komodor

Why Your Kubernetes Cluster Will Fail: Lessons from 1 Million Real-World Incidents

SCALE 23x | 2026

Hello SCALE 23x Pasadena! I'm Mickael Alliel 👋

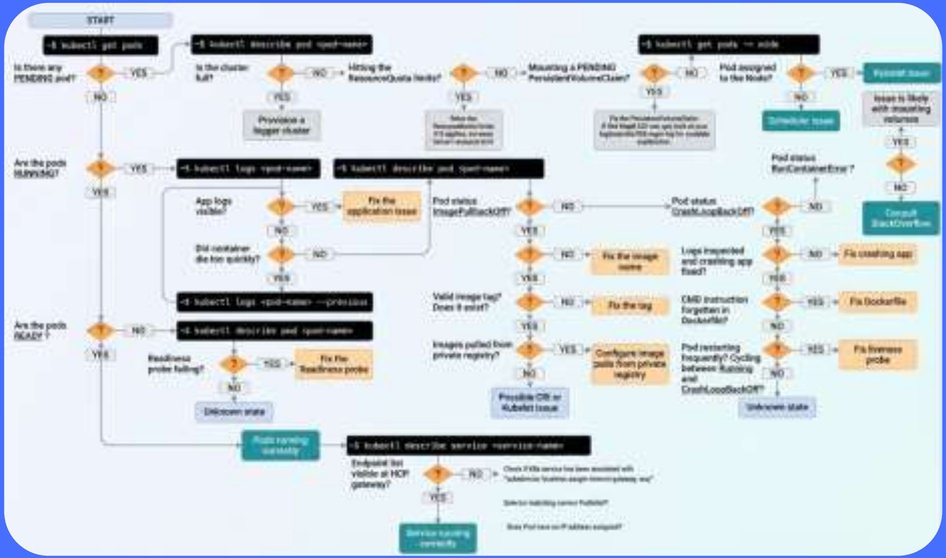
- R&D Tech Lead @ Komodor
- Over a decade of on-call experience
- Tech enthusiast & OSS aficionado
- I can count to 12 with my eyes closed

Agenda

- The Reactive Chaos
- The Million Incident Reality
- The Six Failure Categories
- The Predictability Paradox
- The AI SRE Evolution

komodor

Origin Story: The Reactive Chaos



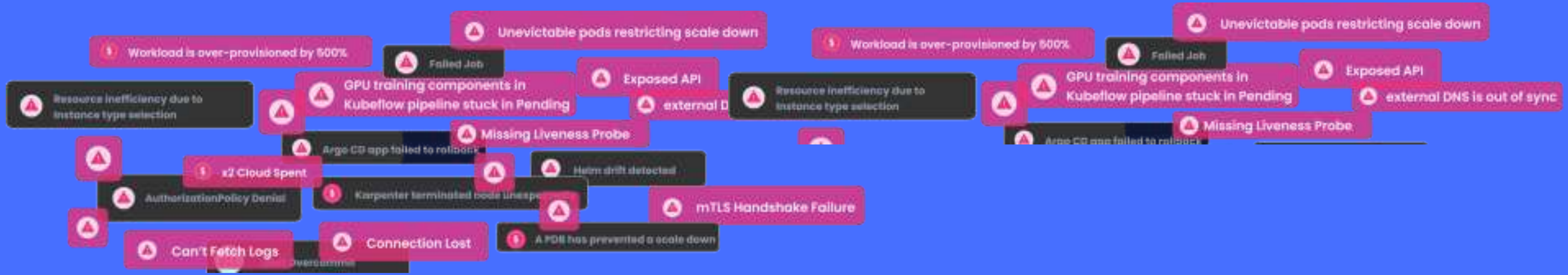
This is a mundane, vanilla, Kubernetes troubleshooting flow…and ***IT SUCKS***!

Especially at 3am when you're not even sure where to start.

In real-life the troubleshooting flows are even more convoluted and sprawling.

Origin Story: The Reactive Chaos

- ~2 years ago we set out to examine how AI can enhance K8s troubleshooting for our customers
- The goal was to create a deterministic runbook for common issues and automate the RCA
- To do that we analyzed over 1,000,000 incidents across dozens of different environments
- We discovered that most incidents fall into one of 6 main categories
- And we envisioned a predictive model that would eliminate the chaos of troubleshooting blindly



komodor

Emerging Patterns: The Six Failure Categories

Resource Exhaustion

- OOMKilled pods
- Memory leaks
- GPU thermal throttling
- CPU throttling
- CPU limits
- Node resource starvation
- Disk pressure evictions

Image & Deployment

- ImagePullBackOff
- Stuck rollouts
- Wrong architecture
- Registry rate limiting
- Image layer corruption
- Init container failures

Config & Secret mgmt.

- Secret rotation breaking apps
- ConfigMap drift
- Env. variable injection failures
- Volume mount permissions
- Base64 encoding errors
- Config validation failures

Cascading Failures

- Multi-pod failures
- Dependency chains
- Network policy blocking communication
- DNS resolution failures
- Shared PVC corruption
- Node failure avalanche

Storage & Persistence

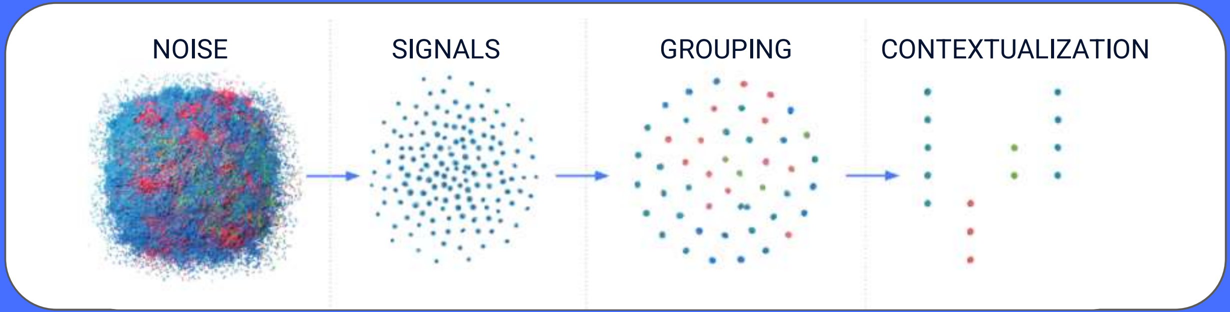
- PVC conflicts
- CSI driver issues
- Volume attachment timeouts
- Filesystem corruption
- StorageClass provisioning fails
- Snapshot/backup failures

App vs Infra

- CrashLoopBackOff
- GPU XID errors
- Network connectivity
- Performance degradation
- Health check false positives

Lessons Learned: The Million Incident Reality

- Very quickly we learned that the reality of production environments is much, much more complex
- Clusters in the wild behave in unpredictable ways and vary significantly between organizations
- The same K8s errors can stem from very different root-causes or require a different solution
- At a certain scale predictability breaks and deterministic approaches fail
- A contextual layer is essential to make AI useful in real-life troubleshooting scenarios



komodor

komodor

Deep-Dive

A Few Real-Life Examples

Prevention Evades: The Predictability Paradox

Let's take a look at '***Resource Exhaustion***' incidents - Same error but a different fix:

```
● Bash(kubectl describe pod -l app=cache-loader -n oom-investigation | grep -A3 "Last State:")
  └      Last State:     Terminated
           Reason:       OOMKilled
           Exit Code:    137
           Started:      Mon, 09 Feb 2026 22:59:59 +0200

● Bash(kubectl describe pod -l app=event-processor -n oom-investigation | grep -A5 "Last State:")
  └      Last State:     Terminated
           Reason:       OOMKilled
           Exit Code:    137
     … +3 lines (ctrl+o to expand)

● Everything is working as planned:

  cache-loader — OOMKilled immediately on startup (no Finished time visible yet because it dies so fast). CrashLoopBackOff with
  rapid restarts. Fix: increase memory limit to 256Mi.

  event-processor — Ran for ~13 seconds before being OOMKilled (Started 22:59:42, Finished 22:59:55). Gradual memory leak pattern.
  Fix: fix the application code to release accumulated data.

  Both pods are in CrashLoopBackOff with OOMKilled reason, exhibiting the distinct diagnostic patterns described in the plan.
```

Prevention Evades: The Predictability Paradox

Let's take a look at '***Cascading failure***' incidents - Same error but a different root-cause:

Current Pod Status

Pod	Ready	Status	Restarts
redis-cache	1/1	Running	0
order-processor	0/1	CrashLoopBackOff	6
notification-service	1/1	Running	6
data-store	1/1	Running	0
etl-worker	0/1	CrashLoopBackOff	6
dashboard-api	0/1	CrashLoopBackOff	5

Chain 1: Order Processing Pipeline

```
notification-service ──HTTP──> order-processor ──TCP──> redis-cache
   (crashing)                     (crashing)              (Running ✓)
```

Root Cause: The redis-cache **Service** has selector app: redis, but the pod has label app: redis-cache. The Service resolves to zero endpoints, so order-processor gets "connection refused" when connecting to port 6379. notification-service then crashes because order-processor is down.

Fix: Patch the Service selector:

```
kubectl patch service redis-cache -n cascade-failure -p '{"spec":{"selector":{"app":"redis-cache"}}}'
```

Chain 2: Data Analytics Pipeline

```
dashboard-api ──HTTP──> etl-worker ──HTTP──> data-store
  (crashing)              (crashing)            (Running ✓)
```

Root Cause: The data-store pod returns **malformed JSON** from its /query endpoint (trailing comma: "value": 41.8,}). It returns HTTP 200, so it looks healthy, but etl-worker fails on json.loads() with a JSONDecodeError. dashboard-api then crashes because etl-worker is down.

Fix: Fix the /query response in the data-store deployment – remove the trailing comma from the JSON payload so it becomes valid JSON, then redeploy.

Prevention Evades: The Predictability Paradox

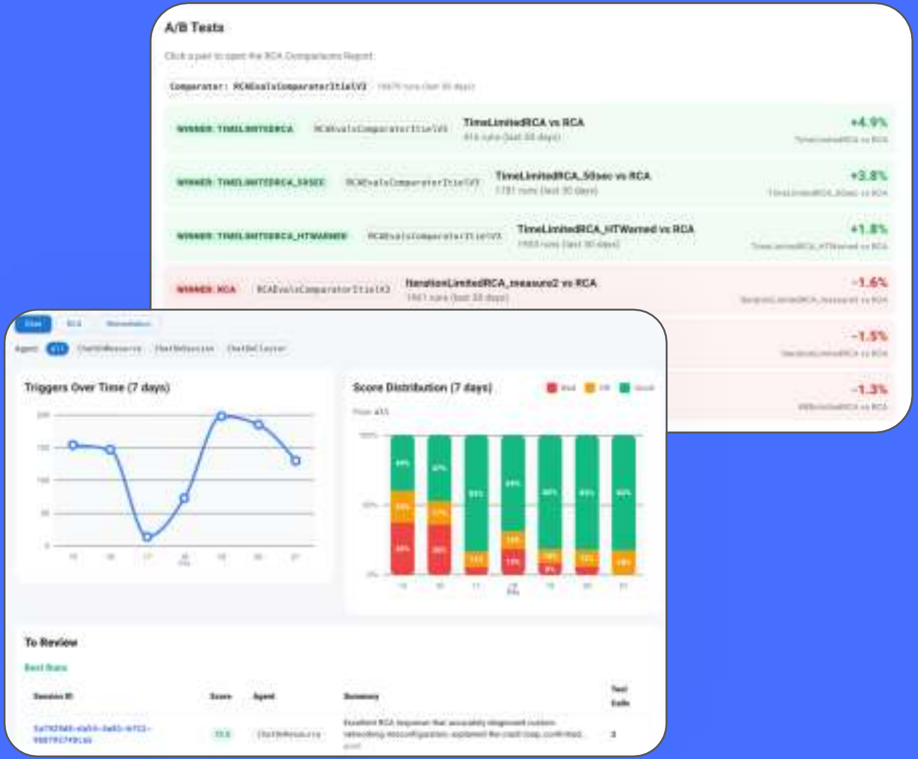
Let's take a look at '***App vs. Infra***' incidents - Same error but a different root-cause:

```
Namespace: restart-investigation

Deployment 1: metrics-api — Infra misconfiguration
- Healthy Python HTTP server on port 8080, serving /healthz → 200 OK
- Liveness probe misconfigured to check port 9090 (nothing listens there)
- Events show: dial tcp ...:9090: connect: connection refused
- Logs look clean — the app itself is perfectly healthy
- Fix: Change probe port from 9090 → 8080

Deployment 2: report-generator — Application bug
- Python HTTP server on port 8080, works fine for ~20 seconds
- After 20s, a blocking operation on the main thread makes the server unresponsive
- Liveness probe correctly configured on port 8080, but the app stops responding
- Events show: context deadline exceeded (Client.Timeout exceeded while awaiting headers)
- Logs show: WARNING: Report generation blocking main thread, request will hang...
- Fix: Move the blocking operation to a background thread

Both pods are accumulating restarts with "Liveness probe failed" events, but the root causes (and error
 messages) are completely different.
```

komodor

The runbook didn't work against real-world incidents

- Divergence level too high from one incident to another
- Handles new context poorly
- Each change in the runbook caused to much noise and regression - very hard to reiterate
- Different organizations (or even different teams) handle investigations different
- When 80% of the cases involve some unique aspects - it's no longer an edge case



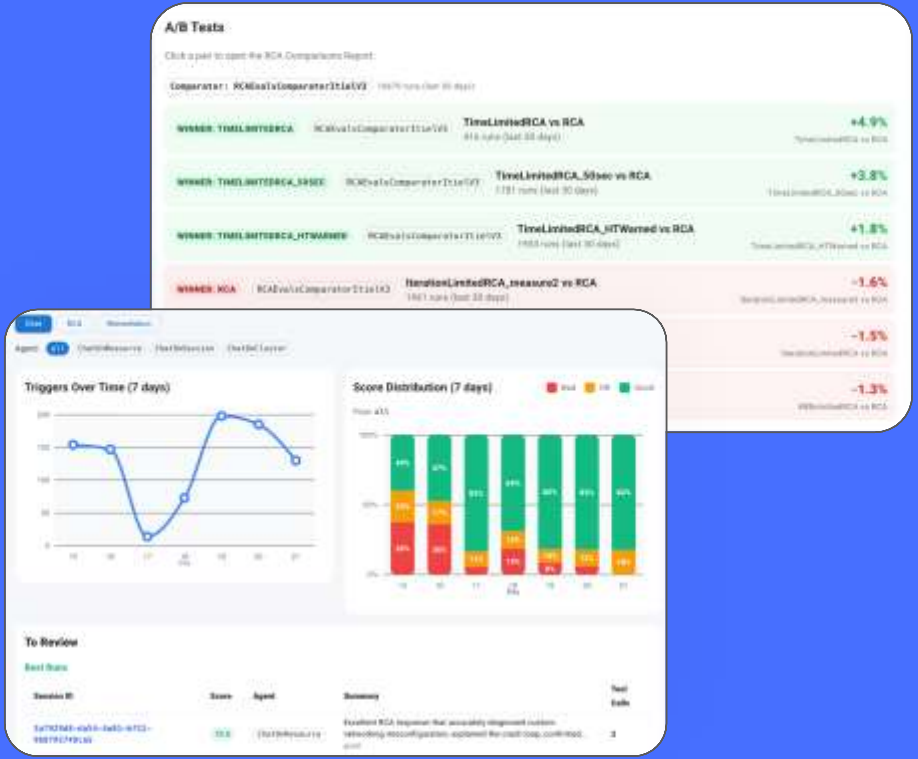
So have we failed?

So have we failed?

(It's a conference talk so no 😅)

The Solution: Agentic AI SRE

- Context(!) - handles large (but not too large) context really well.
- Organizational knowledge base and architectural blueprints as the base for investigations
- Multi-agentic with narrow focus on specific tasks and deep domain expertise (e.g helm agent)
- Evals: LLM-as-a-Judge, Golden Scenarios, Failure Playground, Shadow Agents for fast iterations
- The benchmark: What would a good SRE do?



The Solution: Agentic AI SRE (aka Klaudia)

- Serves endless production clusters for investigations
- Expanding her capabilities on a daily basis
- Very easy to introduce changes which is a must given our world

The Solution: Agentic AI SRE (aka Klaudia)

- Serves endless production clusters for investigations
- Expanding her capabilities on a daily basis
- Very easy to introduce changes which is a must given our world

komodor

Final Thoughts

Things to Consider When Building or Buying an AI SRE

komodor

Thank You!

Try Klaudia AI For Free

komodor

Let’s connect on LinkedIn !