THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

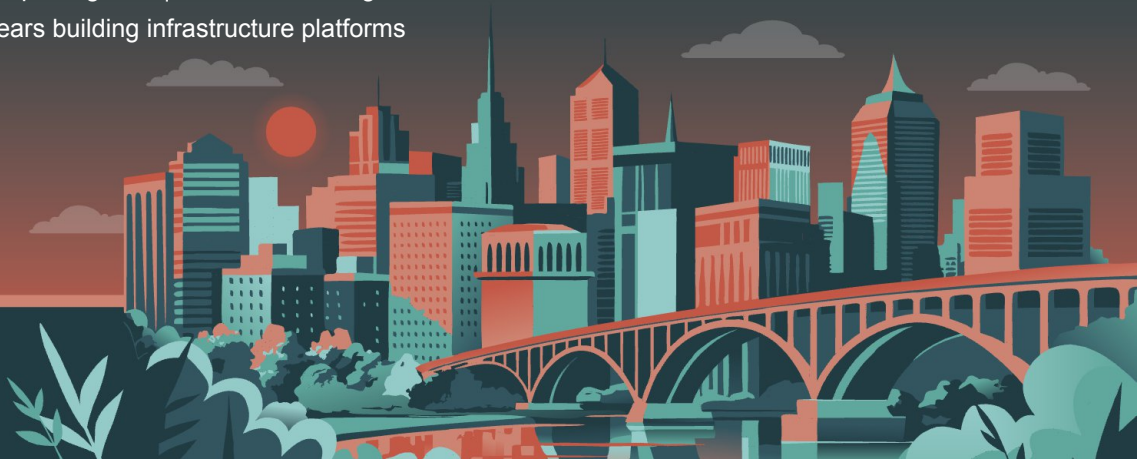
33 million invoices a year.
2 million small business merchants.

On a platform that broke at scale.

"We didn't migrate systems. We migrated assumptions."

Monolithic to Cloud Native: Lessons from Migrating Heroku to EKS at Scale

Mateen Ali Anjum
Staff DevOps Engineer | Phono Technologies Inc.
12 years building infrastructure platforms



THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

The Platform

Fast-Growing SaaS

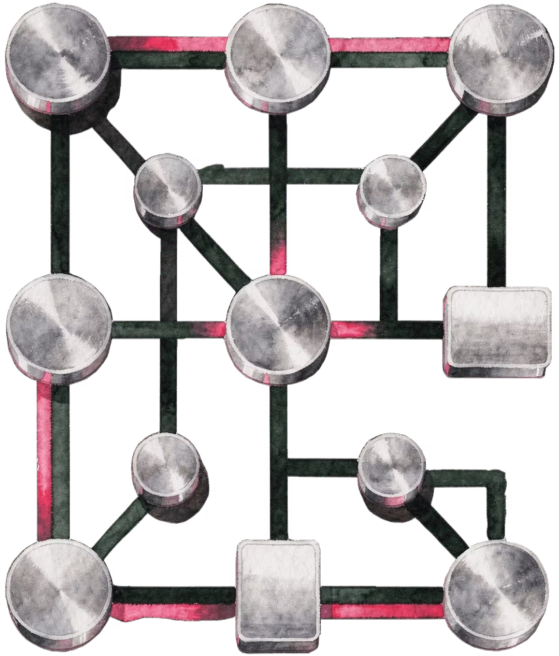
Invoicing platform with 47 Node.js microservices on Heroku. SQS for events, Redis for sessions.

Small Team

10-person engineering team — only 2 of us on platform. Enterprise clients with strict SLAs.

The Real Monolith

The title says "monolithic." The services were already micro. *The platform was the monolith.*



What Broke at Scale

The Numbers

700ms

API Latency p99

45min

Deploy Pipeline

12

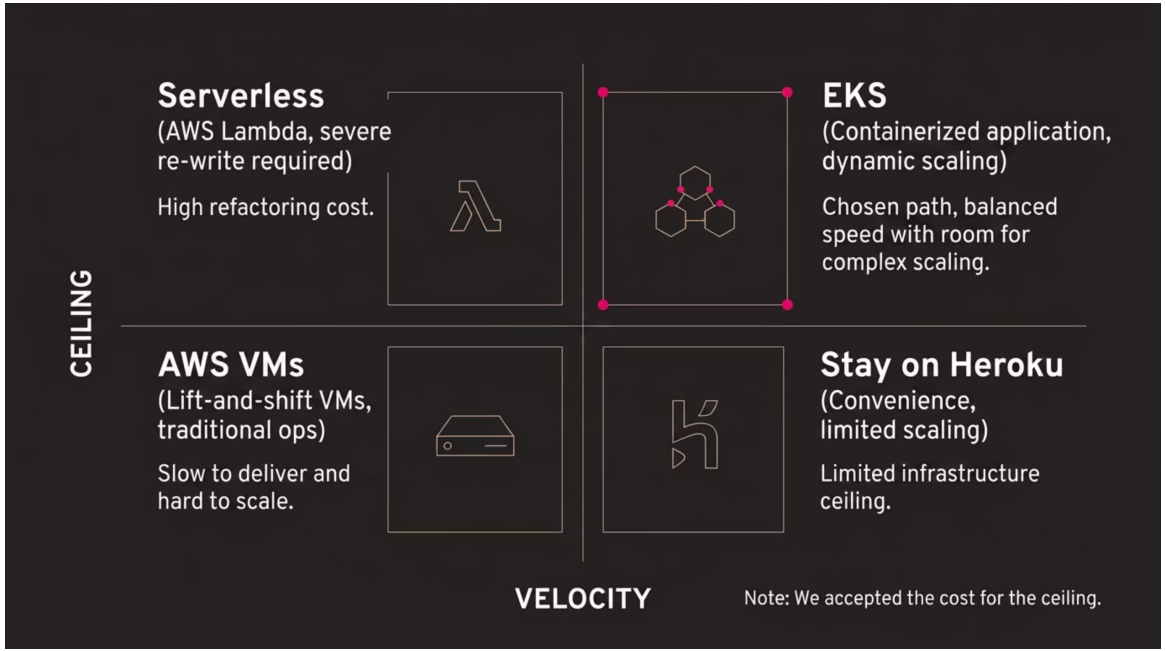
Monthly Incidents

Systemic Failures

- → Heroku dyno scaling ceiling hit
- → Enterprise SLAs we couldn't meet
- → No container observability
- → PaaS premium stopped buying operational leverage

The Decision Framework

We evaluated four paths on two axes: **velocity gain** vs. **ceiling for scale**.



EKS: Highest risk, highest ceiling, the seed planted for the final rubric.

We failed three times before it worked.

ATTEMPT 1

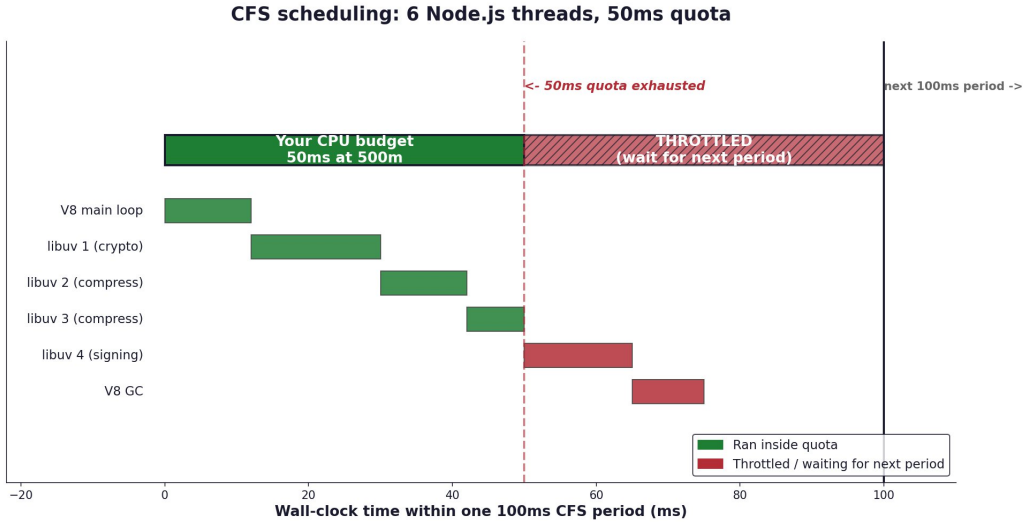
The Invisible Throttle

⚠ **SYMPTOM:** PDF service P99 went from 800ms to 9 seconds. Dashboards showed only 35% CPU.

Diagnosis

CFS scheduler enforces CPU limits in 100ms slices. At 500m, you get 50ms per period. Node.js libuv spawns 4 threads; V8 GC runs separately — 6 threads fighting for 50ms. Crypto stretches 15ms → 200ms.

Key metric: `container_cpu_cfs_throttled_periods_total`



Lesson: A 500m CPU limit isn't a number. It's a 50ms-per-100ms scheduling rule.

ATTEMPT 2

The DNS Amplification Tax

Heroku `resolv.conf`

```
nameserver 10.0.0.2
search ec2.internal
options ndots:1
```

EKS `resolv.conf`

```
nameserver 10.100.0.10
search default.svc.cluster.local
options ndots:5
```

10x

DNS packets per lookup

api.stripe.com (2 dots) under ndots:5

12M

Unnecessary queries/day

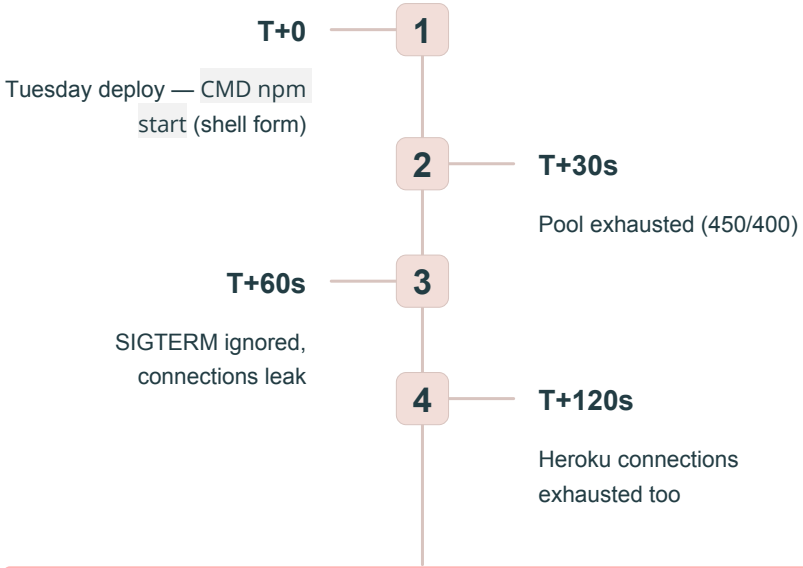
150K Stripe calls × 10 packets, across all integrations

Lesson: ndots:5 turns every short hostname into 10x DNS amplification.

ATTEMPT 3

The Connection Pool Death Spiral

Timeline



Root Cause

Shell form: CMD npm start
PID 1 = /bin/sh → swallows SIGTERM

Exec form: CMD ["node","server.js"]
PID 1 = node → receives SIGTERM

Config	Connections
Heroku PaaS	80
EKS naive	450
EKS + PgBouncer	~80
EKS + PgBouncer + SIGTERM fix	~80, clean shutdowns

Lesson: PID 1 is a contract. Shell form breaks the contract.

The Pattern

> Why didn't our dashboards catch any of this?

CFS

Defaults are invisible.

DNS

Amplification is multiplicative.

Connection Pool

PID 1 betrays you.

> *"We didn't migrate systems. We migrated assumptions."*

Incremental	Observable	Reversible

Why build a platform instead of raw AWS per team?

Because the four decisions below cost less than carrying them per-team.

The Four Decisions That Mattered Most

1

Traffic Shifting

Istio (CNCF graduated)

2

Observability

Prometheus + Grafana + Thanos

3

PR-Driven Infrastructure

Atlantis (MPL 2.0)

4

GitOps

Flux (CNCF graduated)

Supporting cast (CNCF/OSI-approved): Kubernetes, Helm, Backstage, cert-manager, external-dns, Terragrunt, PgBouncer
Security guardrails: IRSA (workload identity), external-secrets, Trivy (image scanning), NetworkPolicy, admission webhooks
Adopted-when-OSS: Terraform (BSL since 2023-08), Redis (RSAL/SSPL since 2024-03), Elastic (SSPL since 2021-03)
Not OSS by design: GitHub Actions, AWS SQS, ElastiCache, RDS — managed services, accepted trade-off.

Decision 1: Traffic Shifting (Istio)

> *"Istio is heavy. Our adoption was light."*

Rejected Alternatives

- ~~DNS-based routing~~ — no fine-grained control
- ~~ALB weighted rules~~ — missed our end-to-end rollback SLO
- ✓ **Istio VirtualService** — chosen

Canary Flipbook

Rollback at any step: 1 config change shifts traffic back in seconds. No redeploy. No DNS propagation. mTLS came free with the mesh.

Decision 2: Observe Before You Migrate

> *"You cannot migrate what you cannot measure."*

Prometheus + Thanos

Long-term metric storage across clusters for durable historical comparison.

Elastic Stack

Centralized logging with structured query across both environments.

Grafana Side-by-Side

Same metrics for Heroku and EKS displayed simultaneously during migration.

2-Week Baseline

Two full weeks before moving the first byte.
Baselined: p99 latency, error rate, DB connection count, DNS query rate, CFS throttle ratio, deploy lead time.

Decision 3: PR-Driven Infrastructure (Atlantis)

> *"Who ran apply from their laptop?" — On-call engineer, 2am*

01

Open a PR

Atlantis runs `terraform plan` automatically

02

Review the Diff

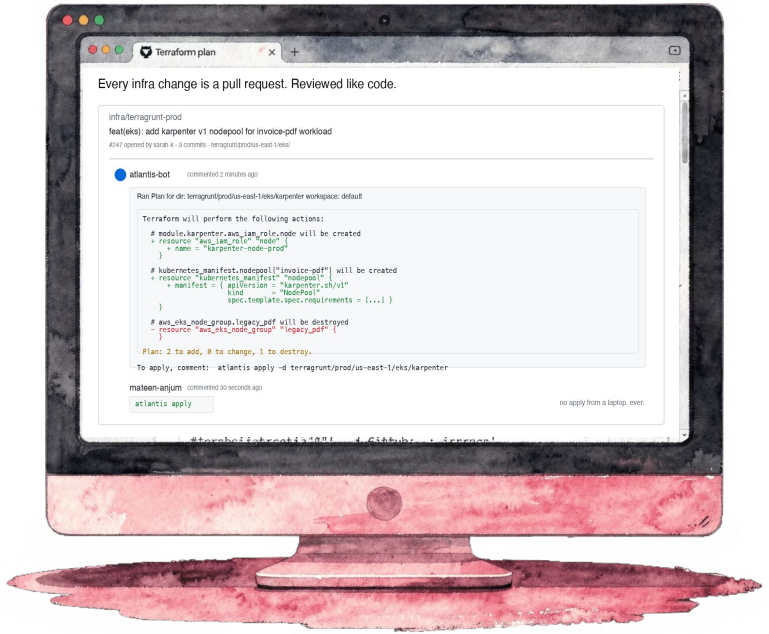
Plan output appears directly in the PR comment

03

Comment "atlantis apply"

Executed and fully audited — no laptop applies

Self-service for the team. No Mateen-as-bottleneck.



Decision 4: Deploys Are Git Commits (Flux)

HelmRelease CRDs

Declarative deploys, Kustomize-native — no imperative scripts.

Drift Detection

Auto-corrects manual `kubectl apply` — cluster state always matches git.

Team Adoption

Within a month, everyone worked through git. No exceptions.

Git Log (real)

Deploys are git commits. Rollbacks are git reverts.

```
infra-flux/HEAD on main

$ git log --oneline -5

a4f1c92  feat(invoices-api): bump to v1.4.7   (Mateen Anjum, 12 minutes ago)
c87b2d1  Revert "feat(invoices-api): bump to v1.4.7"   (Sarah K., 8 minutes ago)
e3b9a4d  feat(reminders-svc): bump to v2.1.0   (Diego M., 2 hours ago)
9f4d61a  chore(flux): rotate kustomize-controller image   (Mateen Anjum, yesterday)
220e7c5  feat(payments): bump to v3.8.2 + new HelmRelease values   (Sarah K., yesterday)

$ flux get helmreleases -A | wc -l
847        HelmReleases reconciled in last 30 days
```

Every deploy. Every rollback. In git.

Connection Pool Death Spiral: Revisited

Layer 1: Connection math

PgBouncer transaction-mode pooling
80 multiplexed conns vs 450 raw

Layer 2: SIGTERM contract restored

```
CMD ["node", "server.js"]  (exec form)
PID 1 = node, receives SIGTERM
```

Layer 3: SIGTERM handler

Graceful shutdown:
pg client closes pool, drains in-flight queries

Layer 4: Health checks decoupled

/healthz no longer queries Postgres
No more cascade-restart loop

Database cutover: dual-write + checksums + tested at full prod scale

1) Both DBs receive writes via thin pg-client wrapper
2) Background checksum compares both DBs continuously
3) Threshold breach -> halt + alert (we never tripped)
4) Cutover with full production-scale data, not a sample

Cutover was anticlimactic. That's exactly what we wanted.

The results.

Before and After

90%↓
API Latency p99
700ms → 70ms

91%↓
Deploy Time
45 min → 4 min

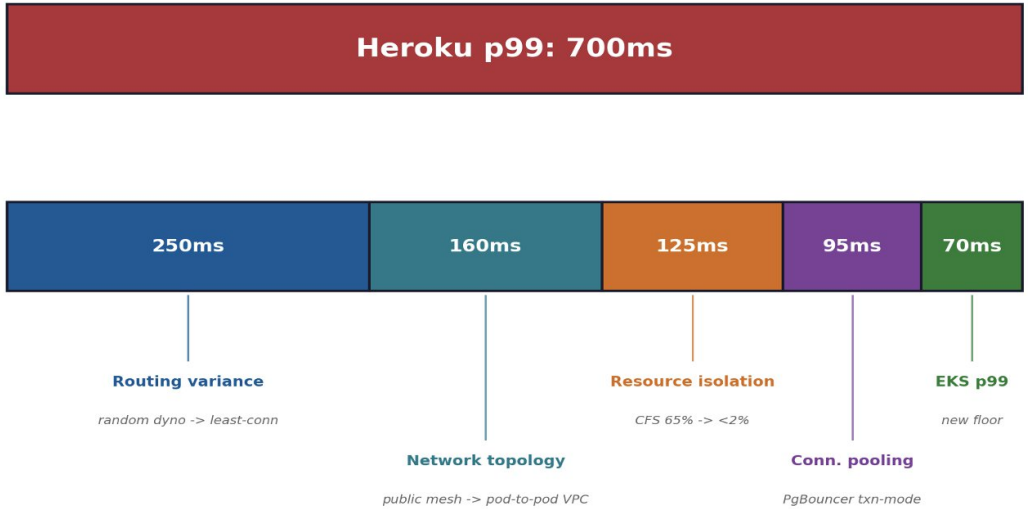
83%↓
Monthly Incidents
12 → 2

50x↑
Deploy Frequency
2/wk → 15/day

60%↓
Cost (Infra only)

30 days before vs 30 days after · production deploys · pager-triggering incidents · infra-only cost

Where the 630ms Went: Latency Attribution



*±10–15ms p99 noise floor

Engineering time absorbed: 2 platform engineers, 5 months full-time. 8 application engineers, ~30% of their time. Nothing is free.

After the Migration: Developer Experience

> *"Simple for developers means complicated for the platform team."*

Heroku: $ git push heroku main
Backstage: Portal → Scaffolder → Git PR → Flux deploy → Service running in < 5 mins

- K8s complexity was **our** problem, not the developers' problem
- Built an Internal Developer Portal with **Backstage** (CNCF Incubating)
- Self-service: new services in **5 minutes** via Scaffolder templates
- Software Templates: 47 templated services, each with vetted defaults

What Almost Stopped Us

Istio Sidecar Startup

Sidecar injection added 8 sec to pod startup. Fix: tuned readiness probe timeouts across all services.

Flux Reconciliation Windows

Reconciliation during peak hours triggered rolling restarts. Fix: scheduled reconciliation windows.

cert-manager TLS Rotation

TLS rotation broke active connections. Fix: graceful connection draining (should have had it from day one).

Compliance & DR: PCI-DSS scope re-mapped, re-certified without finding · Active-passive across two AWS regions, RPO 15 min / RTO 30 min, tested quarterly

Still working on: Backstage cost-attribution dashboards · Istio Ambient mode evaluation

Migration is not over. It's a beginning.

What We Gave Back

49+

CNCF Contributions

12 issues + 16 PRs + reviews in 2026 (DevStats verified)

22+

Merged OSS PRs

Across 14 projects in the past 3 months

Observability (7)

Jaeger UI ×3, OpenTelemetry C++ ×2, .NET, otel-arrow

Kubernetes (7)

KEDA, k8s/website ×2, Agones ×2, Rook, Kmesh

Security + ID (5)

SPIRE ×2, cert-manager, external-secrets ×2

Dev Tools (3)

ko (image builds), go-task ×2

> *"this is a super cool contribution"* — @SgtCoDFish, cert-manager maintainer (PR #8651)

Open Source Is the Equalizer

> *A 2-person platform team in Ontario, Canada running the same infrastructure stack as companies 100× our size. That's only possible because thousands of contributors built the tools we stand on.*

10→100

Engineers on Platform

47→100

Services

2→2

Platform Engineers

When to Migrate / When NOT to Migrate

> *"git push heroku main" is still the best deploy UX I've ever used. Heroku built for me without my having to know how. Heroku still powers thousands of production apps including names you know. Nothing wrong with it.*

✔ MIGRATE if you have:

- 2+ platform engineers
- Steady scaling pressure
- Engineering K8s exposure
- A PaaS limit you've actually hit
- Budget to run two platforms during cutover

🚫 DON'T MIGRATE (yet) if you have:

- Solo platform owner
- Steady-state workload
- Zero K8s exposure, no time to learn
- PaaS still meets your needs

> If your team isn't ready for the highest-risk highest-ceiling option, that's not a failure. That's a correct read of your situation.

TL;DR + Thank You

> *"We didn't migrate systems. We migrated assumptions."*

Scan for resources

- **Every platform hides a different class of failure**
- **Instrument before you migrate, not at migration time**
- **Reversible beats fast**

Thank you. Questions?

phonotech.ca/ossna26