IMPLEMENTATION · CONSULTING · BLUEPRINT

The Logmetry Blueprint

Your telemetry all lands enriched in a lake you own. Your expensive tools receive only what earns its place there. Infrastructure monitoring gets rebuilt as yours instead of rented per node. And an agent you own, running in your own account on your own runbooks, investigates every alert whichever tool fired it, reading your full-fidelity history from the lake you enriched. That is AI put to work on your operations, vendor agnostic all the way down: triage and root cause you run yourself, on models you swap, and the foundation for whatever you buy next.

WHAT IS INSIDE

01	**Foundation**	One collection layer, a lake you own, and every destination a routing choice
02	**Footprint**	Platforms replaced, or cut to what earns its price
03	**AI triage**	Every alert investigated, whichever tool fired it

WHAT THIS IS

This is a reference blueprint, not a fixed design. It is vendor agnostic on purpose, because per-GB and per-host economics behave the same way whatever your stack. Your version of every drawing gets made in the architecture review, on your stack, at the phases you choose. All the review asks of you is your architecture diagram.

WHAT WE BRING

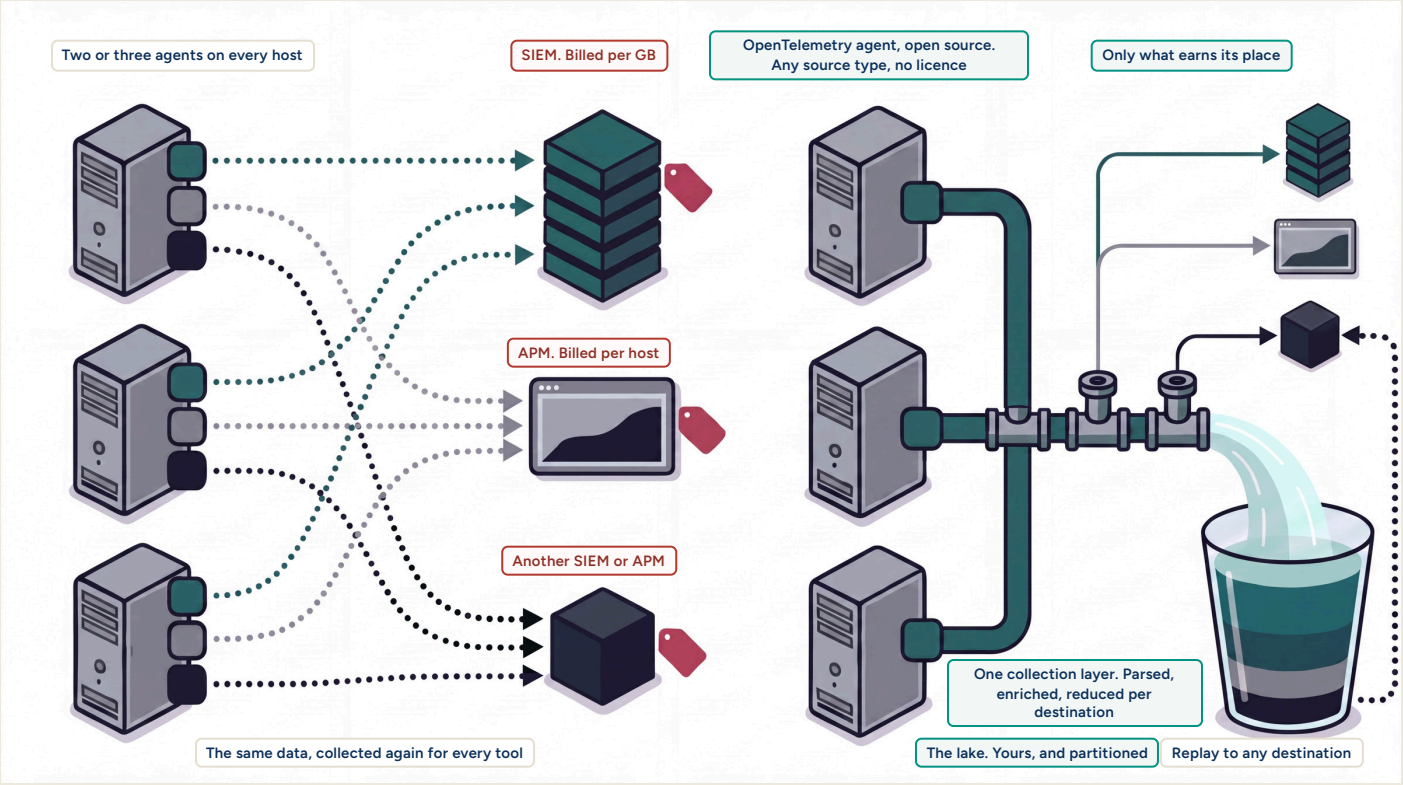
What we bring is engineering. The ground is observability, security, and log management, across every kind of endpoint you run. The core is the collection layer and telemetry control: what gets collected, what it is enriched into, and what it costs you to keep. Built for the estate you actually run, delivered as code you own in your own repositories, down to every last configuration, and operated until you want it back.

None of it is assembled by hand in a console. The pipelines, the rules, the dashboards and the infrastructure underneath are versioned code, shipped through the same tested pipeline every time. That is what AI native means: an estate held as code is one a coding agent can read, review and correct, so the people who run it do not have to become programmers to change it.

The agents on top of it are yours as well: one triaging security alerts, one chasing root cause when performance moves. They are internal, so they learn how your company runs from your own runbooks and playbooks instead of reading a vendor's slice of your estate, and you swap the model underneath them the day a better one ships. All of it is customized to your environment, not configured inside somebody's product. That is the difference between a stack you rent and a vendor agnostic stack you own.

PHASE 01

Foundation



THE ROUTING RULE

Everything lands in the lake, full fidelity, enriched in flight, and partitioned the way it will be asked for: by host, by service, by team, by pipeline, by time. Enrichment is what everybody promises. Partitioning is what makes it answerable, and an investigation is only ever as good as what it can retrieve.

That is the real product of this phase. The ingest saving is immediate, and it is the smaller half. What you are building is the foundation every triage and root-cause agent above it will read. If you control the collection layer, you control where the data goes: each destination receives only what it is good at, shaped and reduced on the way, whether that is the expensive APM, the SIEM, or a monitoring platform you now own.

Why. One vendor-agnostic collection layer, built on OpenTelemetry and collectors nobody licenses, replaces the pile of per-tool agents and puts you in control of where your data goes and what each destination gets.

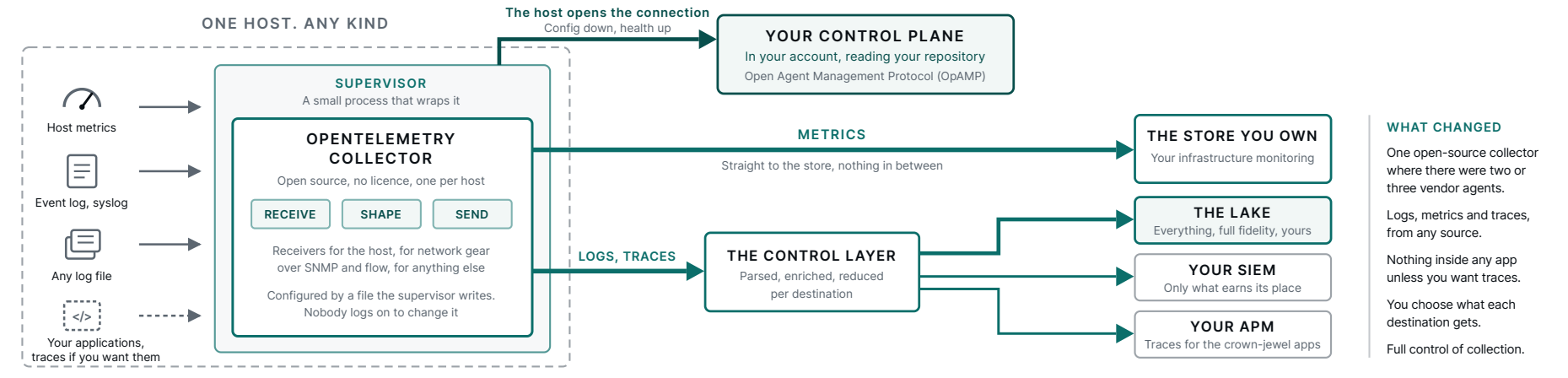
What you own after. The collectors, the pipelines, every routing decision, and a lake that is full fidelity, enriched, and partitioned for retrieval, all as code in your repositories. Swap the AI model behind triage later and each reads the same foundation.

Why it has to be first. You cannot choose where telemetry lands, how much of it, or how clean, without owning the layer that collects it and the pipeline that shapes it. Open collection makes you vendor agnostic at the source, and the pipeline sends each platform only what it needs, the rest wherever you decide, including a platform you build yourself or with us. Every platform you run is being rebuilt around AI, and the agents you buy next are only as good as the data they reach: clean, cheap to keep, and yours.

PHASE 01 · IN DETAIL

Everything starts at the collection layer

Everything starts at the collection layer, and yours is open. Every vendor agent comes off the host and one collector goes on: the OpenTelemetry Collector, open source, no licence, owned by you. Logs, metrics and traces, on premises or in the cloud, from a Windows server to a core switch to an instrumented application. From this layer on, no vendor holds you.



ON THE HOST, BY DEFAULT

Out of the box it reads what a server produces: CPU, memory, disk and network, the Windows event log, syslog, Windows performance counters, and any log file any application writes. That is the whole infrastructure-monitoring estate, with nothing installed inside any application. One process per host, maintained in the open, on the standard every platform vendor now builds against.

NETWORK GEAR AND THE REST OF THE ESTATE

The same collector mechanism, configured differently, does the rest: point a receiver at the network and it polls switches, routers, firewalls and hardware sensors over SNMP, takes syslog and flow from devices, reads virtualization platforms and databases, and scrapes anything that already publishes metrics. Infrastructure monitoring, network included, from one mechanism, with no poller in the middle holding credentials to everything.

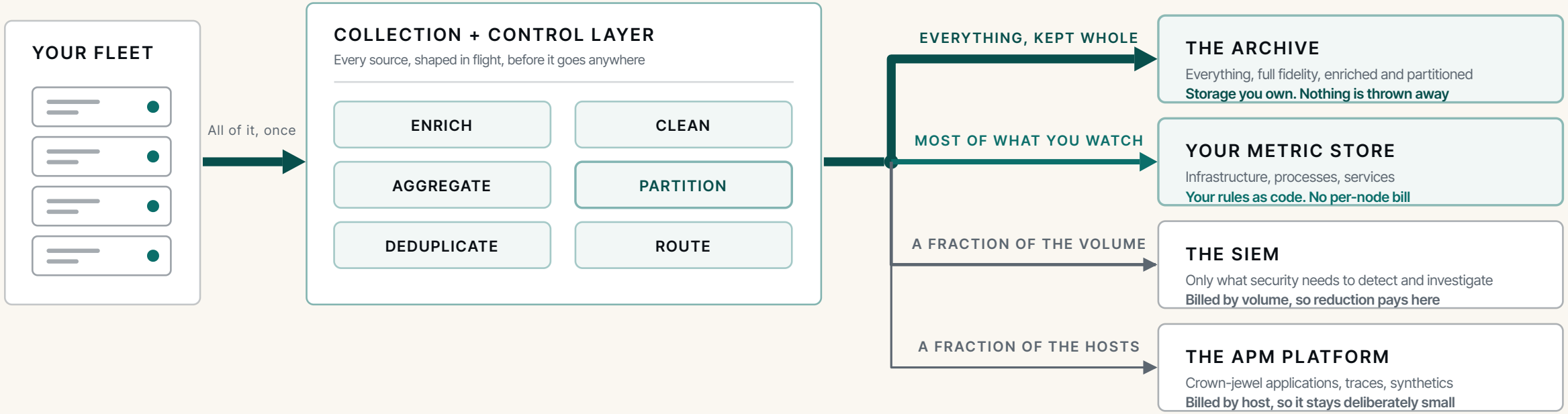
TRACES AND APM, FROM THE SAME PROJECT

Traces come from inside an application, and the same open project covers them: auto-instrumentation at the process level, one flag on a Java service or an installer on .NET, no code change. Your developers instrument nothing unless they want custom business attributes and metrics on top. Crown-jewel applications get their traces into your APM through the same collector and layer. Logs, metrics and traces from one layer: observability unified at the source.

From the first layer on, nobody owns your collection but you. Logs, metrics and traces, infrastructure and applications, on premises and cloud, collected once by open-source software that carries no licence and lives in your repositories. Every destination after this point is a choice you make, not one made for you. A collector is not the product, though. Running thousands of them is, and that is the part every vendor keeps. The next pages are how you own that too.

PHASE 01 · IN DETAIL

Where everything goes



Collected once. Routed four ways. Everything lands in the one you own.

ONE RULE PER DESTINATION

Each destination is good at something different and priced differently, so each gets its own rule. Security tooling receives what security needs and nothing else. The rest lands in the lake at full fidelity, partitioned so it comes back fast and cheap to keep for as long as compliance asks. Replay any of it into any destination when an investigation wants it. So why pay a platform to hold what fires no rule, in case you need it later?

LESS NOISE, RICHER SIGNAL

The noise comes off the security feed and goes to the lake in full instead. What it still receives is enriched harder, because that context is what makes a correlation rule fire, and most estates ingest sources no rule was ever written against. Sending less is only half of it. Everything is kept whole in the lake because investigation is moving to agents, and an agent is only as good as what it can retrieve.

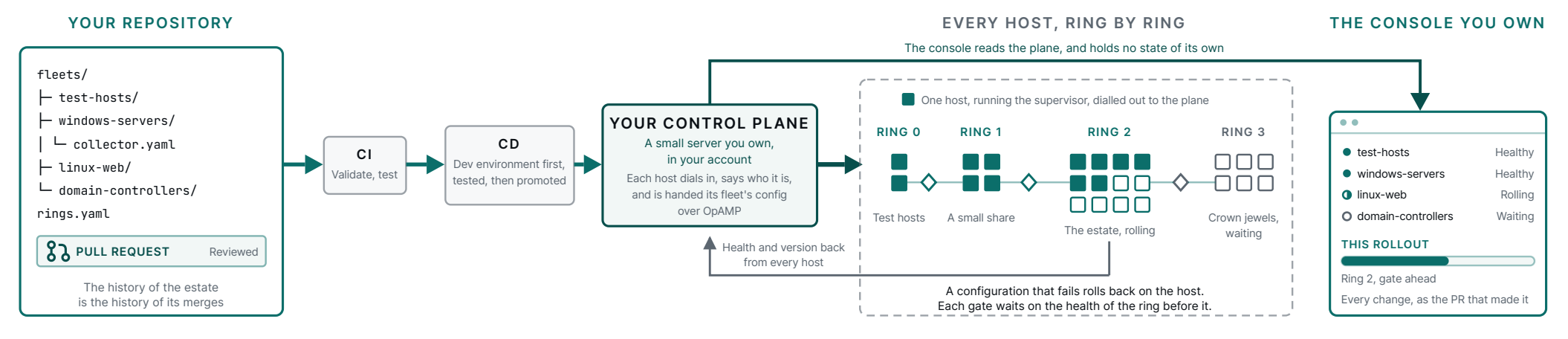
THE TIER THAT NEED NOT BE RENTED

Tracing stays on the crown-jewel applications that earn a per-host price. Ordinary infrastructure and service monitoring comes off per-node pricing, because it is standard enough to rebuild on an open stack, as code, for a fraction of the rent. That used to need a platform team you do not have. Now the rules are files a coding agent edits in plain English, and our engineers build it and stay on it. What you stop paying for is the dependency.

PHASE 01 · IN DETAIL

The fleet, controlled as code

Your fleet is a repository: one folder per fleet, a configuration file per collector role, and a label on every host, written when the host is built, that says which fleets it belongs to. Each host dials in to a small control plane in your account, says who it is, and is handed the configuration for its fleets. Nothing else touches a host.



THE CONSOLE YOU OWN

The same pane every platform vendor rents you for their agent: every host, its health, what it is running and whether that matched what was sent, the rollout in progress. Built for you, running in your account, reading your repository. It holds no state of its own to lose.

EVERY CHANGE IS A PULL REQUEST

A new collector configuration, a new fleet, a host moving between fleets: a reviewed change in a structured repository, merged and then deployed. The history of your estate is the history of its merges, and an estate written that way is one a coding agent reads with you.

ROLLOUTS BY RING, ROLLBACK BY DEFAULT

A few test hosts first, then a small share of the estate, then the rest, then the crown jewels. Each ring waits on the health of the last. A configuration that fails is rolled back on the host itself, and a new collector version reaches a domain controller only after it has proved itself everywhere else.

NO PER-AGENT LICENCE

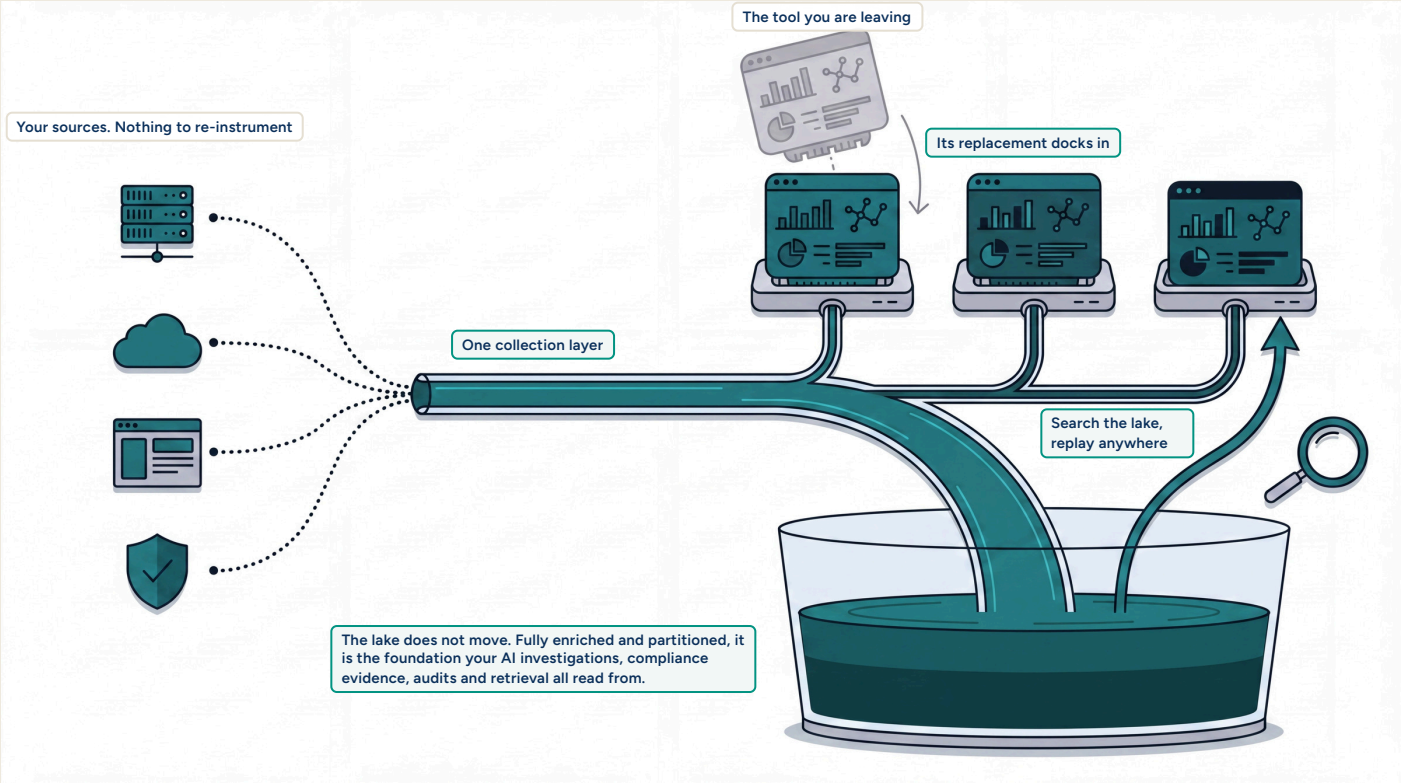
Nothing here carries a per-host, per-agent or per-gigabyte licence: not the collector, the protocol, the plane or the console, and not the infrastructure monitoring alerts and dashboards the vendor platform used to own, now rebuilt on open source on your infrastructure, same alerts, same coverage. What you pay for is the engineering.

Everyone else rents you the pane of glass over their agent, and what you see in it is what they chose to show. This one is in your repository and your account, over an agent nobody owns. That is what vendor agnostic means in practice: not a promise about the future, but a fleet you could hand to the next engineer, or the next vendor, tomorrow.

PHASE 01 · THE FINAL STATE

Vendor agnostic

A COMPLETE ENGAGEMENT



This phase is a beginning, not an end state. A platform can be swapped for a better one from here, and in time replaced by tooling you build and own, because the data it would need is already yours and already enriched. That is what vendor agnostic finally means. It is also the ground the next phase stands on, because AI triage is only ever as good as what it can retrieve, and what it retrieves here is data you control rather than data a vendor rents back to you. None of it needs us in the room to keep running.

DESTINATIONS BECOME PLUGS

A destination is a few lines in the collection layer's configuration, and that configuration is code you hold in your own repository. Adding a tool, retiring one, or feeding the same stream to both is an edit, reviewed and versioned like anything else your engineers ship. None of it is possible while the agent on the host belongs to the vendor you are leaving, which is why collection had to be open first.

Nothing on your hosts changes when a destination does. The replacement takes live data beside the tool it replaces for as long as you want the comparison, and the lake backfills its history so it starts with your past rather than an empty index.

All of it is configured as code in your own repository. Routing, enrichment, alert rules and the fleet itself in one place, versioned, tested, and open to read. An estate written that way is one an AI can read with you, so what a node is sending, what changed this morning, or where a cost came from becomes a question your team asks in plain English and has answered from the configuration itself.

Why. The tools you pay the most for should be the easiest ones to leave.

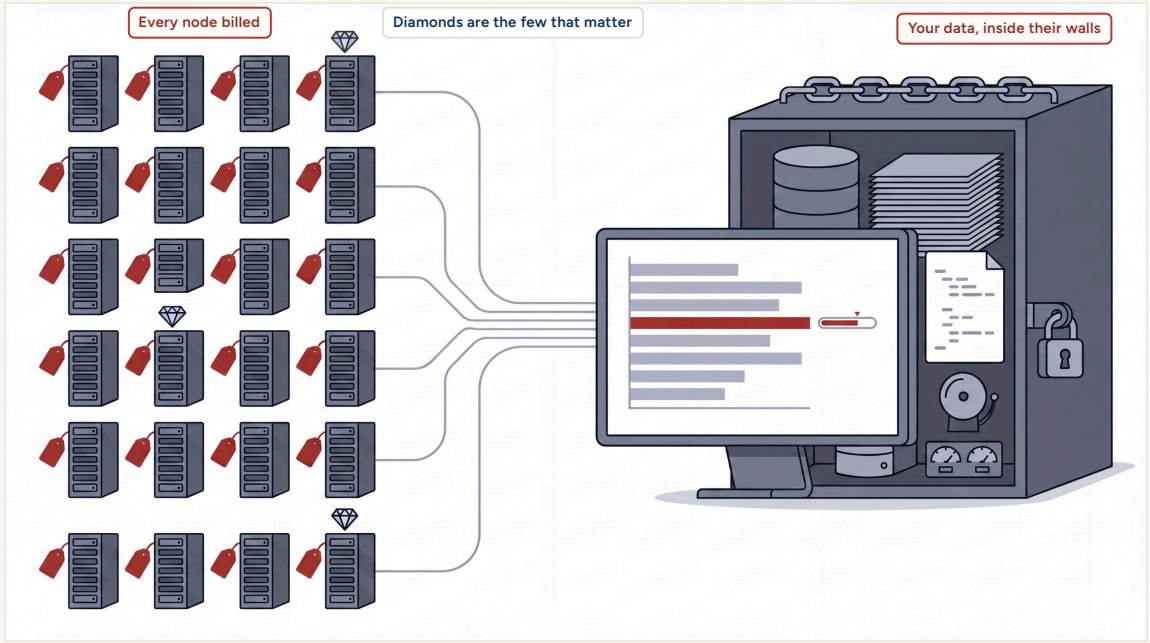
What you own after. A stack where adding, replacing, or leaving any tool is a routing change rather than a migration.

A valid place to stop. Data independence is half of it. The other half is control: what each destination receives, what it is shaped into on the way, and what that costs. Their agent collects everything and bills you for it. You send each platform only what it is good at, keep the rest in a lake you can replay from, and negotiate every renewal after that from strength.

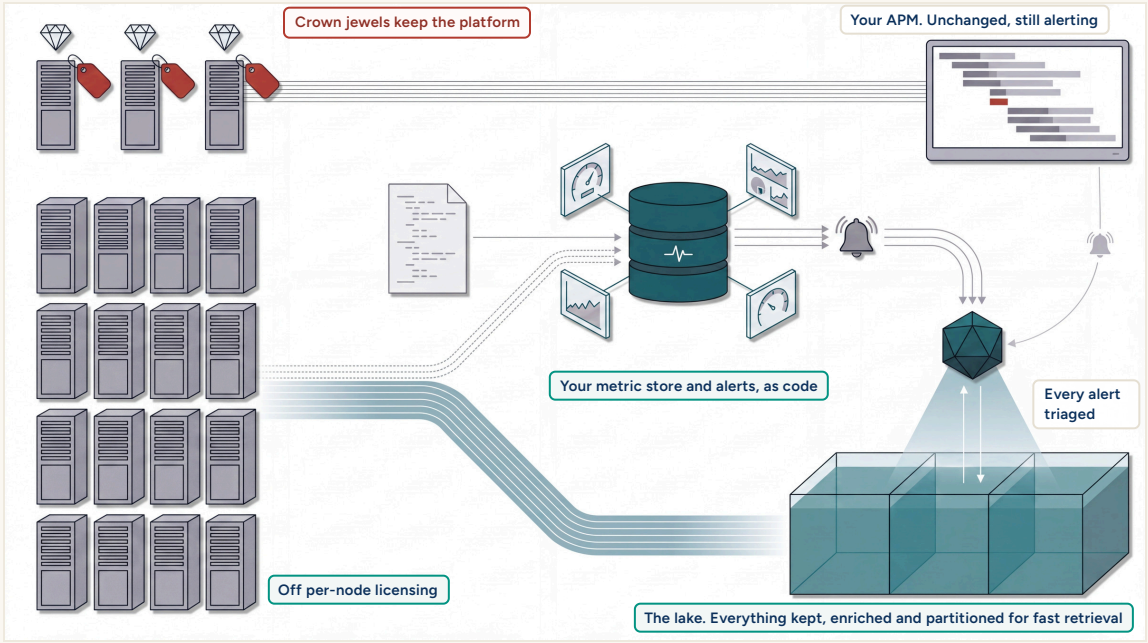
PHASE 02

Footprint reduction

TODAY



AFTER



RELOCATED, NEVER REMOVED

Crown-jewel applications and synthetic monitoring still earn a per-host price. The open collector can feed your APM with traces, and for the deepest tracing, method-level with the vendor's own auto-instrumentation, that vendor's agent earns its place on those few hosts: advanced tracing analytics is what Dynatrace and Datadog are good at. Metrics and logs stay on the open collector everywhere.

The rest is not. Service up, process running, CPU, memory and disk, the standard rules quiet. No tracing, often no logs, just metrics off ordinary nodes, and on-prem estates are almost entirely this. It is where the node count comes from, and so the bill.

That tier moves to an open-source engine you own: not a vendor, not a protocol you license, self-hosted or managed in your cloud. Chosen with you, Prometheus and ClickHouse among the proven ones. Its rules carry the same thresholds, durations and routing you already write in Datadog, SolarWinds or Dynatrace, nothing different, and they come out of each platform through its API as versioned code. Here the product is the alerts, not the dashboards.

Why. Every alert and every rule is rebuilt and proven to behave exactly as it does today, tier by tier and in writing, before a single host comes off per-node pricing. The Phase 03 agent that follows is a second path on top, not a replacement: the alert still fires as it always did, and now also reaches something that investigates it. Coverage goes up, because the alerts nobody ever reached get worked too.

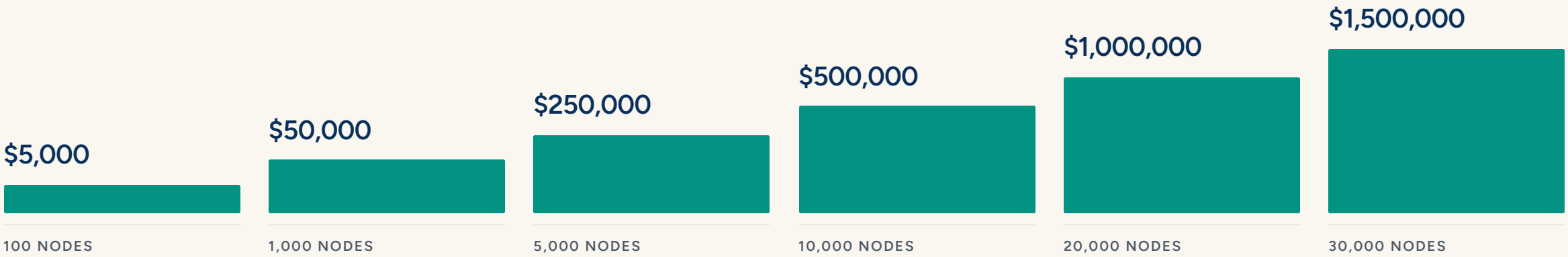
What you own after. Platforms replaced where they can be and cut back to a small core where they cannot, with everything else running on collection, alerts and dashboards you own.

What this adds. Traces and logs now land enriched in the lake as well, so the same trace ID your APM fires on can be followed there by the agent, which is where investigation is heading. It also closes a skills gap: the rules are plain YAML files, changing one means asking a coding agent in plain English, and who operates what is left, us or your team, is a choice.

PHASE 02 · IN DETAIL

What the per-node model costs

Infrastructure monitoring is sold by the node, at fifty dollars or more per node every year. That is not our figure. Published list pricing across the major platforms runs higher, and fifty is roughly where a large estate lands after volume discounting, so the bill is a straight multiplication by the size of the estate. The licence is only part of what you are buying. Your history sits on their disks, your rules are written in their language, and the intelligence you add on top runs on their modules at their price, reasoning only over what you sent them. These are good platforms for the work that genuinely belongs on them. Send that work and no more, and let everything else live in your own estate, as your own code, on a stack you can leave.



READ THIS AS THE METER, NOT THE SAVING

This is what the per-node model charges at each size, every year, before anyone has read a dashboard. It is not a quote for what we take off the bill. Crown-jewel tracing stays where it earns its price, and so does synthetic monitoring. What that leaves is the bulk of most estates: ordinary nodes running ordinary processes, watched by basic checks collected just as well another way. The stack that replaces them costs a fraction of that, and the subtraction is yours to do on your own node count.

WHY THE LADDER STEEPENS, AND WHAT IS CHANGING

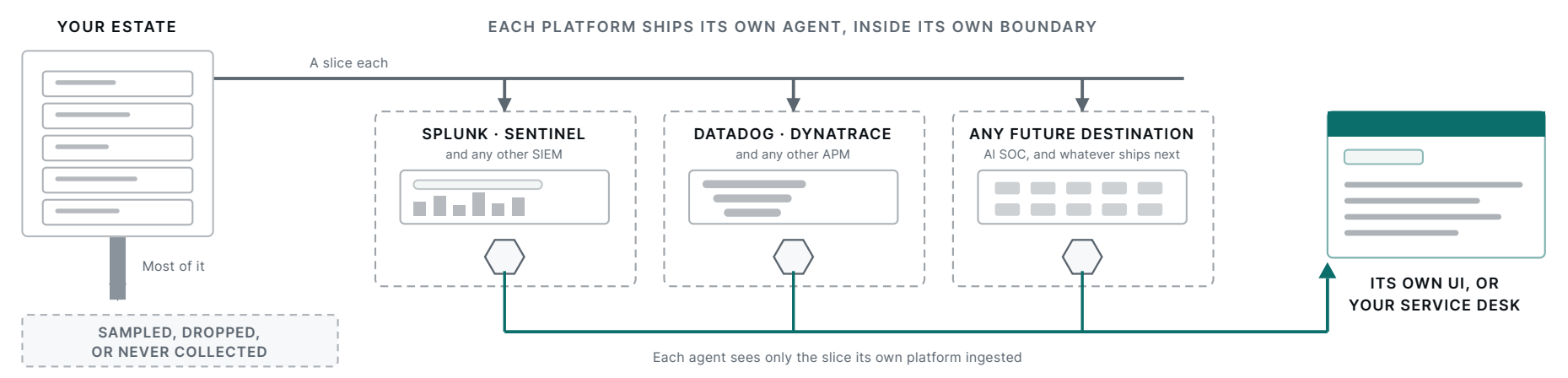
Per-node pricing grows with the estate whether or not those nodes produce anything anyone reads, and renewals move in one direction. The tier that drives the count is almost never the tier anyone is looking at. It is ordinary infrastructure, and that is exactly the tier that does not need to be rented.

Even tracing is changing. Traces and logs collected by the open collector land in the lake carrying the same trace ID, which is how OpenTelemetry ties the span, the log line and the metric together, so an agent can follow a trace ID your APM raised into everything the collector kept around it, for the cost of a query. The deepest tracing stays on the few hosts that earn it. The rest of the investigation is moving to the lake.

PHASE 03

How triage is built today

Every platform you run is shipping an agent of its own. Datadog and Dynatrace have them, Splunk and Microsoft Sentinel have them, and the platforms that do not will. Before you buy another one, it is worth being precise about what an agent that lives inside a platform can actually see.



WHAT IT DOES WELL

It lives inside the platform. An alert fires there, the agent reads the data that was ingested there, and it hands back a written breakdown in that platform's own console or pushes it into your service desk. When an alert is simple and everything it needs is already in that platform, its agent will close it, and you should keep it for exactly that.

THE DATA IT CAN SEE

It reasons over one platform's copy of your estate. Not the full record, because only what earned its way into that licence is in there. Not the sources that went somewhere else. And nothing was parsed, cleaned or enriched on the way in the way foundation work does it, so the agent starts from whatever shape that vendor happened to store.

THE CONTEXT IT DOES NOT HAVE

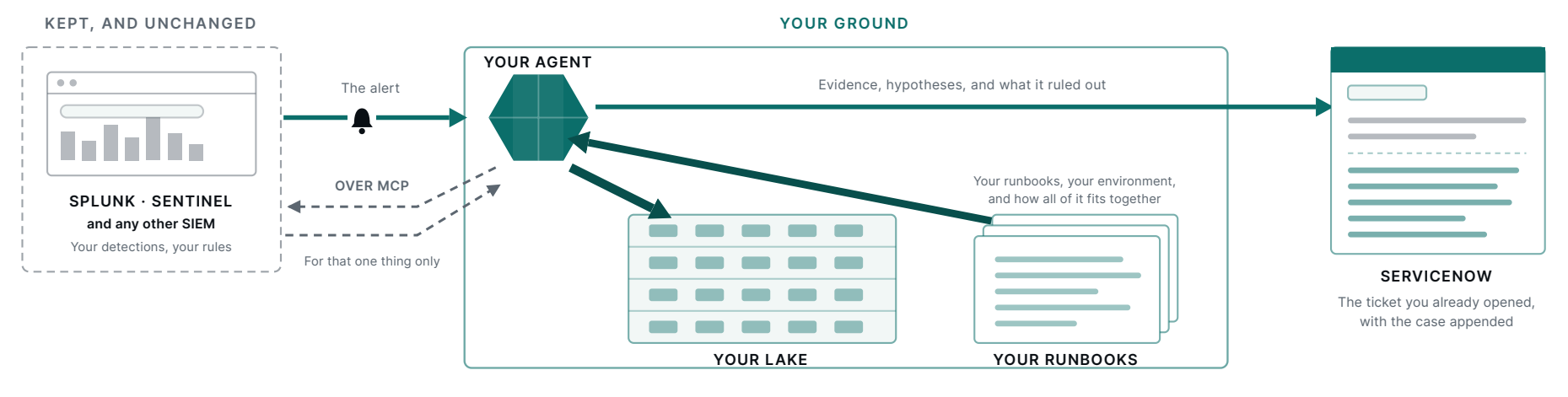
It has no runbooks. It does not know which service depends on which, who owns what, what changed this morning, or what normal looks like on a Tuesday. That is the context an investigation actually turns on, and no vendor can put it in the box, because it is a description of your organisation rather than a feature of their product.

Where it stops is the complex one. An incident that crosses several sources, several services and several teams needs the full-fidelity record and it needs your runbooks. For that agent to have either, you would have to ingest your entire estate into that one platform, which no budget allows and no licence is priced for, and even then the runbooks would still not be there. An agent only works at the speed of the data it can reach. No amount of model quality closes a gap that is about access, and that is the ceiling every page after this one is about raising.

PHASE 03 · IN DETAIL

Your agent, the SIEM you keep

Your SIEM is not going anywhere, and it should not. The detections, the correlation rules and the years of tuning behind them are among the most valuable things you own. This holds for any SIEM you run, now or later. We do not rebuild them the way we rebuild infrastructure monitoring. What changes is where the thinking happens.



HOW THE INVESTIGATION RUNS

The alert reaches your agent directly. It reads your lake first, because that is where the full record lives, and your runbooks, because that is where your environment is described. When it needs something only the platform holds, it calls that platform for that one thing over the platform's own MCP server, and where a platform does not publish one we build it. One route per platform, used for what only that platform has.

WHAT LANDS ON THE TICKET

Everything the investigation found is appended to the ticket your service desk already opened. Not a verdict on its own. The evidence, the hypotheses it tested, the ones it ruled out and the reason it ruled them out. Your analyst opens one ticket and reads a case rather than starting one.

RUNNING BOTH AGENTS

Keep the platform's own agent. For a critical alert that is specific to that platform, on data that already lives inside it, that agent does the job well and there is no reason to take it away. Complex troubleshooting is the other case. Once an incident crosses sources, services and teams it needs the full record and the runbooks, and neither of those is in there. That is the work this agent exists for, and the two are not in competition.

Nothing was migrated to make this work and nothing was switched off. What moved is where the reasoning happens, and that is the whole argument, because reasoning is only ever as good as the context underneath it: your runbooks, your rules, your lake, and the enrichment already done inside it.

PHASE 03 · IN DETAIL

The half nobody can sell you

This is the half of the answer nobody can sell you, because it is a description of your organisation rather than a feature of anyone's product. It is also the half that decides whether the model on top of it is any good.

It is what people mean when they say an AI needs context, and in plain terms it is two things together. Your data, cleaned and enriched and kept whole rather than sampled. And a set of ordinary files, written in plain English, that explain how the pieces relate: which service depends on which, who owns what, what normal looks like on a Tuesday. Neither one is worth much on its own. The data without the explanation is a haystack, and the explanation without the data is a wiki.

YOUR LAKE

Everything, at full fidelity, enriched and partitioned by data type, host, service and time, which are the axes an investigation actually pivots on. The agent crosses sources that never shared an index, which is exactly where the human ritual dies. A badly partitioned lake is the one thing that makes an agent slow and expensive.

YOUR RUNBOOKS

Not documentation nobody reads. How your teams actually work: which service depends on which, who owns what, how escalation runs, what normal looks like on a Tuesday. We write the first set with you as plain files in plain English, which is the format a model reads best. Then your team owns them and adds one after each investigation, and the agent's own telemetry shows where a triage lost time, so they sharpen instead of rotting.

YOUR ENVIRONMENT

Your service catalogue, your ownership map, your change records and your configuration records, wherever they live today. Retrieved alongside the lake so the agent knows what a hostname means inside your organisation, not only what it means in a log line.

WHY IT DOES NOT SHIP IN A BOX

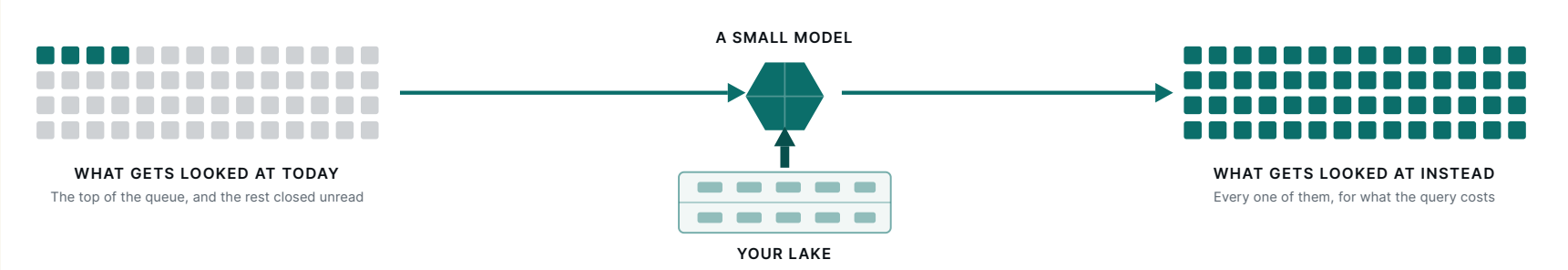
None of this ships in a box. Anyone can sell you a model, and within a year everyone will sell you the same ones. What nobody can sell you is the record of how your estate is put together and how your people work inside it. That is built once, with you, and it is yours afterwards.

This is what makes the intelligence efficient rather than merely present. It is also the ground the next thing stands on, because an agent that cannot describe your estate accurately is never going to be trusted to change it.

PHASE 03 · IN DETAIL

The cheap model does the endless work

Not every question deserves the same brain, and owning the ground the intelligence runs on is what lets you decide, question by question. That is a control lever before it is anything else, and what it buys is the work that never got done, because nobody budgets for the investigations they were never going to run.



THE CHEAP TIER

Low and warning alerts, lookups, retrieval, the compliance question somebody asks on a Tuesday afternoon. A small model, frontier or open weight, running against the partitioned lake all day. That tier is where the volume lives: severities that get closed unread today, at counts no team was ever going to work through, every one of them a thing somebody once thought was worth alerting on.

THE TIER THAT EARNS IT

The criticals. The correlations that cross four sources and a change record. The ones where a wrong answer costs somebody a night and a customer a morning. The best reasoning available, pointed at the small number of investigations that earn it.

TWO AGENTS, AND THE MODEL IS YOURS

Security triage and root cause are different problems and they get different agents, each with its own playbooks and its own model. And the model underneath either one is yours: point at something better the day it ships, run it in your account, and swap it with an edit in a repository you own rather than a renegotiation.

WHAT THAT BUYS

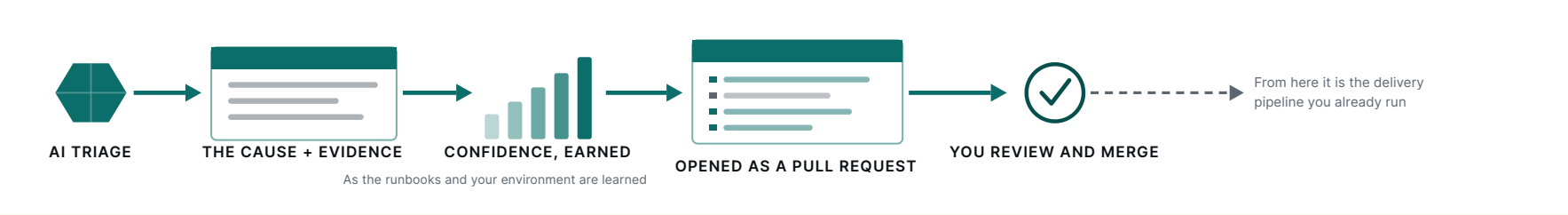
Searches that run whether or not anybody raised a ticket, hunting what nobody thought to write an alert for. Evidence pulls that used to be a two week ticket, answered the same day. And the people who own the services asking about their own service, in their own words, without first learning three query languages. Coverage is the product, and none of it is reachable through a metered module bolted onto one platform.

Buy the intelligence as a module and the vendor has already decided what every question is worth asking. Own it and you decide, per query, and you keep deciding as the models change underneath you. That is what turns triaging the top of the queue into triaging the queue.

PHASE 03 · IN DETAIL

Where this goes

Triage comes first, always. Getting to root cause quickly, on every alert rather than the few a budget allows, is what is available now and what pays for itself. It is also the ground everything after it stands on.



WHAT IS AVAILABLE NOW

An alert arrives, the agent investigates it against the full record and your runbooks, and a person reads a case instead of assembling one. Every alert, rather than the few anyone has time to reach, and on the tools that are already firing them.

THEN THE FIX IS PROPOSED

Where the estate is declared as code, the agent does not have to stop at explaining. It opens a pull request against the repository the resource lives in, with the reasoning attached, and a person reviews and merges. The fix arrives as a proposal, which is the only form of it anybody should accept from a machine.

WHAT THAT MAKES YOUR TEAM

Your team stops being the thing that finds out what happened and becomes the thing that decides what to do about it. Reviewers and owners rather than operators. No administrator has to become an engineer first, and that is the gap that closed.

WHAT GATES IT

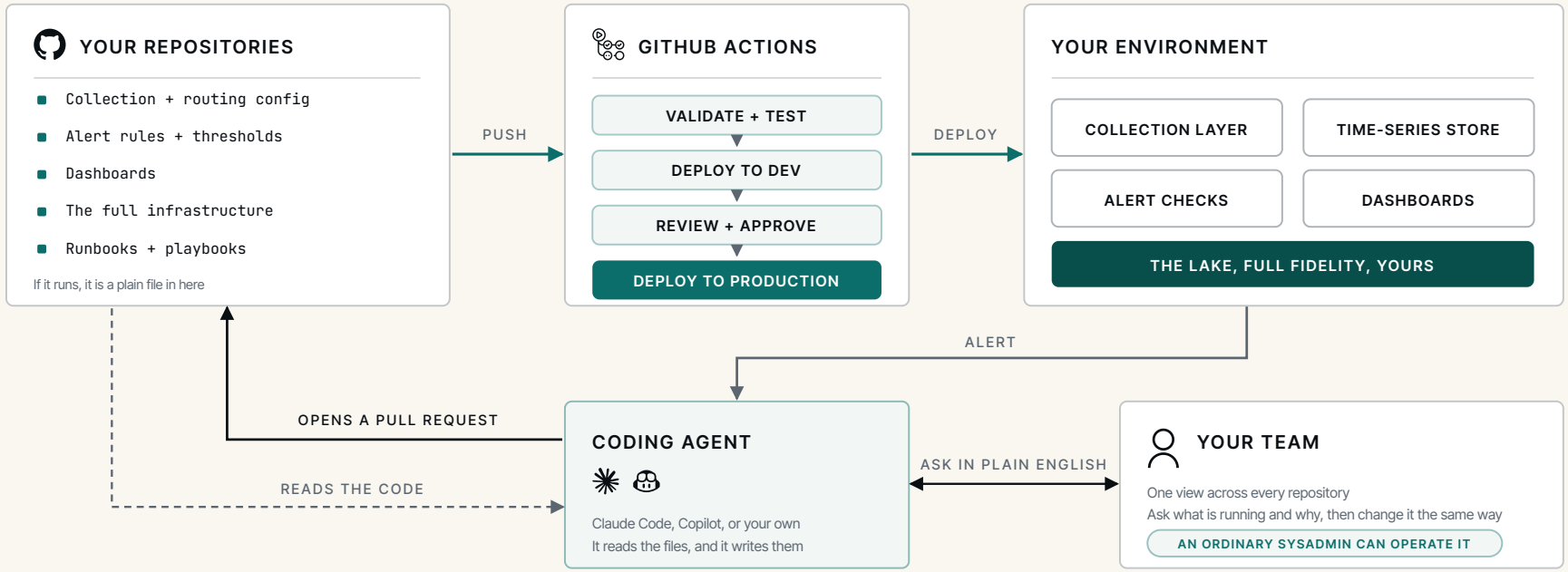
Two things gate it, and neither can be bought. The first is how much of the estate is declared as code. Phase 02 puts the monitoring platform itself there, and your own services are written as code already. What is usually left is the infrastructure those services run on, the servers, databases, load balancers and network rules the monitoring was watching. Coverage there is uneven, and much of what exists is scripts somebody runs rather than definitions that go through review. Where it is declared as code, a failing one is a file somebody corrects and reviews like any other change, and getting it there is work we can do with you. The second gate is whether the agent has earned the trust, which is what a year of auditable triage is for.

None of it is reachable without the foundation. The lake, the context and the control over the model are what make an agent good enough to be trusted, and being trusted is what turns triage into the fix.

HOW ALL OF IT IS BUILT

Everything as code

Start with the collection layer and the destinations, because those are the platforms your security and your performance rest on. Then do not stop there. Everything this blueprint builds is code in repositories you own, and the payoff compounds the further that spreads, because an agent can only reason about what it can actually read, and a console is where that knowledge goes to become unreadable. We make ourselves as replaceable as we make your vendors, and the build survives its builders.



WHEN IT GOES DOWN

The platform that watches everything is infrastructure too. As code, its recovery story is one line: redeploy from the repository.

WHEN YOU NEED TO ASK

What is running, why it is configured this way, how the repositories fit together. Your team asks in plain English and an agent answers.

WHEN SOMETHING BREAKS

The agent that triaged the alert opens the pull request that fixes it, and a person merges it. That works because the fix is a file.

WHO OPERATES IT

Adding a rule is adding a file, so an ordinary administrator runs this. You hold the repository, and we are a retainer rather than a dependency.

HOW ALL OF IT IS BUILT · IN DETAIL

What AI native actually means

Most estates are configured in consoles. A setting here, a script there, a dashboard somebody built and left behind. None of it can be read as a whole, by a person or by anything else. Holding it as code is not a delivery preference, it is the thing that makes all four of these possible.

ANYONE CAN ASK

What is running, why it is configured this way, what changed this morning. The answer comes out of the files rather than out of whoever set it up, so it survives that person leaving and it does not begin with learning a query language. People who do not write code get answers too.

CHANGING IT IS AN EDIT

What used to be a click in a console is a few lines in a file. You describe the change to a coding agent, it makes the edit, and it goes through the same pipeline as everything else: reviewed, tested and validated before it ships. Safe by process rather than by care.

THE HISTORY IS THE AUDIT TRAIL

Every change is a commit with an author, a date and a reason. When something breaks, the change that broke it is findable, whether it landed this morning or two quarters ago. That is the question a dashboard has never once answered.

A TICKET BECOMES A PULL REQUEST

Much of the service desk queue is not incidents at all. It is requests: onboard this source, add an alert for that host, raise a threshold. The agent picks the ticket up over your service desk's own MCP server, writes the change into the repository that rule or source lives in, and opens a pull request. Somebody reviews and merges, and the pipeline ships it. That only works because the thing being asked for is a file.

THE OBSERVABILITY STACK

Collection, routing and enrichment, the lake, and the alert rules and dashboards on every destination, including the platforms you keep.

THE ESTATE UNDERNEATH

The infrastructure itself declared as code, so a resource that is wrong is a file that is wrong, and correcting it is the same motion as everything above.

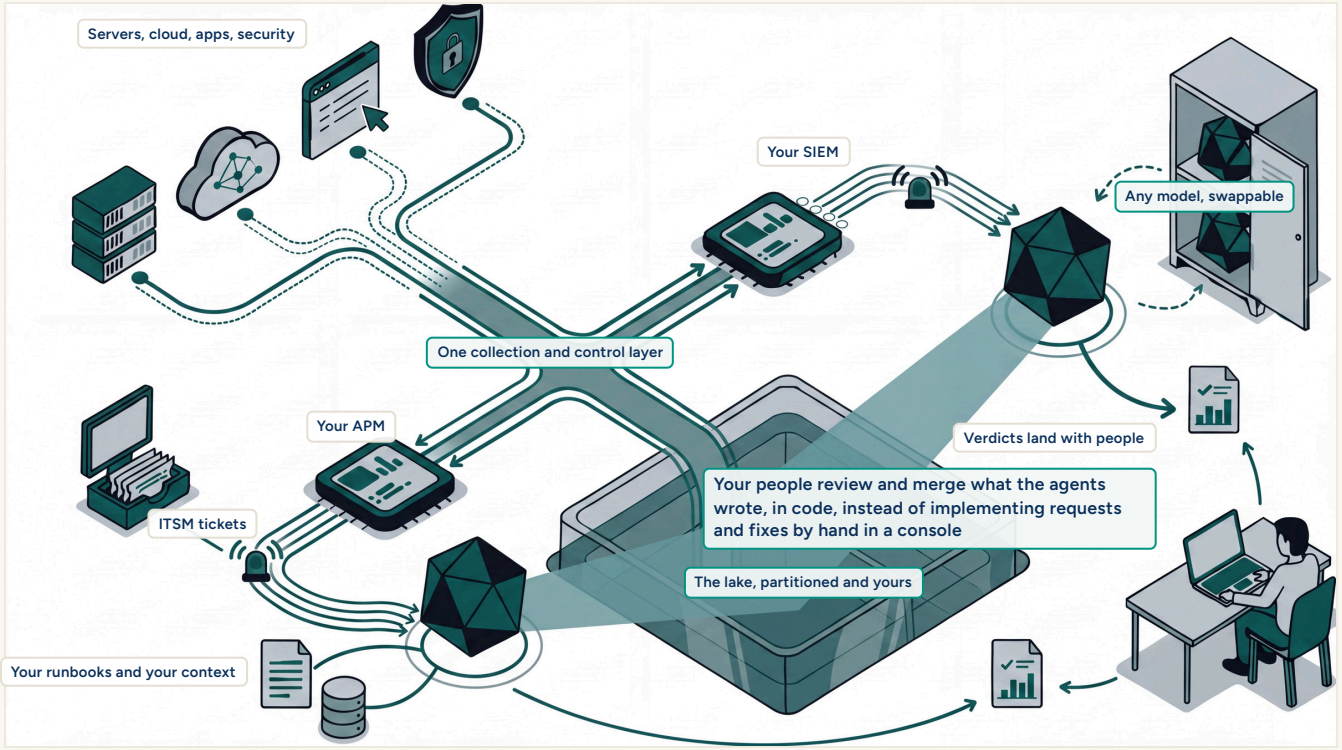
THE KNOWLEDGE AROUND IT

Runbooks, service ownership, and the notes explaining how the parts relate, kept as files beside the code they describe rather than in a wiki nobody opens.

Your platforms sell a version of this too. Theirs is an assistant living inside their own console, so it reasons about what that console holds and stops at its edge. The same idea over an estate you hold as code reaches the collection layer, the destinations, the lake and the infrastructure underneath, because every part of it is readable. That is what AI native means here, and it is why the phases run in the order they do.

WHERE IT LANDS

The machine, complete



TWO TO THREE YEARS FROM NOW

Work the phases and this is the estate you run in two or three years: telemetry and cost under control, everything landing enriched in a lake partitioned the way investigations move, so it serves compliance, audit and the agents at once.

Agents work each alert as it fires, one on security triage and one on root cause, and they hunt the history for what nobody thought to alert on. Findings land in the ticket your service desk already uses. By then the queue runs both ways: a request for a new source or a new alert, and a fix an agent proposes off its own finding, both arrive as pull requests rather than as console work.

From outside nothing moved. Same sources, same destinations, same service desk. In between sits a layer you own, where the models are yours and the runbooks are your team's.

The handoff is the point. We build it, we operate it, and then we teach your own administrators to run it: how the configuration is written, how to change it by describing what you want to a coding agent, and how the pipeline tests and validates every change before it ships. Not how to become engineers.

After that, a new rule, a new destination or a new collection is a small standard edit, and once the playbooks are written the triage largely runs itself. Keep us on retainer for the harder work, because you want to rather than because you have to. No administrator has to become an engineer first. That is the gap AI closed, and it is why a partner you can replace is one you keep by choice.

Start with the architecture review.

Bring your architecture diagram and we draw your version of this: the foundation alone, triage on the lake you already have, or your own open stack replacing what you rent. It grows in phases, so none of it has to happen at once. No system access, no obligation.

logmetry.io · info@logmetry.io