Cloud Native Observability Ecosystem: Scenario-Oriented Selection Framework and Use Case Analysis

Zhang Jingtong[1]

Nanjing University of Information Science and Technology, Nanjing, China
202283910012@nuist.edu

Abstract. With the widespread adoption of cloud-native architectures (e.g., microservices, Kubernetes), observability has become core to ensuring system stability and efficient troubleshooting. The fragmented CNCF observability landscape, spanning metrics, logs, traces and alerting, often leads to tool selection dilemmas for teams with different constraints. We propose a scenario-oriented selection framework across four dimensions (team, business scale, resource budget, technical requirements) and validate it via three use cases (small team, large-scale cross-regional, edge IoT). The framework offers practical tool combinations and implementation guidelines to balance functionality, cost and maintainability.

Keywords: Cloud Native · Observability · CNCF Ecosystem · Technical Selection Framework · Microservices · Edge Computing

1 Introduction

1.1 Research Motivation

The shift from monolithic applications to cloud-native architectures has significantly improved development efficiency and system flexibility, driven by technologies such as microservices decomposition, Kubernetes dynamic scheduling, and edge computing extension. However, this transformation has introduced unprecedented operation and maintenance (O&M) complexity: distributed call chains involving multiple microservices across nodes/regions increase fault-location difficulty; dynamic infrastructure changes render traditional static monitoring ineffective; heterogeneous deployment environments (public cloud, private cloud, edge) demand unified observability; and explosive growth of monitoring data challenges storage and analysis capabilities.

Observability, defined by the "three pillars" (metrics, logs, traces), has become the key solution to these challenges. Metrics provide quantitative insights into system behavior (e.g., QPS, response time), logs record discrete events (e.g., error messages), and traces track request flows across microservices [7]. The CNCF ecosystem has emerged as the de facto standard for cloud-native observability, hosting core tools such as Prometheus (metrics), Loki (logs), and Jaeger

(traces) [1,5,2,3]. However, existing research and practices have critical gaps: lack of systematic scenario-oriented selection guidelines, insufficient consideration of scenario constraints (e.g., edge resource limitations), and weak practicality of theoretical frameworks. Enterprises often face dilemmas such as over-investment in complex toolchains (small teams) or inadequate scalability (large enterprises). This paper aims to fill these gaps with a scenario-oriented framework validated by practical use cases.

1.2 Contributions

This paper makes three key contributions:

- Systematically collates core tools in the CNCF observability ecosystem, clarifying their technical characteristics and applicable boundaries.
- Proposes a scenario-oriented selection framework with four core dimensions and three typical templates, converting abstract decisions into operable steps.
- Validates the framework through three representative use cases (small team, large-scale cross-regional, edge IoT), providing replicable implementation guidelines for common cloud-native scenarios.

1.3 Paper Structure

Section 2 reviews the evolution of cloud-native observability and core CNCF tools; Section 3 details the framework's design principles, core dimensions, and scenario templates; Section 4 verifies the framework through three use cases; Section 5 discusses limitations and future directions; Section 6 concludes.

2 Background and Related Work

2.1 Evolution of Cloud-Native Observability

The evolution of observability in software systems can be broadly divided into three stages:

1. **Monolithic Application Era (pre-2015):** Host-level monitoring tools (e.g., Nagios, Zabbix) and centralized logging (ELK stack) dominated, designed for static single-host environments. These approaches struggled with the dynamism of later cloud-native systems.
2. **Early Cloud-Native Era (2015–2019):** CNCF standardization emerged. Prometheus (2016) became the de facto metrics standard with pull-based collection and Kubernetes-native discovery; Jaeger (2017) addressed distributed call chain tracing; Loki (2018) optimized logging with label-based indexing. The "three pillars" solidified, but signal correlation remained limited.
3. **Mature Cloud-Native Era (2020–present):** Emphasis on unified observability via OpenTelemetry (Otel), lightweight edge adaptation (e.g., Prometheus Agent, Fluent Bit), and intelligent capabilities (AI-driven anomaly detection and RCA).

Table 1: Representative CNCF observability tools and scenarios.

Category	Representative Tools	Core Capabilities / Typical Scenarios
Metrics Monitoring	Prometheus; Thanos	Time-series + PromQL; global query and long-term store for large/geo-distributed setups
Logging	Loki; Fluent Bit	Label-based logs; lightweight collection/filtering; tight Grafana integration; edge friendly
Distributed Tracing	Jaeger	End-to-end traces; latency breakdown; microservice bottleneck diagnosis
Alerting	Alertmanager	Routing, grouping, inhibition; multi-channel notifications for Prom alerts

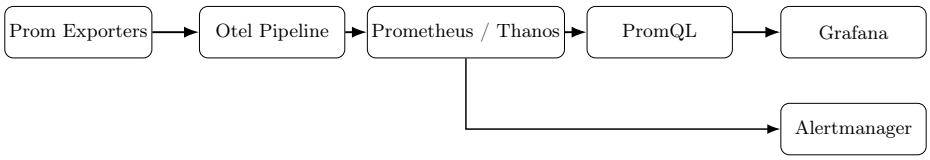

Fig. 1: Representative metrics pipeline: exporters to pipeline to Prometheus/Thanos, queried by PromQL into Grafana; alerts via Alertmanager.

2.2 CNCF Observability Ecosystem

The ecosystem can be viewed in layers: collection/ingress, transport/pipelines, storage, query APIs, and visualization/alerting. Representative projects include Otel Collector, Fluent Bit, Prometheus/Thanos, Loki, and Jaeger, surfaced via PromQL/LogQL/trace query APIs into Grafana and Alertmanager.

2.3 Limits in Existing Guidance

Existing guidance is often tool-centric rather than scenario-centric and exhibits several limitations: (i) lack of selection guidance tailored to team size and enterprise scale; (ii) insufficient adaptation to edge environments where traditional stacks can exhaust resources; (iii) high integration and maintenance costs demanding deep cloud-native expertise; (iv) weak cross-signal correlation across metrics/logs/traces; and (v) error-prone manual configuration that scales poorly with microservice growth. These gaps motivate our scenario-oriented framework.

2.4 Related Work Comparison (Qualitative)

Table 2 contrasts prominent open-source and SaaS options across qualitative dimensions frequently discussed in practice and reports [4,6,7].

Table 2: Qualitative comparison of observability stacks.

Dimension	Prom+Thanos	Grafana Mimir	VictoriaMetrics	SaaS (e.g., Datadog)
Scalability (metrics)	High;objstore; fan-out	High; horiz. scale	High;single-binary	High; managed
Operability	Med; S3/GCs + compaction	Med; Helm/operator	Med; simple deploy	High(outsourced)
Cost control	Goodinfra control	Good (self-host)	Good (efficient store)	Variable (usage-based)
Ecosystem compat.	Excellent (CNCF-native)	Excellent (PromQL)	Good (PromQL-comp.)	Good;vendor features
Data sovereignty	Full control (self-host)	Full control	Full control	Limited; region-bound
Time-to-value	Moderate (helm+tuning)	Moderate	Moderate	Fast (turnkey)

3 Scenario-Oriented Selection Framework

3.1 Design Principles

We follow three principles: (i) scenario adaptability first (avoid over/under engineering), (ii) CNCF ecosystem compatibility (reduce lock-in and ensure interoperability), and (iii) balanced trade-offs across functionality, resource, cost, and scalability.

3.2 Core Dimensions

We profile scenarios with four dimensions grounded in operational constraints and non-functional requirements:

1. **Team size and capability:** number of O&M engineers, cloud-native expertise (Kubernetes, PromQL, tracing), availability of dedicated SRE/platform engineers.
2. **Business scale and complexity:** microservice count, DAU/QPS, deployment topology (single vs. multi-cluster, cross-region), call chain depth and variance.
3. **Resource budget:** cluster resource ceilings (CPU/memory/storage/bandwidth), infrastructure type (on-prem/public cloud/edge), and cost sensitivity (e.g., object storage and egress costs).
4. **Technical requirements:** functional scope (metrics/logs/traces/alerting, cross-region aggregation, offline caching) and non-functional SLOs (availability, latency), retention periods, and correlation needs.

Table 3: Scenario templates and recommended combinations.

Template	Key Characteristics	Recommended Stack / Notes
Small Team & Lightweight	<10 services; single cluster; DAU <100k; part-time O&M	Prom+Grafana+Loki+AM; single-node; metrics 7d; logs 3d; Helm one-click; golden signals
Cross-Region Large-Scale	>100 services; multi-region; 99.99% SLO; 90d retention	Per-cluster Prom; Federation/Thanos (global query+LTS); Jaeger; Loki; traceID correlation; downsampling
Edge IoT	Constrained nodes (2vCPU/4GB); intermittent links; offline cache	K3s; Prom Agent; FB; MinIO cache; selective sync; local Grafana; cloud aggregation

These dimensions convert abstract choices into concrete constraints (e.g., "3 engineers, 8 services, 50k DAU, single cluster" maps to the small-team template), reducing over-engineering for small teams and under-engineering for large deployments.

3.3 Process

We follow a compact four-step loop: (1) scenario profiling (team/scale/budget/requirements), (2) dimension scoring to surface constraints (e.g., retention, SLOs), (3) template matching (Small team / Cross-region / Edge IoT), and (4) implementation and iterative tuning (deployment, sampling, retention, alert routing).

3.4 Scenario Templates

Table 3 summarizes three common templates used throughout the use cases. AM denotes Alertmanager; FB denotes Fluent Bit.

Application Steps. (1) Scenario profiling using the four dimensions; (2) Template matching with explicit inclusion/exclusion criteria; (3) Architecture design (deployment mode, retention, sampling, alert routing); (4) Iterate based on observed reliability, latency, cost, and operator toil.

3.5 Illustrative Scoring (Brief)

For reproducibility, teams can assign 1–5 scores on the four dimensions and select the template with the highest weighted sum; weights emphasize constraints (e.g., retention or global query for cross-region). We omit the full matrix table for brevity.

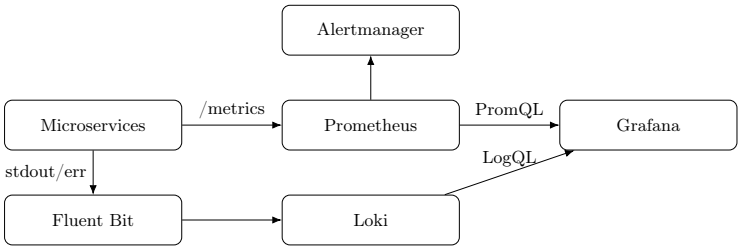

Fig. 2: Use Case 1: lightweight stack for small teams.

4 Use Cases and Analysis

4.1 Use Case 1: Small Team Lightweight Microservices

Scenario. A startup with ∼5–8 microservices on a 3-node Kubernetes cluster (2 vCPU / 8 GB each). Three full-stack engineers with limited cloud-native expertise; goals: core SLO dashboards (QPS, latency, error rate), error log triage, basic threshold alerts.

Stack. Prometheus (single), Grafana, Loki (single) with Fluent Bit, Alertmanager. Prometheus scrape interval 30s; metrics retention 7d; logs retention 3d.

Design Notes. Helm-based install; focus on golden signals; avoid over-engineering (e.g., no Thanos).

Effect Evaluation. Total resource consumption is 0.8 vCPU and 1.6 GB memory; deployment time ≈25 minutes with Helm; average fault localization time reduced to ≈8 minutes (e.g., payment API timeouts due to DB pool exhaustion identified via dashboards and logs). Eight valid alerts were triggered with no observed false positives. Compared to alternatives, ELK+Prometheus consumed significantly more resources, while SaaS offerings improved time-to-value at noticeably higher recurring cost.

4.2 Use Case 2: Large-Scale Cross-Regional E-Commerce

Scenario. >150 microservices across three regions (multi-cluster). Strong SLOs (99.99%) and 90-day metrics retention; dedicated platform team.

Stack. Per-cluster Prometheus and Loki; Jaeger Agents/Collectors per region; global Prometheus Federation and/or Thanos for long-term storage and global query; Grafana enterprise dashboards.

Design Notes. Downsampling in Thanos; centralized alert routing (P0 phone, P1 chat/email); correlation via traceID.

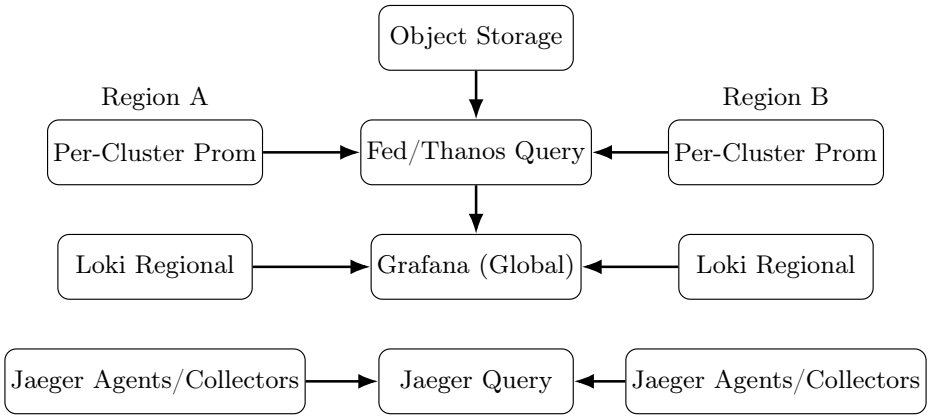

Fig. 3: Use Case 2: cross-regional aggregation with Federation/Thanos and global queries.

Effect Evaluation. Achieved 99.995% uptime; cross-region metric aggregation latency < 300 ms; mean time to resolve (MTTR) for distributed incidents reduced from 30+ minutes to 5–8 minutes; seamless scale-out to two additional regions without architecture rewrites. Relative to a centralized single-Prometheus approach, the design avoided storage and query bottlenecks and reduced cross-region data egress costs by 80% via local collection and global downsampling.

4.3 Use Case 3: Edge IoT Monitoring

Scenario. Thousands of devices across plants; K3s edge clusters; constrained nodes (2 vCPU / 4 GB), intermittent connectivity; local alerts and offline caching required.

Stack. K3s, Prometheus Agent, Fluent Bit, MinIO for edge caching; cloud Prometheus for aggregation; local and cloud Grafana.

Design Notes. Selective sync (abnormal/aggregated); no local TSDB on agent; bandwidth savings.

Effect Evaluation. Edge resource consumption 0.3 vCPU and 1.2 GB memory ($\approx$30% of node capacity); bandwidth reduced by 85% compared with full-fidelity upstreaming; no data loss observed during 2–3 hours of disconnection windows; local alerts fired within 1 s and remote alerts within 5 minutes after connectivity recovery. The approach outperformed traditional Nagios+Rsyslog in edge/cloud collaboration and scalability.

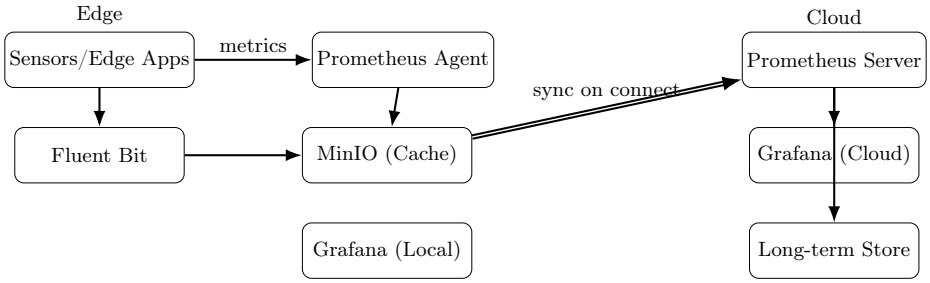

Fig. 4: Use Case 3: edge caching and selective synchronization to cloud.

5 Discussion and Limitations

5.1 Trade-offs and Practical Considerations

Scenario-oriented selection emphasizes fitness over maximal functionality. Small teams should prefer single-node deployments and shorter retention to minimize cost and operational overhead. Large-scale deployments benefit from global query layers (Federation/Thanos) and downsampling, but introduce complexity (object storage, compaction, query fan-out). Edge scenarios prioritize lightweight agents and offline durability, accepting eventual consistency to save bandwidth.

5.2 Security, Compliance, and Data Governance

Telemetry often contains sensitive metadata. Centralized log stores and cross-region aggregation must consider encryption at rest/in transit, multi-tenant isolation, and data residency. OpenTelemetry pipelines can enforce redaction and sampling; Loki and Jaeger support multi-tenant modes. For regulated industries, retention policies and audit trails should be codified.

5.3 Limitations of This Work

Our framework is qualitative and does not provide quantitative thresholds (e.g., exact service counts or DAU breakpoints). The three templates do not cover serverless-only or multi-cloud mesh scenarios. We do not evaluate proprietary/SaaS offerings, which may simplify operations but raise cost and data-sovereignty concerns.

5.4 Future Directions

We plan to: (i) add serverless and multi-cloud templates; (ii) provide a scoring matrix and a simple tool to recommend templates from inputs; (iii) integrate anomaly detection and RCA via OpenTelemetry signals; (iv) extend edge guidance with store-and-forward policies and cost models.

Table 4: Self-hosted vs. SaaS observability (qualitative).

Dimension	Self-hosted (Prom+Thanos/Loki/Jaeger)	SaaS (e.g., Datadog)
Time-to-value	Medium; deploy+tune Helm/ops	Fast; turnkey
Operability	Needs SRE/infra (obj store, compaction, upgrades)	Outsourced ops; vendor SLAs
Cost model	Infra + labor; predictable at scale	Usage-based; can spike
Data sovereignty	Full control; on-prem/region placement	Limited by vendor regions/policies
Ecosystem/lock-in	CNCF-native; portable	Vendor features/integrations; lock-in risk
Advanced features	Build as needed; OSS ecosystem	Built-in analytics/AI; rapid pace

5.5 Self-hosted vs. SaaS (Qualitative Comparison)

Table 4 summarizes qualitative trade-offs often considered in practice when choosing between self-hosted open-source stacks and SaaS platforms.

6 Conclusion

We proposed a scenario-oriented framework for selecting cloud-native observability stacks across four dimensions and validated it with three representative use cases. The framework helps teams balance functionality, cost, and maintainability while remaining compatible with the CNCF ecosystem. Future work includes quantitative scoring, serverless/multi-cloud templates, and integrating AI-driven anomaly detection to further reduce operational toil.

References

1. Cncf cloud native landscape. `https://landscape.cncf.io`, accessed: 2025-12-23
2. Grafana loki documentation. `https://grafana.com/docs/loki/latest/`, accessed: 2025-12-23
3. Jaeger: open source, end-to-end distributed tracing. `https://www.jaegertracing.io/docs/`, accessed: 2025-12-23
4. Opentelemetry specification. `https://opentelemetry.io/docs/specs/`, accessed: 2025-12-23
5. Prometheus documentation. `https://prometheus.io/docs`, accessed: 2025-12-23
6. Thanos: Global, scalable, highly available prometheus setup. `https://thanos.io/tip/`, accessed: 2025-12-23
7. Majors, C., Fong-Jones, L., Miranda, G.: Observability Engineering: Achieving Production Excellence. O'Reilly Media (2022)