LOGZILLA DOCUMENTATION

Transforming Event Streams

LogZilla transforms normalize, enrich, and filter incoming events between ingestion and storage so search, dashboards, and triggers see consistent fields

Data Transforms · Generated June 12, 2026 · logzilla.ai/docs/data-transforms/transforming-event-streams

Data Transformation

Data transformation prepares incoming events for analysis by normalizing fields, extracting attributes, and enriching context. Transformation occurs after ingestion and before storage so that search, dashboards, triggers, and forwarding operate on consistent data.

When transformation is needed

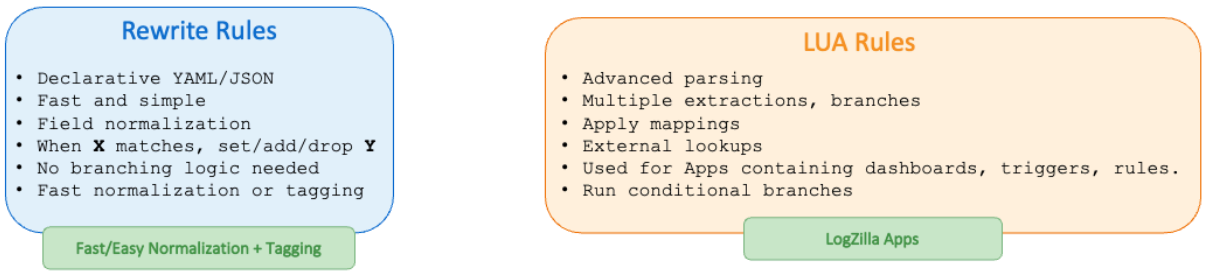
- Ingested messages contain vendor-specific formats that require normalization.
- Events arrive as unstructured JSON where key values must be extracted into searchable fields.
- Noisy events or keepalive messages need to be dropped early.
- Host, program, severity, or message text must be standardized.
- Additional context must be added (for example, device role, site, or business owner) for downstream filtering and reporting.

How transformation works (conceptual)

- Events enter through supported receivers, such as syslog or the HTTP Event Receiver.
- The parsing engine evaluates installed transformation rules in a defined order.
- Declarative rewrite rules apply simple match-and-modify changes.
- Lua rules perform advanced parsing and enrichment and are commonly delivered as part of LogZilla Apps.
- The resulting enriched event is stored and becomes available for search, dashboards, and alerts.

Choosing the right tool

Use the following guide to select an approach based on the problem to solve.



- Simple normalization or tagging
  - Match on basic conditions and update a field, add a tag, or drop the event.
  - Prefer rewrite rules.

- Complex extraction or conditional logic
  - Extract multiple values, transform message text, apply mappings, or run conditional branches.
  - Prefer Lua rules.
- Consistent packaging and reuse
  - Bundle vendor-specific parsing with dashboards and triggers for consistent rollout and updates.
  - Prefer a LogZilla App (internally uses Lua rules).

Decision quick reference

- If the change can be expressed as "when this matches, set/replace/add that" without complex logic, prefer rewrite rules.
- If the change requires branching, multiple extractions, external data lookups, or message reformatting, prefer Lua rules.
- If the organization wants a repeatable, versioned package that also includes dashboards and triggers, create or install a LogZilla App.

Rewrite rules summary

- Declarative, fast, and easy to audit.
- Authored as YAML. JSON is also supported when preferred, but YAML is recommended for simplicity.
- Best for simple field updates, tagging, quieting noise, or lightweight normalization.
- Evaluated in a deterministic order; processing can stop on first match when configured.

Lua rules summary

- Advanced logic for parsing and enrichment.
- Suitable for vendor formats, complex extraction, mapping, or reformatting.
- Often paired with configuration files and shipped within LogZilla Apps.
- Support safe reload workflows and error reporting through standard tooling.

Operational considerations

- Favor clarity and maintainability; keep rewrite rules simple and push complexity into Lua where appropriate.
- Minimize field proliferation; reuse common tags for dashboards and alerts.
- Test on representative samples before rollout; monitor parser performance and rule errors.

Related reading

- LogZilla Apps (https://www.logzilla.ai/docs/administration/logzilla-apps)
- Syslog pipeline customization (https://www.logzilla.ai/docs/administration/syslog-pipeline-customization)
- HTTP Event Receiver (https://www.logzilla.ai/docs/receiving-data/http-event-receiver)

- Command Line Tools —- Data Commands (https://www.logzilla.ai/docs/command-line-tools/data-commands)