Elasticsearch vs. OpenSearch vs. Loki vs. Quickwit vs. ClickHouse: The Complete Guide

2026-05-14

A field guide to Elasticsearch, OpenSearch, Loki, Quickwit, ClickHouse for 7+ year log archiving: storage, operations, security, UX, and a full OVH MKS build.

This is the index and reading guide for a five-part series comparing Elasticsearch, OpenSearch, Grafana Loki, Quickwit, and ClickHouse as backends for long-term log archiving — retention measured in years, not days. It is for anyone choosing or operating a log-storage backend: the comparison is empirically grounded across storage tiering, operations, security, and day-to-day usability, and closes with a full working ClickHouse pipeline running on OVH Managed Kubernetes Service (MKS).

A few things recur across all five parts. The 7-year retention horizon is the constant frame of reference throughout — a very different problem from keeping the last 30 days searchable, because cost per GB and the operational burden of keeping a system alive for years outweigh hot-tier query latency. The concrete ClickHouse-on-OVH-MKS build from Part 5 is used repeatedly as the running reference point: when Part 1 discusses TTL-based tiering, Part 3 discusses encrypted disks, or Part 4 discusses cold-tier query latency, it is the same ClickHouse deployment being described from a different angle.

This series is fairly broad, but it is not, and cannot be, complete. Every use case carries its own constraints — a regulatory obligation, a legacy dependency, a team's operational maturity, a query pattern not covered here — that can turn a general comparison into the wrong call for that specific situation. Treat this guide as a starting point, not a checklist to apply unmodified: it is always worth measuring your own log volume and query patterns, and discussing your use case with people who know it, before committing to any of the systems below.

How to read this guide

If you are choosing a backend from scratch, start with Part 1 — it sets up the axes the rest of the series uses. If you already run one of these five systems and want to harden or scale it, jump to the part that matches your gap.

- Part 1 — Comparison: storage tiering, compression, resource consumption, query languages, SaaS options
- Part 2 — Operations: setup, ingest pipeline, ECS vs. OTel, backup & DR, observability, alerting

- Part 3 — Security & Compliance: encryption, RBAC, WORM / S3 Object Lock
- Part 4 — UX, Dashboards & Alerts: UI layers, log search UX, cold-tier query behaviour, dashboards, alerting
- Part 5 — Use Case: Vector, Istio/Envoy, nginx, Java on OVH MKS — complete ClickHouse ingestion setup and monthly awffull report

Part 1 — Comparison

Audience: platform and architecture decision makers choosing a log-storage backend.

Part 1 lays out the axes that actually matter for a 7-year archive rather than for day-to-day operational logging: automated storage tiering, object storage integration, compression ratio, resource consumption during ingest and compaction, query language, and available managed offerings. It closes with a worked sizing example (100 GB/day raw) carried through compression ratios and an OVH object storage cost projection for all five systems, plus a section on systems evaluated and rejected for this specific use case (VictoriaLogs, Graylog).

Key takeaways:

- Elasticsearch's ILM + Frozen Tier is the only fully automated hot-to-frozen tiering pipeline of the five; OpenSearch's equivalent (ISM + Searchable Snapshots) does the same job for free.
- Loki has the lowest ingest and raw-storage cost of the five, but fulltext search without label filters is effectively a distributed grep — Quickwit needs up to 98% less CPU for the same query.
- ClickHouse's SQL TTL tiering is the most flexible of the five, and its columnar compression (5–10× on structured logs) gives it the second-lowest long-term storage cost after Loki.
- Quickwit has the best ingest-to-search trade-off architecturally, but remains pre-1.0 with no known managed offering after its acquisition by Datadog.
- At 100 GB/day over 7 years, the projected OVH object-storage-only cost ranges from ≈€3,600 (Loki) to ≈€20,400 (OpenSearch) — before compute.

Read Part 1 — Comparison →

Part 2 — Operations

Audience: platform engineers who will run the cluster day to day.

Part 2 moves from the storage model to what it takes to keep each system alive for years: minimum node counts for HA, the ingest pipeline (which shipping agents support which backend, Kafka as a decoupling buffer), the ECS-vs-OpenTelemetry field-naming decision that has to be made before the first log line ships, and the operational split between Kubernetes and bare-metal deployments. It ends with backup and disaster recovery mechanics and a comparison of built-in observability across all five systems.

Key takeaways:

- Elasticsearch and OpenSearch need 3 master-eligible nodes minimum for quorum; ClickHouse needs 3 nodes for its built-in Keeper quorum; Quickwit's stateless architecture is the lightest to scale but adds a second stateful dependency — PostgreSQL — for its metastore.

- Committing to ECS or OpenTelemetry semantic conventions before ingesting the first log line avoids silent query failures years later, when `log.level` and `severity_text` coexist in the same pipeline and nobody remembers why.
- ClickHouse has no built-in snapshot API; `clickhouse-backup` (Altinity, open-source) is the de-facto tool and must be explicitly scheduled and tested — unlike the Snapshot API in ES/OpenSearch, which has been production-hardened for over a decade.
- ClickHouse's `system.*` tables (`query_log`, `part_log`, `metric_log`) turn operational questions into SQL queries against the cluster itself — no external monitoring stack required to answer "which query used the most memory last week."
- OVH Object Storage has no per-request API fees, which removes an entire cost category (GET-request billing on cold-tier scans) that matters on AWS S3 and GCS.

Read Part 2 — Operations →

Part 3 — Security & Compliance

Audience: security and compliance engineers responsible for access control and audit evidence.

Part 3 covers what a compliance-grade archive actually needs: encryption in transit and at rest (and the meaningful difference between OS-level disk encryption and ClickHouse's application-layer encrypted disks), RBAC down to the row and column level, audit logging, and how WORM immutability is implemented in practice via S3 Object Lock — since none of the five systems provide end-to-end WORM guarantees natively.

Key takeaways:

- Elasticsearch and OpenSearch store Lucene segments unencrypted at the application layer — protection depends entirely on the underlying disk being encrypted; ClickHouse's `encrypted` disk type (AES-256-CTR) encrypts data before it reaches the filesystem.
- OpenSearch matches Elasticsearch's RBAC, field-level, and document-level security, and adds free audit logging — Elasticsearch's equivalent requires an Enterprise subscription.
- ClickHouse's SQL `GRANT/REVOKE` and Row Policies give the most granular access control of the five, down to individual columns, and `system.query_log` provides always-on, free, SQL-queryable audit logging.
- S3 Object Lock (fully supported on OVH) is the practical WORM mechanism for all five systems, but it only covers the tier actually stored in object storage — hot-tier data on local SSD in ES/OpenSearch/ClickHouse is not covered until it moves to the cold tier.
- ClickHouse's built-in key rotation is "key rolling," not re-encryption: new parts use the new key, but existing parts keep the old key until they are merged away or expire via TTL — for a 7-year archive that can mean years of dual-key configuration.

Read Part 3 — Security & Compliance →

Part 4 — UX, Dashboards & Alerts

Audience: the engineers and auditors who actually run queries against the archive day to day.

Part 4 is about what happens when someone opens a browser to investigate an incident — or, years later, a compliance auditor queries a 4-year-old event trail. It compares the built-in UI

layers (Kibana, OpenSearch Dashboards, versus Grafana as the shared layer for Loki/Quickwit/ClickHouse), log search ergonomics, dashboard building and sharing, and — the part that matters most for a multi-year archive — how each system actually behaves when a query has to reach into cold object storage.

Key takeaways:

- Kibana's async search API runs slow queries server-side with a progress indicator and survives the browser tab closing; Loki's query frontend times out after 1 minute by default and Quickwit after 10 seconds — both need explicit configuration changes for archive-wide queries.
- ClickHouse has no query timeout by default and is the fastest of the five against cold S3 data for selective queries, because columnar part-skipping reads only the relevant column files regardless of data age.
- Grafana's mixed-datasource dashboards and free Public Dashboards (GA since Grafana 10) are a structural advantage for Loki, Quickwit, and ClickHouse that Kibana cannot match — Kibana requires an Enterprise licence for equivalent unauthenticated public embedding.
- Kibana Discover's field-extraction sidebar and ML-based pattern detection remain the most purpose-built log investigation UX of the five; Grafana Explore has no equivalent auto-populated field sidebar for Loki or ClickHouse.
- Neither ClickHouse nor Quickwit has native alerting — Grafana Alerting (free, Alertmanager-compatible) is the standard and only production-grade path for both.

Read Part 4 — UX, Dashboards & Alerts →

Part 5 — Use Case

Audience: platform engineers implementing a concrete ClickHouse-based log pipeline.

Part 5 shows what the decision looks like once it has actually been made: ClickHouse as the log store, Vector as the ingestion agent, running on OVH MKS with OVH Object Storage as the warm and cold tiers. It covers seven log sources (Envoy/Istio, nginx, Java, Tomcat, Go, Kubernetes system logs, and a catch-all unmatched table), the full ClickHouse schema and storage policy, service account and human-user authentication (LDAP, JWT/OIDC), and a monthly awffull report generated directly from ClickHouse via SQL — no separate log-shipping path needed for the reporting use case.

Key takeaways:

- A three-tier storage policy (local SSD → warm S3 → cold S3 Infrequent Access) with `TTL ... TO VOLUME` clauses moves data automatically; no manual lifecycle scripting is required once the policy is defined.
- No log line is silently dropped: route misses and parser aborts both land in a dedicated `unmatched_logs` table, and Vector's disk-backed buffers (1 GiB per sink) absorb ClickHouse restarts without losing events.
- Structured ClickHouse fields let AWFFull's monthly report be reconstructed on demand via a single SQL query — one system serves both live Grafana dashboards and static, shareable HTML reports.

- Service accounts (Vector, the awffull CronJob) authenticate with `sha256_password` plus `HOST` restrictions; human users and Grafana access go through LDAP or JWT/OIDC — the `IDENTIFIED WITH` clause is the enforcement boundary between the two.
- The Altinity clickhouse-operator, the Altinity Terraform provider, and the `community.clickhouse` Ansible collection cover the full lifecycle — Kubernetes cluster, schema-as-code, and bare-metal — without a paid ClickHouse tier anywhere in the stack.

Read Part 5 — Use Case →

The through-line

Three themes run across all five parts, and noticing them makes the series cohere:

Object storage as the leveller. Every system's long-term economics come down to how much of the data actually lives in object storage and how gracefully it queries from there. Loki and Quickwit put everything in S3 from day one; Elasticsearch, OpenSearch, and ClickHouse only reach that state after a hot/warm phase — and the systems that reach it earliest also have the lowest 7-year storage cost.

Automation versus flexibility. Elasticsearch's ILM is the most automated tiering mechanism of the five and needs the least ongoing tuning; ClickHouse's SQL TTL is more flexible — any column, any condition — but that flexibility means schema and storage-policy design decisions made at deployment time affect query performance for the entire retention period. Neither is strictly better; they trade automation for control.

The theoretical comparison becomes concrete in Part 5. Every abstract trade-off from Parts 1–4 — TTL-based tiering, encrypted disks, RBAC granularity, cold-tier query latency — reappears in Part 5 as an actual storage policy, an actual `CREATE USER` statement, an actual SQL query against a 7-year-old partition. Reading Part 5 after the first four parts turns the comparison tables into a system you could deploy this week.

Related deep-dives

Several angles on the same problem have their own dedicated series on this blog:

- AWS vs. GCP vs. Azure vs. OVHcloud: The Complete Guide to Managed Log Archiving — the managed-service angle on the same problem: what happens when a cloud provider runs the backend for you instead of self-hosting one of the five systems above.
- SigNoz on OVH MKS: The Complete Guide — a full observability stack built on the ClickHouse approach from Part 5, extended to metrics and traces alongside logs.

Where to start

If you have no strong prior, read the parts in order — each one assumes the storage model from Part 1. If you are here to solve a specific problem (encryption, alerting, a slow cold-tier query), use the summaries above to jump directly to the part that names it, then follow the cross-links back into the others as needed.

Start with Part 1 — Comparison →