SRE AS A SERVICE — PLATFORM ENGINEERING & RELIABILITY

Always On.

The SRE playbook for platforms that can't afford to fail: SLOs and error budgets, observability, incident response, and the engineering discipline behind zero downtime — from the team that has kept Cardano infrastructure up since the network's foundation.

0
DOWNTIME ON CARDANO SINCE FOUNDATION

99.9%+
SLO-BACKED UPTIME TARGETS

24/7
MONITORING, ALERTING & ON-CALL

THE UNCOMFORTABLE TRUTH

Hope is not an operating model.

Every platform is "reliable" until the night it isn't. The difference between teams that stay up and teams that apologise is not luck or headcount — it's an operating discipline.

Site Reliability Engineering is what happens when you treat operations as a software problem: **engineers who run production — and write code to make it run itself.** Born at Google and battle-tested across two decades of the world's largest systems, SRE replaces "keeping the lights on" with explicit targets, measured reliability, and automation that removes the work instead of documenting it.

The stakes keep rising. Always-on customer expectations, real-money workloads, and blockchain-native infrastructure — where downtime means missed blocks, slashed stakes and lost revenue — have made reliability a boardroom number. Organisations adopting SRE practice report dramatic reductions in customer-facing incidents within the first year; the ones that don't discover their reliability posture from a status page, in public.

> **Reliability is a feature. It has a spec, a budget, an owner — or it has none of those things, and you find out which during an outage.**

This guide distils the SRE discipline into the decisions that matter for leadership: what to measure, what to automate, how to respond when things break, and what to demand from anyone — internal or external — who runs your platform.

IN THIS GUIDE

- **SLOs & error budgets** — reliability with a spec and a price
- **Observability** — the four golden signals, and alerts that mean something
- **Toil & automation** — everything as code, nothing done twice by hand
- **SREs who code** — why scripting on the fly changes outcomes
- **Incident response** — detect, mitigate, learn without blame
- **Buying SRE well** — what to demand from any provider

SLOS & ERROR BUDGETS

Reliability with a spec and a price.

"As reliable as possible" is not a target — it's an unlimited budget with no owner. SRE replaces it with three connected instruments.

SLI

Measure what users feel

A Service Level Indicator is a number that tracks the user's experience — request success rate, latency of successful responses, freshness of data. Not CPU. Not "server up". **What the customer would notice.**

SLO

Set an explicit target

A Service Level Objective is the promise: e.g. 99.9% of requests succeed each month. It should be as reliable as the business needs — **and no more.** Every extra nine multiplies cost; users can't tell the difference past a point.

ERROR BUDGET

Spend the difference

100% minus the SLO is the error budget — unreliability you're allowed. Budget healthy? Ship fast, take risk. Budget burned? **Feature work pauses, reliability work takes over.** The trade-off becomes a rule, not an argument.

What this changes for leadership

- **Reliability gets a number the board can read.** Not a feeling, not a dashboard of greens — a target, and performance against it.
- **The speed-vs-stability fight ends.** The error budget decides when to push and when to pause — no more escalation battles between product and ops.
- **Alerting gets honest.** Page humans when the budget is burning — user impact — not when a disk hits 80%. Meaningful alerts protect both sleep and response speed.
- **Vendors become accountable.** An SRE partner who won't contract against SLOs is selling effort, not outcomes.

> A target you never miss is not ambition — it's an SLO set too low. A target you always miss is fiction. Both waste money.

OBSERVABILITY

Monitoring tells you when. Observability tells you why.

You can't defend an SLO you can't see. Four golden signals cover most of what matters — and the discipline is in what you refuse to alert on.

SIGNAL	WHAT IT TELLS YOU	THE NUANCE MOST TEAMS MISS
Latency	How long requests take — the user's experience of speed.	Track successful and failed requests separately: a 500 error served in 10ms is fast and still a failure.
Traffic	Demand on the system — requests, transactions, blocks, sessions.	Traffic shape matters as much as volume: a spike that never came can be as alarming as one that did.
Errors	The rate of failed requests — explicit failures and quietly wrong answers.	The dangerous errors are the silent ones: stale data, partial responses, a node that's up but not in consensus.
Saturation	How full the system is — headroom before degradation.	Systems degrade before they fail. Saturation is your early warning — but it's a planning signal, rarely a paging one.

Metrics, events, logs, traces

Monitoring answers "is it broken?" Observability answers "why?" — by correlating metrics, events, logs and traces so an engineer can follow one bad request through the whole system at 3am without guessing. Build that path before you need it.

Alert on symptoms, not causes

Page a human only for user-visible impact or a burning error budget. Everything else is a ticket for daylight hours. Every noisy alert you tolerate trains your team to ignore the one that matters — alert fatigue is how well-monitored systems still fail publicly.

> The 3am page should mean one thing: users are hurting, or about to be. Everything else can wait for coffee.

TOIL & AUTOMATION

Everything as code.
Nothing done twice by hand.

Toil is manual, repetitive operational work that scales with the system — and it quietly eats engineering teams. SRE treats it as a defect with a budget.

Cap toil, then attack it

Google's benchmark: no more than 50% of SRE time on toil — the rest goes to engineering that removes the toil. Without the cap, operational load grows with the system until the team does nothing but firefight. The cap forces the investment that breaks the cycle.

The two-times rule

Any manual step executed twice — during an incident, a deployment, an upgrade — becomes an automation candidate immediately. Runbooks are a waypoint, not a destination: today's runbook is next quarter's script, and next year's self-healing workflow.

Infrastructure as code, all of it

The entire estate — environments, nodes, validators, networking, secrets, monitoring — defined in code: reproducible, auditable, reviewable. No snowflake servers, no undocumented manual steps, no knowledge that leaves in someone's head.

Immutable and reproducible

Environments are rebuilt, not repaired. If you can recreate any component from code in minutes, whole categories of risk disappear — configuration drift, botched patches, and the "nobody knows how this box was set up" server every estate hides.

> Every hour of toil you automate away is compound interest: it pays out on every deploy, every upgrade, every incident, forever.

The clean-handover test: if your operations partner vanished tomorrow, would you inherit a documented, code-defined, reproducible platform — or a pile of scripts and tribal knowledge to reverse-engineer? Ask before you sign, not after.

OPERATORS VS ENGINEERS

SREs who code change what's possible.

Most "SRE" providers staff operators: people who know the infrastructure and follow the runbook. When reality leaves the runbook — and it always does — the difference between an operator and an engineer is the outage's length.

WHY IT MATTERS AT 3AM

When the runbook runs out, we write code.

Our SREs don't just know the infrastructure — **they write scripts on the fly to get things done when the situation demands it.** A migration that needs a one-off data check across a thousand nodes. An incident where the fastest safe fix is a script that drains, verifies and rebalances — written, reviewed and run in minutes. An upgrade path the vendor never documented. Operators escalate and wait. Engineers ship the fix.

Every one of those on-the-fly scripts then follows the two-times rule: reviewed, hardened and folded into the automated estate — so the improvisation of this incident becomes the automation of the next.

What that looks like in practice

- **Incidents:** mitigation isn't limited to restart-and-pray. Custom tooling gets written mid-incident when it's the fastest safe path to recovery.
- **Migrations & upgrades:** one-off verification, backfill and cutover scripts — built for the job, not forced from generic tools.
- **Protocol changes:** blockchain-native work — node upgrades, validator operations, consensus quirks — handled by people who read the source, not just the docs.
- **Glue nobody sells:** exporters, checkers and integrations for the corners of your estate no vendor covers.

> The question for any reliability partner: when something happens that has never happened before, can your people build the fix — or only escalate it?

INCIDENT RESPONSE

Fast to mitigate. Ruthless about learning.

Incidents are inevitable; chaos is optional. The SRE incident discipline has four movements — and one cultural rule that makes them work.

MOVEMENT	WHAT GOOD LOOKS LIKE
1 · Detect	The system tells you before the customer does. SLO burn-rate alerts catch user impact in minutes; the incident clock starts on detection, not on the support ticket.
2 · Triage	Severity assessed against user impact, a single incident commander named, roles split — one person leads, others fix, someone communicates. Nobody debugs and coordinates at once.
3 · Mitigate	Restore service first, find root cause later. Rollback, failover, drain, rate-limit — or script the fix live when the runbook runs out. Every manual step performed twice gets automated.
4 · Learn	A blameless postmortem within days: timeline, contributing factors, and tracked actions that actually close. The system and process failed, not a person — that's what makes people tell the truth.

Blameless is a hard rule, not a vibe

The moment a postmortem asks "who", engineers stop reporting near-misses — and near-misses are your cheapest reliability data. Blameless culture isn't kindness; it's how you keep the information flowing that prevents the next incident.

Practice before it’s real

Game days and chaos experiments — deliberately injecting failure in controlled conditions — turn the first real regional outage into a rehearsed procedure. On-call rotations stay humane and sharp: rested engineers, meaningful pages, clear escalation.

> An incident you learned nothing from is one you’ve scheduled to repeat.

CHANGE & CAPACITY

Most outages are self-inflicted. Change is where reliability is won.

The majority of production incidents follow a change someone made on purpose. You don't get reliability by changing less — you get it by changing safely, constantly.

Progressive rollouts

No change hits 100% of production at once. Canary first, then staged expansion, with automated health checks against the SLIs at every step — and automated rollback the instant they degrade. The blast radius of a bad change should be a design decision, not a surprise.

Rollback as a first-class feature

Every change ships with its undo. If rolling back requires a meeting, you don't have a rollback — you have a second incident. Rehearse it like a fire drill; test restores, not just backups.

Capacity planned, not discovered

Saturation trends feed forecasts; forecasts feed scaling decisions before the traffic arrives. Load-test to know your true limits — the worst time to learn them is during the spike that matters. Automated scaling absorbs the routine; planning absorbs the exceptional.

Upgrades without drama

Protocol upgrades, hard forks, breaking dependency changes — the scheduled events that break unprepared estates. Rehearsed on replicas, scripted end-to-end, rolled out progressively. In blockchain, a missed upgrade window isn't embarrassment; it's missed blocks and slashed stake.

> **Freezing change doesn't protect reliability — it stockpiles risk and releases it all at once, later, in a bigger batch.**

The error budget closes the loop: when the budget is healthy, changes flow fast; when it burns, the bar rises automatically. Speed and safety stop being a negotiation and become a control system.

BUILD, HIRE, OR BUY

What to demand from anyone who runs your platform.

Senior SREs are scarce, expensive, and hard to keep challenged on a single estate. Buying reliability as a service can beat hiring — if you hold the provider to the same bar this guide sets.

DEMAND	WHY IT SEPARATES REAL SRE FROM REBADGED SUPPORT
SLO-backed accountability	Explicit reliability targets in the engagement — uptime, latency, correctness — with performance reported against them. Effort-based contracts are support tickets wearing an SRE badge.
Design-build-operate, one team	Whoever operates the platform should be able to build it — and vice versa. Reliability bolted onto someone else's architecture is always weaker than reliability engineered in from day one.
Engineers, not just operators	People who can write the script, the exporter, the migration tool — on the fly when needed. Ask directly: "when the runbook runs out, what do your people do?"
Everything as code, clean handover	The whole estate code-defined and reproducible, yours to take in-house whenever you choose. A provider who resists this is building lock-in, not infrastructure.
24/7 that's real	Follow-the-sun or genuine on-call rotations, symptom-based paging, incident command discipline, blameless postmortems you get to read.
Workload-native expertise	If you run blockchain infrastructure, demand node and validator operations experience — consensus, staking mechanics, protocol upgrades. Generic cloud ops learns those lessons on your stake.

> The outcome you're buying is a number on a dashboard the board can read: the platform stayed up, and nobody on your team had to think about it.

PROOF

This discipline, in production.

We have operated Cardano infrastructure since the network's foundation — with zero downtime. Nodes, relays and the platform around them: designed, built and run to explicit SLOs by the same engineers, through every protocol upgrade and hard fork the network has shipped. In a world where downtime means missed blocks and lost revenue, that record is the product.

0
DOWNTIME ON CARDANO SINCE FOUNDATION

99.9%+
SLO-BACKED UPTIME TARGETS

0
INFRA HEADCOUNT FOR YOU TO HIRE

WHY ICAN

Design · Build · Operate

We architect for reliability from day one, build the estate as code — cloud, nodes, validators, RPC endpoints — and run it to defined targets with observability and 24/7 on-call.

SREs who code

Our engineers don't stop at the runbook. They script fixes, tooling and migrations on the fly — then fold every improvisation back into the automated estate.

Blockchain-native

Node and validator operations, consensus behaviour, staking mechanics and protocol upgrades — expertise earned in production, on workloads where downtime is slashed stake.

Tell us what needs to stay up.

Tell us about the platform you're running — or the one you need built. We'll come back with a pragmatic plan to design, build and operate it to clear reliability targets.

INFO@ICANGROUP.CO.UK
+44 7470 473113
WWW.ICANGROUP.CO.UK