OPEN SOURCE SUMMIT INDIA · 2026

A TALK FOR PLATFORM ENGINEERS

AI as a Platform Engineer.

Explaining Kubernetes failures — not just detecting them.

Patterns we're piloting on a multi-region production platform.

SPEAKER

Paranitharan Kalaiselvan

Principal Engineer · Comcast

WHEN

Tue Jun 16 · 2:50 PM

Lotus 3 · #OSSUMMMIT

PART ONE

The problem.

Kubernetes gave us building blocks. Failures got harder, not easier.

BEFORE WE START

One number to hold on to.

MTTR · PER DEPLOY FAILURE

30 min → 5 min

Across a multi-region production platform. From `kubectl` spelunking to next action.

Cross-region debugging used to be four tabs and a Slack thread.
We changed that. Here's how.

WHERE WE ARE

Kubernetes is not a developer platform.

THE CLOUD FOUNDRY ERA

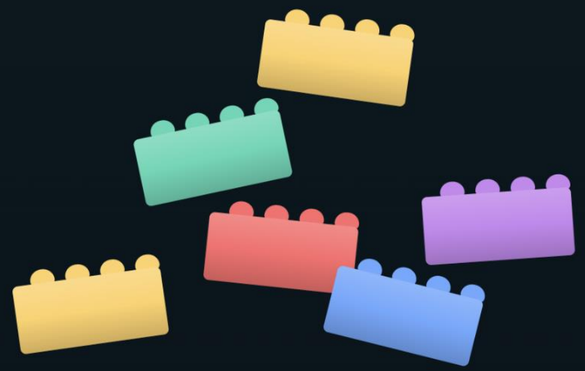
```
$ cf push
```

my-app
RUNNING

One command. The platform handled everything.

THE KUBERNETES ERA

```
$ kubectl apply -f ...
```



Legos. **You** build the house.

WHERE WE ARE

Kubernetes is not a developer platform.

THE CLOUD FOUNDRY ERA

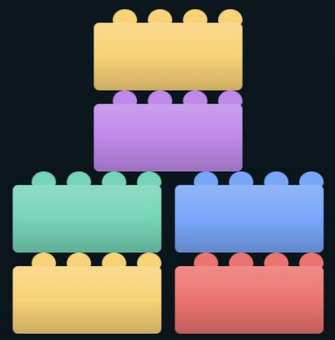
```
$ cf push
```

my-app
RUNNING

One command. The platform handled everything.

THE KUBERNETES ERA

```
$ kubectl apply -f ...
```



Legos. **You** build the house.

WHAT TEAMS ACTUALLY BUILD

The stack on top of Kubernetes.

Developer Interface	portal · CLI · git push
Platform APIs · Custom CRDs	Application · Database · Queue
Controllers & Reconciliation	ArgoCD · Flux · custom operators
Kubernetes	pods · services · networking · storage

THIS IS THE PART

Every platform team writes Custom CRDs.

They are the seams generic AI tools cannot see.

The goal: recreate cf push — with Kubernetes flexibility.

Every layer above K8s exists to give developers one thing back: a single command that ships code.

WHEN SOMETHING BREAKS

"Why did my app deployment fail?"

my-app · cluster-prod

```
$ kubectl describe deployment my-app

Events:
Type Reason Message
Normal ScalingReplicaSet Scaled up replica set my-app-7d4b to 2
Normal ScalingReplicaSet Scaled down replica set my-app-6c3a to 0
Warning FailedCreate Error creating: pods "my-app-7d4b-x9k2" forbidden:
exceeded quota PLATFORM
Warning Unhealthy Readiness probe failed: connection refused APP
Warning BackOff Back-off restarting failed container APP
```

Five events.
Three root causes.
Zero clarity.

KEY SIGNAL | **ResourceQuota exceeded** → unambiguously a **platform** issue. Every platform should expose this.

WHERE EXISTING TOOLS MEET THEIR LIMIT

Great tools.
Outside your platform.

OPEN SOURCE · EXCELLENT

k8sgpt · k8s-mcp-server

- ✓ Reads K8s objects and any CRD
- ✓ Explains generic K8s errors
- ✓ Great for cluster-level diagnosis

They **aren't trying** to know your controller logic, your ownership rules, or your platform's institutional memory. That's a different job.

THE NEXT LAYER UP

Built into your platform

- ✓ Everything generic tools do
- ✓ Knows **your** controller logic
- ✓ Encodes **your** ownership rules
- ✓ Uses **your** knowledge base

PART TWO

The approach.

Rules decide. AI explains. Determinism first, language second.

THE KEY INSIGHT

Let rules decide. Let AI explain.

AI DECIDES EVERYTHING

"I think this is a platform issue..."

- Non-deterministic
- Hard to trust
- Hard to audit
- Hallucinates under pressure

RULES DECIDE, AI EXPLAINS

```
if ImagePullBackOff + auth_err
→ PLATFORM
```

- ✓ Deterministic
- ✓ Auditable
- ✓ Trustworthy
- ✓ Plain-English when ready

Deterministic classification first. AI summary second.

HOW CLASSIFICATION WORKS

Platform checks go first — always.

◆ **Cluster unreachable?** NO → **YES → Platform · connectivity**

◆ **Platform resource missing?**
`e.g. ResourceQuota exceeded, ImagePullBackOff (auth)`
NO → **YES → Platform · controller**

◆ **Build failed?** NO → **YES → App · build · inspect logs / deps**

◆ **Deploy failed?** **YES → App · runtime → drill in ↗**

DRILL INTO APP FAILURES

Pull the right signal.

`CrashLoopBackOff` →

Pull application logs

Most likely an **app code** issue — stack trace, bad config, missing env var.

`Readiness probe failing` →

Check connectivity & deps

App is up but can't reach **DB / cache / upstream**. Often a **dependency** issue, not the app itself.

*Platform checks first — including **resource quota**. If the infrastructure is starving, the developer is **never** blamed.*

HOW TO WIRE AI INTO YOUR PLATFORM

The AI explanation layer.

SYSTEM PROMPT

PEEK INSIDE

Platform knowledge

```
## PRIORITY ORDER
1. Container log analysis
2. Configuration change detection
3. Platform error pattern matching
## CORE RULES
- Only summarize facts in evidence
```

USER MESSAGE

K8s signals (live)

events · container logs · ResourceQuota · HPA state · probe status

AI LLM temp = 0

STRUCTURED RESPONSE

ROOT CAUSE

"Resource quota EXCEEDED"

REMEDIATION

"Reduce requests · raise quota"

WRITTEN BACK TO

```
kind: Application
status:
  explanation: "..."
  remediation: [...]
```

WHAT V1 TAUGHT US

The lessons that built the guardrails.

LESSON 01 · THE HARD ONE

More freedom = more hallucination.

v1 had a loose prompt. The model made up resource names. We tightened progressively — priority order, schema-bound output, temperature zero. Hallucinations dropped to near-zero. **Boring prompts win.**

LESSON 02 · THE NON-NEGOTIABLE

PII scrubbing before the LLM.

Container logs have secrets, tokens, customer IDs. We added a **scrubbing layer** that strips PII before evidence ever reaches the model. Security and accuracy in one pattern.

LESSON 03 · THE PHILOSOPHY

Rules own classification. Always.

Early versions let the model classify ownership. Consistency drifted. Auditability vanished. **Now: rules decide owner, AI explains.** Determinism survives.

We iterated to honest. Honest is what shipped.

PART THREE

In practice.

From theory to a real failure — and what you can take home.

TWO TRIGGERS, TWO AUDIENCES

When the agent runs.

AUTOMATIC · AT DEPLOY TIME

When a deploy fails.

TRIGGER	Reconcile detects failure
AUDIENCE	Developer
SURFACE	`status:` on the CR

Unblocked without filing a ticket.

ON DEMAND

When the app misbehaves.

TRIGGER	Operator invokes diagnosis
AUDIENCE	Platform engineer / on-call
SURFACE	CLI & CR status

Cross-region diagnosis in 5 minutes.

Same agent. Same prompt. Different button.

One pattern. Two audiences who never have to wait for each other.

A REAL FAILURE, END TO END

A missing env var — caught across three regions.

STEP 01

Developer ships

```yaml
# new image tag
apiVersion: apps.example.io/v1
kind: Application
metadata:
  name: antimatter
  namespace: app-prod
spec:
  image: registry/antimatter:
    2026050506
  replicas: 3
```

Rolling deploy across 3 regions.

STEP 02

Pods crash everywhere

- Application CRD ✓
- Deployment ✓
- Pod · region-1 💥
- Pod · region-2 💥
- Pod · region-3 💥

`CrashLoopBackOff · all 3 regions`

STEP 03

Agent responds

APPLICATION_CRASH_LOOP · APP TEAM

Still running on the previous version.
No service disruption — this is a failed update.

ROOT CAUSE

Missing `DATABASE_URL` env var. App fails during FastAPI startup → exits → crash loop.

WHAT CHANGED

New tag `2026050506` dropped the env var. Previous (2026-03-16) had it.

REMEDIATION

1. Verify `env:` in your manifest
2. Add `DATABASE_URL` from `secretKeyRef`
3. Reapply — old pods serve until new ones are healthy

✓ Cross-region failure **collapsed** to one explanation ✓ Diff **caught** — without git blame ✓ Developer reads it on the **CR they already watch**

BEFORE VS. AFTER THE EXPLANATION LAYER

From "it just failed" to "here's why".

BEFORE — without AI AFTER — with AI

```
$ kubectl describe application awesome-app

Status:
Deployment Status:
- Region: region-1
Cluster: cluster-1
Health: false
State: CrashLoopBackOff
Exit Code: 137
Restart Count: 12

Events:
- Warning FailedCreate pods forbidden
- Warning BackOff Back-off restarting
- Warning Unhealthy Readiness probe failed

# What now? Quota? Memory? Probe? Image?

→ AI explanation layer engaged...


$ kubectl describe application awesome-app

Status:
Deployment Status:
- Region: region-1
Cluster: cluster-1
Health: false
- Region: region-2
Cluster: cluster-2
Health: false

Diagnosis:
Classification: APPLICATION_CRASH_LOOP
Owner: ApplicationTeam
Summary: Resource quota EXCEEDED
CPU: 400m / 1, Memory: 512Mi / 1Gi
Root Cause: |
Deployment exceeds namespace quota in both regions.
With 2 pods × 400m CPU / 512 MB, total usage exceeds
quota → pods evicted by Kubernetes.
Remediation:
```

STEP 1 · WITHOUT AI

Developer sees **CrashLoopBackOff**, exit codes, scheduling errors. **What now?**

STEP 2 · AGENT FIRES

Rules classify the failure. Evidence is scrubbed and packaged. LLM gets **structured signals**.

STEP 3 · WITH AI

Same CR. Same `kubectl describe`. Now it tells the developer the **why** and the **fix**.

CALLBACK TO WHERE WE STARTED

Remember those numbers?

MTTR · PER FAILURE

30 min → 5 min

From kubectl spelunking across regions, to next action.

PER DIAGNOSIS

8–16k

tokens

Structured evidence in. Plain English out.

COST

Linear in failures

Not pods. Not deploys. Cost grows with the thing you want to reduce.

That's where this lands. **One CR status field, four ordered rules, one well-shaped prompt.**

TAKE THIS BACK TO YOUR PLATFORM

A four-step framework.

01

Map failure domains

What can break?

02

Define ownership rules

Platform first. Always.

03

Normalize signals

One shape, many sources.

04

Add AI — last

Polish, not foundation.

WHERE THIS PATTERN GOES NEXT

Failures are just the start.

01 — SHIPPED · TODAY'S DEEP-DIVE

Diagnose failures

Explain why a deploy broke — the explanation layer we just walked through.

02 — EXPLORING

Read release notes

Parse upstream changelogs across your add-ons. Surface breaking changes. Flag impact on your CRDs before you upgrade.

03 — EXPLORING

Plan upgrades

Recommend safe target versions for cluster add-ons. Day-2 operations without the spreadsheet.

04 — EXPLORING

Onboard developers

Answer "how do I provision a database?" against **your own** CRDs and docs — not a generic LLM guess.

AI isn't replacing the platform engineer. It's accelerating one.

IF YOU REMEMBER NOTHING ELSE

Three things.

01 Explanation > Detection.

Knowing something broke is table stakes. Knowing **why**, **whose**, and **how to fix** — that's what matters.

02 Rules first. AI second.

Rules own classification. AI never decides ownership — it only explains the **why**.

03 AI is infrastructure — not magic.

Circuit breakers, structured prompts, fallbacks. With the right guardrails, AI is a reliable platform layer.

Every platform team has one job: make developers faster.
LLMs are the biggest leverage we've had since `cf push`.

NOW YOU

Questions?

Architecture · prompt engineering · rolling this out to skeptical developers — anything.

PARANITHARAN KALAISELVAN · COMCAST

Thank you.

SPEAKER

Paranitharan Kalaiselvan

Principal Engineer · Comcast

LET'S CONNECT

linkedin.com/in/paranitharan-kalaiselvan

OPEN SOURCE SUMMIT INDIA 2026 · #OSSUMMMIT