Logging Architecture: Expanding InfoLogger into a Unified, Scalable Logging Platform

Athanasios Papadopoulos
CERN, Geneva, Switzerland

Abstract
We introduce a logging architecture that integrates ALICE InfoLogger [1] and DDS log files into a unified platform. Using Fluent Bit [3] for collection, OpenSearch [2] for indexing, and Kafka [4] for distribution, the system decouples ingestion, storage, and visualization. Dashboards in Grafana [5] and alert routing via Alertmanager [6] provide operators with real-time visibility and actionable alerts. The design enhances scalability and resilience while preserving current workflows and enabling future extensions such as anomaly detection and cross-system correlation.

Keywords
CERN report; ALICE; EPN; logging; InfoLogger; OpenSearch; Kafka; Fluent Bit; Grafana; Alertmanager.

1 Introduction

Logging for large-scale distributed systems must support high ingestion rates, low-latency inspection, robust retention, and flexible downstream consumption. In ALICE, InfoLogger [1] has long served as the primary path for log ingestion and operator display. While this design has proven reliable, it is tightly coupled and does not natively accommodate new, high-volume data sources such as DDS log files.

To address these limitations, we propose an expanded architecture that unifies InfoLogger and DDS logs in a common platform. Built on open-source technologies, the system separates ingestion, storage, and visualization, improving scalability and resilience while preserving existing InfoLogger workflows. It also establishes a foundation for advanced capabilities such as anomaly detection, correlation analysis, and automated alerting. The following sections describe the current system, the proposed architecture, and its component roles, before concluding with a discussion of advantages, operational considerations, and future directions.

2 Current Architecture

InfoLogger clients send messages to `infoLoggerD`, which forwards them to an `infoLoggerServer`. Messages are archived in a SQL database and streamed to InfoBrowser clients for real-time visualization. This design is operationally simple but tightly couples ingestion, storage, and display.

A critical limitation is that it *ignores all log files generated directly by the processes*, such as the high-volume DDS logs. These files remain outside the InfoLogger pipeline and cannot be indexed, searched, or correlated with InfoLogger messages. At present, the only way to access them is through direct inspection on the host machines via the terminal, which severely restricts their usability for monitoring and analysis. Moreover, the current stack provides no native alerting pathway: alerting rules cannot be set up against either the live stream or the SQL archive, leaving operators to rely on manual inspection.

3 Proposed Architecture

The new design decouples ingestion, collection, alerting and display, standardizing on widely supported open-source components (Fig. 2):

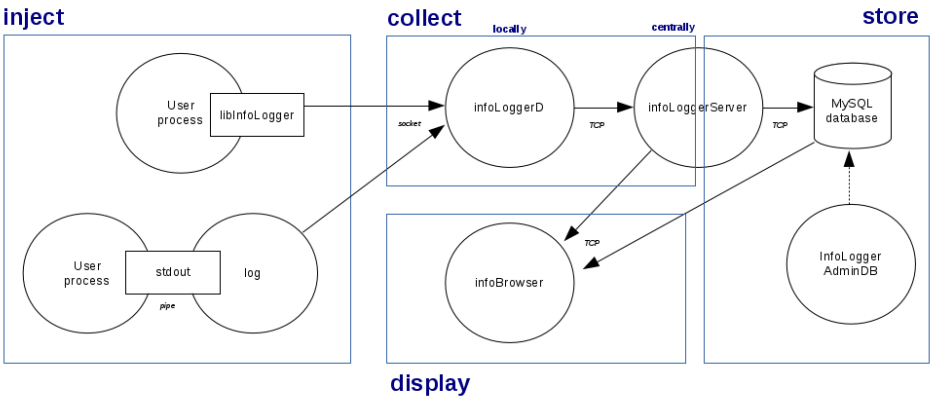

Figure 1: Current InfoLogger-based dataflow.

- **Injection:** InfoLogger output [1], DDS log files, and additional log types supported by Fluent Bit plugins [3] (e.g. systemd journals, syslog, hidden service logs such as ODC and DDS internals, JSON/NDJSON application logs, or container/runtime logs) are captured on each node by a local Fluent Bit collector.
- **Collection:** Collectors parse, tag, and filter logs, forwarding them to (i) OpenSearch [2] for indexing and search and (ii) Kafka [4] for selected topics (e.g. live feed). A Fluent Bit aggregator subscribes to Kafka topics to produce a consolidated stream without tying collectors directly to extra consumers.
- **Display and Alerting:** Grafana dashboards [5] provide real-time and historical views. Alerts are evaluated on top of OpenSearch (including anomaly detection and query-based rules) and routed via Alertmanager [6] for delivery and visualization.

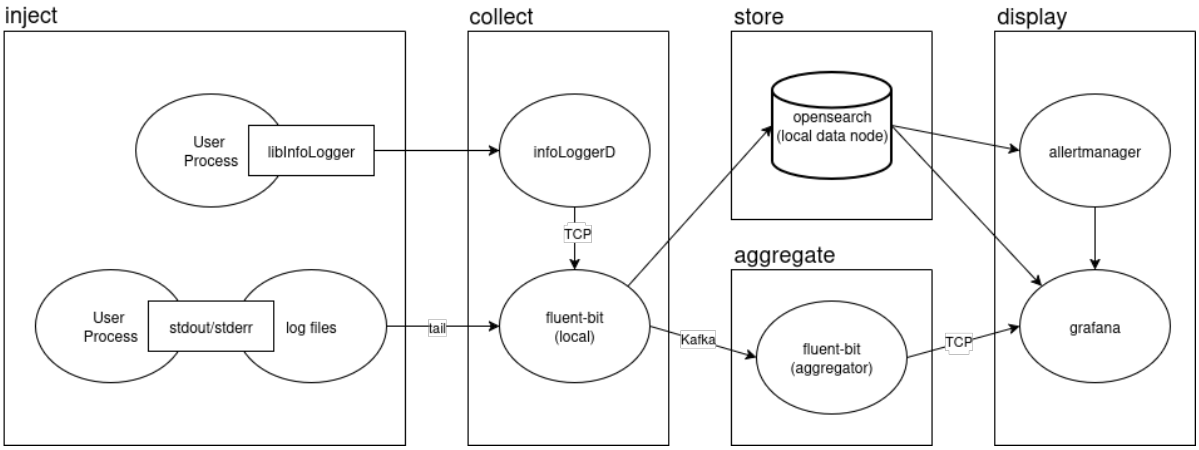

Figure 2: Proposed logging architecture dataflow.

3.1 Component Roles

3.1.1 Fluent Bit (Collectors and Aggregator)

Collectors run on each node, ingesting InfoLogger output and local DDS log files, performing parsing, tagging, and filtering, and forwarding to OpenSearch and Kafka. The aggregator subscribes to Kafka topics, merges streams across nodes, and exposes a low-latency feed for dashboards and other consumers.

3.1.1.1 *Index families for DDS logs.*

DDS-derived logs are split by a Fluent Bit filter into two OpenSearch index families: *generic-logs-info* for informational entries with severity `I`, `INFO`, or empty; and *generic-logs-other* for warnings, errors, critical, and debug messages. This separation enables differentiated lifecycle policies—shorter, higher-compression retention for informational chatter and longer, more query-friendly retention for the rest.

3.1.2 OpenSearch

A distributed OpenSearch cluster indexes logs with index-family–specific lifecycle policies [2]. At present, a single multi-node cluster is used because it is the recommended and simplest option to configure and operate. In the future, alternative models (e.g. hybrid or cross-cluster search) could be explored if workloads prove too demanding, although significant performance gains are not expected.

3.1.2.1 *OpenSearch Prometheus Metrics*

. In addition to log indexing, OpenSearch exposes an embedded Prometheus exporter for cluster and node metrics (e.g. shard states, indexing pressure, query latency). While not part of the current design, this exporter is planned for future use to monitor node performance.

3.1.3 Kafka

Kafka [4] provides a durable, scalable event backbone. It is used selectively: not as the primary ingestion path for all logs, but for services that benefit from decoupling such as the live feed aggregator. Since OpenSearch is already distributed and the critical destination for all logs, connecting Fluent Bit directly to OpenSearch avoids introducing an extra mediator.

For future correlation and dependency analysis, Kafka topics (e.g. `infologger-errors`) will feed both graph-construction services and correlation engines as independent consumers.

3.1.4 Grafana

Grafana [5] offers customizable dashboards for real-time monitoring and historical search across InfoLogger and DDS indices.

3.1.5 Alertmanager

Alerts are unified via Alertmanager [6], originating from OpenSearch rules or Grafana evaluations. OpenSearch is the preferred layer for alert logic because it supports rich query conditions and anomaly detection directly against indexed data. Alertmanager acts primarily as a routing and silencing service, with visualization in Grafana.

3.2 Dashboards

Three primary operator views are provided: an InfoLogger dashboard (high-rate, operator-centric signals) (Fig. 3), a DDS log dashboard (file-derived telemetry) (Fig. 4), and a dedicated live feed sourced from the aggregator, which complements historical search by providing rapid situational awareness.

4 Discussion

4.1 Advantages

The proposed architecture provides several key benefits:

– **Unified ingestion:** InfoLogger, DDS, and additional log families are integrated into a single platform via Fluent Bit [3].

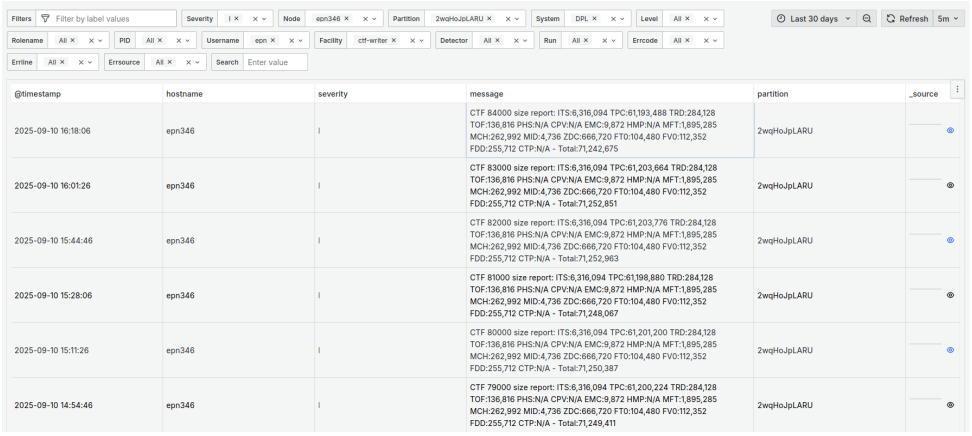

Figure 3: Example InfoLogger dashboard view.

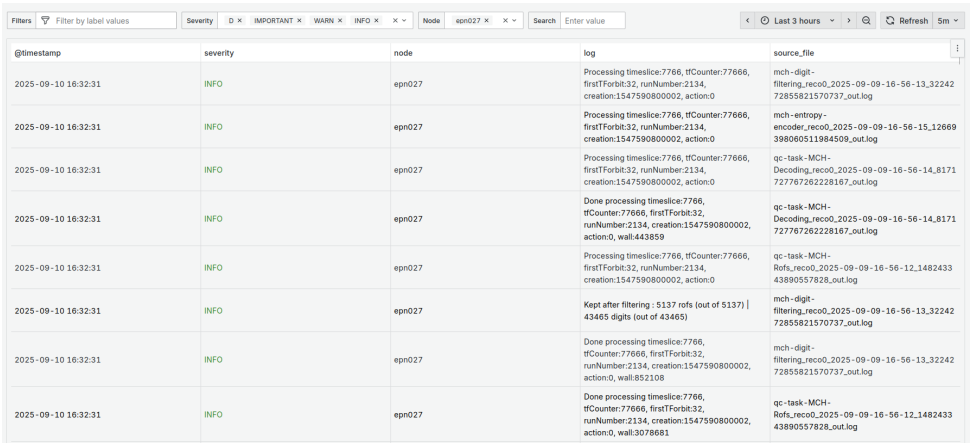

Figure 4: Example DDS log dashboard view.

- **Scalability and resilience:** OpenSearch [2] supports distributed indexing, while Kafka [4] decouples producers and consumers for selective use cases such as live feeds and correlation analysis.
- **Optimized retention:** Per-index lifecycle policies balance storage cost with query performance.
- **Modern monitoring and alerting:** Grafana dashboards [5] offer customizable real-time and historical views, with centralized alerting routed via Alertmanager [6].
- **Extensibility:** Kafka streams and Fluent Bit plugins enable additional consumers, correlation engines, and integration of further log sources without disrupting the core workflow.

4.2 Operational Considerations

From the operator perspective, InfoLogger usage remains unchanged. The main difference is the availability of three complementary Grafana dashboards (InfoLogger logs, DDS logs, and a live aggregator feed).

DDS logs are denser than InfoLogger logs, making storage planning critical. Careful index-family design and lifecycle policies are required to control growth while ensuring useful retention. Node-scoped alerts can be supported because Fluent Bit enriches indices with hostnames, though such alerts are expected to be uncommon.

For asynchronous nodes, a proposed configuration setting in `infoLoggerD` would suppress for-

warding to the legacy InfoLogger server and instead direct messages only to Fluent Bit, ensuring consistent collection in OpenSearch without duplicating traffic.

4.3 Future Directions

Planned developments focus on deployment practices and analytical extensions:

- **Container orchestration:** Deploying with Kubernetes or similar platforms will streamline scaling, updates, and monitoring.
- **Advanced analytics:** Kafka streams may feed anomaly detection models, correlation engines, or graph-based dependency mapping tools.
- **Template mining and metrics extraction:** Automated log template discovery and metric derivation will support anomaly detection and performance monitoring.
- **Cross-system integration:** The architecture lays the groundwork for linking logs to knowledge graphs or Bayesian networks, enabling richer failure diagnosis and prediction.

Acknowledgements

The author thanks colleagues in the ALICE collaboration, and specifically the EPN team, for their valuable feedback and support throughout this project.

Bibliography

[1] O2-Infologger, `https://github.com/AliceO2Group/InfoLogger/tree/master`.

[2] OpenSearch Project, `https://opensearch.org/`.

[3] Fluent Bit, `https://fluentbit.io/`.

[4] Apache Kafka, `https://kafka.apache.org/`.

[5] Grafana, `https://grafana.com/`.

[6] Alertmanager, `https://prometheus.io/docs/alerting/latest/alertmanager/`.