human review as friction to be engineered away, so the top of the model becomes an agent that changes production unsupervised.

Most teams would take full autonomy if they could trust it, prove what it did, and satisfy a regulator with the evidence. For most estates that combination is not available yet. What is available is a system built to reach it, running at the threshold you are ready for and moving up as confidence accumulates. You keep a hand on the wheel and make corrections while the record builds, and the system does not have to be rebuilt when you are ready to take it off.

That reframes the question. Rather than asking how much autonomy a system offers today, ask whether it can operate at the level you are comfortable with now and still go further without a migration.

Today approval is present at every stage of the curve, including the top, and it does not hold autonomy back. Producing a change is days of work, while reading one that arrives with its reasoning, a bounded diff, and a rollback path takes minutes. Move the production of changes to agents and approval stops being the constraint.

Stage 5 is therefore an agent that runs a workflow end to end across the fleet, on its own trigger, and still lands a reviewed merge. Where the review gate sits is something you configure. Require a human on every change to a production database, let a nightly CVE patch merge on its own, and move that line as the confidence builds. Criterion 2 covers what to ask about how that control works.

SECTION 03

What to evaluate

The six criteria below determine how far up the curve any given system can take you. They apply to anything you might put agents on: a commercial product, an assembled open-source stack, or something your own team builds.

None of them is a wishlist item. Each one gates specific stages, and the gate is noted under every criterion. A system can satisfy one criterion well and fail another badly, and where it fails tells you exactly where it will stall. That matters because a team aiming for Stage 3 does not need everything a team aiming for Stage 5 needs, so **score each criterion against the stage you are trying to reach** rather than against the best answer available.

The six criteria at a glance

1. Where agents execute, and with what permissions
Without this, you stop at Stage 2.

2. Control, approval, and audit
Without this, you stop at the security review.

3. How a change reaches production, and what it leaves behind
Without this, you stop at Stage 2.

4. How far the system can reach
Without this, incident work stops at Stage 2 regardless of everything else.

5. What the system knows, and what it retains
Without this, you stop at Stage 3.

6. Whether it survives your enterprise
Without this, you stop at procurement.

CRITERION 1

Where agents execute, and with what permissions

Require execution in a defined environment holding only the permissions the task needs.

An agent running on an engineer's laptop inherits everything that laptop holds, which in most organizations includes a long-lived cloud credential with far more access than the task needs. You cannot constrain it usefully, and you cannot prove afterwards what it touched.

Kubernetes RBAC, workload identity, and short-lived credentials issued from a secrets manager are all reasonable foundations. Server-side agent execution also buys parallelism, which matters from Stage 4 upward because forty clusters means forty concurrent agentic runs.

Without this, you stop at Stage 2.

CRITERION 2

Control, approval, and audit

Require control over what an agent reaches, who can invoke it, and a durable record of what it did.

This criterion covers three things that often get bundled together and should be evaluated separately. What an agent is allowed to reach. Who is allowed to invoke it. And what evidence exists afterwards about what it did.

The evidence matters most and often is the least well specified. You want a durable log of every tool call the agent made, every decision it took, and every change it produced, held for as long as your auditors require and queryable when someone asks what happened. This is a higher standard than you hold humans to today, since an engineer who shells into a box at two in the morning leaves almost no trace behind.

The largest enterprises add a further requirement. Before an agent operates in production it has to be registered with internal systems and integrated with the firm's AI governance tooling. Your platform team will not have built most of the agents running in the estate, but they will be the team that answers for what an agent did and cleans up after it. That makes agent governance your problem regardless of where the agent came from. Ask any vendor how they have handled that registration process elsewhere, because a system that has never been through it will need you to solve it for them.

Without this, you stop at the security review.

CRITERION 3

How a change reaches production, and what it leaves behind

Require every change to arrive through a path a human can inspect, approve, and roll back.

Every autonomous system needs a path into production that an engineer can inspect and reverse. Most enterprises already run one: a pull request against a repository, protected merges, and a pipeline that reconciles declared state onto the infrastructure. Your GitOps reconciler, your infrastructure as code tool, and your source control platform already do this work between them.

Insist that agent changes go through that path, rather than through a separate route the vendor controls. Reusing it inherits the controls you already operate and cuts the internal approvals needed to adopt anything new. Where formal change management applies, the merge should block until the linked change record is approved, rather than someone reconciling the two by hand.

Reviewability is what platform leaders most often under-specify. A gate only scales if an engineer can clear it quickly, which needs three things arriving with every change: the reasoning that produced it, a diff bounded to the fault instead of a wholesale rewrite, and a stated rollback path. Ask to see ten real changes a system produced for a customer at your scale, not one prepared for a demo.

Without this, you stop at Stage 2.

CRITERION 4

How far the system can reach

Require the system to act at whichever layer the fault lives, including application code.

Most incidents resolve to application code. A system that only touches infrastructure will investigate the failure, identify a bad query or a memory leak in a service, and then stop and write a ticket. Your engineer still has to open the repository and fix it, which is the part that was going to take the longest anyway.

Ask whether the system can raise a pull request against an application repository as readily as against a Terraform module or scale a cloud resource, and what it does when the fault turns out to sit in code rather than configuration.

Without this, incident work stops at Stage 2 regardless of everything else.

CRITERION 5

What the system knows, and what it retains

Require the system to read your infrastructure as it is now, and require the knowledge it accumulates to stay yours.

Agents reason over what they can see. A system reading live state from clusters, repositories, pipelines, and cloud accounts reaches different conclusions from one reconstructing events out of logs after the fact. Cost attached to resources belongs in the same picture, because right-sizing decisions are otherwise guesswork.

Retention is the second half of this criterion, and it is the one some vendors gloss over. Every system will be described as improving with use, and few descriptions say where the improvement lives. Ask four questions.

1 Are corrections made at review captured at all?	2 Where does that knowledge sit, and can your team read and edit it?
3 Does it travel with your estate or with the vendor?	4 How often is it refreshed?

Context and infrastructure knowledge held as versioned instructions in your own repositories is inspectable, reviewable, and portable. If that context is held inside a hosted model, it's none of those, and you find out what that costs at renewal.

🔒 **Without this, you stop at Stage 3.**

CRITERION 6

Whether it survives your enterprise

Require evidence at your scale, in your regulatory posture, before you are the test case.

Critical infrastructure should not be a vendor's first production test case. This criterion is about evidence rather than architecture.

Ask what the largest estate under management looks like, counted in clusters and in environments. Ask whether the system runs self-hosted, whether it operates air-gapped, and whether it works through an internal model gateway if your organization mandates one. Ask which SSO provider, ticketing system, observability platform, and alerting webhooks are already integrated in production somewhere other than a roadmap. Then ask for a reference at your scale and in your regulatory posture.

🔒 **Without this, you stop at procurement.**

SECTION 04

Questions to take into the room

These are written to be asked out loud, of any vendor and of your team if you are considering building your own solution. The note under each group describes what a weak answer sounds like.

Execution and permissions

- [ ] Where does the agent run, and what identity does it hold while it is running?
- [ ] Show me the permission scope for one workflow, and what happens when the agent requests something outside it.
- [ ] How many changes can the system produce concurrently across a fleet?

WEAK ANSWER

A description of the user's permissions rather than the agent's, or a concurrency number with no fleet behind it.

Control, approval, and audit

- [ ] What is recorded for a single run, how long is it kept, and can I query it?
- [ ] Who is allowed to invoke this agent, and where is that configured?
- [ ] We have to register agents with our AI governance tooling before production. How have you handled that elsewhere?

WEAK ANSWER

Anyone with console access can invoke it, or any response that treats the governance registration requirement as new information.

Path to production

- [ ] Show me the last ten changes the system produced for a customer at our scale, in the order they happened.
- [ ] What arrives alongside a change so that I can review it in under five minutes?
- [ ] Our changes require an approved change management request (e.g., a ServiceNow record). Does the merge block on it, or do we track that separately?
- [ ] What is the rollback path when a merged change turns out to be wrong?

WEAK ANSWER

One carefully chosen example, or a rollback story that depends on someone noticing issues quickly.

Reach and context

- [ ] Walk me through an incident where the root cause turned out to be in application code.
- [ ] Where does the system read cluster state from, and how current is that state at the moment it reasons?

WEAK ANSWER

An answer built entirely on log ingestion, or an incident story that ends at a recommendation.

Retention and portability

- [ ] Show me where the system stores what it learned from the last correction, and show me editing it.
- [ ] If we stop using this in two years, what will we keep?

WEAK ANSWER

The knowledge exists but nobody on your team can see it, edit it, or take it with them.

Enterprise fit

- [ ] What is the largest estate you run today, and can I speak to the team that runs it?

WEAK ANSWER

A logo without an estate size, or a reference in a materially different regulatory posture.

SECTION 05

Comparing approaches to infrastructure autonomy

Teams usually start by considering three routes: build it themselves, buy a point solution, or extend what observability already gives them. Each is a reasonable answer to part of the problem but each stalls for a different reason. A fourth route, a control plane that operates the estate with agents built into it, is the newest of the four, so it is worth understanding what the other three run into before evaluating it.

ROUTE 1

Build it yourself

Every large enterprise builds something, and the real question is how much. Assembling agent harnesses, tool gateways, and search over runbooks gets a capable team to Stage 3 inside a year. The higher stages are where it breaks down: unattended fleet-wide execution requires a live model of your infrastructure, permission scoping per workflow, an audit record that survives a compliance review, and policy evaluation on every action. The integration then becomes permanent, and the maintenance liability lands on your team.

ROUTE 2

Point solutions, including AI SRE

Root cause analysis is one item in the operations backlog. Provisioning, decommissioning, scaling, cost, CVE remediation, and fleet upgrades are the rest of it. Tools in this category observe the estate from outside the system that changes it, so most of them stop at a recommendation and act read-only. A recommendation is Stage 2 however good the reasoning behind it, and covering the remaining backlog means building or buying a second system.

ROUTE 3

Observability

Datadog and its peers see everything and change nothing. There is no execution path into infrastructure and no history of writing changes. Visibility is necessary, but nothing in the stack applies a fix, so an engineer remains the connection between the alert and the infrastructure.

Approach	Where it stalls	Cost carried	What your team still owns
Build	Stage 3, Stage 4 with sustained investment	Engineering years, then permanent maintenance	The whole system, including everything integrated into it
Point solution	Stage 2 for advice, Stage 3 in narrow workflows	License per tool, plus a second platform for the rest of the backlog	Every workflow the tool does not cover
Observability	Stage 2	License plus ingestion, which rises with the estate	Execution, entirely
Control plane with agents	Nothing structural, if the six criteria are met	License, plus migrating fleet operations onto it	Standards, guardrails, and review

Table 2: Contrasting the different approaches to infrastructure autonomy

SECTION 06

How Plural approaches this

Plural is a control plane for enterprise infrastructure with agents built into it. Measured against the six criteria, this is where it stands.

Agents execute server-side inside your own clusters with permissions scoped per workbench, never on a laptop, with read-only and read-write tiers and a persistent log of every job. Changes reach production as pull requests against your existing repositories, reconciled by Plural's GitOps engine, and the merge can be blocked until the linked change request is approved by your change advisory board. Policy decides which operations are denied, which require a human, and which are known-safe enough to approve automatically, and those policies are version controlled and can be tested against real runs before they take effect.

Agents read live state from a semantic graph spanning Git, infrastructure as code, Kubernetes, CI/CD, and cloud accounts with cost attached to resources, and they act at whichever layer the fault lives, including application code. Your standards, runbooks, and the corrections your engineers make at review live as versioned skills in your own repositories, so what the system learns is yours to read, edit, and keep. Plural installs self-hosted and egress-only, runs air-gapped, and works through your own model gateway.

AGAINST THE SIX CRITERIA

- **Execution**
  Server-side, scoped per workbench
- **Control & audit**
  Versioned policy, log of every job
- **Path to production**
  PRs, GitOps, CAB-gated merges
- **Reach**
  Infrastructure and application code
- **Knowledge**
  Live graph, skills in your repos
- **Enterprise fit**
  Self-hosted, air-gapped, your gateway

Proven in production

300+	10x	90%	~5,000
Clusters running a bank's core infrastructure	Kubernetes footprint per platform engineer	Upgrade workflows automated	Edge clusters, no local IT

GLOBAL FINANCIAL INSTITUTION

A global financial institution runs the core infrastructure for the bank comprising more than 300 clusters on Plural. Each platform engineer now operates ten times the Kubernetes footprint they did before, and the organization delivered its roadmap three times faster the following year.

FORTUNE 500 CYBERSECURITY

A Fortune 500 cybersecurity company automated 90 percent of its upgrade workflows with Plural. A cycle that had consumed a senior team for twelve weeks was executed in a single afternoon by the most junior engineer on the team.

GLOBAL ENGINEERING CONSULTANCY

A global engineering consultancy built its enterprise-wide internal developer platform on Plural and budgets it as critical infrastructure alongside its cloud provider rather than as an operational cost line.

NATIONAL RESTAURANT OPERATOR

A national restaurant operator runs roughly five thousand edge clusters using Plural, one per site, with no local IT at any of them.

Two ways to test Plural

SELF-SERVICE

Cloud sandbox

The cloud sandbox is a self-service trial of the full product with no installation and no credit card. Most teams have agents running inside an hour.

GUIDED

Proof of concept

A guided proof of concept runs against your most acute workflow, whether that is cluster upgrades, CVE remediation, incident root cause, or cost. It runs in your environment or in Plural's, including air-gapped, with weekly working sessions alongside the Plural team.

NEXT STEP

See it on your own clusters

Book a demo plural.sh

plural

The AI-native control plane for enterprise infrastructure.

NEXT STEP

Contact us to book a demo.

LEARN MORE

plural.sh

