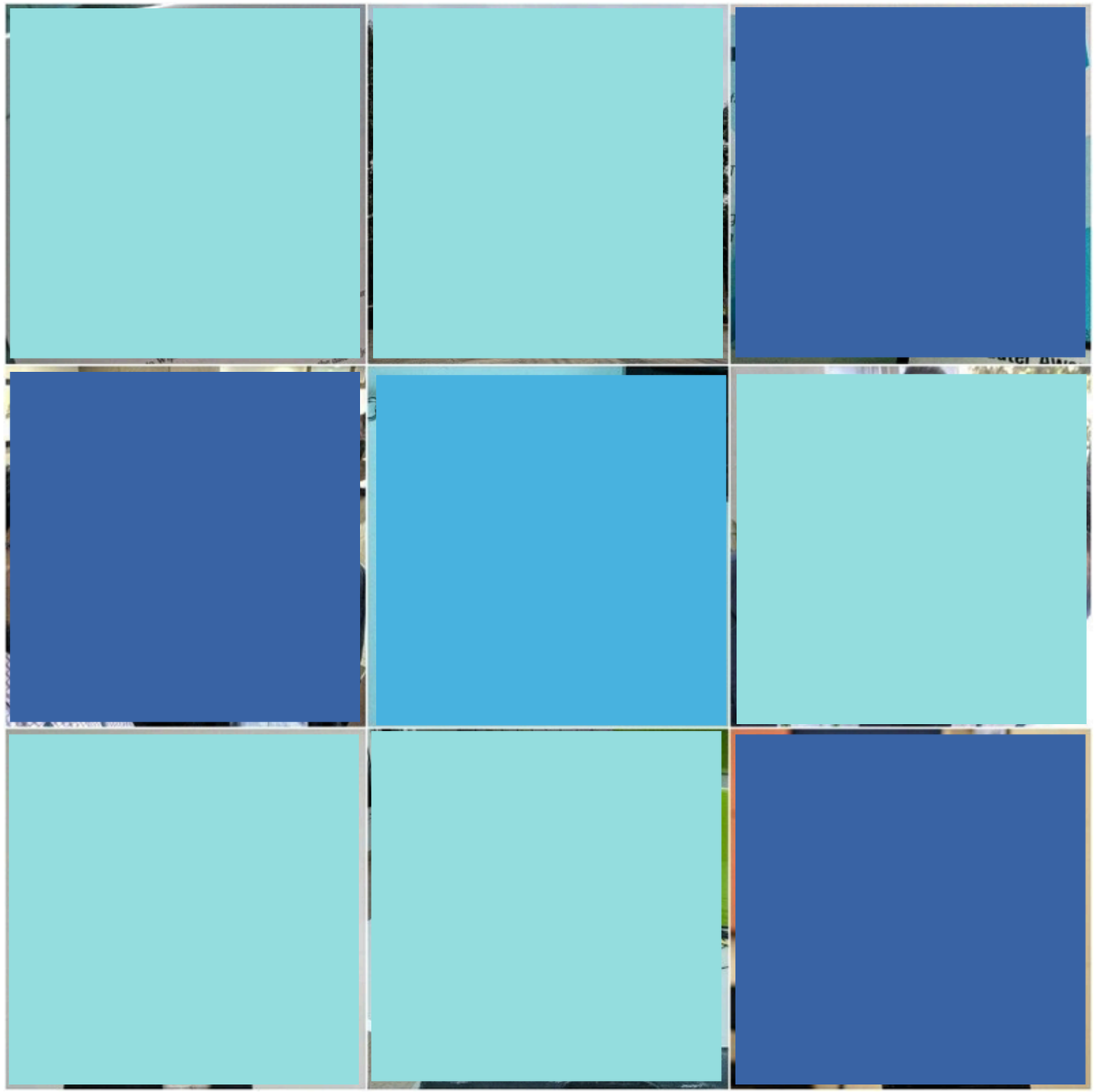
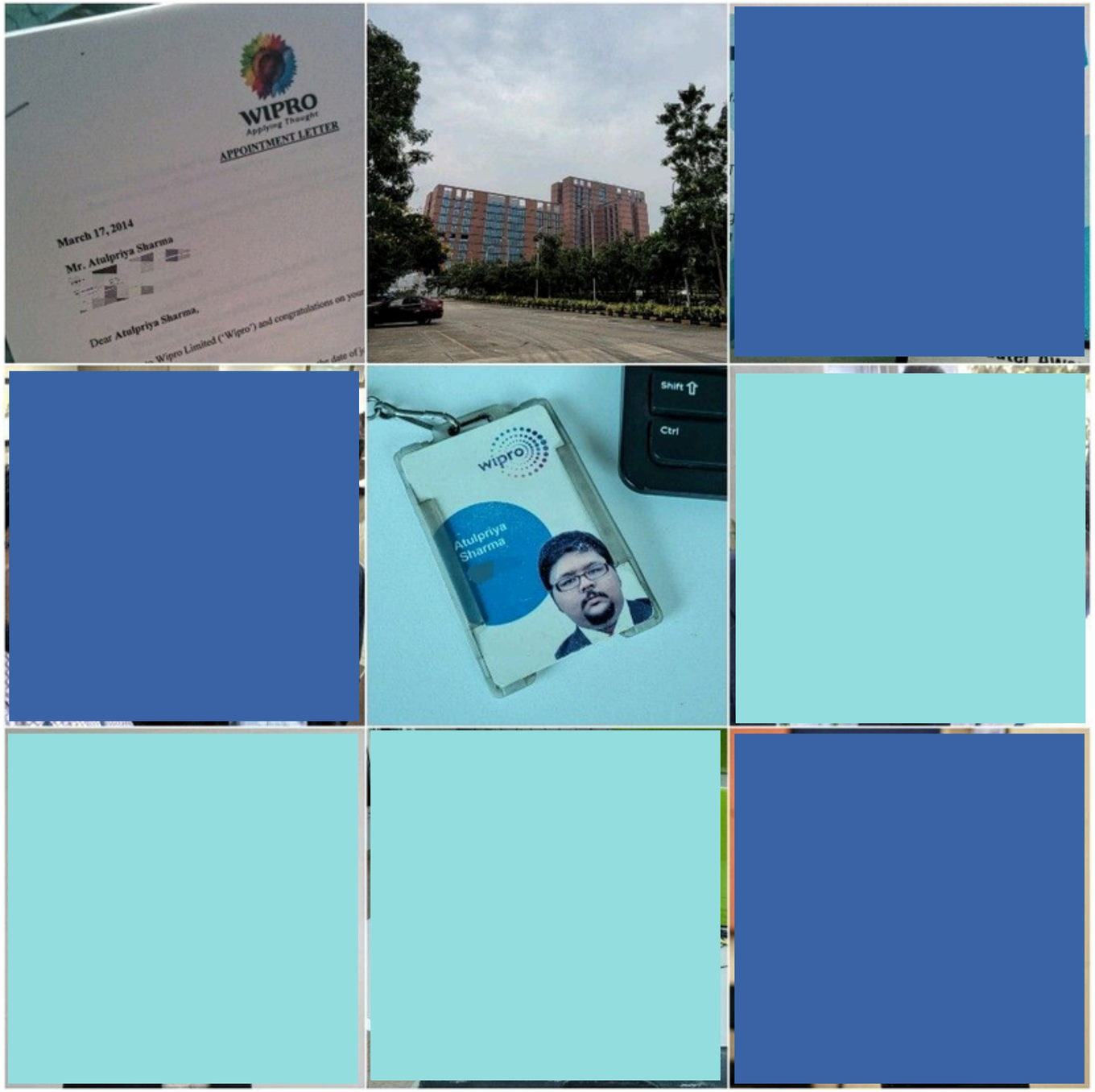


Quick Poll



2014

Joined a consulting company.
Wanted to be a developer.
Trained on a vendor testing suite.
Put on manual QA.



2018

Automation testing using the client's own C# framework.
A web portal for my own QA team.



2021

Power BI dashboards for a state health department.
Android apps using Xamarin.

I'm not telling you this because it's unusual.

I'm telling you because it isn't.

Golden Paths

Guardrails

Platform Engineering

Self Service

Standardization

Internal Developer Portal

Platform Teams

Platform Engineering's Missing Chapter

Services Organizations

November 9, 2026



Platform Engineering Day
NORTH AMERICA

KubeCon | CloudNativeCon
North America 2026

Atulpriya Sharma

- Hyderabad, India
- Principal Developer Advocate, Improving
- CNCF Ambassador, Co-Chair, KubeCon India'25, Co-Chair Platform Engineering TCG, CNCG Hyderabad



SCAN ME

@TheTechMaharaj

Atulpriya Sharma

SCAN ME

@Atulmaharaj

https://socialmaharaj.com



The first thing it cost

Nobody asked me to build it.

A portal. Which apps my team had tested.
No ticket. No owner. No mandate.

Then I rolled off.



The second thing it cost

I went deep twice. Neither one travelled.

A client's own C# framework. Two years.
Only ever existed inside their walls.

Xamarin. Two years.
Microsoft ended support, May 2024.



I never found out what the next floor was building.

Nobody asked me what I had built. I never asked them.

There was nowhere to put the answer.

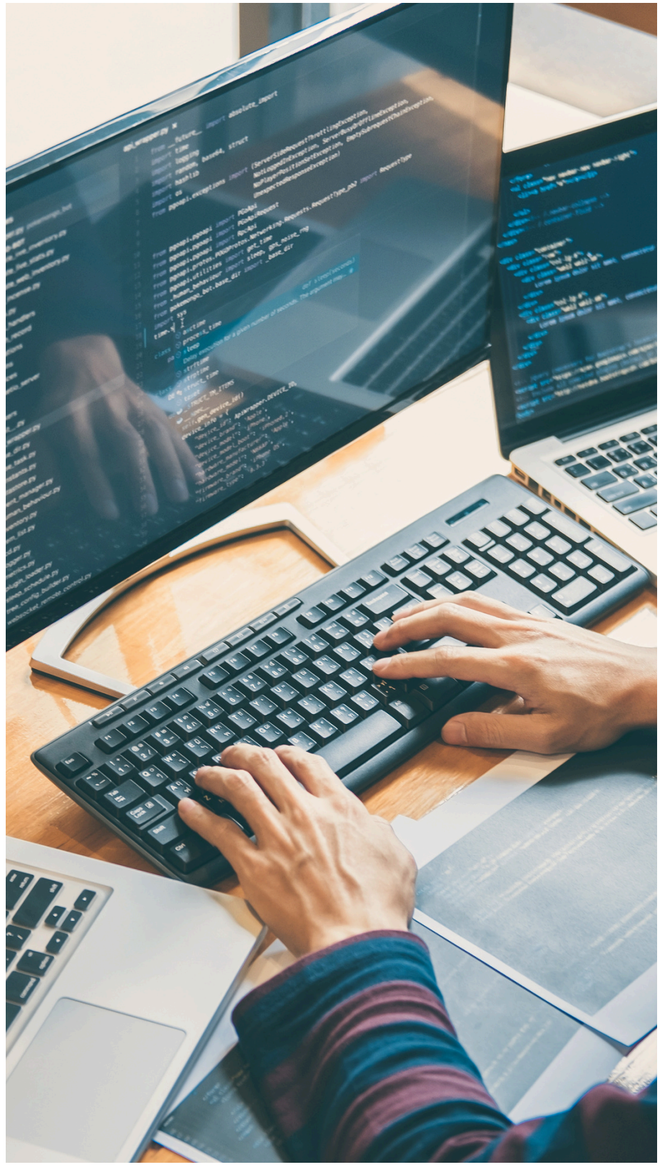
Each project team had a different way of delivering.
There was no standard way.



What platform engineering got right

2022: 45% of engineering organisations had platform teams.
2026: Gartner predicted 80%.
Backstage, open sourced 2020, now 3,400+ organisations.

Golden paths. Paved roads. Self-service. Cognitive load down.



Our Own Definition

"an integrated collection of capabilities defined and presented according to the needs of the platform's users"

"...no more and no less than an enterprise's internal users"

Platforms Whitepaper
Platforms Technical Community Group



The environment belongs to the client.

The team dissolves with the engagement.

The time is billable.

The users are internal.
The runtime never is.



The thing you already have

Delivery Capital

What the organisation learns, across engagements.

Today it lives in:

- One senior consultant's head.
- A Slack thread nobody can search.
- A repo on a laptop that left.

The only thing you own.
And nothing collects it.

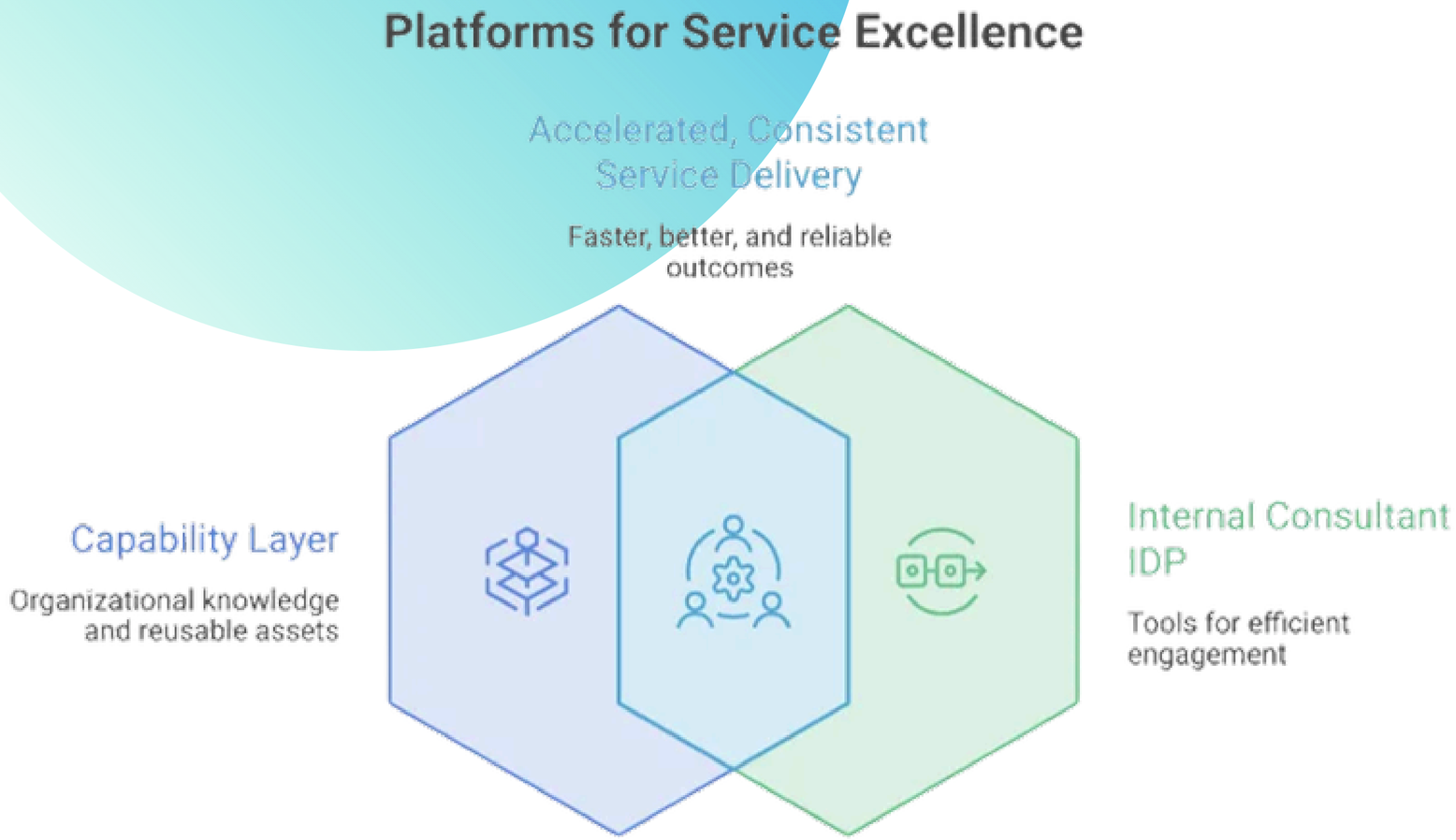




You can't standardise the environment.

So standardise the approach.

The shape of it



One travels to the client. One never leaves.

The thing you don't have

Capability Layer

Somewhere to put the learnings

The decisions you only make once, written so the next team can use them, and the module that goes with each.

A wiki records what someone did.
This is what you open on day one.



The other half

Internal Consultant IDP

Tools for efficient engagement.

Sandboxes to test patterns safely.
Demo clusters for scoping calls.
Onboarding in days, not weeks.

This half is an ordinary IDP.
Build it the normal way.





The same five decisions. Every engagement. From scratch.

Cloud. IaC. CI/CD. Observability. Monitoring.

Area 1 of 5

Cloud

The playbook assumes one cloud.
Chosen once. Yours.

Different cloud per client.
AWS. GCP with ArgoCD already running.
Air-gapped, with no tool access at all.

The client bought their cloud before they bought you.



What carries

Cloud

You can't standardise the cloud.
Only the choosing.

Reference architectures per use case.
Distributions you actually support.
What you need from their account.

Artifact: which distribution for a client like this, and the two you rejected.



Area 2 of 5

IaC

The playbook assumes one estate.
One module library. Yours.

You inherit someone else's.
Half written. Undocumented.
And you may not be allowed to refactor it.

You don't get a greenfield. You get a handover.



What carries

IaC

You can't own the estate. You can own the entry.

Module patterns that survive a move.
A migration path you've walked.
What to ask before touching state.

Artifact: a takeover checklist. What you read first, and what you don't touch in week one.



Area 3 of 5

CI/CD

The playbook assumes one pipeline standard, enforced.

Different CI per engagement.
Usually one you don't control and can't change.
Rebuilt from scratch, every time.

The pipeline isn't the asset. The decisions are.



What carries

CI/CD

You can't pick the CI. You can pick what you depend on.

Patterns that assume nothing below.
GitOps structure that ports.
What you need from their CI, day one.

Artifact: the day-one list of what you need from a client's CI.



Area 4 of 5

Observability

The playbook assumes one stack, instrumented once.

The client owns the telemetry.
You may have no production access.
You leave before the data matures.

You instrument it. Someone else reads it.



What carries

Observability

You can't own the stack. You can own the baseline.

The signals you always turn on first.
Dashboards that assume no backend.
SLOs you propose per engagement type.

Artifact: a baseline instrumentation set that drops into any environment.



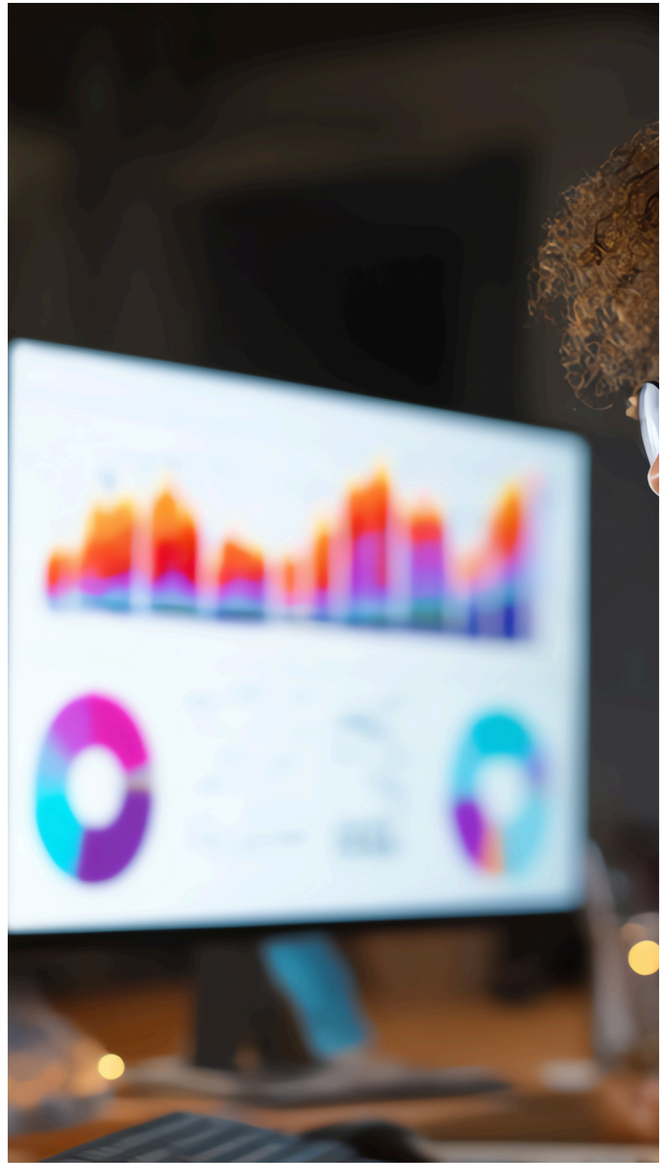
Area 5 of 5

Monitoring

The playbook assumes you run the alerts.

Someone else gets paged.
Often after you've gone.
Inheriting alerts they didn't write.

The person woken up can't fix the alert.



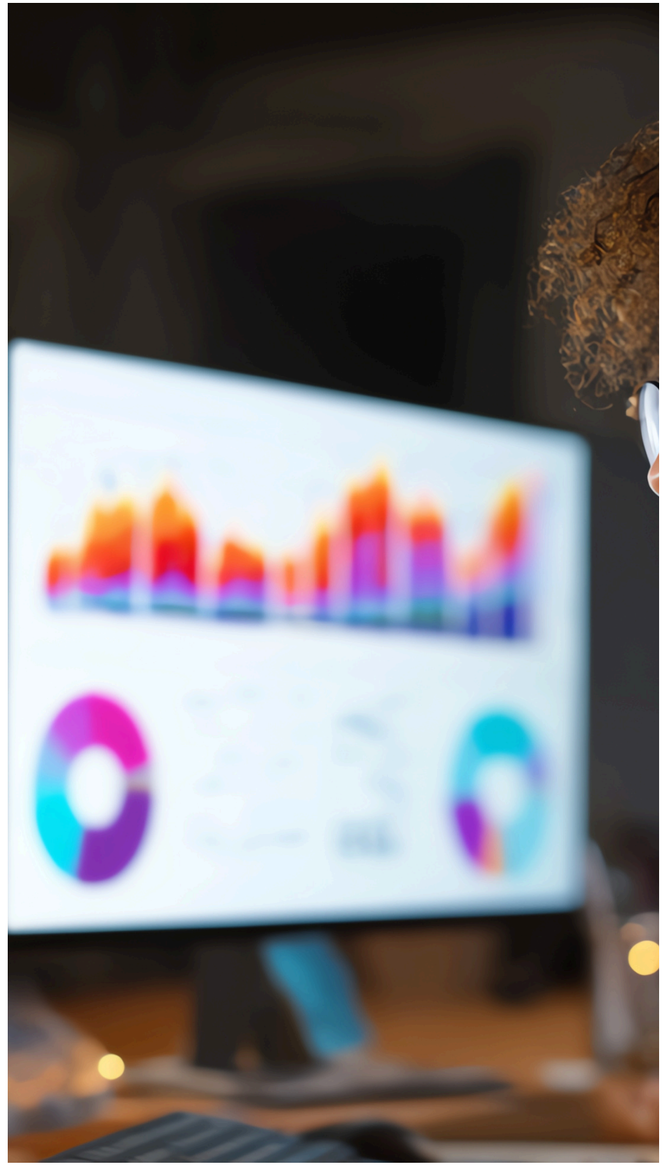
What carries

Monitoring

You can't stay on call. You can hand over cleanly.

Which alerts go with the handover.
Which ones you switch off on exit.
A named owner against each.

Artifact: an alert taxonomy, written before you need it.



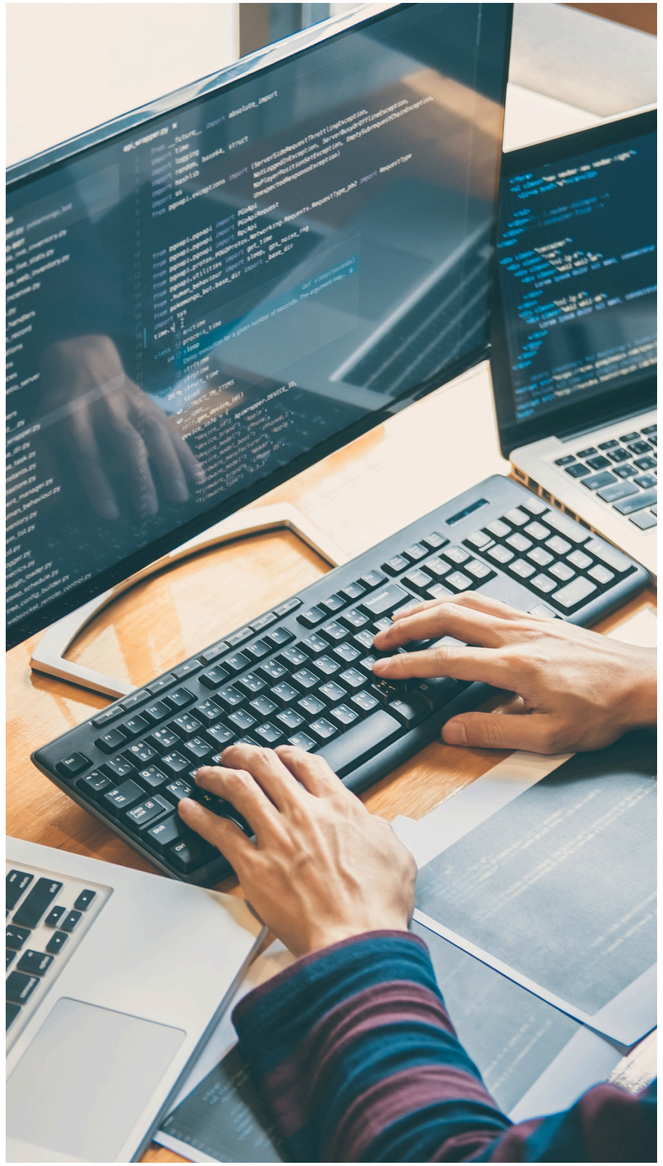
What one looks like

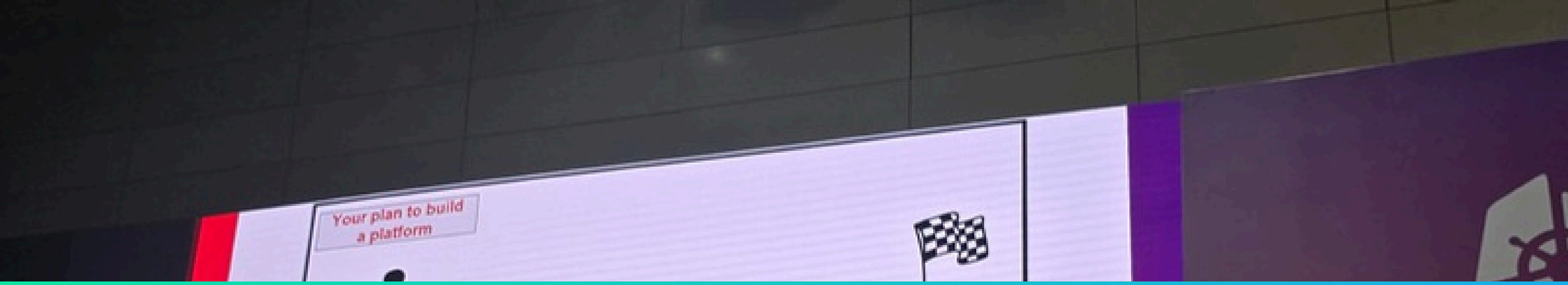
Decision record

Here’s an illustrative example

- Client shape: regulated, GCP, Argo CD running
- Picked: their Argo CD, our repo layout
- Ruled out: our own Argo install, Flux
- Why: two controllers fight over one cluster
- Day one: repo access, a namespace, an approver
- Alerts: their Alertmanager, our rules, an owner

The next team opens this on day one.





So why doesn't every services company have one?

And how would you know if it was working?

The hard part

Why nobody has one

Each one is built into how services firms run.

Ownership. Nobody holds the mandate.
Rotation. Whoever builds it rolls off.
Identity. Billable is valued. Internal isn't.

Pitch your platform to the leadership keeping these in mind.



How you'd know

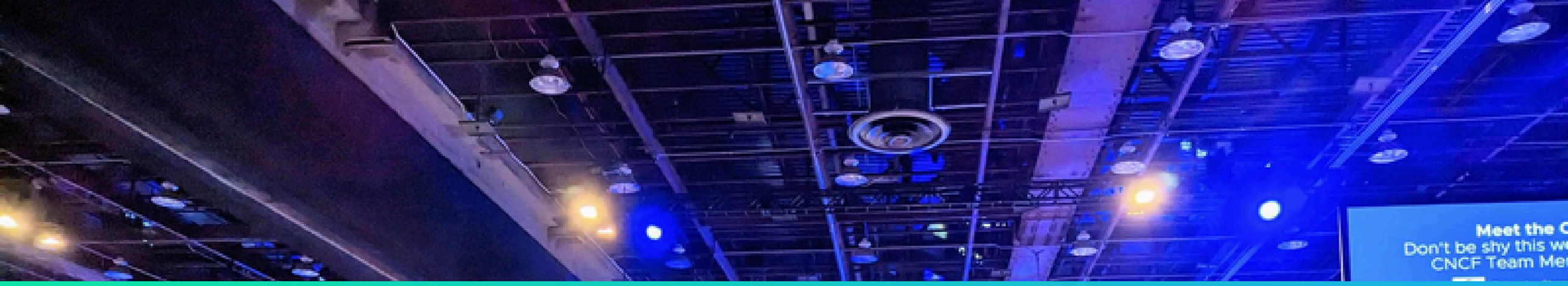
Not DORA

One consulting firm: delivery time down 40%.

Time-to-ramp.
Delivery consistency.
Proposal strength.
Engagement margin.

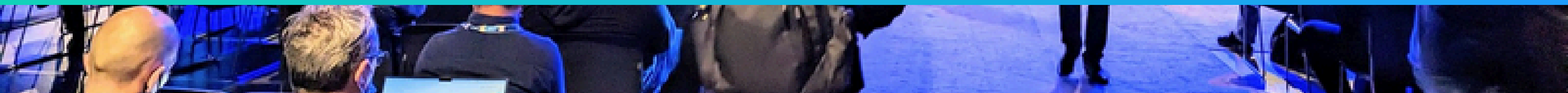
These are the numbers that get it funded.





Somebody joined a services company this week.

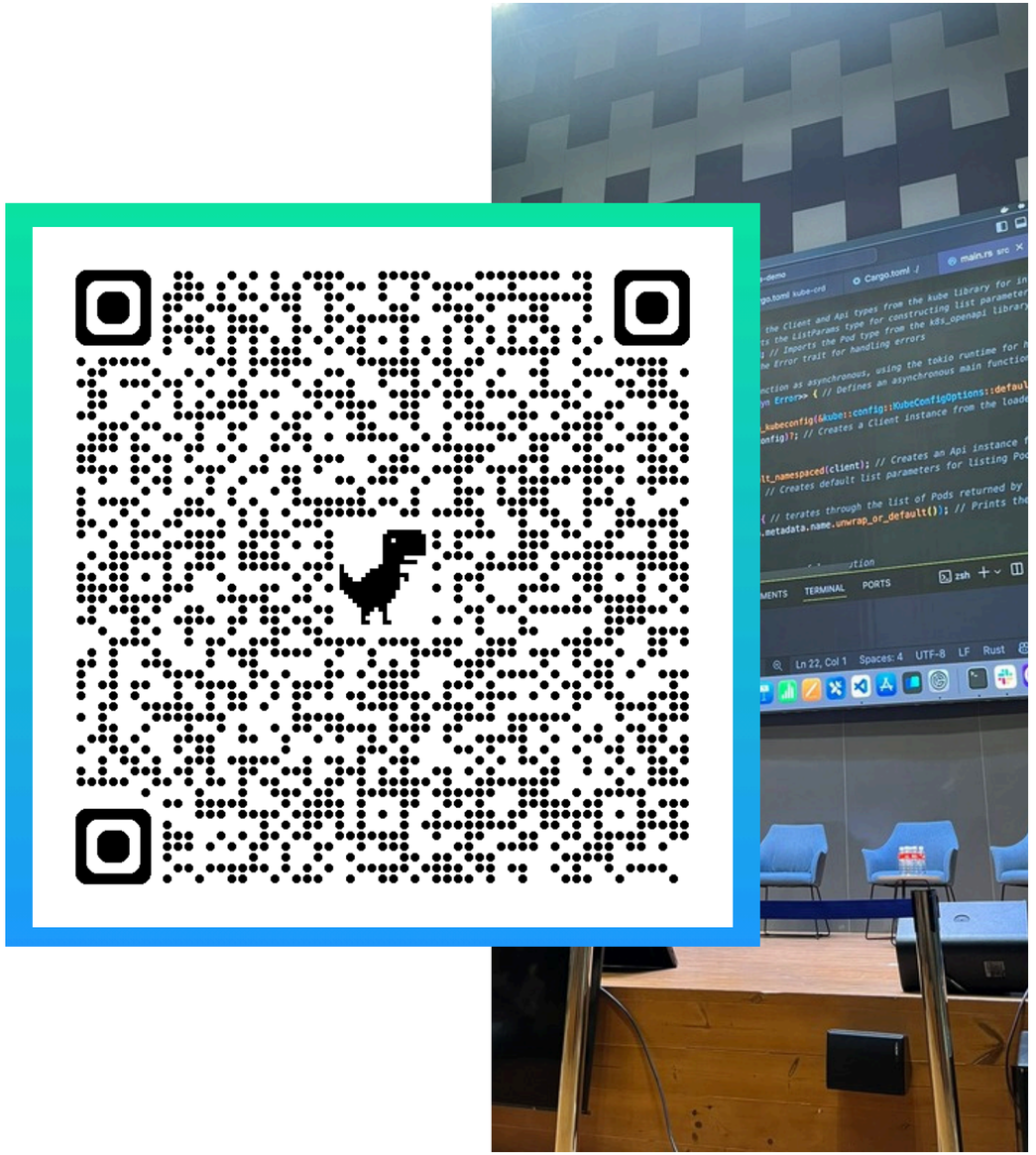
Five disconnected engagements, or the patterns from twenty?



Read the blog post

Platform engineering isn't a universal implementation pattern, it's an organizational optimization strategy.

Product companies optimize for consistency.
Service companies optimize for adaptability.



Join Us

Here's everything Platform Engineering that's happening at KubeCon →





Events by the Platform Technical Community Group:

- **Platform Coffees: every morning**
- **Platform Engineering Booth: Tuesday, 3- 7 PM**
- **Platform Lightning Talks**

Thank You

Let’s take the discussions offline →

