

Observability for IT Operations

Moving beyond monitoring with correlated telemetry,
OpenTelemetry and service level objectives

IT INFRASTRUCTURE & MANAGEMENT

Published October 2026 · cybertechinsights.com

EXECUTIVE SUMMARY

From “is it up?” to “why is it slow?”

Traditional monitoring answers questions you knew to ask in advance: is the server up, is the disk full, is CPU above a threshold? Modern applications — spread across cloud services, containers, APIs and third-party SaaS — fail in ways nobody predicted. Observability is the ability to understand what is happening inside a system from the data it produces, including problems you have never seen before.

This whitepaper explains the difference between monitoring and observability, the telemetry signals involved, the role of the OpenTelemetry standard, and a phased roadmap IT operations teams can follow without an all-at-once platform replacement.

Key takeaways

- Monitoring and observability are complementary; observability adds the ability to ask new questions.
- Collect metrics, logs and traces with shared context so they can be correlated.
- Standardise instrumentation on OpenTelemetry to avoid vendor lock-in.
- Alert on user-facing service level objectives (SLOs), not on every component metric.
- Plan for telemetry cost from day one: sampling, retention tiers and cardinality limits.

CONCEPTS

Monitoring vs. observability

	Monitoring	Observability
Question answered	Is a known condition happening?	Why is the system behaving this way?
Approach	Predefined checks, thresholds and dashboards	Rich, correlated telemetry explored on demand
Best for	Known failure modes and capacity	Distributed systems and novel failures
Typical output	Alerts and status pages	Investigations, root cause and SLO reporting

TELEMETRY

The signals that matter

- **Metrics** — numeric measurements over time (request rate, error rate, latency, saturation). Cheap to store and ideal for alerting and trends.
- **Logs** — timestamped records of discrete events. Structured logs (for example JSON with consistent fields) are far easier to search and correlate than free text.

- **Traces** — the end-to-end path of a single request across services, showing where time is spent and where errors originate.
- **Events and profiles** — deployments, configuration changes and code-level performance profiles add the “what changed?” context that speeds up diagnosis.

The value comes from correlation: a latency spike on a metric should lead directly to the slow traces and the related log lines, using shared identifiers such as trace IDs and service names.

STANDARDS

Why OpenTelemetry

OpenTelemetry is an open-source, vendor-neutral project under the Cloud Native Computing Foundation that provides APIs, SDKs and a collector for generating and exporting metrics, logs and traces.

Instrumenting applications once with OpenTelemetry means you can send data to one or several back ends and change tools later without re-instrumenting code.

- Start with automatic instrumentation for common languages and frameworks, then add custom spans for key business transactions.
- Deploy the OpenTelemetry Collector to filter, sample, enrich and route telemetry centrally.
- Agree naming conventions (service, environment, version) so data from different teams lines up.

OPERATIONS

Alert on what users feel: SLOs

Thousands of component-level alerts produce fatigue, and fatigued teams miss real incidents. Service level objectives focus attention on user experience. An SLO such as “99.9% of checkout requests complete successfully in under 500 ms over 30 days” defines an error budget — the amount of unreliability that is acceptable. Alert when the budget is being consumed too quickly, and use the remaining budget to balance reliability work against new features.

- Choose a small number of SLOs per critical service, based on availability and latency.
- Page people only for fast error-budget burn; route slower trends to tickets.
- Review every page: if it was not actionable, change or remove the alert.

COST

Keeping telemetry spend under control

Telemetry volume grows with every new service, and costs can grow faster than the value delivered. Build cost controls into the design:

- **Sampling:** keep all error and slow traces, and sample routine successful requests.
- **Retention tiers:** keep recent, detailed data in fast storage and move older data to cheaper tiers.
- **Cardinality limits:** avoid unbounded labels such as user IDs on metrics.
- **Ownership:** show each team the volume and cost of the telemetry it produces.

ROADMAP

A phased approach

Phase	Focus	Outcome
1. Foundations	Inventory services, standardise structured logging, deploy the OpenTelemetry Collector.	Consistent, searchable telemetry.
2. Correlation	Add distributed tracing to critical user journeys; link metrics, logs and traces.	Faster root-cause analysis.
3. SLOs	Define SLOs and error budgets for top services; rework paging rules.	Fewer, more meaningful alerts.
4. Optimise	Sampling, retention tiers, cost reporting, change-event correlation.	Sustainable cost and continuous improvement.

Checklist: observability readiness

- Critical services and user journeys are documented, with owners.
- Logs are structured and share service, environment and trace identifiers.
- Traces cover the most important user journeys end to end.
- SLOs exist for top services, and paging is based on error-budget burn.
- Telemetry volume and cost are visible to the teams that create them.

About Cyber Tech Insights: a B2B publication providing research and resources for technology and business decision-makers. This document is for general information only and does not constitute professional advice. © 2026 Cyber Tech Insights.