LOGZILLA DOCUMENTATION

Rewrite Rules

Declarative LogZilla rewrite rules in YAML or JSON to normalize program, host, and message fields, add user_tags, and drop noise events

Data Transforms · Generated April 29, 2026 · logzilla.ai/docs/data-transforms/rewrite-rules

Rewrite rules

Rewrite rules provide a fast, declarative way to normalize and enrich incoming events. Rules match on event attributes and perform straightforward updates such as setting fields, adding tags, replacing text, or dropping noise before storage.

When to use rewrite rules

- Simple match-and-modify changes that do not require branching logic.
- Standardizing `program`, `host`, or message text.
- Adding `user_tags` for dashboards, filters, or alerts.
- Quieting low-value events (for example, frequent keepalives).
- Parsing simple key-value pairs from messages.

YAML is recommended for authoring rewrite rules due to readability and change review. JSON is also supported if preferred by organizational standards.

How rewrite rules work (conceptual)

- Rules are evaluated in a deterministic order.
- Within a rule group, rules are processed top to bottom.
- Some groups can stop on first match to reduce processing.
- A matching rule can update core fields, add tags, perform controlled replacements in text fields, or drop the event.

Common actions include:

- Set or normalize `program`, `host`, or `message`.
- Add `user_tags` (for example, `site`, `role`, `device_type`).
- Replace text using safe patterns.
- Parse simple key-value pairs embedded in the message.
- Drop events that match well-defined noise patterns.

Authoring guidelines

- Keep rules simple and focused; prefer many small rules over one broad rule.
- Use consistent tag names across teams to support shared dashboards and alerts.
- Avoid high-cardinality tags unless necessary for operations.
- Document intent in the rule comments to aid reviews.
- Prefer adding structured `user_tags` rather than rewriting message text unless normalization is required.

Managing rules via CLI

Refer to the CLI section for full syntax and options: Command Line Tools —- Data Commands (https://www.logzilla.ai/docs/command-line-tools/data-commands).

Typical operations include:

```
# List rules and review status
logzilla rules list

# Enable, disable, or reload rules after changes
logzilla rules enable "My Rewrite"
logzilla rules disable "My Rewrite"
logzilla rules reload

# Check for recent rule errors
logzilla rules errors
```

Verification workflow

- Generate representative events. For custom JSON input, see the HTTP Event Receiver (https://www.logzilla.ai/docs/receiving-data/http-event-receiver).
- Reload rules and confirm the expected changes in search results.
- Inspect tags and fields with targeted queries and widgets.
- Monitor parser health using:

```
# Parser statistics overview
logzilla events parser-stats

# Inspect values observed for key fields and tags
logzilla events values --scope fields --limit 50
logzilla events values --scope tags --limit 50
```

Best practices

- Start with a small set of rewrite rules and expand iteratively.
- Use rewrite rules for simple, deterministic edits; escalate to Lua for multi-step extraction, external lookups, or complex formatting.
    - Keep rule names and comments clear to simplify audits and reviews.

Capability quick reference

- Normalize core fields
  - Standardize `program`, `host`, or `message` text for consistency.
- Add or update tags
  - Enrich events with stable, searchable `user_tags` (for example, `site`, `device_role`).
- Search-and-replace in selected fields
  - Perform controlled text substitutions to clarify messages.
- Parse simple key=value pairs
  - Extract basic attributes embedded in message text.
- Drop matched noise
  - Suppress well-defined low-value events (for example, periodic keepalives).
- Stop further processing
  - End rule evaluation after a successful match when configured.

When more advanced logic is required:

- Use Lua rules for multi-step extraction, conditional branches, message reformatting, or external mappings/lookups.
- Package logic as a LogZilla App when distributing parsing together with dashboards and triggers for consistent rollout.

Match conditions (shared with forwarders)

Many LogZilla components, including rewrite rules and forwarders, use a common `match` structure to select events for processing. A `match` element contains:

- `field`: The event field to inspect (for example, `program`, `host`, `message`, or a user tag).
- `op`: The comparison operator.
- `value`: One or more values to compare against.

Multiple entries in a `match` list act as a logical AND: all conditions must be true for the event to match.

Commonly used fields in match conditions include:

- `program`: Syslog program/tag field.
- `host`: Source hostname.
- `message`: Event message text.
- `facility`: Syslog facility.

- `severity`: Syslog severity.
- `MSWin EventID`: Windows Event ID (requires the `ms_windows` app).
- User tag fields: Any user tag created by parsers or apps.

The following operators are supported in `match` conditions and are shared across rewrite rules and forwarders:

Operator	Description	Example
`eq`	Equals	`program` equals `"sshd"`
`ne`	Not equals	`program` not equals `"sshd"`
`lt`	Less than (numeric)	`severity` less than `4`
`le`	Less than or equal (numeric)	`severity` less than or equal `3`
`gt`	Greater than (numeric)	`severity` greater than `5`
`ge`	Greater than or equal (numeric)	`severity` greater than or equal `6`
`=*`	Wildcard match	`message` contains `"*error*"`
`!*`	Not wildcard match	`message` does not contain `"*debug*"`
`=~`	Regular expression match	`message` matches `"^ERROR:"`
`!~`	Not regular expression match	`message` does not match `"^INFO:"`

For wildcard operators (`=*`, `!*`), use `*` as wildcards in the value (for example, `"*keepalive*"` matches any message containing "keepalive").

Format examples (YAML and JSON)

YAML is recommended for readability. JSON is also supported.

```
rewrite_rules:
  - match:
      - field: message
        op: "=*"
        value:
          - "*keepalive*"
    drop: true
```

```
  - match:
      - field: program
        op: eq
        value:
          - "netd"
    tag:
      site: west-dc
      device_role: edge
```

JSON example (equivalent):

```
{
  "rewrite_rules": [
    {
      "match": [
        { "field": "message", "op": "=*", "value": ["*keepalive*"] }
      ],
      "drop": true
    },
    {
      "match": [
        { "field": "program", "op": "eq", "value": ["netd"] }
      ],
      "tag": { "site": "west-dc", "device_role": "edge" }
    }
  ]
}
```

Related reading

- Overview and decision guide (https://www.logzilla.ai/docs/data-transforms/transforming-event-streams)
- Lua rules (https://www.logzilla.ai/docs/data-transforms/lua-rules)
- LogZilla Apps (https://www.logzilla.ai/docs/data-transforms/creating-custom-apps)
- Command Line Tools —- Data Commands (https://www.logzilla.ai/docs/command-line-tools/data-commands)