INFRASTRUCTURE AS CODE · PLATFORM ENGINEERING

IaC Declares How. An IOM Declares What's Allowed.

Infrastructure as Code captured the build. It did not capture the authority that decides whether the build should run.

ABSTRACT

Infrastructure as Code was a genuine advance: it made infrastructure version-controlled, reviewable, and reproducible, and it retired an era of hand-built drift. Its success produced a natural belief — that if the whole estate is described as code, intent is already captured. It isn't. IaC is declarative about *implementation*: it states how to build a desired state. It is silent about *authority*: what that state is allowed to be, why, who owns its blast radius, and whether a given change should be permitted at all. A Terraform module can create a security group; nothing in the code knows whether that security group violates a segmentation boundary that lives in no file. This paper draws the line precisely — IaC executes declared implementation; an **Infrastructure Operating Model (IOM)** holds declared intent and reconciles reality against it — and shows that the two are not rivals. IaC becomes the IOM's execution arm. The IOM is the authority that decides whether the code should run.

1. The advance IaC actually made

Infrastructure as Code earned its place. It replaced hand-built, snowflake infrastructure with definitions that are version-controlled, peer-reviewed, diffable, and reproducible. It made "rebuild it from scratch" a routine operation instead of a disaster-recovery gamble, and it gave teams a shared, inspectable artifact where there had been tribal knowledge and console clicks. None of what follows diminishes this. IaC is one of the most important operational advances of the last fifteen years, and an IOM does not replace it.

But every advance creates its own characteristic blind spot, and IaC's is a specific and seductive one. Because IaC definitions are *declarative* — you state the desired end state rather than scripting the steps — it is easy to conclude that declaring infrastructure as code is the same as declaring intent. It is not, and the gap between the two is exactly where governance has quietly gone missing.

> ***"It's all in the code" is true of the implementation and false of the authority. The code says how to build. It does not say what is allowed to be built.***

2. The declarative trap

The trap is the inference from "our infrastructure is fully described as code" to "therefore our intent is captured." The two feel identical because both use the word *declarative*, but they declare different things. IaC is declarative about **implementation**: it specifies the resources, their configuration, and their relationships required to reach a desired state. Intent is a declaration about **legitimacy**: what states are permissible, under what constraints, owned by whom, and which changes may proceed.

A concrete case makes the gap visible. A Terraform module declares a security group that opens a port between two services. The code is valid, it applies cleanly, and it is the source of truth for *that resource*. What the code cannot express — and does not contain anywhere — is the architectural rule that production and non-production must never share a trust boundary, or that this particular service class may never accept inbound administrative traffic. If the security group violates one of those rules, IaC will build it anyway, correctly and reproducibly, because the rule was never something IaC was designed to hold. The code declares the *how* flawlessly and is silent on the *whether*.

> ***IaC is declarative about implementation. Intent is declarative about legitimacy. The shared word hides that they are different kinds of fact.***

3. Four things the code cannot say

The boundary becomes precise when you enumerate what authority requires and IaC structurally lacks. None of these are deficiencies in IaC; they are simply outside what implementation code is for.

It cannot state what is allowed, only what to build

A definition describes a target state. It carries no representation of the space of *permissible* states around that target, so it cannot determine whether the target itself is legitimate. The constraint that would make that judgment — the segmentation rule, the trust boundary, the ownership policy — lives in no module, and a plan that violates it looks identical to one that honors it.

It cannot adjudicate between conflicting-but-valid code

Two engineers can write two modules that are each syntactically valid, each apply cleanly, and each contradict the other's architectural assumptions. IaC has no notion of which is *correct*, because correctness is a question about intent, and intent is not in the code. The merge conflict it can resolve; the legitimacy conflict it cannot see.

It reconciles against itself, not against authority

Drift detection in IaC compares running state to the *last applied configuration* — drift from the code, not drift from what was authorized. If the code itself encodes a non-conforming state, IaC will faithfully detect drift away from that non-conforming state and "correct" reality back toward the violation. Its reconciliation loop is closed over its own prior output, with no external reference to intent.

It executes; it cannot refuse

This is the decisive one. IaC's purpose is to make the declared state real. It has no faculty for evaluating a change against authority and *declining* to apply it on legitimacy grounds. Policy-as-code guardrails bolt a partial check onto the pipeline, but they validate rules within a boundary; they do not hold a cross-domain, authoritative model of intent, and they live inside the execution path they are meant to police.

4. Why “more code” doesn’t close the gap

The intuitive fix is to push intent *into* the code: encode the policies as policy-as-code, capture the constraints as more modules, make the descriptions richer. This helps at the margin and is worth doing, but it does not reach authority, for the same reason aggregation never reaches it. Intent expressed as more execution artifacts is still bound to the execution layer: it is enforced where it is defined, scoped to the pipeline that runs it, and versioned as code rather than held as an independent reference that execution must answer to.

Authority has to sit *outside* and *above* the thing it governs, or it cannot govern it — a system cannot be the sole judge of its own legitimacy. No volume of additional declarative code becomes authority, because code declares how to build while authority declares what is legitimate, and the second has to be stated as a reference the first is checked against rather than folded into the first. The gap is structural, not a matter of how much of the estate is yet captured in modules.

> ***You cannot close an authority gap with more execution. A system that is its own only reference cannot tell you whether it is wrong.***

5. The cooperative architecture: IaC as the IOM’s execution arm

Stated correctly, IaC and the IOM are not competitors at all; they are adjacent layers with a clean division of labor. The IOM holds declared intent — allowable states, constraints, ownership, blast-radius semantics — as an authoritative model independent of any execution tool. A proposed change, whether it originates from a Terraform plan, a pipeline, or an autonomous agent, is validated against that model *before* it executes. If it conforms, it proceeds; if it violates intent, it is refused or escalated. IaC then does what it is genuinely excellent at: making the now-authorized state real, reproducibly and at scale.

In this arrangement IaC loses nothing and gains a governor. Its definitions remain the execution mechanism and the implementation source of truth; what moves out of IaC is the burden it was never built to carry — the authoritative account of what is legitimate. The IOM does not read or rewrite Terraform; it sits above it, deciding whether the plan should run and reconciling the running result against intent. Platform teams keep their tooling, their workflows, and their reproducibility, and acquire something they never had: a layer that can say no before the apply, on grounds the code could never represent.

> ***IaC declares how to build. An IOM declares what is allowed to be built — and validates the plan against it before a single resource is created.***

6. What platform engineering gains

For the teams closest to IaC, the practical payoff is direct. Architectural standards stop being wiki pages that review boards check by hand and become constraints validated against every plan automatically, so conformance is continuous rather than periodic. Blast radius is bounded by a declared dependency and ownership model rather than discovered during an incident. Conflicting-but-valid changes are adjudicated by the model instead of by whoever wins the pull-request argument. And autonomy becomes safe: an agent generating and applying IaC can be permitted to act precisely because something independent can refuse an illegitimate plan before it executes. The faster and more automated the pipeline becomes, the more valuable the layer that can gate it.

7. Conclusion

Infrastructure as Code captured the build, and that was a real and lasting advance. What it did not capture — because it was never its job — is the authority that decides whether a build should happen at all: what is allowed, why, owned by whom, and conforming to which constraints. That authority is a different kind of declaration than implementation code, and it has to be held above the execution layer as a reference the code answers to, not inside it as more code. An Infrastructure Operating Model is that layer. It does not replace IaC or compete with it; it governs it, and in doing so turns reproducible execution into governed execution. The code already says how. The operating model is what finally says whether.

This is a category paper in the IOM Standard library. It establishes the boundary between declarative implementation and declared authority; it does not restate the normative requirements of the IOM Standard. The IOM Standard is an open, vendor-neutral specification — learn more at theIOM.org.