

— THE COMPLETE GUIDE

The Night Shift Handbook.

Why another dashboard won't cover the night shift, what the work between an alert and a fix actually consists of, and how to hand it to a crew of specialist coworkers — with autonomy you widen one pattern at a time.

12 chapters 3 frameworks 4 coworkers

CONTENTS

What this handbook covers

This guide is for the people responsible for keeping infrastructure running at 3am: heads of ITOps and infrastructure, SRE and platform leads, service-desk managers, and the operators of datacenter estates. It assumes you already have monitoring. The question it answers is who — or what — should be doing the work that monitoring generates.

01	Why more dashboards didn't help	03	02	The work between alert and fix	04
	Visibility was never the constraint			Where the night shift actually goes	
03	Framework 1 — the correlation math	05	04	Specialists, not a generic bot	06
	Counting what your team really handles			Why a crew beats one agent	
05	Context is the real input	07	06	Three resolution modes	08
	Telemetry, tickets, Slack and email together			Autonomous, human-in-the-loop, advisory	
07	Framework 2 — the trust ladder	09	08	Orchestrating across layers	10
	Widening autonomy one pattern at a time			When an incident spans the stack	
09	Governance you can approve	11	10	The cost dimension	12
	Policy, audit, rollback, cost			FinOps and GreenOps in the decision	
11	Framework 3 — the readiness checklist	13	12	Getting started	14
	Twelve questions before you start			The first thirty days	

THREE FRAMEWORKS YOU CAN REUSE

- **The correlation math** — counting alerts, conditions and real incidents to find where your team's time goes.
- **The trust ladder** — four rungs from observe to autonomous, climbed per pattern on evidence.
- **The readiness checklist** — twelve questions worth answering before a coworker touches production.

01 CHAPTER ONE

Why more dashboards **didn't** help

Most infrastructure teams have spent a decade improving visibility. The dashboards are better, the metrics are richer, the alerting is more precise. And someone still gets paged at 3am.

That is not a failure of the tools. It is a category error about what the tools were for. A dashboard answers the question "what is happening?" It does not answer "what should be done about it?", and it certainly does not do it. Every improvement in visibility has increased the amount a person can see without reducing the amount a person has to do.

More tools just mean more tabs. You don't need more visibility. You need someone to do the work.

The tab problem

Compute, storage, database, network and facilities each arrived with their own console, their own alerting logic and, usually, their own on-call. An incident that touches two of them becomes a coordination exercise before it becomes a fix — two people, two screens, one call, comparing what each can see. Adding a sixth tool does not help. It adds a sixth tab.

FOUR SIGNS VISIBILITY IS NO LONGER YOUR CONSTRAINT

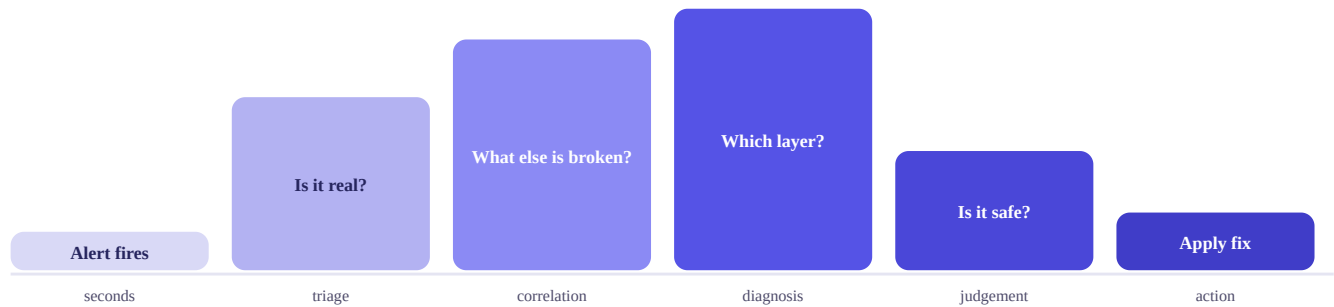
- You can see the problem clearly, and it still takes hours to resolve.
- Most overnight pages end with **"no action required"**.
- Cross-layer incidents begin with a call to assemble the right people.
- The fastest resolutions depend on who happens to be on shift.

02 CHAPTER TWO

The work between alert and fix

An alert and a fix are usually minutes apart in effort and hours apart in time. The gap is filled with work that no dashboard performs.

The work between an alert and a fix



A dashboard shows the first bar. The tall middle — correlation and diagnosis — is still a person, at whatever hour it happens.

FIGURE 1 — WHERE THE NIGHT SHIFT ACTUALLY GOES

Reading, not thinking

The tallest bars — deciding whether an alert is real, finding what else is broken, identifying which layer owns the cause — are mostly **reading work**. They require pulling together telemetry, recent changes, open tickets, the Slack thread where someone mentioned a deploy, and the email from a vendor about maintenance. The judgement at the end is often quick. Assembling enough context to make it is slow.

That distinction matters for automation. Reading work is exactly what a coworker with access to every signal source does faster than a person who must open each one in turn. The judgement step — is it safe to act? — is where you decide how much authority to delegate, and that is a policy choice rather than a technical one.

WHAT THIS IMPLIES

- The largest time savings come from **correlation and context**, before any action is automated.
- Value appears even in **advisory mode**, because the reading work is done either way.
- Measure pages avoided and time-to-context, not only mean time to resolve.

03 FRAMEWORK ONE

The correlation math

Before evaluating any coworker, count what your team actually handles. The numbers usually make the case without further argument.

The correlation math



Illustrative — the middle step is an example ratio; your conditions-per-incident figure comes from your own data.

FIGURE 2 — FROM ALERTS TO THE PAGES THAT MATTER

COUNT	HOW TO GET IT	WHAT IT TELLS YOU
Alerts per day	Total fired across every monitoring source	The raw interruption load before any grouping
Distinct conditions	Alerts sharing a cause and time window, grouped	How much duplication a person currently absorbs
Real incidents	Conditions that required any action	The work that genuinely needed doing
Known patterns	Incidents resembling a prior resolved one	The candidates for autonomous resolution
Cross-layer share	Incidents touching more than one domain	Where orchestration saves the most coordination

Reading the result

The published example — **900 alerts a day becoming roughly 30 incidents** — is a thirty-to-one ratio. Yours may differ, but the shape is nearly universal. The gap between the top and bottom rows is time your team spends confirming that nothing needs doing, and the "known patterns" row is the part a coworker can take entirely.

04 CHAPTER FOUR

Specialists, not a generic bot

A generic agent knows a little about everything. A specialist knows exactly what to do the moment it sees a familiar signal — because it has seen that pattern before, in that layer.

Compute	Storage	Database	Network
Servers, containers and Kubernetes running hot. Right-sizes and scales, escalates when unsafe.	Capacity, latency and IOPS across block, file, object and SAN/NAS. Tiers and rebalances.	Slow queries, locks and storage pressure. Tunes indexes, or files a root-caused ticket.	Packet loss, latency and config drift. Rolls back bad changes — or holds for approval.
Autonomous	Human-in-the-loop	Auto-fix + ticket	Advisory

The modes shown are the defaults the product describes, and they reflect something real about each layer. Scaling compute within bounds is reversible and low-consequence. Expanding storage commits capacity. A database change can affect every application that queries it. A network change can affect everything. The appropriate default authority differs because the blast radius differs.

The mode belongs to the pattern, not the coworker. One specialist can act alone on a known pattern and hold for approval on a novel one.

Why specialisation compounds

A specialist accumulates a pattern history within its domain. Each resolved storage incident makes the next storage judgement sharper, in a way that a single general agent spreading attention across every layer does not replicate. Over time the boundary between what a coworker resolves alone and what it escalates moves — but it moves separately for each layer, at the pace that layer's evidence supports.

05 CHAPTER FIVE

Context is the real input

An alert-driven agent sees what fired. A context-aware coworker sees why — because it reads the same sources your engineers would open, all at once.

Consider what an experienced engineer actually does when paged. They check the telemetry, yes. But they also look at the change calendar, scroll back through the team channel for anyone who mentioned a deploy, check whether a ticket is already open, and glance at the vendor email about scheduled maintenance. The telemetry says what is wrong. The rest says why, and whether it is already being handled.

SOURCE	WHAT IT CONTRIBUTES	WHAT IS LOST WITHOUT IT
Device telemetry	What is failing, and how badly	Everything — this is the minimum
Logs & traces	Where in the path the failure originates	The ability to separate symptom from cause
Tickets	Whether this is known, already worked, or recurring	Duplicate effort on problems already in hand
Slack	What people noticed, changed or suspected	The informal context that often holds the answer
Email	Vendor notices, maintenance windows, change approvals	Scheduled explanations for apparently unexplained events
Security signals	Whether an anomaly is operational or adversarial	The risk of remediating an attack as a fault

LUMI · WHY IT WAS DISMISSED

Why It Was Dismissed

Lumi's overnight orchestrator run correctly determined this alert was a **false positive** because:

1. **Current state is healthy:** Device pings successfully now
2. **No structural failure:** All 40+ monitored items are operational
3. **Resources are fine:** No CPU spike, no memory exhaust, no thermal throttling
4. **No correlation with changes:** No deployments, config edits, or recent incidents on this device
5. **No blast radius:** No downstream services reported impact

No action required. This is typical noise from edge router monitoring — intermittent ICMP loss over the Internet is normal; ICMP is not reliable enough for SLA-critical alerting.

The reasoning a person would have to reconstruct — health, structure, resources, recent changes, blast radius — arrives written out, so the next reader can check it rather than redo it.

This is why the product describes itself as reading "telemetry, tickets, Slack and email" rather than just monitoring. A ticket that arrives with root cause, impact and fix already filled in is only possible if the coworker read the same material the analyst would otherwise have had to gather.

06 CHAPTER SIX

Three resolution modes

The most common objection to automated operations is that it is all or nothing. The coworker model removes that choice by setting authority per pattern.

MODE	WHAT HAPPENS	WHEN TO USE IT	WHAT IS LOGGED
Autonomous	Resolves end to end, no approval step	Known, reversible, low-risk patterns	Full decision trail and outcome
Human-in-the-loop	Proposes a fix, waits for a named approver	Consequential or capacity-committing changes	Proposal, evidence, and approval
Advisory	Recommends; your team decides and acts	Novel patterns and high blast radius	Recommendation and whether followed

Approval that isn't a bottleneck

Human-in-the-loop only works if approving is faster than investigating. That depends on what arrives with the request. A proposal with the evidence behind it — the correlated signals, the pattern it matches, the expected effect — lets the approver confirm a conclusion. A bare "approve rollback?" forces them to start the investigation the coworker was supposed to have done.

If approving takes as long as investigating, you have not delegated anything. You have added a step.

Advisory is not a lesser mode

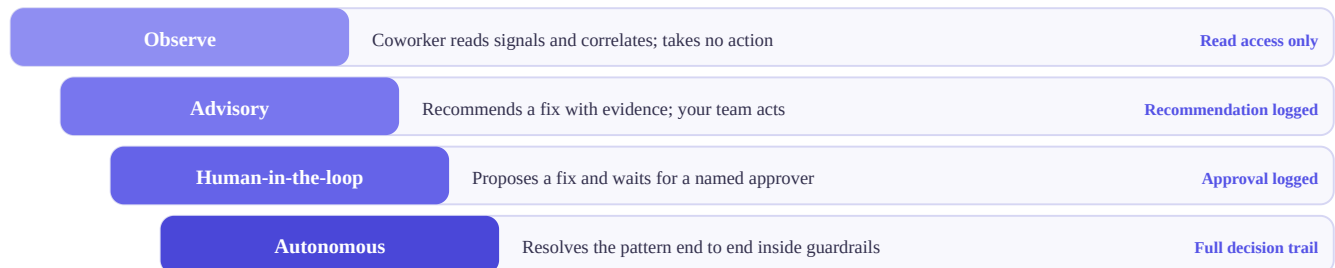
Teams sometimes treat advisory as a trial period to escape. It is better understood as the right permanent setting for some patterns — network changes with estate-wide reach, for instance — where the value is in the diagnosis and the context, not the execution. A recommendation that arrives correctly root-caused has already done most of the work.

07 FRAMEWORK TWO

The trust ladder

Autonomy is widened on evidence, one pattern at a time. The ladder makes that process explicit, so it can be governed rather than argued about.

The trust ladder — set per pattern, not per product



Each rung is climbed one pattern at a time, on the evidence of clean resolutions at the rung below.

FIGURE 3 — RESOLUTION MODES AS A LADDER

1	Observe	Read-only correlation and diagnosis. Move up when: the incident stories it produces match what your engineers conclude.
2	Advisory	Recommendations with evidence. Move up when: recommendations for a pattern are followed consistently and hold.
3	Human-in-the-loop	Proposed fixes executed on approval. Move up when: approvals for a pattern are routine and the fixes verify cleanly.
4	Autonomous	End-to-end resolution inside guardrails. Stay when: outcomes remain clean; drop a rung if a pattern drifts.

How to use it

The ladder is per pattern, per coworker. On any given day a mature estate will have some compute patterns autonomous, most storage patterns at human-in-the-loop, and most network patterns advisory — and that is a healthy state, not an incomplete one. The discipline is in moving a pattern up only on evidence, and down promptly when evidence changes. Rollback on drift, which the product builds in, is what makes moving up a reversible decision.

08 CHAPTER EIGHT

Orchestrating across layers

The incidents that cost the most are the ones that span the stack — and they are precisely the ones separate teams and separate tools handle worst.

A checkout slowdown shows up as compute pressure. The cause is storage saturation. A connection pool nearing its limit makes it worse. Three layers, three teams, three consoles — and the first twenty minutes are spent establishing that it is one incident. The Lumi Orchestrator exists for this case: it correlates signals across layers into a single incident, decides the resolution path, and dispatches the right coworkers in the right order.

```
LUMI — ORCHESTRATE ● live

signal checkout-api p99 alert · storage IOPS warning · 2 Slack threads

# correlate: same window, same dependency graph
correlate → 1 incident: "checkout-api degraded, storage-fleet saturated"

decide --guardrails
  → Storage Coworker: rebalance tier → autonomous
  → Compute Coworker: scale checkout pool (+2) → autonomous
  → Database Coworker: pool near limit → human-in-the-loop

✓ rebalanced · scaled · DBA approved · resolved
```

Sequencing matters

Order is not incidental. Scaling compute into a saturated storage tier makes the problem worse; rebalancing storage first, then scaling, resolves it. A human bridge works this out by discussion. An orchestrator with the dependency model works it out before dispatching anything — and each coworker still acts at the resolution mode its pattern allows, so a cross-layer incident can be partly autonomous and partly gated in the same run.

09 CHAPTER NINE

Governance you can approve

Every action runs inside guardrails you set. The goal is speed without losing oversight — and the four properties below are what make that claim checkable.

Policy-governed

You define what is automated, what needs approval, and what stays advisory — as configuration, reviewable by security and risk, not as convention held in someone's head.

Full audit trail

Every decision, recommendation and change is logged and traceable, so an incident can be reconstructed from records rather than from memory and chat history.

Safe by default

Changes are policy-checked before execution, and Lumi rolls back when something drifts — which is what makes granting autonomy a reversible decision.

Cost-aware

FinOps and GreenOps context weighs spend and sustainability alongside uptime, so the fastest fix is not automatically the most expensive one.

One control plane

These properties are not bolted onto the coworker afterward. They come from running on **CERNE**, the same control plane that governs LumiOps AI's SRE, Security, FinOps and ITSM orchestrators. The practical consequence is that the policy model, the audit format and the approval routing are the same across every automated function — so a security team reviews one governance model, not one per tool.

Guardrails are not a brake on autonomy. They are what makes autonomy something you can actually turn on.

10 CHAPTER TEN

The cost dimension

Most operational automation optimises for one thing: restoring service. A cost-aware coworker weighs how service is restored, not only whether.

The default response to a performance problem is usually more capacity. Scale the pool, expand the volume, add a node. It works, it is fast, and it quietly becomes permanent. Over a year, an estate that resolves every slowdown by adding resources accumulates a bill that no single decision justified.

SYMPTOM	CAPACITY-FIRST RESPONSE	COST-AWARE ALTERNATIVE
Slow queries	Scale the database tier	Tune the index causing the pressure
Storage latency	Expand the volume	Tier or rebalance hot data
CPU pressure	Add nodes	Right-size the workload that is over-provisioned elsewhere
Network congestion	Upgrade the link	Roll back the config change that shifted traffic

Each alternative is slower to identify and cheaper to run — which is exactly the trade a person under pressure at 3am will not make, and a coworker with FinOps and GreenOps context can. The healthcare case in our companion set is a direct example: index tuning replacing database scale-ups as the default response.

QUESTIONS WORTH ASKING OF ANY AUTOMATED RESPONSE

- Did it restore service by **fixing the cause** or by adding capacity around it?
- Will the capacity it added be removed when the pressure passes?
- What is the **energy and carbon** cost of the chosen fix versus the alternative?

11 FRAMEWORK THREE

The readiness checklist

Twelve questions worth answering before a coworker acts on production. The signal is whether each answer exists as configuration, as documentation, or only as intention.

- | | |
|---|--|
| ☐ Which signal sources will the coworker read on day one? | ☐ Are tickets, Slack and email included, or only telemetry? |
| ☐ Which patterns are candidates for autonomous resolution? | ☐ What mode does each coworker start in, per pattern? |
| ☐ Who approves human-in-the-loop actions, as a role? | ☐ Does the approver receive the evidence with the request? |
| ☐ What bounds each autonomous action — ceilings, retries? | ☐ How is drift detected, and what triggers rollback? |
| ☐ Where is every decision logged, and who can read it? | ☐ Which cost and sustainability limits apply to fixes? |
| ☐ What evidence moves a pattern up the trust ladder? | ☐ What evidence moves a pattern back down? |

Scoring the answers

Sort responses into three categories. **Configured** means someone can show you the setting. **Documented** means it is written down but enforced by convention. **Intended** means it lives in a plan. You are ready to start in advisory mode with most answers intended. You are ready to move a pattern to autonomous only when the approval routing, the bounds, the rollback trigger and the audit trail for that pattern are configured.

SCORE	WHAT IT MEANS	READY FOR
Configured	A setting someone can show you, enforced by the platform	Autonomous resolution for that pattern
Documented	Written down, enforced by convention and review	Human-in-the-loop, with approval as the enforcement
Intended	Agreed in principle, not yet written or set	Observe and advisory only

Start advisory with good intentions. Go autonomous only with configured guardrails.

12 CHAPTER TWELVE

Getting started

The fastest route to value is narrow and read-only. Breadth and autonomy come later, on evidence.

STEP	WHAT TO DO	WHAT IT PRODUCES
1 • Count	Run the correlation math on a recent month of alerts	A baseline in alerts, conditions and incidents
2 • Connect	Point the coworker at telemetry, tickets, Slack and email	Context-aware incident stories from day one
3 • Observe	Run read-only and compare its conclusions to your team's	Evidence of diagnostic accuracy, risk-free
4 • Advise	Enable recommendations across all four coworkers	Pre-solved tickets and a measurable time saving
5 • Gate	Move repeatable patterns to human-in-the-loop	Approved fixes with a clean verification record
6 • Widen	Promote proven, reversible patterns to autonomous	Incidents that resolve before anyone is paged

What the first month should prove

Three things, agreed before you start. Does correlation reduce what reaches a person on your real alert volume? Are the incident stories correct when your engineers check them? And would you have been comfortable with the fixes it recommended, had they been applied? Those are answerable in weeks. Questions about long-term autonomy are answered later, by the trust ladder, one pattern at a time.

THE FIVE THINGS WORTH TAKING AWAY

- **Visibility was never the constraint.** The work between alert and fix is.
- **Specialists beat generalists.** A crew per layer resolves more without escalating.
- **Context is the input.** Tickets, Slack and email turn a page into a pre-solved ticket.
- **Authority is per pattern.** Autonomous, human-in-the-loop and advisory coexist.
- **Governance makes it possible.** Policy, audit and rollback are what let you turn it on.

IN CLOSING

Get your night back

Every infrastructure team has the same unspoken arrangement: the estate runs around the clock, and so, in effect, does someone on the team. The night shift is not a rota entry. It is the assumption that a person will do the reading, the correlating and the deciding, whenever it is needed.

That assumption no longer has to hold. A crew of specialist coworkers can read every signal source at once, correlate a thousand alerts into the few incidents that matter, resolve the patterns they know inside the guardrails you set, and hand you the rest with the evidence already assembled. None of that removes engineering judgement. It removes the hours of collection work standing between an engineer and the moment where judgement is actually needed.

BOTTOM LINE

Same team, same stack — a very different on-call. Meet the coworker who works the night shift.

Meet the coworker who **works the night shift.**

See it take a real incident from alert to resolved — on your stack.



GET IN TOUCH
lumiops.ai/contact-us



POC
~2 weeks to value



LINKEDIN
linkedin.com/company/lumiops