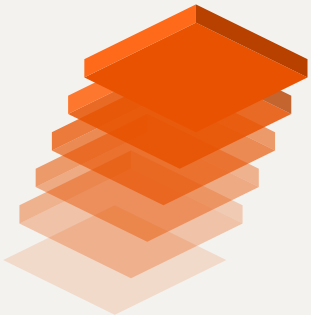
encore

The AI-Ready Infrastructure Playbook

Code moves at AI speed.
Infrastructure does not. A report for engineering leaders.



Executive summary

AI has made code much cheaper to produce, but it has not done the same for infrastructure, and that mismatch is what now limits how fast software actually reaches production. This playbook explains what is slowing delivery down, what AI-ready infrastructure looks like, and how leading teams are rebuilding their delivery systems around it.

01 **More code means more infrastructure.**

Each service, database, queue and permission an AI-assisted team produces still has to be provisioned, configured and validated. Infrastructure change volume rises with code volume.

02 **Infrastructure as code does not absorb that volume.**

IaC is the hardest category of code to review, because a diff shows what a change intends but not what it will actually do, and the errors that matter often apply without failing. A human reviewer rarely catches them from the diff, and neither does an AI reviewer.

03 **The difference is the feedback loop.**

AI performs on application code because compilation, tests and local execution return a verdict in seconds. Infrastructure has little that compares, so in most Terraform estates the verdict tends to arrive only once the change is already in production.

04 **The requirement follows from the loop.**

Infrastructure work needs what application code already has: somewhere real to run, errors that show up at build time, and output the author can actually read back, all before production.

05 **Platforms that derive infrastructure from code meet that requirement.**

Infrastructure is declared as typed resources inside the application and provisioned from those declarations. Encore, examined in chapter 6, is one implementation.

Review capacity was never the bottleneck. Infrastructure code cannot be fully validated until it runs, and in many organisations that first happens in production.

© 2026 Encore

1 More code means more infrastructure

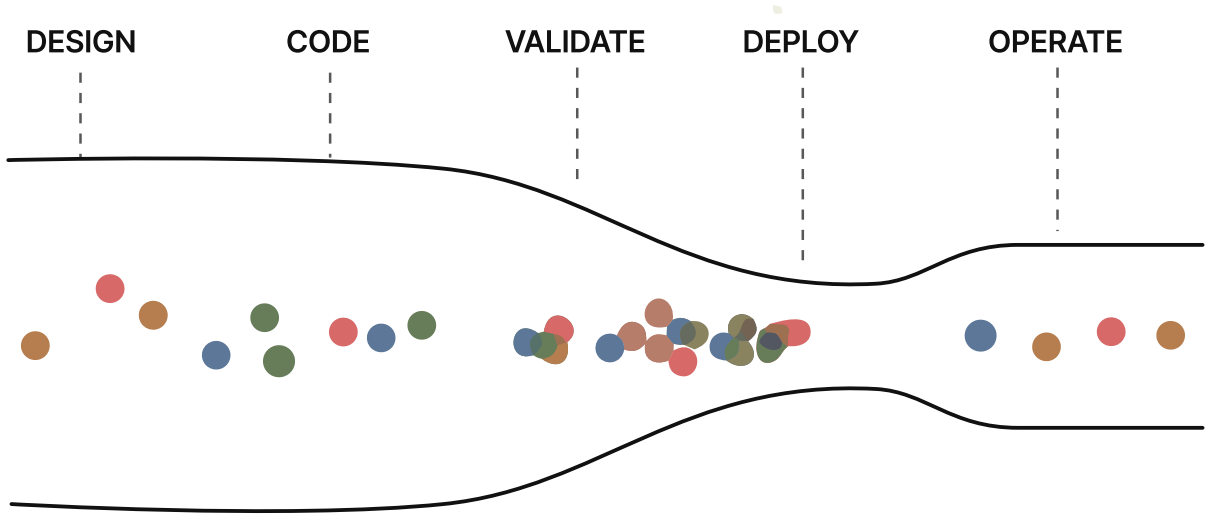
Every feature drags infrastructure along with it: the tables, queues, secrets and permissions a change depends on are not application code, and most of them only surface late, once the feature is already being built.

A single instruction such as “add signup emails with a retry queue” turns into one reviewable diff of business logic plus roughly nine infrastructure objects: a topic, a dead-letter queue, a subscriber role, a provider secret, an egress rule, retry configuration, a delivery-state table, environment variables, and alerting. Only the first of those is covered by a test.

Until now, the rate of infrastructure change was capped by how fast humans could write the features in front of it, and that cap is now gone. 93% of organisations report at least one AI-caused infrastructure incident, and 67% report that development has moved ahead of infrastructure.[1]

FIGURE 01 — WHERE THE WORK ACCUMULATES

DESIGN → OPERATE



Code generation was one stage of the lifecycle, and AI compressed it. Validation, provisioning and deployment still run at their previous speed, so work accumulates in front of them. Illustrative, not to scale.

The broader data points the same way: two thirds of software firms have rolled out generative AI tooling, and those that have report a 10 to 15% productivity gain, since writing and testing code accounted for only 25 to 35% of the path from idea to launch.[2] Compressing that one stage pushes the constraint downstream instead of getting rid of it: AI-authored pull requests wait 4.6 times longer for a first review,[3] and 45% of deployments containing AI-generated code lead to problems.[4]

[1] Spacelift, “The Infrastructure Automation Report 2026: The AI Readiness Gap”, n=406 IT and platform leaders.

[2] Bain & Company, “From Pilots to Payoff”, Technology Report 2025. [3] LinearB, “2026 Software Engineering Benchmarks”, 8.1M pull requests.

[4] Harness, “The State of AI in Software Engineering 2025”, 900 engineers and leaders.

© 2026 Encore

2 Why infrastructure code resists review

A common objection to the previous chapter is that AI reviews code well, so AI-assisted review should be able to absorb the extra volume, and for application code it can. Infrastructure is different, because a change to it is easy enough to read but hard to judge from the diff alone.

A change to application code can be checked locally in seconds, by the compiler and its tests or by just running it. A change to infrastructure is a claim about a system that is not there at review time, and whether it is correct depends on the live state of the target account and on conditions that only appear under load, neither of which the diff shows.

FIGURE 02 — WHERE EACH KIND OF MISTAKE GETS CAUGHT

CAUGHT BEFORE PRODUCTION

THE MISTAKE	IN APPLICATION CODE	IN TERRAFORM
Syntax or type error	Compiler, instantly	Validate, instantly
Wrong behaviour	Tests, run locally in seconds	No local equivalent exists
Over-permissive IAM policy	n/a	Applies cleanly; found in an audit or a breach
Missing backup or retention	n/a	Applies cleanly; found when you need it
Undersized pool or timeout	Load test	Applies cleanly; found under real traffic
Plan built on stale state	n/a	Invisible until apply

The four failure modes an infrastructure reviewer most needs to catch are in the lower rows, and none is visible in the diff. The same limitation applies to an AI reviewer, because the information needed to judge the change lives in the target account, which the diff never shows.

Model performance follows the same split. On EvalPlus, the standard Python benchmark, GPT-4 passes 86.6% of tasks; on IaC-Eval, its Terraform equivalent, it passes 19.4%.[5] This is not about HCL syntax either: on Amazon's SWE-InfraBench, which uses AWS CDK instead, the best model still gets only 34% of single attempts right.[6]

[5] Kon et al., "IaC-Eval", NeurIPS 2024, 458 scenarios. [6] AWS, "SWE-InfraBench", NeurIPS 2025 workshop.

© 2026 Encore

One change in detail

Take a change an agent might open, a new service that needs a database, a security group and an IAM policy. It reads cleanly and **`terraform plan`** succeeds, but neither says much about whether it is safe.

FIGURE 03 — ONE CHANGE, FIVE OPEN QUESTIONS

AGENT-GENERATED PR

```
# infra/main.tf
+ resource "aws_db_instance" "users" {…}
+ resource "aws_security_group" "svc" {
+   ingress { from_port=0 to_port=65535 }
+ }
+ resource "aws_iam_role_policy" "svc" {
+   Action="s3:*" Resource="*"
+ }

validate ✓plan ✓   6 to add
```

WHAT THE DIFF AND THE PLAN DO NOT TELL YOU

IAM	Is the access exactly what the service needs, or more?
NETWORK	Can the service actually reach its database and queue?
LIFECYCLE	Will applying it replace or destroy something in use?
STATE	Will it collide with existing resources, quotas or drift?
RUNTIME	Will the service run once the infrastructure is live?

The ingress rule opens every port and the policy grants every S3 action, yet both pass validate and plan. Not one of the five questions can be settled from the diff or the plan, because each depends on the live account the change runs against rather than the change itself.

Teams have already adjusted in practice: 78% use AI to generate infrastructure code without thorough review, and a third would push AI-written HCL to production without reviewing it at all.[1]

A reviewer can read the intent here and still not know the outcome, and the only thing that answers these questions is running the change somewhere first.

© 2026 Encore

3 The feedback loop

AI does well on application code because of the loop it works inside. When an agent gets something wrong, it finds out in seconds and can try again at almost no cost, and none of the failed attempts leave a mark.

Infrastructure has nothing like that loop, because the only way to run a change is to apply it to a real cloud account. The loop closes between staging and production, hours later, with users already exposed to the result.

FIGURE 04 — TWO SESSIONS, SAME AGENT, SAME DAY

ABRIDGED TRANSCRIPTS

WORKING ON APPLICATION CODE

```
10:02:04 run tests
10:02:06 2 failing
10:02:19 edit · run tests
10:02:21 1 failing
10:02:44 edit · run tests
10:02:46 passing
10:03:10 open pull request
```

Three corrections in 66 seconds. The environment answered each one, and no one else saw the attempts.

WORKING ON INFRASTRUCTURE CODE

```
10:02:04 terraform validate
10:02:07 valid
10:02:12 terraform plan
10:02:41 6 to add, 1 to change
10:04:00 open pull request
14:20:00 merged, applied
02:14:31 paging on-call
```

No correction was possible. Both checks passed, and the first informative signal arrived sixteen hours later.

The left session iterates because the environment gives back a verdict the author can act on. The right one cannot, since validate and plan pass whether or not the change is right, so the only real verdict comes from production.

The industry data shows the same pattern, with DORA finding that every 25% increase in AI adoption came with a 7.2% decrease in delivery stability and concluding that AI amplifies whatever system it is dropped into.[7] Where the infrastructure loop only closes in production, what gets amplified is incidents.

None of this is fixed by a better model. Even one that wrote flawless Terraform would be working blind, because the file it writes cannot contain the live state that its correctness depends on.

> "A high-quality platform amplifies the effects of AI adoption."
>
> DORA Research, 2025

[7] DORA, "Accelerate State of DevOps" 2024 and "State of AI-assisted Software Development" 2025.

© 2026 Encore

4 The IaC ecosystem has not addressed this

You might reasonably expect the tooling ecosystem to be working on this, but the record of the last three years does not really bear that out.

FIGURE 05 — WHAT THE ECOSYSTEM SPENT 2023–2026 ON — PUBLIC RECORD

AUG 2023	Terraform moves to the Business Source License. The community forks it as OpenTofu. Licensing.
FEB 2025	IBM completes its $6.4B acquisition of HashiCorp; the roadmap now runs through IBM. Ownership.[8]
2025	GitLab drops its Terraform templates over the licence; Fidelity migrates 50,000+ state files to OpenTofu. Migration.[9]
2026	Two divergent dialects, and 78% of teams generating both with AI. Still no way to run the thing before production.[1]

That is three years spent on licence changes, acquisitions and forks. None of it touched the feedback loop, and the fork now has AI tooling generating two diverging dialects where there used to be one.

The layer was under strain before any of this: 89% of organisations use IaC, but only 6% report full coverage of their cloud, roughly a third of codified resources have drifted from their definitions, and 48% of teams make out-of-band production changes several times a week.[10] Definition and reality routinely disagree, and that is exactly when a plan tells you the least.

WHY MATCHING HUMAN QUALITY IS NOT ENOUGH

Human-written infrastructure fails at some rate too. Production absorbs those failures today only because human throughput keeps their number down. Push the change volume up tenfold and the same per-change quality gives you ten times the incidents, so once volume is the goal, simply matching human quality no longer gets you there.

From here, teams can go one of two ways, either keeping a human reviewing every infrastructure change and accepting infrastructure as a permanent brake on everything AI produces upstream, or changing what has to be written in the first place so the loop can close before production.

[8] IBM Newsroom, Feb 2025. [9] OpenTofu blog, Oct 2025; GitLab 18.0 breaking changes.
[10] Firefly, "State of IaC" 2025–2026 and "The Bad IaC Tax".

© 2026 Encore

5 What AI-ready infrastructure requires

The loop from chapter 3 implies a narrow requirement, that whoever changes infrastructure, whether a person or an agent, can run the result and see what it did before it reaches production. Four conditions make that possible.

01 **Real infrastructure, off the critical path**

A change has to be able to create the resources it needs and run against them in an environment nobody else depends on, using real queues, databases and permissions instead of mocks or plan output.

YOU HAVE IT WHEN

the full system can be created and broken without affecting anyone.

02 **Failures that surface at build time**

The core problem with HCL is that wrong configuration often applies without raising any error. When the same requirements are written as typed declarations, a missing database or a wrong permission turns into a build failure instead of an incident.

YOU HAVE IT WHEN

the compiler, not the reviewer, stops the incorrect change.

03 **A single source of truth**

When infrastructure is derived from the application code, there is no second codebase to keep in sync and nothing to drift, and the work stays in the kind of code models are actually good at.

YOU HAVE IT WHEN

a feature and its infrastructure ship as one pull request.

04 **A readable return signal**

A loop needs a way back. Traces, logs and errors from the run have to be produced automatically, in a form an agent can actually parse, or iteration stops after a single attempt.

YOU HAVE IT WHEN

a failed run can be diagnosed without dashboards or assistance.

An infrastructure change can finally be **checked before it ships**, and review becomes what it already is for application code, a second opinion on something that has already run.

It also helps with governance, because the platform holds the credentials and state, so an AI tool only ever sees business logic and typed declarations.

NOTE WHAT IS NOT ON THIS LIST

A better AI reviewer, a policy engine on the existing pipeline, or a bigger module library would each improve the current loop at the margins, but none of them lets you see what a change actually did before customers do.

© 2026 Encore

6 Infrastructure derived from code

These four conditions point to a category of platform, not one specific product, defined by deriving infrastructure from the application code itself rather than maintaining it alongside. Encore is a concrete example here, because its model is public and specific.

In this model the developer, or the agent, declares what the service requires in ordinary TypeScript or Go, next to the code that uses it.

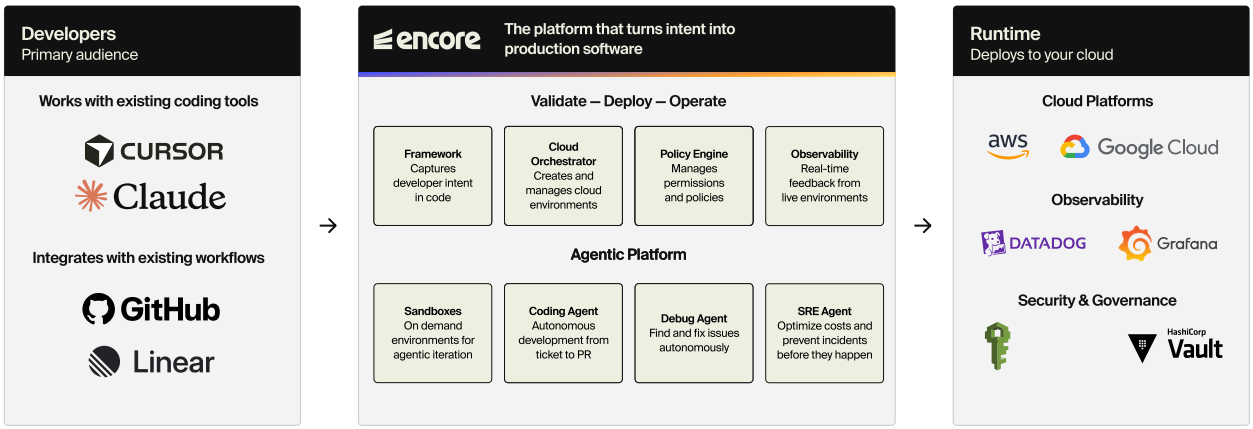
SIGNUP/SIGNUP.TS

```
// the database and the queue are declarations, not YAML
const users = new SQLDatabase("users", { migrations: "./migrations" });
const signups = new Topic<SignupEvent>("signups", { deliveryGuarantee: "at-least-once" });
```

The platform reads those declarations by static analysis and works out what to provision in each environment, with which permissions, and what to trace. Sensible defaults cover what the declaration leaves unsaid, from instance sizing to IAM scope, and each can be overridden in the dashboard without editing the code. The same definition runs on a developer machine, in a preview environment created per pull request, and in the organisation's own AWS or GCP account. Because that same definition is what runs locally, getting a change working on your machine tells you how it will behave in production, and the loop from figure 04 closes two stages earlier.

FIGURE 06 — ONE IMPLEMENTATION OF THE MODEL

ENCORE



The developer's tools on the left, the platform that turns declarations into running environments in the middle, and the organisation's own cloud on the right. Nothing in the developer's path is a separate infrastructure codebase.

An agent can then work on infrastructure the way it already works on application code, declaring a resource, running the service against it, and reading back the trace to correct. The environment that returns the verdict is created for the change and discarded with it, so the number of attempts carries no operational cost. Running the change also settles the five questions Figure 03 left open, because it has actually run, not just been planned.

© 2026 Encore

What the repository stops holding

Most of what an AI tool can read and edit in a Terraform-based service is configuration that cannot be run without a cloud account, the same material a reviewer cannot verify. Deriving infrastructure from declarations takes it out of the repository altogether.

FIGURE 07 — WHAT STOPS BEING WRITTEN

ONE SERVICE, BOTH MODELS

TERRAFORM-BASED SERVICE

```
signup/signup.ts
infra/main.tf
infra/variables.tf
infra/iam.tf
infra/terraform.tfstate
Dockerfile
.github/workflows/deploy.yml
.env.production
```

Seven of the eight files cannot be exercised without a cloud account.

INFRASTRUCTURE DERIVED FROM CODE

```
signup/signup.ts
signup/migrations/1_users.up.sql
—
—
—
—
—
—
```

Everything remaining compiles, runs and is covered by tests.

The same service under both models. The files removed on the right are the ones chapter 2 identified as unreviewable.

Because the platform holds the credentials and deployment state, an AI tool reading the repository finds business logic and typed declarations, never production keys.

The cloud account, the security model and operational responsibility all stay with the organisation. The second codebase describing them goes away, and with it a class of error that used to surface only in production.

Deleting infrastructure carries a risk that adding it does not. Whether an existing firewall rule or IAM grant is still needed is hard to tell from the code, and removing one that turns out to be load-bearing causes an outage, so the safe choice is to leave it in place. Permissions and network rules pile up, and the estate drifts steadily towards over-permissive. When each resource exists because a line of application code asks for it, removing that code removes the resource and its access with it, so the question never has to be asked.

> “Before Encore, deploying with Terraform was so frustrating. Even for small changes, we’d be stuck dealing with out-of-date Terraform state. With Encore, it’s a headache I don’t need to have.”
>
> **Mason Stewart, CTO**
> BOOKSHOP.ORG

© 2026 Encore

7 Where to start

Two questions tell you whether this report is describing your organisation, and you can answer both without a vendor. First, where does an infrastructure mistake actually get caught today? Second, what would it be worth to catch it a stage or two earlier?

FIGURE 08 — WHERE IS AN INFRASTRUCTURE ERROR CAUGHT TODAY? BY STAGE

At build time	Uncommon. The loop is already closed, and this report describes a problem you do not have.
In code review	The control is a person reading intent. Chapter 2 sets out the four failure modes that survive it.
In staging, sometimes	A partial loop, shared and slow to reset, and bypassed under deadline.
In production	The condition this report describes, and it only gets worse the better your AI adoption goes.

Most organisations land on one of the lower two rows. For the second question, put one upcoming service on the model from chapter 6 while everything else keeps running, and read the four numbers below before and after. Nothing migrates, and the comparison is against your own baseline.

FIGURE 09 — FOUR NUMBERS, BEFORE AND AFTER TWO READINGS

METRIC	BEFORE	AFTER	CHANGE
Lead time Commit to running in production			
Review wait Pull request opened to first review			
Change failure rate Deploys requiring a corrective change			
Infrastructure errors caught late Your answer to figure 08, counted			

Fill in the before column while everything is still unchanged. The last row is the diagnostic above, counted rather than guessed at.

© 2026 Encore

About Encore

Encore is a development platform for teams whose code output has outgrown their infrastructure. Developers and AI agents declare the infrastructure a service needs, from databases and queues to cron jobs and secrets, as typed resources in ordinary TypeScript or Go, and Encore provisions and runs them in your own AWS or GCP account, with tracing, metrics and logs generated automatically.

HOW IT WORKS

Declared, not configured

Resources are typed objects in your code, not free-form YAML or HCL. There is nothing for an agent to mis-key or hallucinate, and an unsafe declaration fails at compile time rather than in production.

Sensible defaults, yours to override

Instance sizing, connection pooling, retention and IAM scope ship with production-ready defaults, each adjustable from the dashboard without touching the code.

Runs beside what you have

Adopt it one service at a time. Existing Terraform and cloud resources keep running alongside, with nothing to rip out or migrate.

Because the infrastructure lives in the application code, there is nothing separate to review or keep in sync, and a change can be run and checked before it reaches production. That is the loop this report argues for, available by default rather than as a programme to stand up.

"Encore is our foundation for all new development. Since adopting it, we've seen a 2–3x increase in development speed and 90% shorter project lead times."

GROUPON Josef Sima, Engineering Director

"We're saving $200k+ annually compared to additional hiring. The ROI is outstanding, easily 10x."

Carla Daniel Stocks, CTO

Documentation, quick-starts and a free tier are at encore.dev.

© 2026 Encore

Close the loop before production

See what your team ships once infrastructure can be tested before production instead of after.

Get started free

Book a demo

SHARE THIS INTERNALLY

A 7-slide presentation summarising how to make infrastructure keep pace with AI-generated code, ready to share with your team.

Download the deck