From Dashboards to Defenses: Building Autonomous Resilience at Scale

This talk explores strategies for building autonomous, self-defending systems that can proactively detect and mitigate issues before they impact users and the business.

From Dashboards to Defenses

- Growing system demands make **traditional dashboards + firefighting unsustainable**
- Reliability at scale requires moving **beyond observation → into resilience engineering**
- Shift focus: **from watching problems → to preventing and fixing them automatically**

- Goal: Build **self-defending systems** that can:
  - Detect issues autonomously
  - Contain problems before they spread
  - Mitigate impact without human intervention
- Outcome: Protect both **users and the business** proactively

The Reliability Challenge

2 a.m. Pager → Fatigue and burnout

Constantly being paged leads to engineer fatigue and burnout, impacting overall system reliability.

Dashboards Don't Act → Only show problems

Dashboards that only display metrics without triggering automated actions are not enough to maintain reliability.

Heroic firefighting doesn't scale at billions of requests

Engineers can save the day in a crisis, but at billions of requests, heroics can’t keep systems reliable

Mandate: Engineer reliability into the system itself

To achieve reliable systems at scale, reliability must be engineered as a core part of the system design, not an afterthought.

The key challenge is to transition from reactive, human-driven reliability processes to proactive, autonomous systems that can defend themselves and reduce the burden on engineers.

The Observability Gap

- **From Vanity Metrics**

  Transitioning from metrics that don't drive action to actionable signals that inform decision-making.

- **Humans can't parse millions of data points**

  The sheer volume of data makes it impossible for humans to manually analyze and respond to issues.

- **Old Way: Collect everything, hope someone looks**

  The traditional approach of collecting all data in the hope that someone will find it useful is ineffective and leads to information overload.

- **New Way: SLO-driven signals tied to user experience**

  Focusing on Service Level Objectives (SLOs) and using them to derive actionable signals that are directly linked to user experience.

- **Example SLIs: p95_request_latency, availability, checkout_success_rate**

  Defining relevant Service Level Indicators (SLIs) that capture key aspects of the user experience, such as request latency, availability, and conversion rates.

Production Observability — Proven Patterns

- **SLOs & Error Budgets**
  Use SLOs and error budgets to gate release velocity
- **RED / Golden / USE Signals**
  Implement consistent signals across services
- **Golden Dashboards**
  Maintain one dashboard per service, not dozens
- **OpenTelemetry-first**
  Standardize on metrics, traces, and logs
- **Exemplars**
  Use trace IDs in slow/error metrics for one-click debugging

Metrics, Traces & Logs that Scale

Metrics: Counters, Gauges, Histograms

Use low-cardinality, base unit metrics for scalability

Tracing: W3C trace context, tail/dynamic sampling, service maps

Leverage open standards and intelligent sampling to capture meaningful traces

Logging: Structured JSON logs

Design logs for scalability and security with structured formats and intelligent processing

Outcome: Correlated telemetry you can trust

Achieve a unified view of metrics, traces, and logs to drive reliable decision-making

By designing scalable, correlated telemetry using best practices for metrics, traces, and logs, you can build a robust observability platform that provides the actionable insights needed to ensure autonomous resilience.

Change & Client Visibility

Change-Aware Observability

Capture deploy and feature flag events as annotations to track changes

Client-Side Visibility

Monitor user experience metrics like LCP, CLS, crashes, and edge metrics like cache hit percentage and error rates

Capacity & Cost Forecasting

Leverage saturation metrics and team/service cost tags to predict resource needs and optimize spend

Incorporating observability, client-side visibility, and capacity forecasting allows you to proactively monitor the health and performance of your systems from both the backend and the user's perspective.

Detecting Trouble Before It Happens

Static thresholds fail

"Normal" usage patterns are always shifting, making static thresholds ineffective for detecting issues.

Capacity anomaly detectors

Machine learning models that learn usage patterns and detect deviations, identifying potential capacity issues early.

AI models for forecasting

Leveraging AI and time series analysis to forecast spikes in demand or resource usage, enabling proactive mitigation.

Browser error spikes

Monitoring client-side errors and using them as early warning signals

By combining anomaly detection, AI-driven forecasting, and client-side signals, you can build a robust system that identifies and addresses issues before they impact users and the business.

Alerts that Drive Action

Old Way: Alert → human → mitigation

Alerts in the old way required human intervention to diagnose and mitigate issues.

New Way: Alerts as control signals → direct automated action

In the new approach, alerts are used as direct triggers for automated mitigation and remediation actions.

Multi-burn-rate alerts: Fast + slow windows

Alerts use multiple time windows

Goal: Page only for user-impacting issues

The goal is to page humans only for the most critical, user-impacting incidents

By transitioning from human-driven alerts to automated, multi-faceted alerting that triggers direct mitigation, organizations can scale their reliability efforts and ensure that engineers are only paged for the most severe, user-impacting issues.

Tiered Mitigation Strategy

Tier 1: Safe, Reversible

Implement circuit breakers, throttles, and other mitigation actions that are safe and can be easily reversed if needed.

Tier 2: Progressive

Utilize more complex mitigation strategies like regional traffic shifts and human override options to handle more severe incidents.

Tier 3: Notify-only

For complex or novel incidents, leverage automated runbooks to guide human intervention rather than triggering immediate mitigation.

By implementing a tiered mitigation strategy, you can cover the 'easy 80%' of incidents with safe, automated actions, while reserving human intervention for the 'hard 20%' of more complex issues.

Resilience Design Patterns

- **Circuit Breaker**

  Prevent cascading failures by opening the circuit when downstream service is unhealthy

- **Bulkhead**

  Isolate failures in one part of the system from impacting other parts

- **Retry with Exponential Backoff**

  Prevent overload by retrying failed requests with increasing delay between attempts

- **Failover & Load Balancing**

  Route traffic around unhealthy nodes or regions to maintain availability

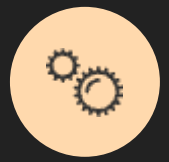
- **CQRS & Saga**

  Scale reads/writes independently and manage distributed transactions reliably

Delivery Pipelines as a Resilience Engine

Manual gates slow you down and add risk

Traditional manual review and approval gates in deployment pipelines introduce delays and increase the risk of errors.



Automated CI/CD: Canary → progressive rollout → auto rollback

Implement a fully automated CI/CD pipeline with canary deployments, progressive rollouts, and automatic rollback capabilities to accelerate releases and improve safety.

Shift Left: Human review moves post-deployment, not pre

Shift the focus of human review and approval from pre-deployment to post-deployment, allowing for faster release cycles while maintaining oversight.

Outcome: Faster velocity + safer releases

By leveraging automated CI/CD pipelines, organizations can achieve a higher release velocity with improved safety and reliability.

Delivery pipelines should be designed as a resilience engine, empowering teams to accelerate releases and improve safety through automation, canary deployments, and post-deployment human review.

Dynamic Rate Limiting

Static limits fail

Static rate limits fail against buggy clients or runaway jobs

Adaptive throttling

Adaptive throttling that learns baseline usage patterns per client/endpoint

Detect deviations

Automatically detects deviations from normal and contains workloads

Prevent incidents

Helps defend against ~30% of major incidents by automatically throttling

Dynamic rate limiting is a key resilience pattern that automatically contains workloads and prevents incidents by adapting to changing conditions, detecting anomalies, and throttling proactively.

The Scar Tissue Principle

Every safeguard born from a painful incident

Each reliability feature or automation was implemented in response to a specific issue or outage.

Culture: Postmortems → automation improvements

Culture of learning from incidents and using them as opportunities

Key question: Could this have been auto-detected/mitigated?

Team evaluates whether the issue could have been proactively detected and automatically mitigated.

Result: Toil engineered away, failures prevented

Continuously improving automation - eliminate manual toil and prevent future failures

The Scar Tissue Principle represents the organization's commitment to learning from every incident and using that knowledge to strengthen their automated resilience capabilities, ultimately preventing future failures and reducing toil.

Building Trust in Automation

Shadow Mode: Log-only

Run automated actions in log-only mode to observe and validate before taking action.

Suggest Mode: Recommend action for human approval

Provide automated recommendations for human review and approval before executing.

Autonomous Mode: Act independently with guardrails

Gradually transition to fully autonomous mode with safeguards and rollback capabilities.

Transparency & reversibility build confidence

Provide visibility into automated actions and ensure the ability to easily reverse them.

Trust is earned, not declared. By gradually building confidence through transparency, reversibility, and progressive autonomy, you can successfully transition to self-defending systems.

Business-Driven Resilience

UI Keep-alives

Detect degraded user flows early

JS Error Tracking

Frontend anomaly signals

Business KPIs

Cart abandonment, revenue dips trigger rollbacks

Reliability is about protecting both users and the business. By tying reliability metrics to user experience and key business outcomes, you can ensure that your system defenses are aligned with real-world impact.

The Blueprint for Self-Defending Systems

- Metrics → Actionable Signals
- Alerts → Closed-Loop Mitigations
- Deployments → Autonomous Pipelines
- Limits → Adaptive Safeguards
- Firefighting → Proactive Automation

“Stop watching, start engineering. Reliability is achieved through autonomous resilience.”

Reliability ≠ Pager duty

Reliability = Autonomous resilience

Every incident = automation opportunity

Step-by-step → Build systems that defend themselves