LOGZILLA CORPORATION

Administrator's Guide

Technical reference for platform operators and integrators

Generated April 27, 2026

logzilla.ai

© LogZilla Corporation · logzilla.ai

Contents

Getting Started 7

Choose Your Path 9

Self Hosted Install 10

Upgrade 12

Troubleshooting 13

Using The Dashboard 15

Dashboard Overview 17

Widgets Overview 22

Pre-Built Widgets 24

Creating Your Own Widgets 28

Search Syntax 38

Search Types 41

Dashboard Import Export 49

Receiving Data 57

Receiving Syslog Events 60

Cisco Ios Configuration 65

Windows Event Forwarding 67

Receiving Snmp Traps 70

Receiving Java Events 72

Juniper Srx Configuration 74

Nginx 76

Paloalto Pan-Os Configuration 80

Aws Cloudwatch Kinesis Setup 84

Http Event Receiver 87

Avaya Communications Manager 92

Linux Bind 94

Data Transforms 97

Transforming Event Streams 99

Rewrite Rules 101

Lua Rules 106

Creating Custom Apps 111

Testing And Verification 116

Rewrite Rule Walkthrough 119

Lua App Walkthrough 122

Creating Triggers 130

Trigger Page 132

Explanation Of Actions 136

Trigger Scripts 143

Event Correlation 151

Event Correlation Overview 156

Event Correlation Types 165

Bgp Network Monitoring Correlation 172

Brute Force Attack Detection 186

Application Performance Monitoring 198

Network Infrastructure Correlation 211

Soar Security Orchestration 221

Windows Event Correlation 234

Advanced Event Correlation Walkthrough 244

Alerts 254

Alerts Overview 256

Automations 260

Trigger Import Export 265

Outgoing Webhooks 269

Forwarding Module 273

Dedup Forwarder Introduction 275

Downstream Syslog Receivers 278

Downstream Snmp Receivers 286

Forwarding To Splunk 289

Forwarding To Event Correlation 291

Logzilla Api 298

Getting Started 300

Core Concepts 304

Api Endpoint Overview 309

Making Queries 315

Event Query Types 319

System Query Types 322

Query Api Parameters 326

Query Export 332

Sample Api Code 333

Administration 338

Software Lifecycle And Release Notes 340

Server Licensing 347

Sending Email 349

Network Communications 353

Syslog Basics 357

Using Tls Tunnels 370

Using Https 374

Server Settings 379

Search Engine Settings 389

Data Archiving And Retention 395

Ldap Authentication 404

Pci Compliance 414

Role Based Access Control 423

Offline Installs And Upgrades 435
Custom Dns 446
Docker Containers 448
Logzilla Apps 453
Logzilla On Vmware 456
Network Tuning 459
System Tuning 463
Server Migration 468
Syslog Settings 472
Syslog Pipeline Customization 476
Http Receiver Settings 479
Syslog Troubleshooting 481
Syslog Relays 484

Command Line Tools 489
Cli Overview 495
System Commands 502
Data Commands 510
Query Basics 518
Advanced Query Types 529
Query Examples 540

Logzilla Ai 556
Installation 558
Using Copilot 561

Logzilla Cloud 564
Ordering And Checkout 567
Cloud Onboarding 569

Kubernetes Deployment 576
Gke Quickstart 579
Prerequisites 582

Postgres 584
Redis 588
Influxdb 591
Common Config And Secrets 595
Storage Module 598
Query Module 604
Api Module 607
Ingest Module 614
Front End 622
Ingress Gke 626
Scaling And Tuning 631
Troubleshooting 633

CHAPTER 01

Getting Started

Getting started with LogZilla: self-hosted Linux installation, upgrade procedures, install troubleshooting, and LogZilla Cloud onboarding

© LogZilla Corporation · logzilla.ai

LogZilla can be deployed as a self-hosted Linux installation or used as a fully managed Cloud service. Choose the path that fits the environment.

Self-Hosted (Linux)

- Install LogZilla (Self-Hosted): Install LogZilla (Self-Hosted)
- Upgrade LogZilla: Upgrade
- Troubleshooting (Install & Startup): Troubleshooting

Note: LogZilla Server runs on Linux. It is not supported on Windows.

LogZilla Cloud (Hosted)

- Ordering and Checkout: Ordering and Checkout
- Onboarding and Configuration: Cloud Onboarding

Choose Your Path

Pick a LogZilla deployment path: self-hosted install on Linux or fully managed LogZilla Cloud, with links to installer steps and onboarding

Choose The Path

New users can start quickly with either a self-hosted installation on Linux or LogZilla Cloud.

Self-Hosted on Linux

LogZilla Server is supported on Linux only. It is not supported on Windows. For Windows event forwarding, see the Windows Syslog Agent in the Receiving Data section.

- Install on Linux (self-hosted):
  - Start here: Install LogZilla (Self-Hosted)
  - Or deploy on Kubernetes: Kubernetes Deployment
- After installation:
  - Explore the dashboard: Dashboard Overview
  - Begin ingest: Receiving Syslog Events

LogZilla Cloud (Hosted)

Prefer a fully managed service with no servers to maintain?

- Ordering and Checkout: Ordering and Checkout
- Onboarding and Configuration: Cloud Onboarding

Self Hosted Install

Install LogZilla Server on Linux with Docker, including sizing tiers for lab through extra-large deployments and the logzilla install preflight

Platform Support

- LogZilla Server runs on Linux. It is not supported on Windows/MacOS.
- Common choices include Ubuntu and Red Hat Enterprise Linux (RHEL).

Requirements

- Latest stable Docker is required.

Tier	Typical EPD	vCPU	RAM	Notes
Installer preflight minimums	N/A	4	8 GB	Docker installed and usable
Lab	up to 50M	8	12 GB	Evaluation/testing
Small	50M-100M	16	24-48 GB	
Medium	100M-1B	32	48-64 GB	
Large	1B-10B	48	128+ GB	Consider Kubernetes above 10B EPD
Extra large	>10B	Deployment Specific	Deployment Specific	Kubernetes

- For sustained ingest above 10B EPD, Kubernetes is recommended. For >20B EPD, Kubernetes is required.
- Current guidance aligns with modern Docker (27+). Use the latest stable version.

Important:

- Do not use Docker from Ubuntu Snap or RHEL distribution repositories.
- Install Docker from Docker.com:

```
curl -fsSL https://get.docker.com | sudo bash
```

Install Steps

Run the LogZilla installer preflight:

```
curl -fsSL https://logzilla.sh | sudo bash
```

Perform the installation:

```
sudo logzilla install
```

- The preflight checks CPU, memory, Docker presence/version, and proxy settings, then instructs to proceed with `logzilla install`.
- If a proxy is detected on Linux, the preflight can configure Docker's proxy settings.

Verify Installation

Verify LogZilla is ready:

```
curl http://localhost/api/ping
{"detail":"ok"}
```

Next Steps

- Learn the UI: Dashboard Overview
- Start ingest: Receiving Syslog Events

Upgrade

Run sudo logzilla upgrade to pull the latest LogZilla containers and restart services in-place, typically within a few minutes

Upgrading LogZilla pulls the latest containers and restarts the services. Run the following with root privileges:

```
sudo logzilla upgrade
```

The upgrade is in-place and typically completes within a few minutes.

Troubleshooting

Resolve LogZilla install and startup issues: RHEL firewalld ports, Docker health checks, disk and memory usage, and primary log file locations

RHEL firewalld (post-install)

On RHEL, the default firewall can block incoming syslog traffic. After installing LogZilla, allow common ports and make the change permanent:

```
sudo firewall-cmd --list-all
sudo firewall-cmd --add-port=514-516/udp --add-port=514-516/tcp --add-port=601/tcp --add-port=6514/tcp
sudo firewall-cmd --runtime-to-permanent
sudo firewall-cmd --list-all
```

Note: Required ports depend on how the environment is configured. The above ports cover typical syslog (RFC 3164/5424) and TLS syslog.

Check Docker health

```
docker info
docker ps | grep lz_
```

Check system resources

- Disk (Docker data):

```
df -h /var/lib/docker
```

- CPU and memory:

```
top
free -h
```

Log locations

- Primary LogZilla log file:

```
/var/log/logzilla/logzilla.log
```

If issues persist, capture recent errors or warnings from the log and share with support.

CHAPTER 02

Using The Dashboard

LogZilla dashboards deliver real-time event views with widgets, role-based access control, multiple filters, and per-widget time ranges

LogZilla dashboards present data in an interactive, real-time view. Widgets are the building blocks of each dashboard. Every widget supports real-time updates so users see the most current and relevant information.

Widgets and dashboards are adaptable. They can be customized to highlight the metrics and data subsets that matter to different audiences.

Key Features

- **Multiple custom dashboards**
  Create separate dashboards for teams, tasks, departments, or goals.
- **Role-Based Access Control (RBAC)**
  RBAC limits access to both data and the user interface.
  - **Data access**: RBAC governs which data sets a user or group can view. For example, in a widget that lists top hosts, a user will only see hosts the role permits.
  - **User interface access**: RBAC controls who can manage dashboards, view notifications, use "online mode" (GeoIP lookups, ICMP, Telnet, SSH), run searches, create or view triggers, run reports, and manage tasks.
- **Multiple filters**
  Apply filters to focus on specific metrics, identify emerging patterns, or remove noise.
- **Per-widget time range**
  Each widget can use its own time window, such as the last hour, day, month, or a custom duration, within the same dashboard.

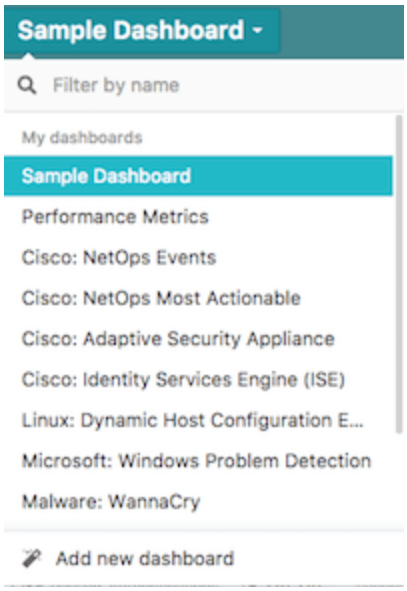
The sections that follow describe each feature and how to configure it effectively.

Dashboard Overview

Navigate LogZilla dashboards: dashboard selector, settings and clone options, import and export, adding widgets, and time range controls

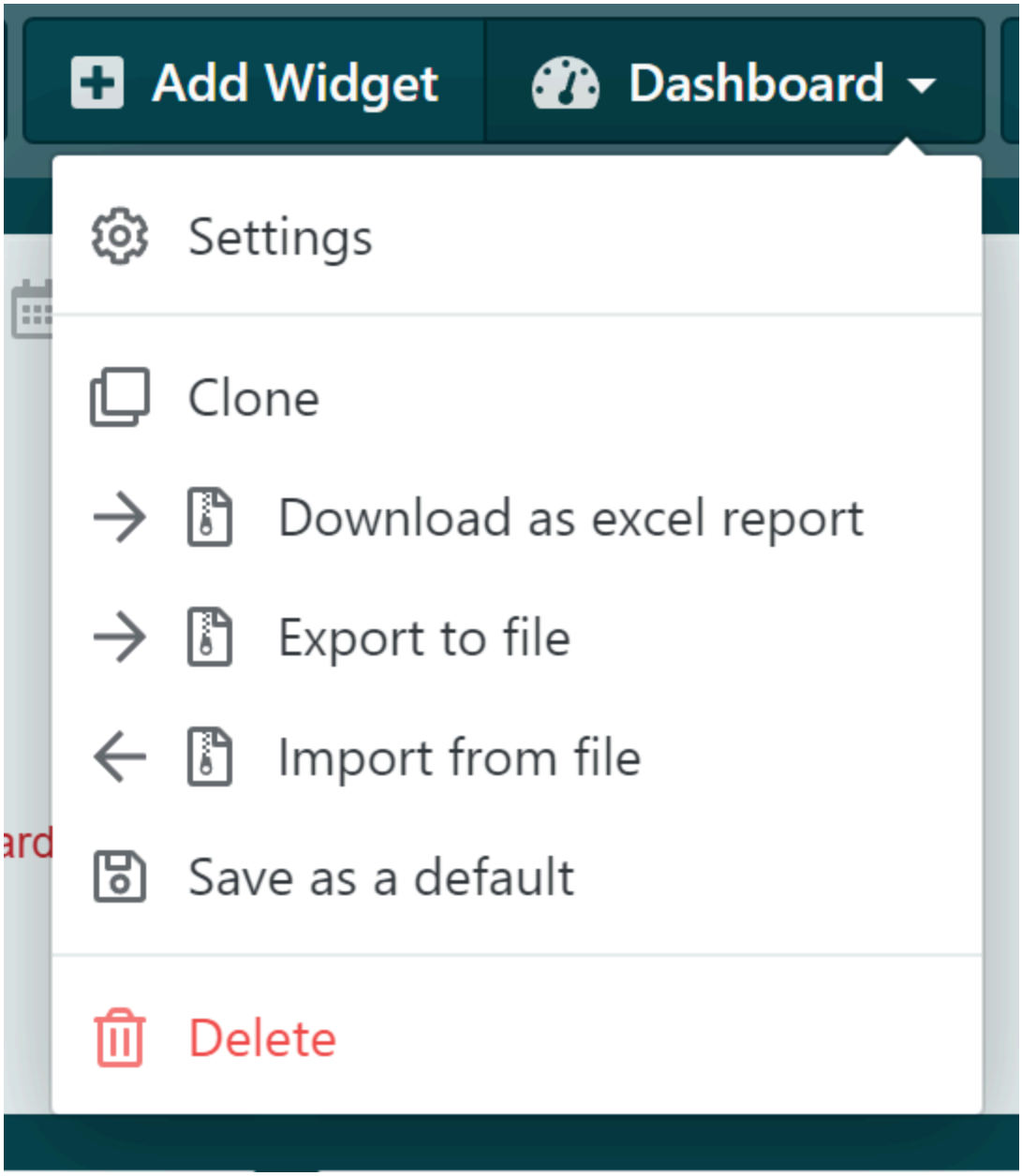
Dashboard Selector

The first drop-down on the left allows selection from existing dashboards or creation of a new one.



Dashboard Options

The dashboard drop-down on the right side of the UI shows available options for `Settings`, `Clone`, `Export to file`, `Import from file`, and `Delete` of dashboards.

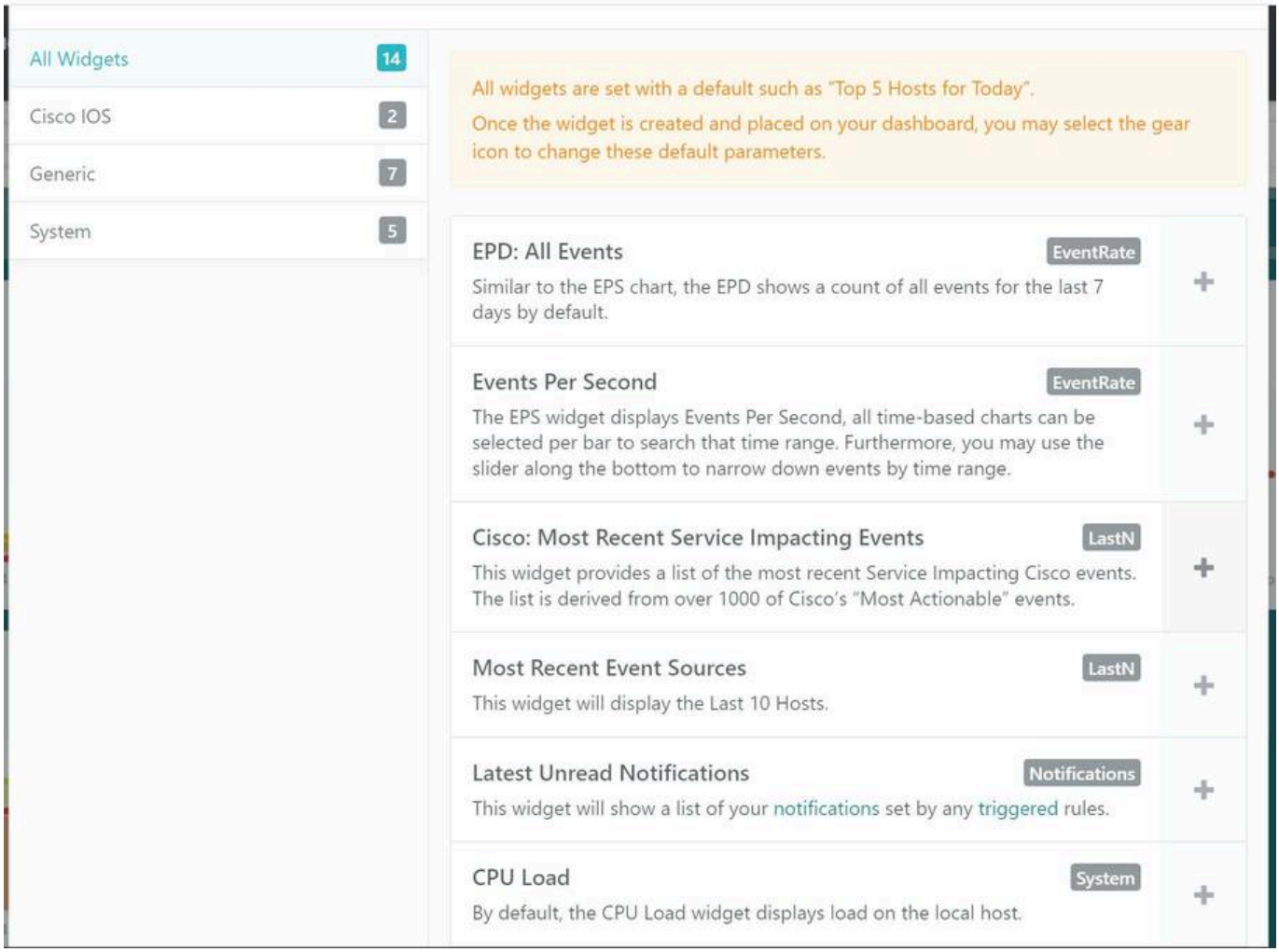


Exporting

Dashboards may be exported to JSON format for modification, sharing, and other purposes.

Adding Widgets

Widget types are listed in the "Add Widget" menu option which allow customization of widget filters once that type is added.



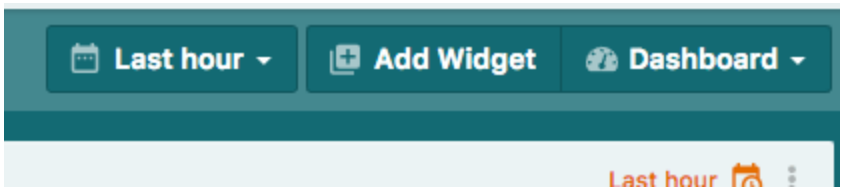
For more information on tailoring widgets to specific needs, see the section "Creating your own widgets".

Time Range Selector

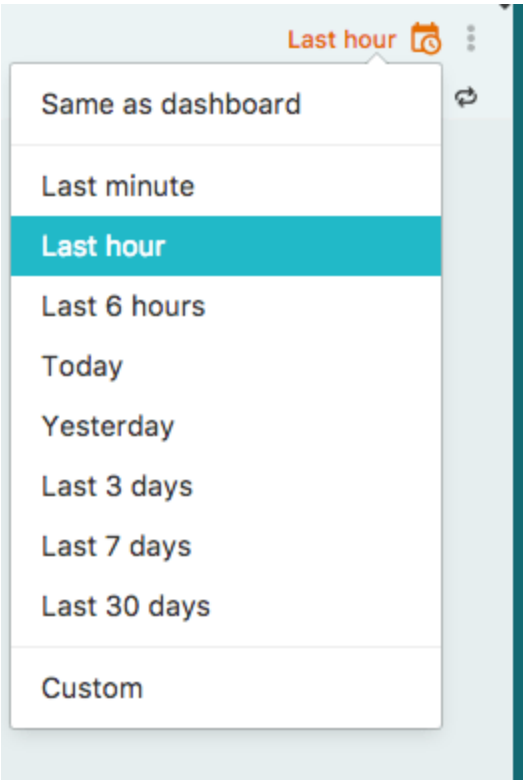
There are two options for specifying the time ranges that will be displayed in widgets. Each widget may be customized on an individual basis for its own time range, or set to use the dashboard's time range selection.

Setting a widget to "Same as dashboard" tells that widget to use the time range set in the dashboard itself.

Dashboard Time Range Selector



Widget Time Range Selector



TV Mode

TV Mode may be used to maximize the LogZilla dashboard as a full-screen view. In TV Mode, all navigation, search, and other interface elements are removed from view so that only the dashboard widgets are displayed. This is particularly useful in large-screen Network Operations Centers.

PaloAlto: Threat
Exit fullscreen mode
Threat Events Per Second
Events Per Second
Average
251
Max
367
Threat Events Per Day
Events Per Day
Average
21.8m
Max
21.8m
Top Blocked Users
SRCUSER
COUNTERS
user_lvpgp 965
user_vkewp 943
user_ttxl1 942
user_vnghj 868
user_umelc 843
user_woldv 570
Traffic vs. Threat Distribution
TRAFFIC
THREAT
Top Destination Locations
DSTLOC
COUNTERS
United States 11,900
Brazil 1,884
Ireland 315
United Kingdom 181
Canada 169
France 50
Poland 50
Germany 23
Japan 23
Singapore 23
Threat Rules
RULE
COUNTERS
Web Default 12,725
Block Outbound URL Categories 1,458
Web Collaborators 144
Web Personnel 74
Web Priviledged Access 72
Threat Actions
block-url
alert
Top 5 Users
SRCUSER
COUNTERS
user_beeez 1,159
user_ttxl1 1,085
user_lvpgp 965
user_vkewp 943
user_vnghj 868
Live Stream: Threat Events
SEVERITY
HOST
FACILITY
PROGRAM
MESSAGE
FIRST SEEN
LAST SEEN
COUNTER

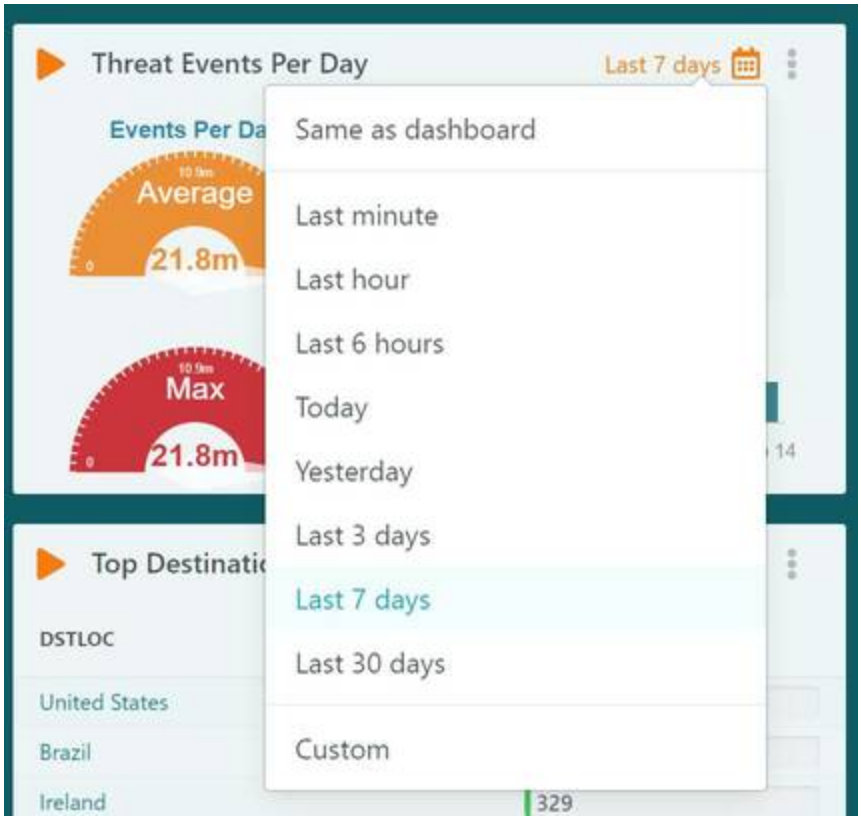
Widgets Overview

Basic LogZilla widget controls: per-widget time range selection, the edit widget modal for customization, and resize and move behavior

Basic Controls

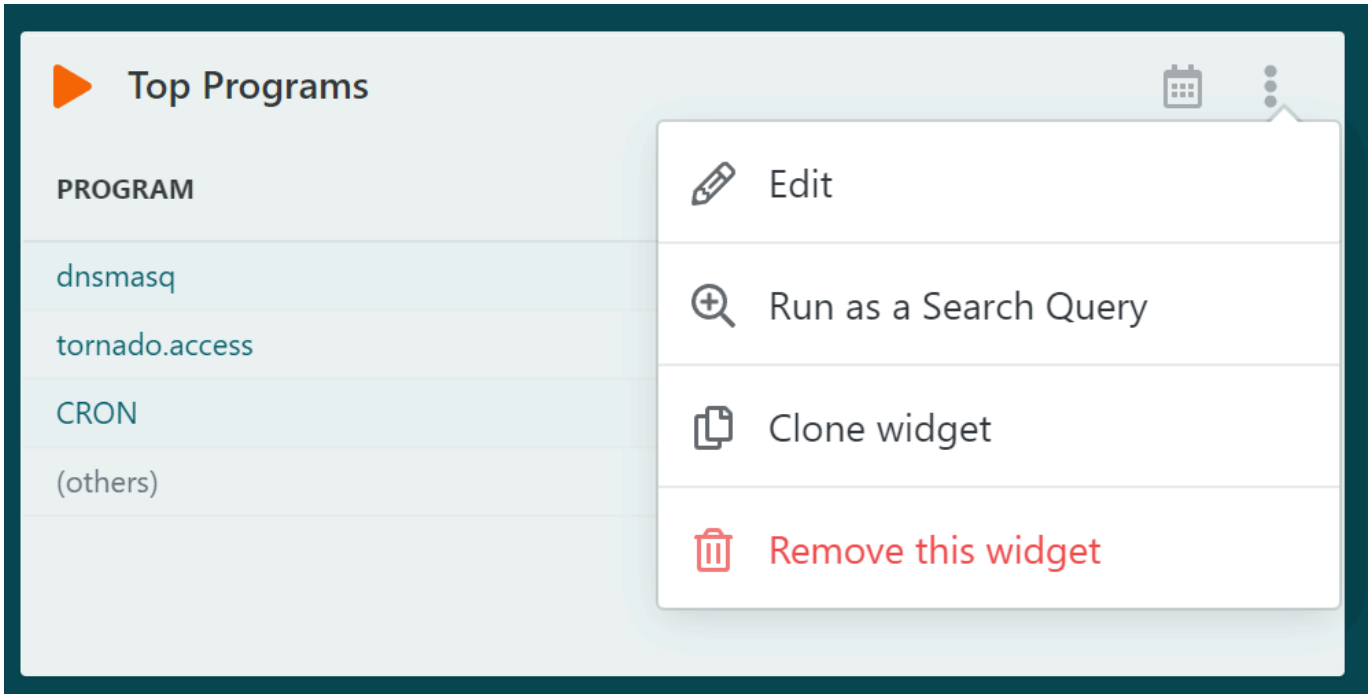
Time Range

This icon allows users to set the time range for each widget. Choices include several pre-set time ranges, custom selections, or matching the dashboard.



Edit Widget

This allows complete customization of widgets. See the section "Creating your own widgets" for more details.



Resizing and Moving

All widgets can be resized and moved so that the layout suits specific needs. Users can move important information to the top of the dashboard, or make it larger for greater visibility.

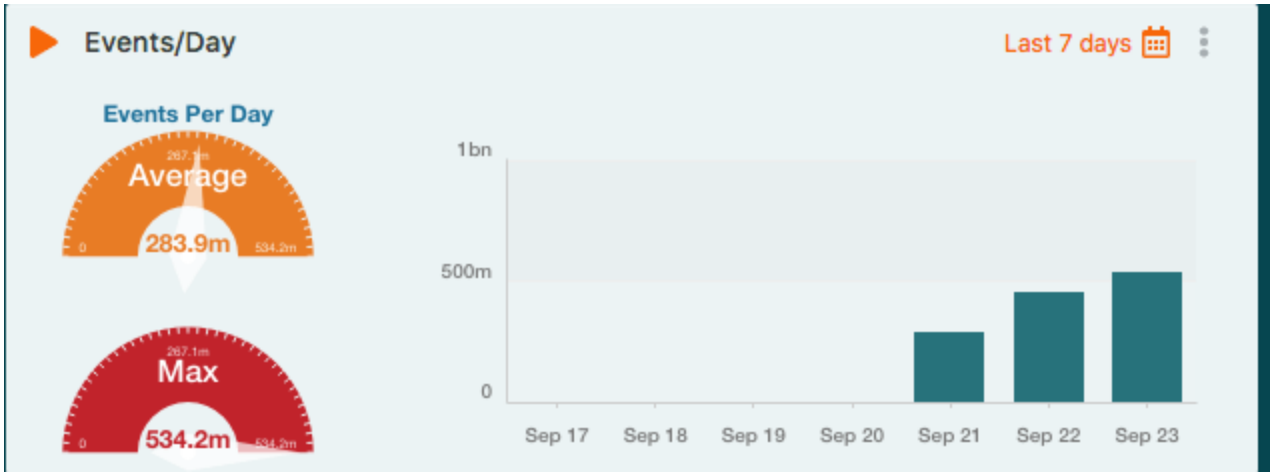
Pre-Built Widgets

Fourteen pre-built LogZilla widgets covering event rate, LastN, notifications, tasks, TopN, system health, and Cisco IOS service-impacting events

LogZilla comes packaged with 14 pre-built widgets organized into categories: Generic widgets (7) including EventRate, LastN, Notifications, Tasks, and TopN widgets; System widgets (5) for monitoring CPU Load, Disk IOPS, Memory Usage, Network Utilization, and Storage; and Cisco IOS widgets (2) for service impacting events. Each widget type can be customized and configured for specific monitoring needs.

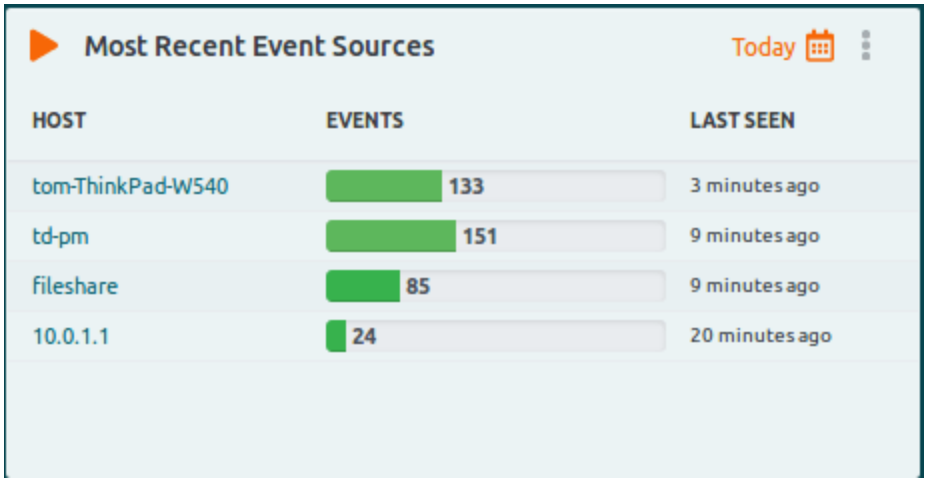
Event Rate Widgets

These widgets give users a quick view of current and long-term event rates. Large changes in these can indicate changes in traffic, configuration problems, or security events, among other things. Spikes in these graphs should be investigated as part of routine maintenance.



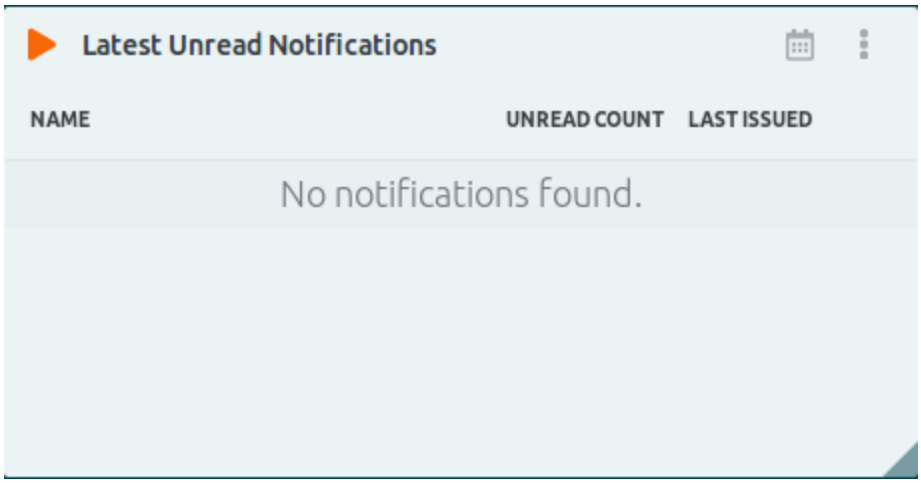
LastN Widgets

The sample widget in this section shows the Most Recent Event Sources. By changing the title and parameters, it can display Mnemonics, Hosts, or other data.



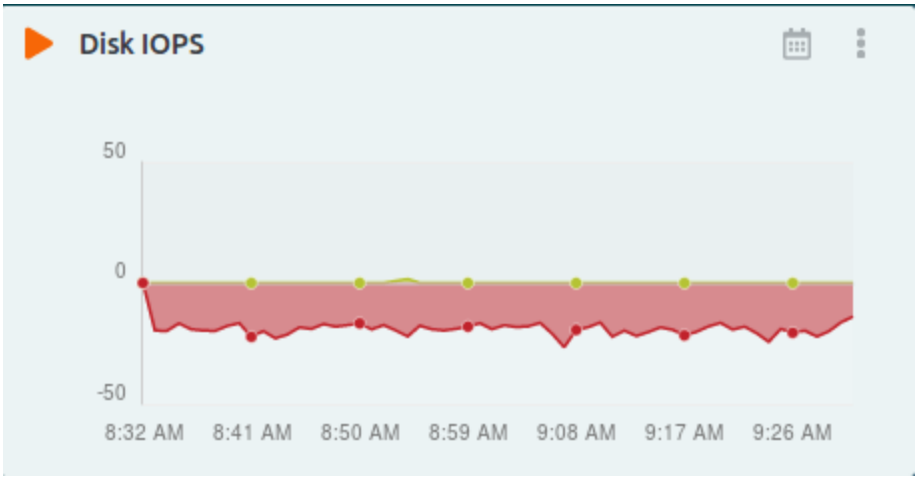
Messaging

This widget displays the Latest Unread Notifications. Notifications are generated by triggers that users create. See the Alerts Overview documentation for more information.



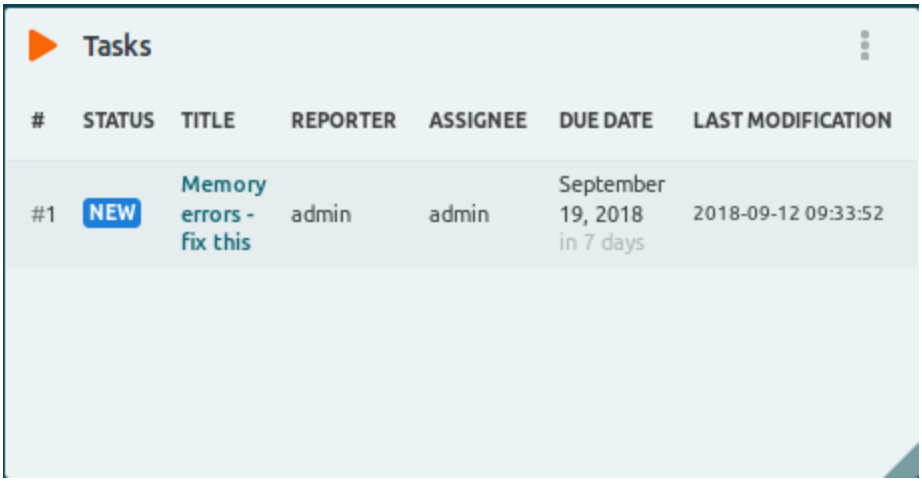
System Widgets

These widgets allow monitoring of the LogZilla server status. Options include CPU Load, Memory Usage, Network Utilization, Storage, and Disk IOPS. For high traffic servers that require 100% uptime, these widgets can help indicate issues before they become downtime.



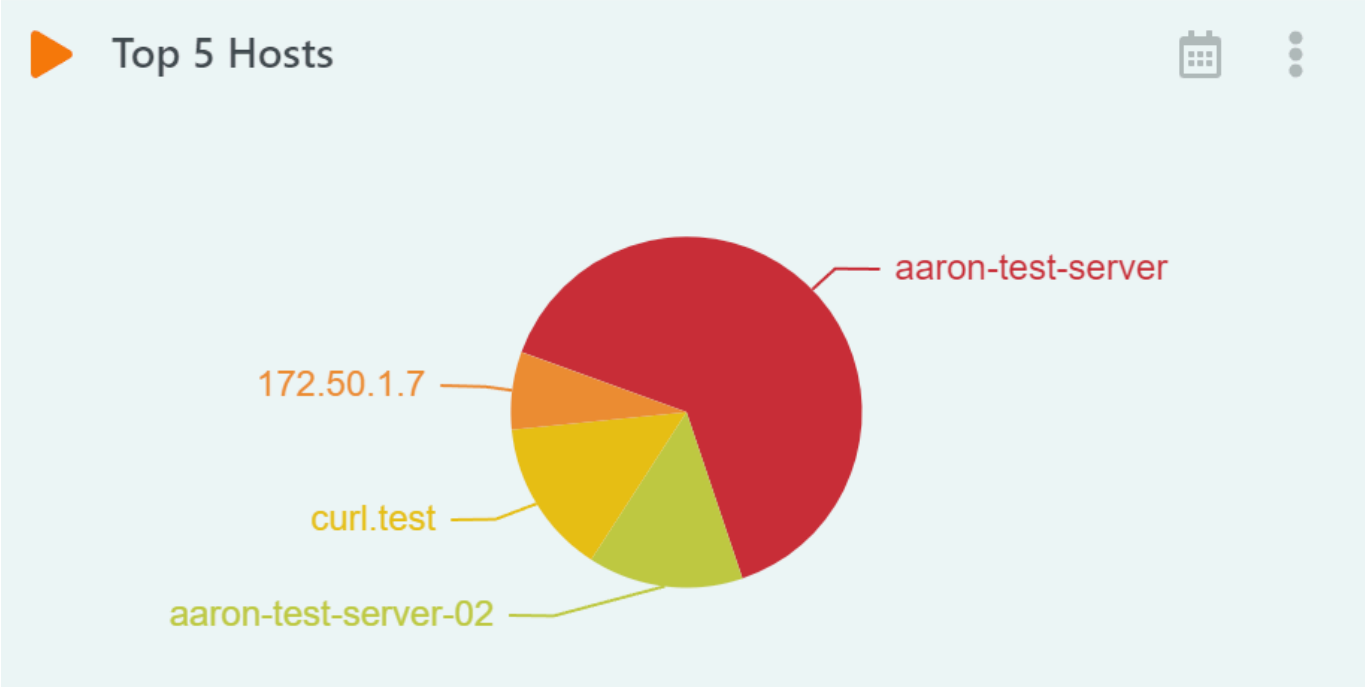
Tasks Widget

This widget displays any tasks created by, or assigned to the user. Tasks can be created on the Tasks page.



TopN Widgets

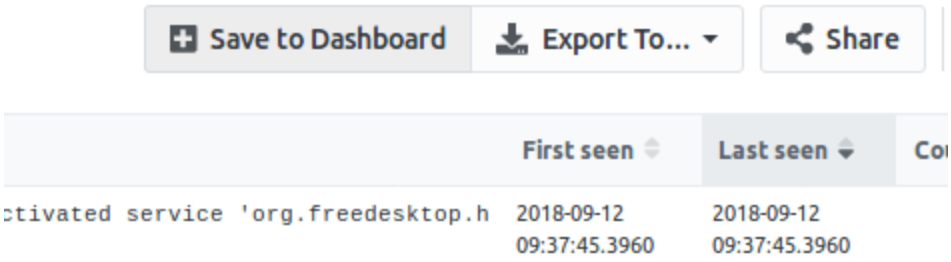
These widgets include Top Hosts and Top Programs. These can also be customized in many ways, and will be covered below in further detail.


Top 5 Hosts
aaron-test-server
172.50.1.7
curl.test
aaron-test-server-02

Creating Your Own Widgets

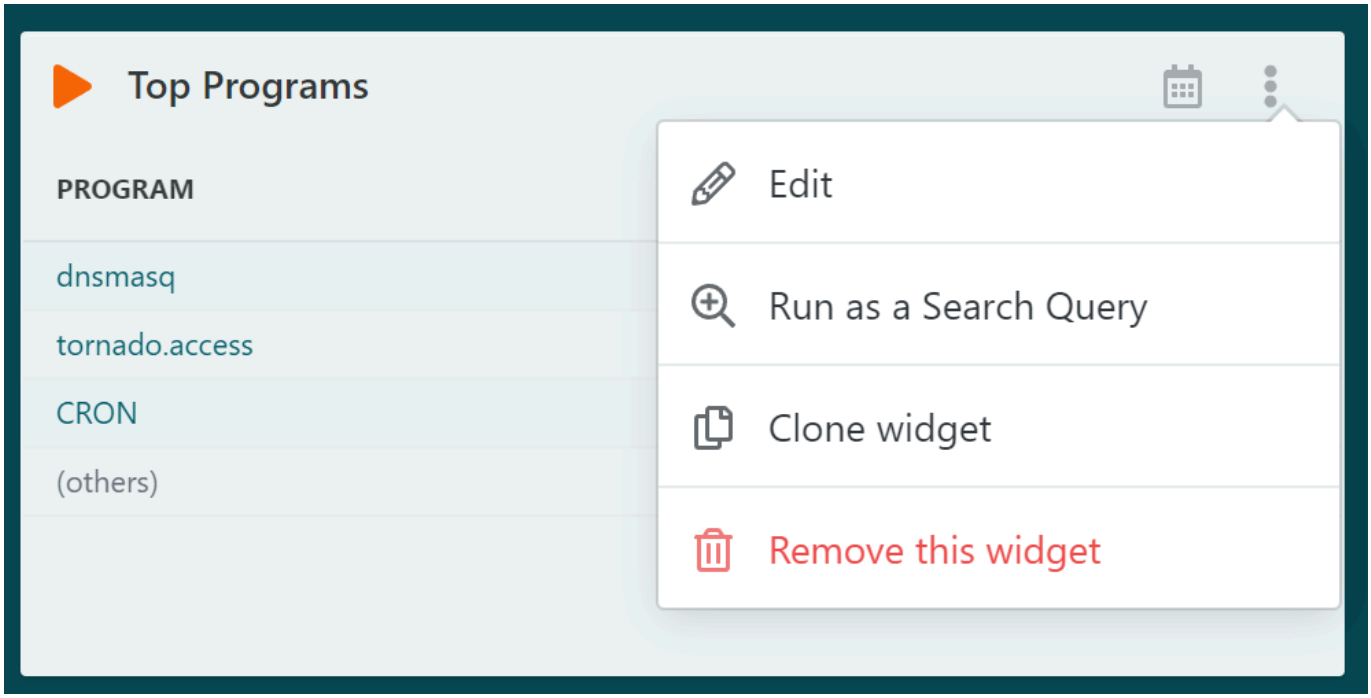
Build custom LogZilla widgets from Save to Dashboard or by editing pre-built widget titles, fields, filters, and user tag criteria

Widgets may be created using either a customizable **pre-built widget** or by performing a search based on the desired filters followed by selecting the **Save to dashboard** button on the search results page.

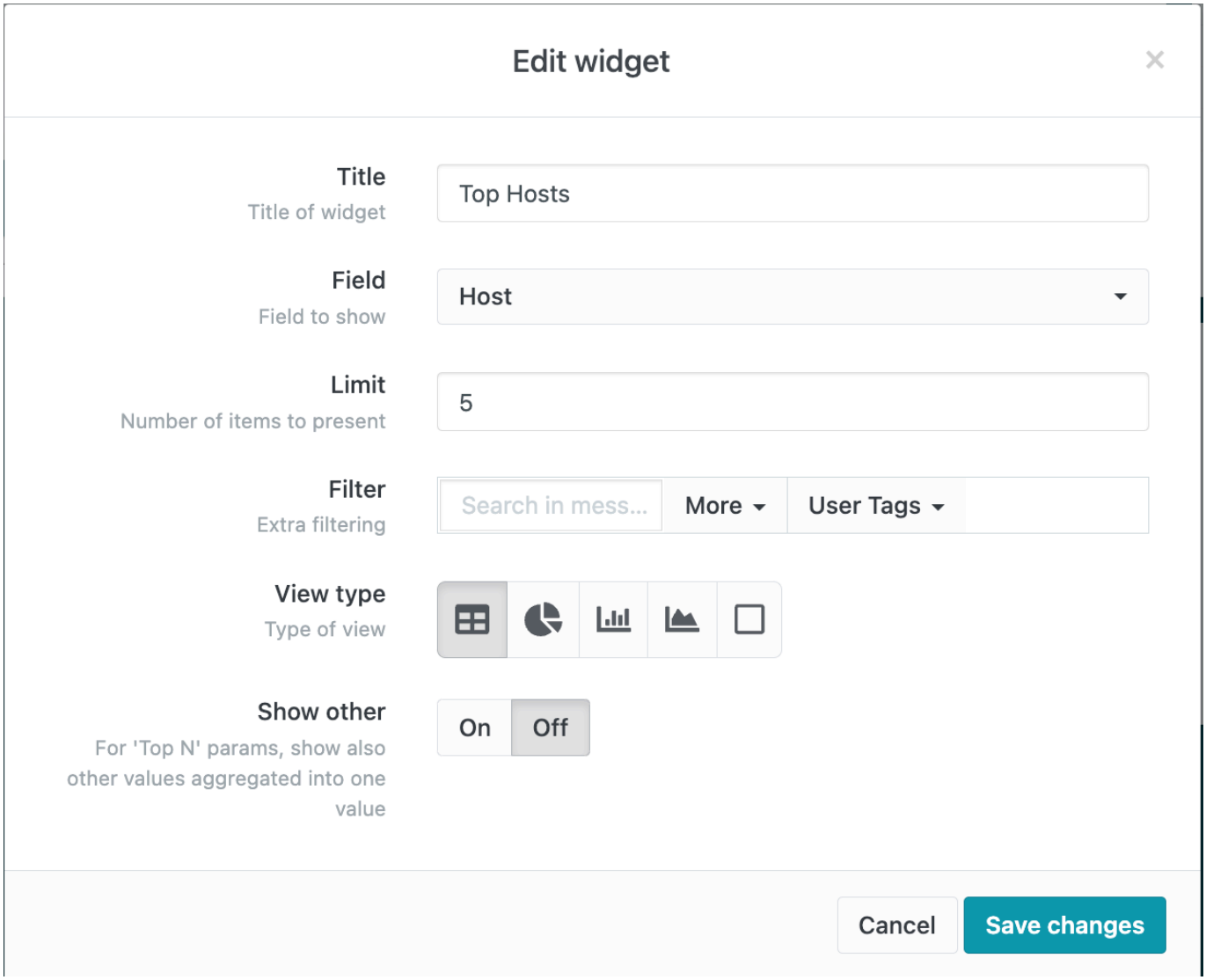


Customizing Existing Widgets

Widgets may be customized to display only the data users wish to see. The following examples show how to set various options for both a `Top Hosts` (which are `TopN`-type charts) and `Top Programs` widgets.

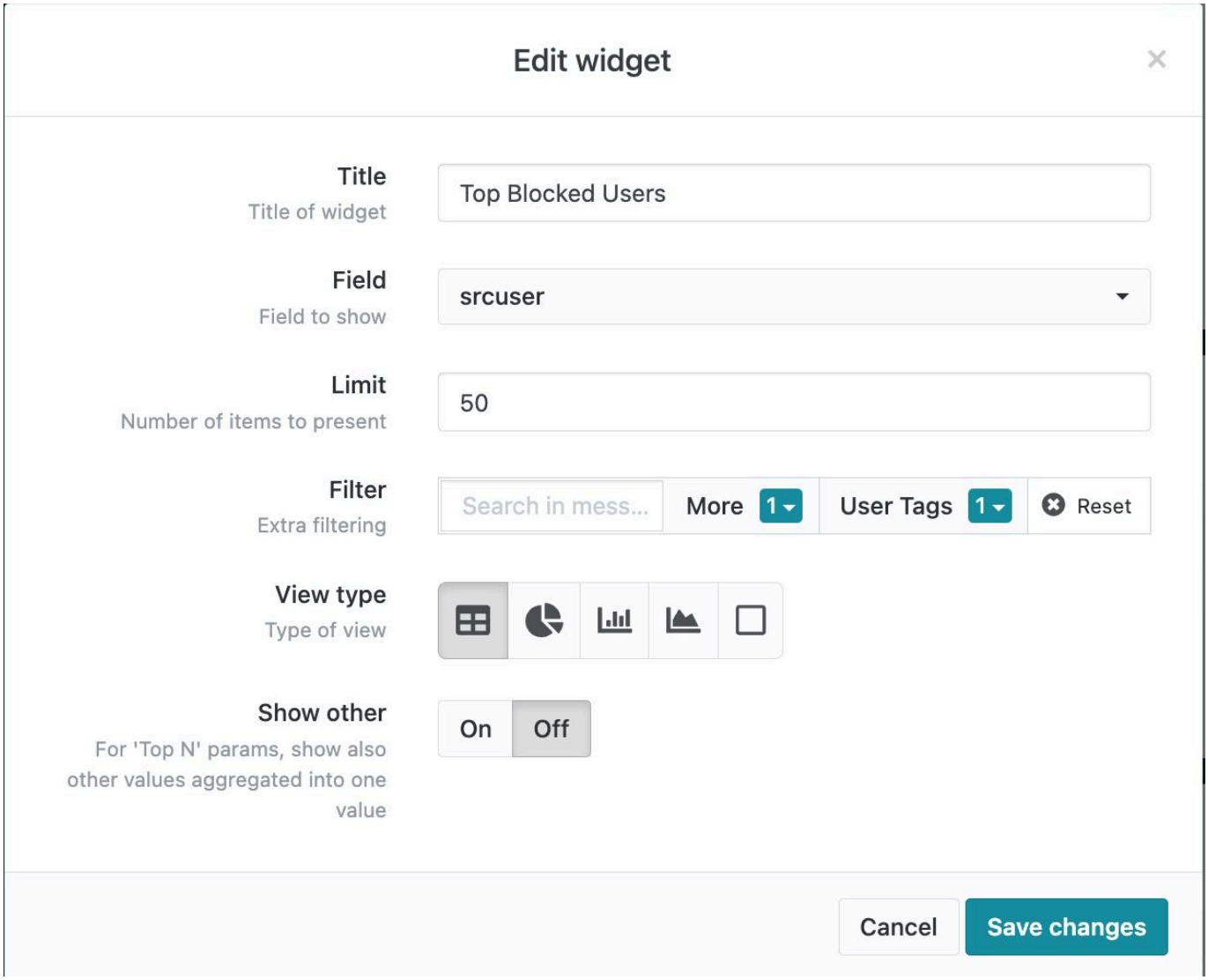


Access the customization menu by clicking the widget's "options" icon (3 dots). Next, select **edit** from the menu to access the widget settings.

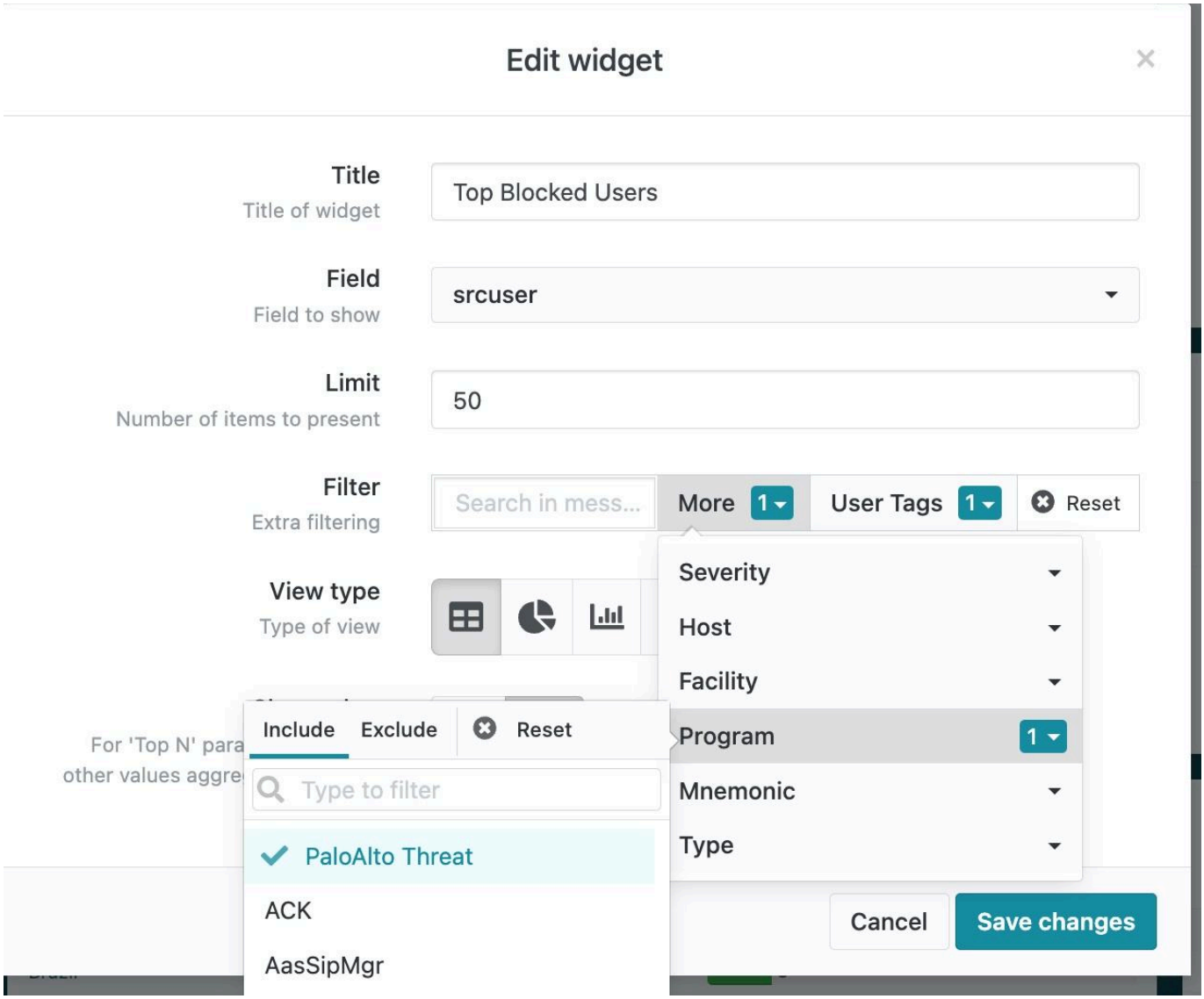


By changing the Title, Description, and Field, users can easily turn this into a widget for showing top severities.

Users can also monitor top Cisco Mnemonics, or add a search term like 'failed login'. The Title and Description need to be modified by the user to be clear about the information shown in the widget. The Field lets users decide what data to display.



The Filter section lets users narrow results similarly to the main search bar in LogZilla. Users add a search term, such as 'failed login', then select all Windows servers from the Host section. This will give a widget that displays only failed login events generated by Windows hosts. Users can similarly filter results by Severity, Facility, Program, Mnemonic, or Type (Unknown, Actionable, or Non-actionable). So, selecting all 'CONFIG' mnemonics would display configuration changes from routers and switches.



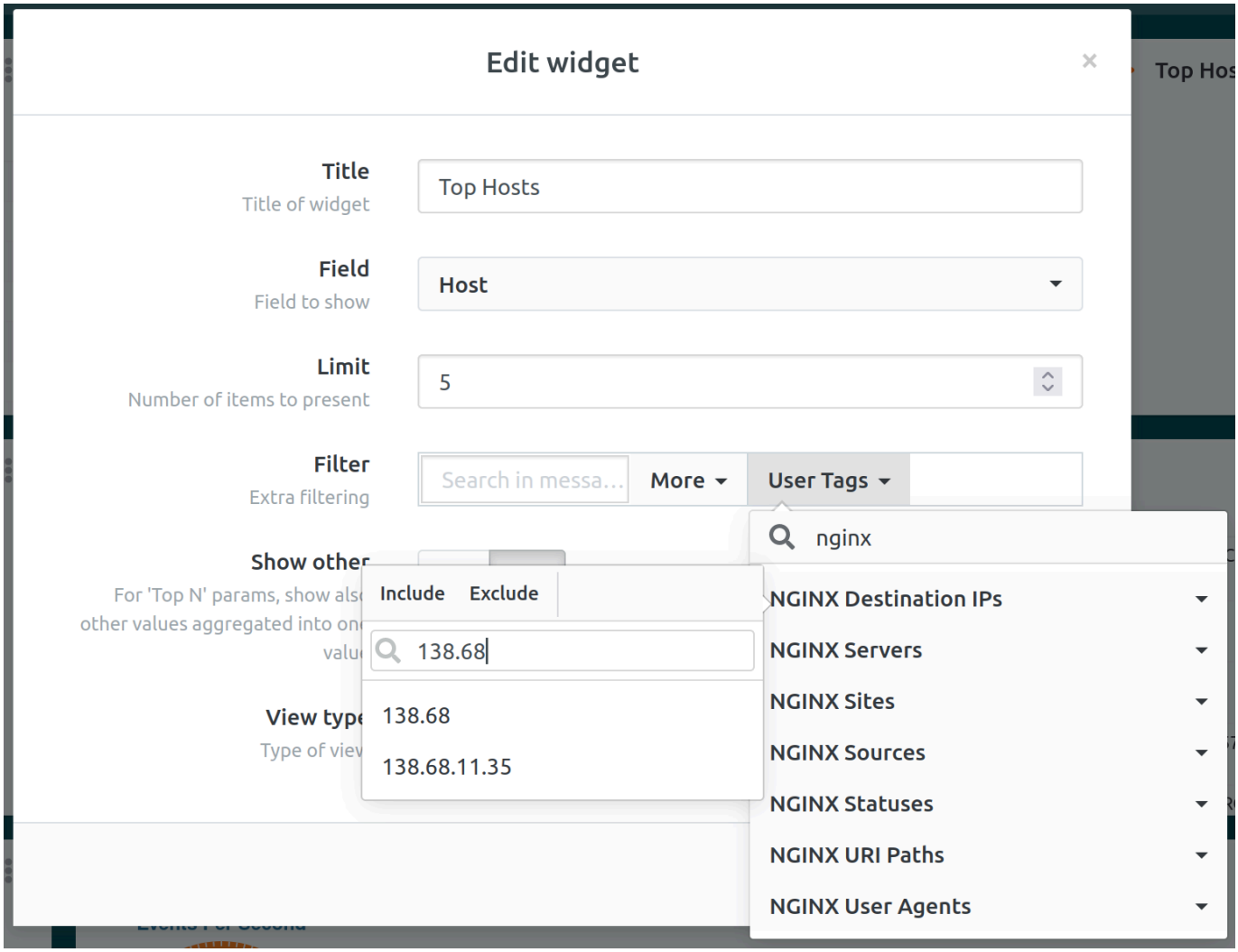
User tags can be used in the filter. User tags are special key/value pairs associated with each individual event. The LogZilla rules can parse the data in each event message and then set specific named (configurable) tags to values from the event data. For example, some common tags are `DstIP` and `DstPort`, respectively representing the destination IP address and the destination IP port for the given event. User tag `DstIP` could for example have value `192.168.0.2`.

The widget can be filtered based on user tags. If the "User Tag" dropdown is selected, optionally at the top of the dropdown a filter for the desired user tag name can be entered (such as if user tag `DstPort` is desired then "Dst" can be entered in the search field at the top of the dropdown, and each user tag with a name containing "Dst", such as `DstPort` will be listed). At maximum ten user tag names will be shown, so to access the user tags that are not shown the search box at the top of the dropdown should be used to narrow down the selection of available user tags.

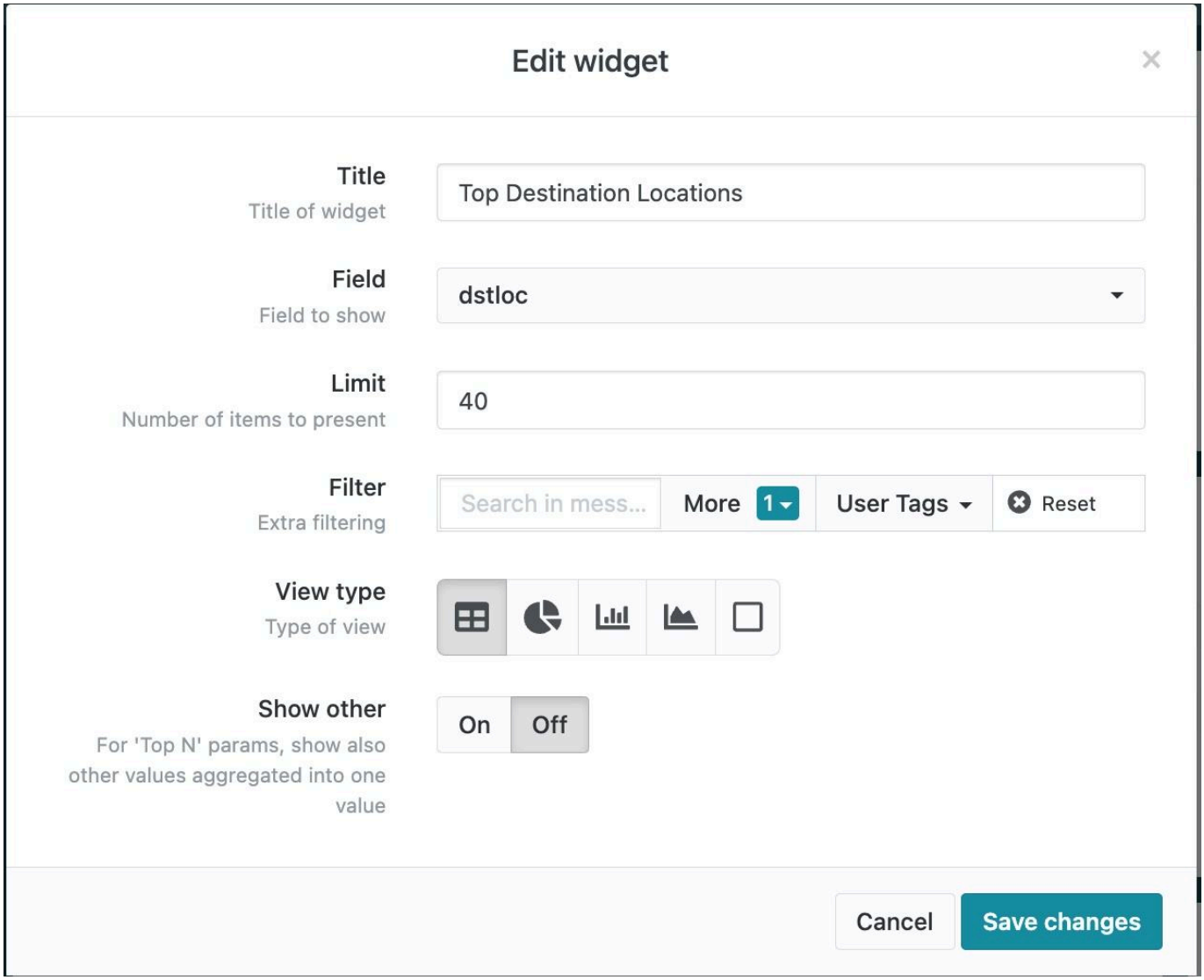
Once the desired user tag is shown it can be clicked to open the values dropdown. The values dropdown allows choosing the particular values for the given user tag either to be included or excluded, such that only those events with the chosen values for the designated user tag will be included in the widget, or those with the chosen values will be specifically excluded from the

widget. The top of this dropdown as well contains the search box to find particular values of interest. Multiple user tag values can be chosen by clicking on each and a checkmark will be shown next to those so designated as an indicator, or the checked ones can be clicked once more to deselect them. User tags that have very many values, such as typically IP addresses, will only show an asterisk (`*`) where the values would be. In this case, the value must be typed directly into the search box.

A special value of `*` can be typed in, then selected. This value has a special meaning: it selects only those events that have *some* value for the designated user tag. This is useful because not every event may contain every user tag. For example, there may be events that have no SrcPort, and those events are not desired to be included. In order to select only those events that have a value for SrcPort, without distinction of what that value is, the `*` filter value should be used.

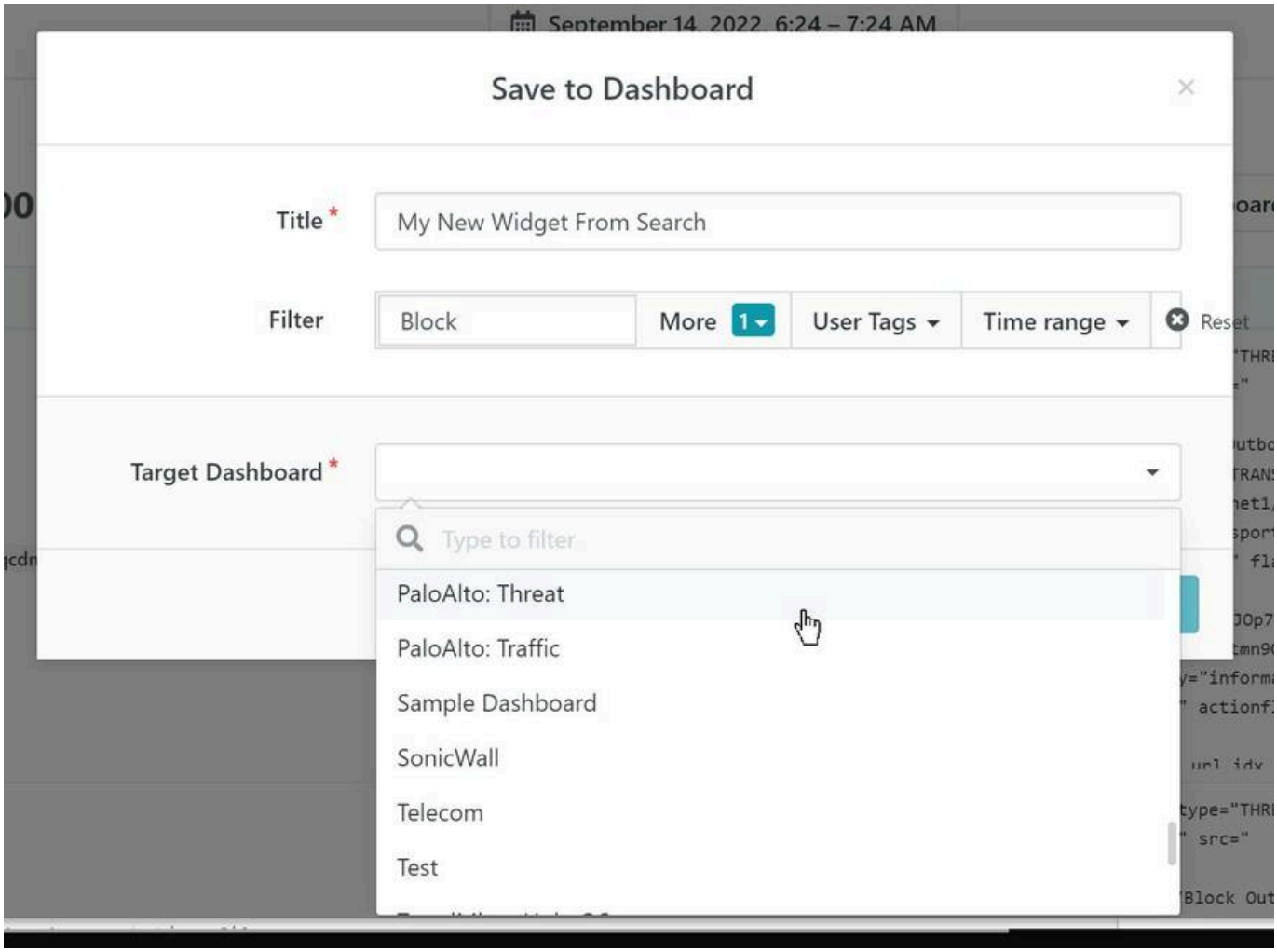


The Limit allows users to control how many results are shown in the widget, while 'Show other' toggles the display of items that don't fit the standard categories of the selected filter. For example, for 'Top N' params, it will also show other values aggregated into one value. The final selection is 'View type', which allows users to select the chart type that best fits the other widget options.

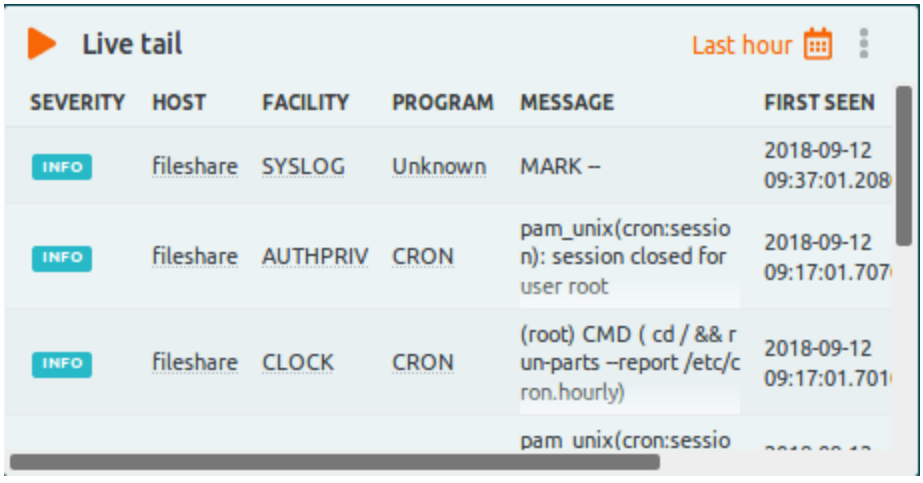


Creating Widgets from Search Results

If users find that they run a particular search on a regular basis, they can click the 'Save to dashboard' button. This will prompt them to name the widget and select the dashboard that it should be displayed on. Users can also modify the search parameters or filters further, if needed.

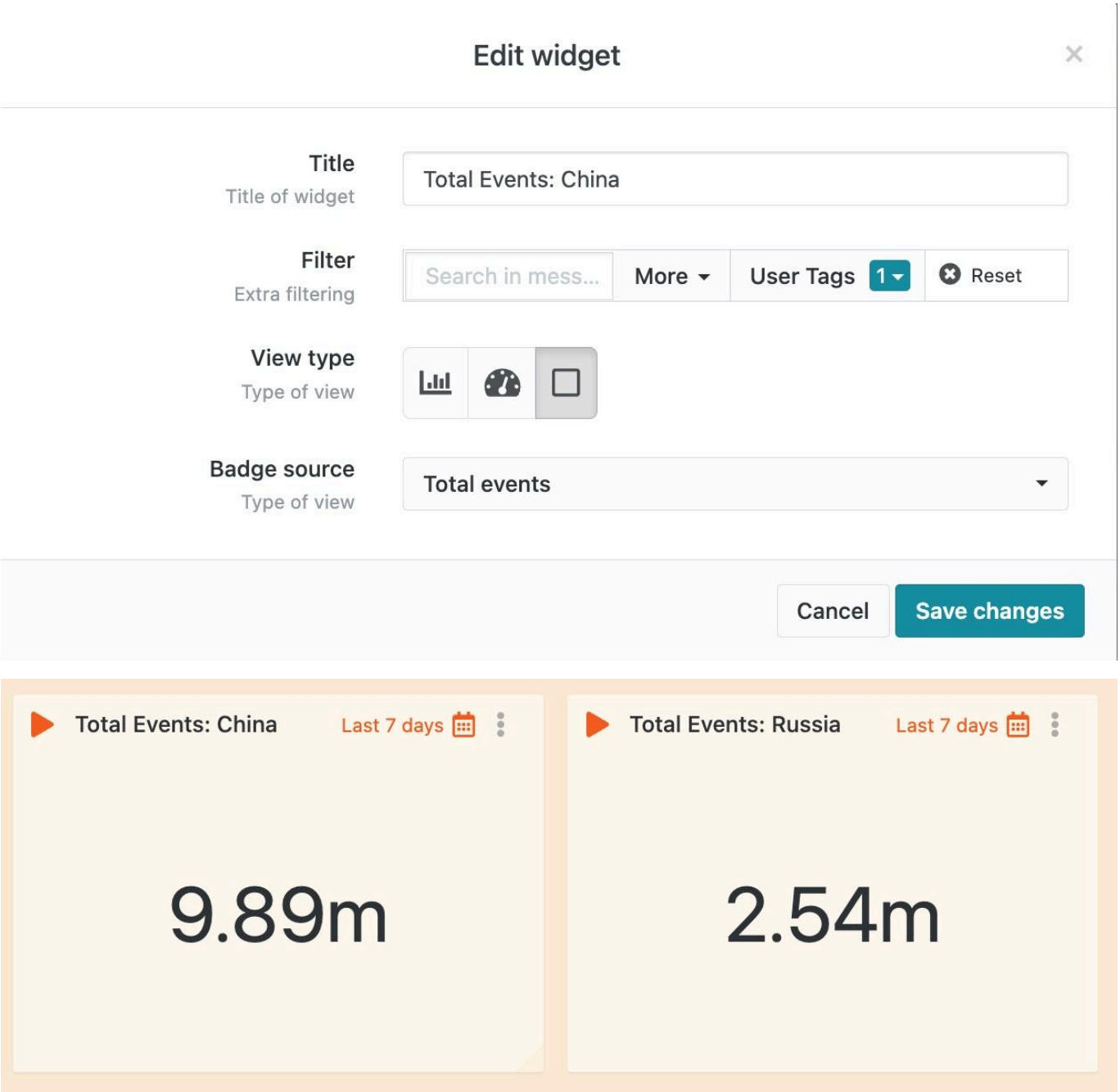


The display will show updated information on a regular basis. This is ideal for keeping up with ongoing network issues, keeping an eye on intrusion attempts, or even knowing when users are locked out after consecutive failed logins.

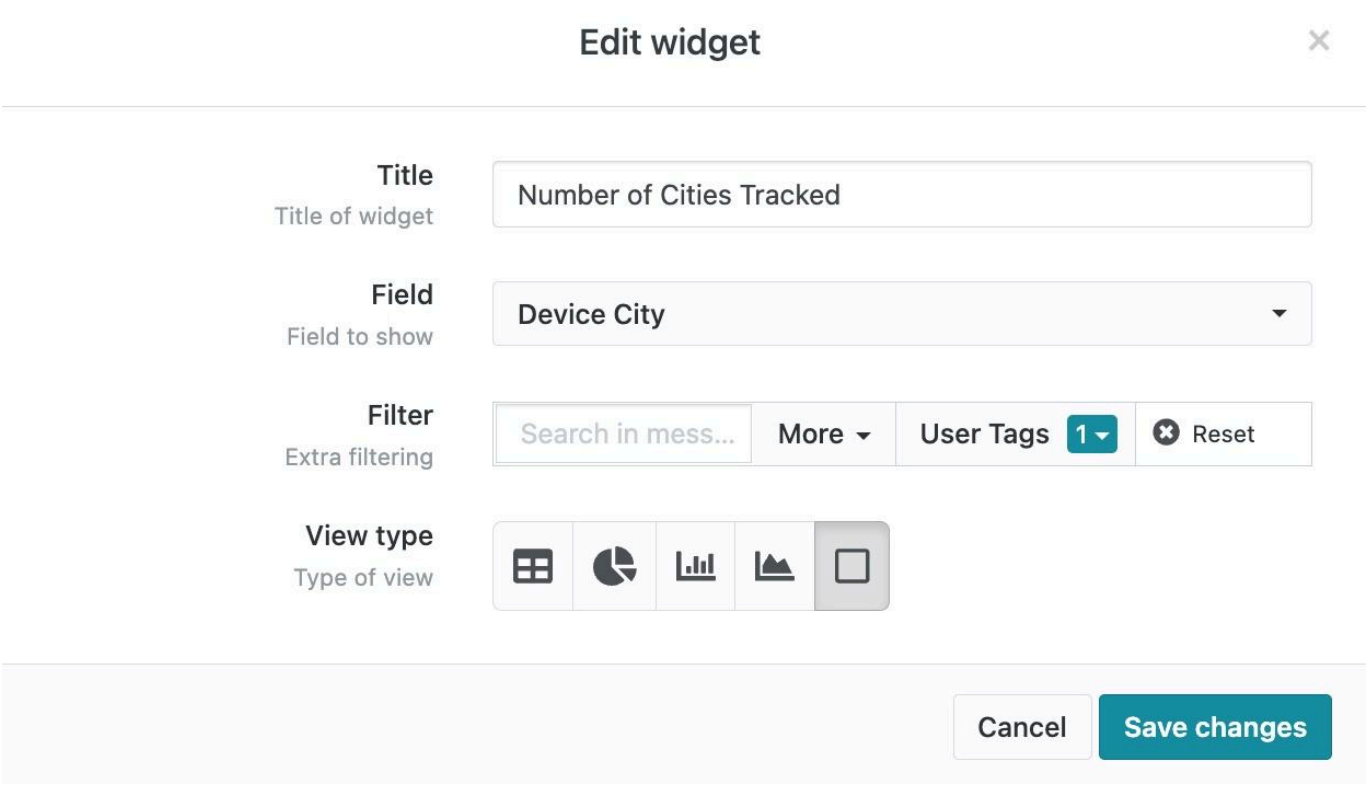


Using Badge (Counter) Type in Rate and TopN Widgets

- **Rate Type Widgets**: For widgets that show event rates, the badge can display the total count or a summary statistic, like the average rate.



- **TopN Widgets**: In TopN widgets, the badge can show the count of unique items in the selected field, providing a quick overview of the diversity in the data.


Edit widget
Title
Title of widget
Number of Cities Tracked
Field
Field to show
Device City
Filter
Extra filtering
Search in mess...
More
User Tags
1
Reset
View type
Type of view
Cancel
Save changes

Number of Cities Tracked
58

To use a badge in these widgets, select 'badge' in the 'View type' option. Customize the title, field, and filters as needed to reflect the data to showcase.

Search Syntax

LogZilla search syntax reference: boolean operators, wildcards, the four-character minimum, and Sphinx prefix and infix indexing rules

LogZilla provides standard boolean-type search syntax much like users would expect when using Google. The only difference is the ability to append a wildcard (`*`).

- All searches are case *insensitive*
- All searches must contain at least 4 characters at a minimum unless otherwise configured by the administrator.
- Wildcard characters (`*`) count toward the minimum character requirement.

Correct Search Syntax

Example 1:

```
hello*
```

Example 2 (prefix/infix wildcard):

```
*hel*
```

Incorrect Search Syntax

Too few characters:

```
hel
```

The 4 character minimum is set in the Sphinx configuration which administrators can adjust. The minimum word length, prefix length, and infix length settings control search indexing behavior. Customers are welcome to contact LogZilla for guidance on modifying these settings.

Boolean Examples

Operator AND

The `AND` is automatically implied when separating search words with a space and **should not** be included in search criteria.

For example, searching on the text `hello world` would return results containing both `hello` and `world`.

Operator NOT

The `!` or `-` operators may be used to find events `NOT` containing the specified text. For example:

```
hello -world
```

Or

```
hello !world
```

Operator OR

A `|` (pipe) operator may be used to find events matching either of the given terms. For example:

```
hello | world
```

Would return all events matching "hello" or "world".

```
hello | other | world
```

Would return all events matching "hello" or "other" or "world".

Boolean Mode Wildcard

Many Network and Systems logs will include names such as `GigabitEthernet1/0/0`, etc. The wildcard feature allows users to specify a search term when they may not know the trailing characters.

For example:

```
gigabitethernet1*
```

Would return results for `GigabitEthernet1/0/0`, `GigabitEthernet1/0/2`, or even `GigabitEthernet100`.

A **prefix/infix wildcard** may also be used:

```
*bitethernet1/*/2
```

Would return results for `GigabitEthernet1/0/0`, `GigabitEthernet1/1/2` but not `GigabitEthernet100`.

Grouping

Note that expression *grouping* can be used. This is surrounding a search expression with parentheses "(" ")" . This must be used in cases in a multi-term search expression is used with an OR operator "|", in order to clarify which terms are handled by the OR. For example, to indicate that you want to find messages that contain the expression "foo bar", OR messages that contain "baz" but *not* "boz", you would do the following:

```
"foo bar" | (baz -boz)
```

Invalid Search Syntax

The following examples show some of the mixed-mode searches which are not supported at this time:

- Searches containing both `OR` and `NOT` operator's combined:

```
hello | -world
```

- Mixed "Phrase" `AND` or `NOT`

```
"hello world" !world2
```

```
"hello world" world
```

- Negative searching without a preceding positive search

```
!hello
```

This would be analogous to searching Google for every word on the internet that does `NOT` contain the word hello. Which, of course, would not be very useful.

Search Types

Three ways to query LogZilla events: the main query bar with severity and host filters, widget data search, and direct URL entry

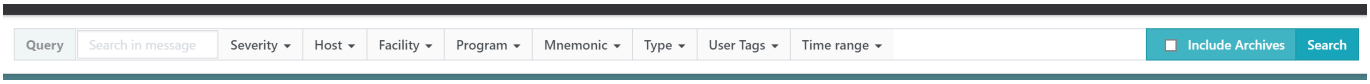
The Search Results page will provide a list of events matching the criteria set by one of:

- The Main Query Bar
- Widget Data Search
- Direct URL Entry

Main Query Bar

The Query Bar provides an easy-to-use interface for setting filters on queries. For syntax on text matching, please refer to the Search Syntax help document.

Main Query Bar



Users may also set more filtering criteria using the query bar such as:

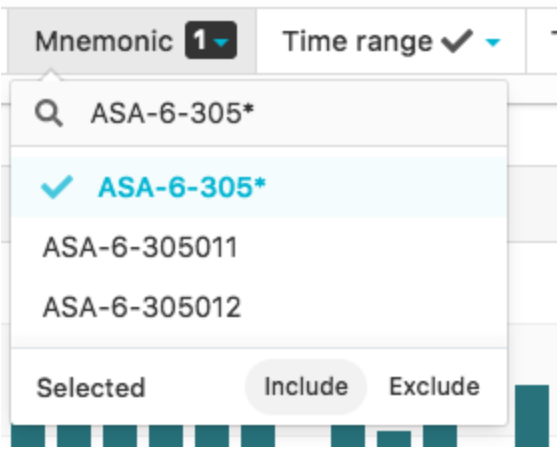
- Severity
- Host
- Facility
- Program
- Cisco Mnemonics
- Time Range
- Status (Actionable, Non-Actionable, Unknown)
- User Tag

Each dropdown provides a list of recently seen entries. Wildcards may be used to search for any unlisted entries in the dropdown.

In the example below, the search results would return all events matching `ASA-6-305*`.

Note that after typing `ASA-6-305*` (case-sensitive) users must **select the wildcard pattern typed in** as seen below in the screenshot (indicated by the blue check mark).

Query Bar Filter Example

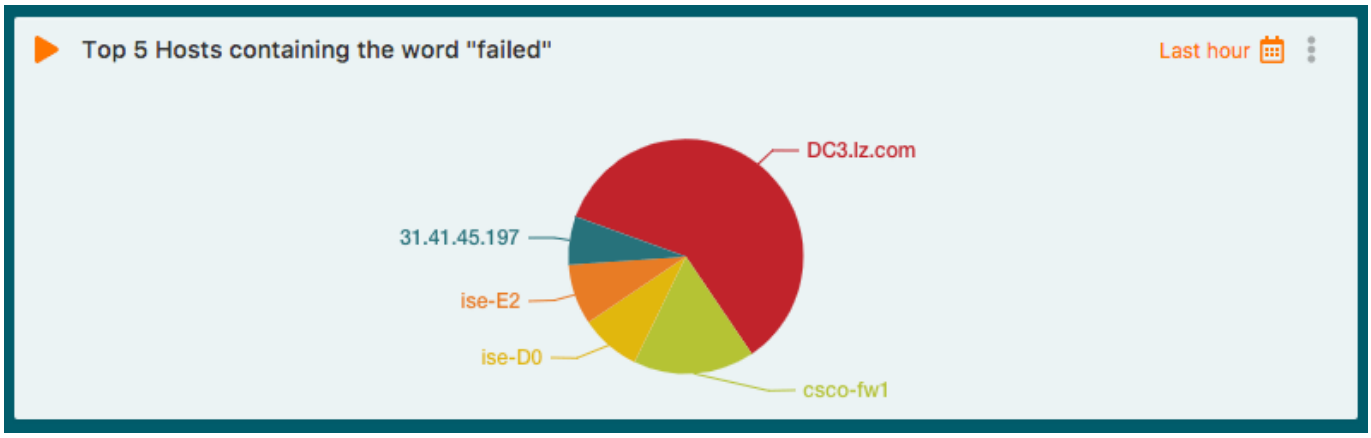


Widget Data Search

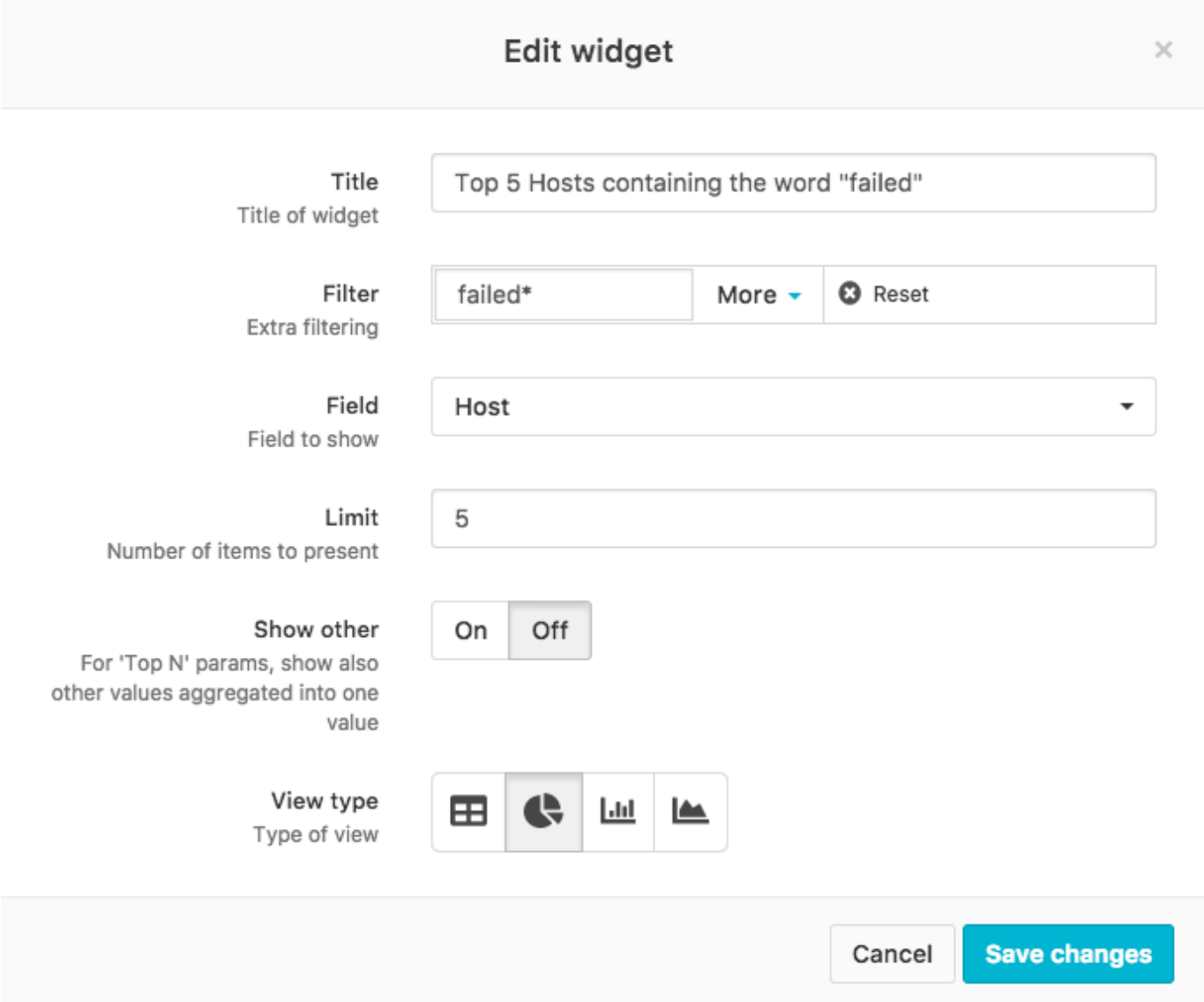
All widgets have an option to perform a search of the data contained in the widget itself. This allows users to perform searches without having to manually enter all of the filter criteria set in that widget.

For example, the widget below has a filter set for showing only the Top 5 hosts which contain the word `failed` in the message.

Top 5 Widget With Filters

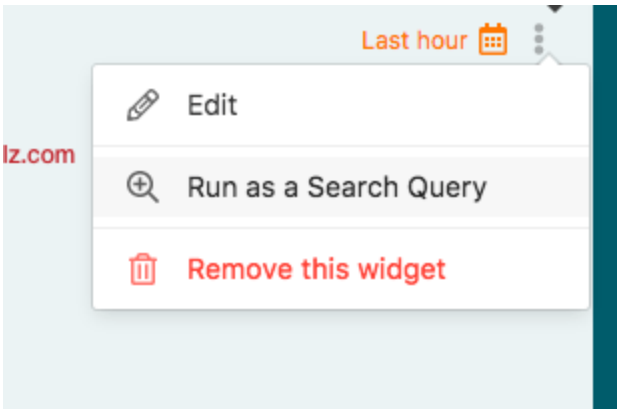


Settings For The Widget Above



To search for all events contained in that widget, simply select the widget handle, then click **Run as Search Query**.

Query From Widget



Direct URL Entry

LogZilla also allows direct searching via the browser's URL by typing the query string along with any desired filter criteria, for example: `http://logzilla.company.com/search?{querystring}`

Usage

- The `search` call must start with a question mark, i.e.: `/search?msg=foo`
- It may contain keys with or without values separated by an `=` (equal) sign or pairs separated by ampersand.
- If multiple values for a single parameter are present in the URL (e.g.: `/search?facility=USER&facility=KERN`), the requested search for these two items will return results for `either` of the two filters (boolean `OR`).

Example

```
http://logzilla.company.com/search?
msg=successful%20auth&facility=USER&severity=Info&time_range=2017-12-13T00:00~14T00:00
```

URL Query String Parameters

What "array" means here

- **array**: Multiple values (strings) can be provided for that parameter. The search will match events where the field equals any of those values (boolean OR).
- **How to pass it in the URL**: Repeat the same key multiple times.
  - Example: multiple facilities
    - `/search?facility=USER&facility=KERN`
  - Example: multiple severities
    - `/search?severity=Error&severity=Warning`
- **Effect**: Matches entries with any of the listed values.
  - E.g., `/search?facility=USER&facility=KERN` returns events with facility USER OR KERN.
- **Applies to**: `facility`, `host`, `mnemonic`, `program`, `severity` (each can be a single `string` or an `array<string>` via repeated params).
- **Full example**:

```
/search?msg=successful%20auth&facility=USER&facility=KERN&severity=Info&severity=Error
```

`msg`

Type: `string`

Search terms are encoded as a Uniform Resource Identifier (URI) component (`encodeURIComponent()` function or equivalent) supporting mixed-mode search syntax searches.

`facility`

Type: `string` **or** `array<string>`

Facility keywords (case-insensitive) are defined in RFC 3164.

Supported Facility Values

Keyword	Description
KERN	Kernel messages
USER	User-level messages
MAIL	Mail system
DAEMON	System daemons
AUTH	Security/authorization messages (note 1)
SYSLOG	Messages generated internally by syslogd
LPR	Line printer subsystem
NEWS	Network news subsystem
UUCP	UUCP subsystem
CLOCK	Clock daemon (note 2)
AUTHPRIV	Security/authorization messages (note 1)
FTP	FTP daemon
NTP	NTP subsystem
AUDIT	Log audit (note 1)
ALERT	Log alert (note 1)
CRON	Clock daemon (note 2)
LOCAL0	Local use 0 (local0)
LOCAL1	Local use 1 (local1)

Keyword	Description
LOCAL2	Local use 2 (local2)
LOCAL3	Local use 3 (local3)
LOCAL4	Local use 4 (local4)
LOCAL5	Local use 5 (local5)
LOCAL6	Local use 6 (local6)
LOCAL7	Local use 7 (local7)

These values may also be found in the LogZilla API on the server at `/api/dictionaries/facility`

```
GET /api/dictionaries/facility
```

host

Type: `string` **or** `array<string>`

Hostname or IP address of the device.

mnemonic

Type: `string` **or** `array<string>`

Cisco mnemonic.

NOTE: Mnemonics should be passed without the `%` prefix as the `%` is a reserved character for URI encoding. e.g.: `SYS-5-CONFIG_I` instead of `%SYS-5-CONFIG_I`

program

Type: `string` **or** `array<string>`

Name of the source program/process.

severity

Type: `string` **or** `array<string>`

Severity name (case-insensitive) as defined in RFC 5424.

Supported Severity Values

Name	Description
`Emergency`	System is unusable
`Alert`	Action must be taken immediately
`Critical`	Critical conditions
`Error`	Error conditions
`Warning`	Warning conditions
`Notice`	Normal but significant condition
`Info`	Informational messages
`Debug`	Debug-level messages

These values may also be found in the LogZilla API on the server at `/api/dictionaries/severity`

```
GET /api/dictionaries/severity
```

`time_range`

Type: `string` **or** `start:iso8601~end:iso8601`

Default: `last_1_hours`

Option 1: Time range preset

Use relative time range preset as defined in the API on the server at `/api/dictionaries/time_range`.

Preset	Description
`last_1_minutes`	Last minute
`last_1_hours`	Last hour
`last_6_hours`	Last 6 hours
`today`	Today
`yesterday`	Yesterday
`last_3_days`	Last 3 days
`last_7_days`	Last week

Preset	Description
`last_30_days`	Last 30 days

Fetch list from API

```
GET /api/dictionaries/time_range
```

Option 2: Date time range

Searches within a specific time range using combined ISO 8601 date/time representation of start and end times, should contain a tilde character (`~`) as the separator (*basic format* is `YYYY-MM-DDTHH:mm:ss.sss~YYYY-MM-DDTHH:mm:ss.sssZ`). If any elements are missing from the end value, they are assumed to be the same as the starting value.

Examples

Cross-day range:

```
2017-12-01T18:00~2018-01-03T01:00
```

Searches from December 1, 2017 at 6:00 PM through January 3, 2018 at 1:00 AM.

Date range (same time each day):

```
2017-11-04~06
```

Searches from November 4, 2017 at 12:00 AM through November 6, 2017 at 12:00 AM.

Same day, different times:

```
2017-08-04T08:00:00~11:00
```

Searches on August 4, 2017 from 8:00 AM through 11:00 AM.

`sort`

Type: `string`

Default: `-last_occurrence`

Name of the field to sort by (`first_occurrence` , `last_occurrence` or `counter`). Prefixing with a negative sign reverses the order.

Dashboard Import Export

Export, edit, and re-import LogZilla dashboards in YAML or JSON format using the UI or the logzilla dashboards CLI for sharing between instances

Dashboard Import and Export

LogZilla supports importing and exporting dashboards so that configurations can be shared across teams or moved between LogZilla instances. The same capability is available through the user interface and the `logzilla` command line.

Dashboard Format in LogZilla

A LogZilla dashboard is stored in either YAML or JSON format. Both formats support the same schema. The following YAML snippet shows the basic structure of a dashboard:

```
- config:
    style_class: infographic
    time_range:
      preset: last_1_hours
    title: Meraki DHCP
  is_public: true
  widgets:
    - config:
        col: 0
        filter:
          - field: program
            value: Meraki DHCP
        row: 0
        show_avg: true
        show_last: false
        show_max: true
        show_min: false
        sizeX: 6
        time_range:
          preset: last_1_minutes
        title: Meraki DHCP Events Per Second
      type: EventRate
```

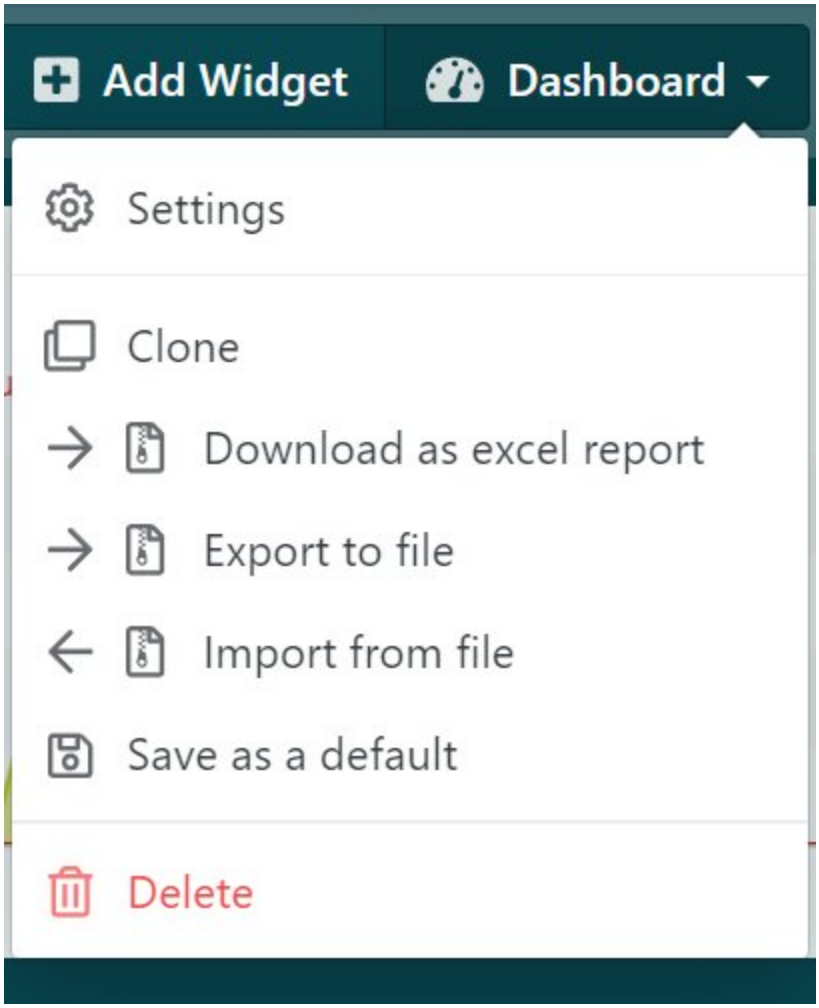
The top-level keys are `config`, `is_public`, and `widgets`. Each widget has its own `config` key describing the widget `type` and the `filter` applied. Layout values include column and row placement (`col` and `row`), display size (`sizeX`), and `time_range`.

Manipulating Dashboards in LogZilla

Dashboards can be manipulated in two ways: through the UI menu on an open dashboard, or through the `logzilla dashboards` CLI subcommand.

Dashboard Manipulation via UI

The dashboard action menu is shown below:



- *Clone*: Create a copy of an existing dashboard.
- *Download as Excel report*: Download the dashboard as an Excel report.
- *Export to file*: Write the dashboard configuration to a file and download it to the local system.
- *Import from file*: Upload a dashboard configuration file and load it into LogZilla.
- *Save as a default*: Designate the dashboard as the default assigned to new users.
- *Delete*: Permanently remove the dashboard from the system.

Dashboard Manipulation via Command Line

The `logzilla dashboards` subcommand provides equivalent capabilities.

Import

To import a dashboard from a YAML or JSON file:

```
logzilla dashboards import -I mydashboards.yaml
```

If the dashboard file contains a typo or a field LogZilla does not recognize, the import fails with an error that names the bad field. The dashboard is not partially imported. Fix the field in the file and run the import again.

Export

To export all dashboards to a YAML file:

```
logzilla dashboards export -O mydashboards.yaml
```

The `-I` flag specifies the input file for the import command; the `-O` flag specifies the output file for the export command.

To use YAML instead of JSON, add the `-F yaml` option to the above commands.

A single dashboard can be exported by first listing available dashboards:

```
logzilla dashboards list
```

which shows the list of dashboards with names and dashboard IDs. The specific dashboard can then be exported as follows:

```
logzilla dashboards export --dashboard-id <dashboard_id> -O mydashboard.yaml
```

Using Dashboards from Apps in LogZilla

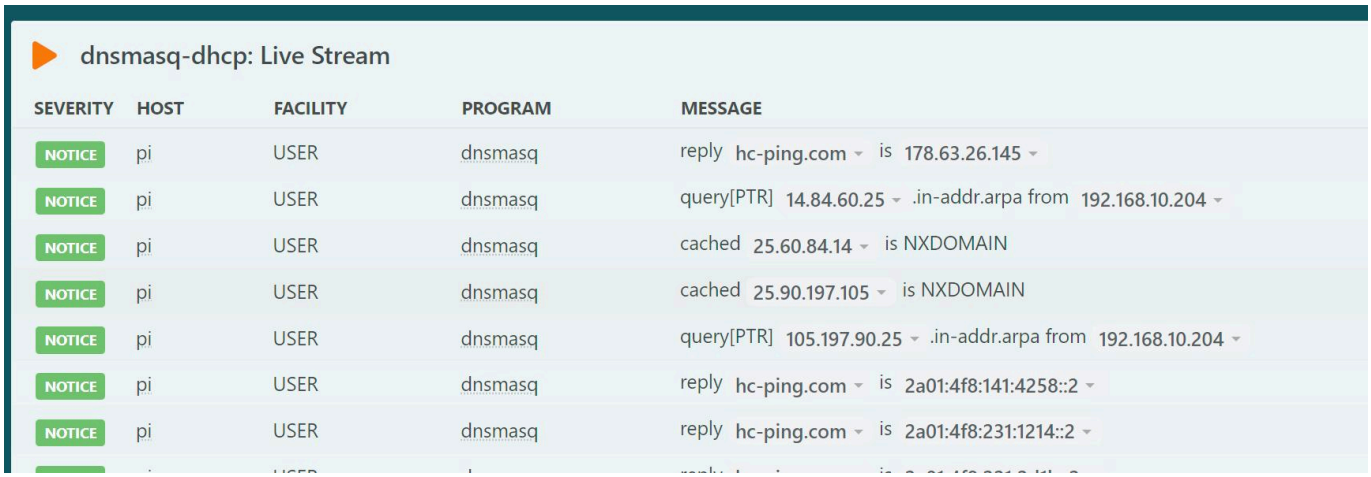
Dashboards are included with LogZilla *apps*. Once an app is installed, the app's dashboards support the full set of actions including cloning, editing, importing, and deleting. Operators modifying an app-supplied dashboard for a specific environment can use the export, edit, and re-import flow described below.

Example: Exporting, Changing, and Importing a Dashboard

The following example walks through exporting, modifying, and re-importing a dashboard.

Scenario: Modifying the Linux DNSmasq App Dashboard

The Linux DNSmasq app includes a dashboard named "Linux: dnsmasq Events" containing a widget called "dnsmasq-dhcp: Live Stream", which presents a continuous stream of incoming DHCP log messages. By default this widget displays events of type "query", "cached", and "reply":



In this example the "cached" events are filtered out of the widget. The procedure is:

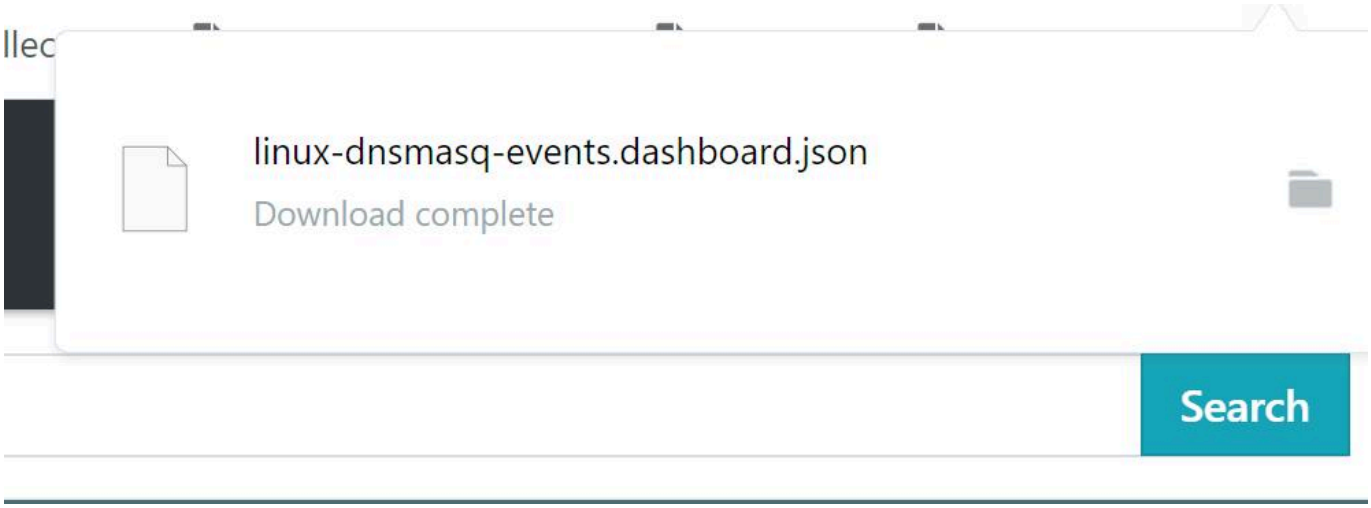
Export the dashboard

Edit the configuration file

Re-import the modified dashboard

Step 1: Exporting the Dashboard

Click *Export to file* on the dashboard menu described earlier. The dashboard configuration file is downloaded to the local system:



Step 2: Editing the Configuration File

The downloaded file contains JSON on a single line. A JSON formatter makes the file easier to edit. The formatted file begins:

```
{
    "config": {
        "style_class": "infographic",
        "time_range": {
            "preset": "last_1_minutes"
        },
        "title": "Linux: dnsmasq Events"
    },
    "widgets": [
        {
```

followed by widget configuration elements. The live-stream widget configuration looks like this:

```
{
    "config": {
        "col": 0,
        "columns": [
            "severity",
            "host",
            "facility",
            "program",
            "message",
            "first_occurrence",
            "last_occurrence",
            "counter"
        ],
        "filter": [
            {
                "field": "program",
                "op": "eq",
                "value": [
                    "dnsmasq*"
                ]
            }
        ],
        "limit": 16,
        "row": 1,
        "sizeX": 6,
        "sizeY": 2,
        "sort": "-first_occurrence",
        "title": "dnsmasq-dhcp: Live Stream"
    },
    "type": "Search"
}
```

To exclude messages matching `*cached*`, add a second filter entry:

```
    "filter": [
        {
            "field": "program",
            "op": "eq",
            "value": [
                "dnsmasq*"
            ]
        },
        {
            "field": "message",
            "op": "ne",
            "value": [
                "*cached*"
            ]
        }
    ],
```

Step 3: Importing the Modified Dashboard

Remove the existing dashboard before importing the modified copy:

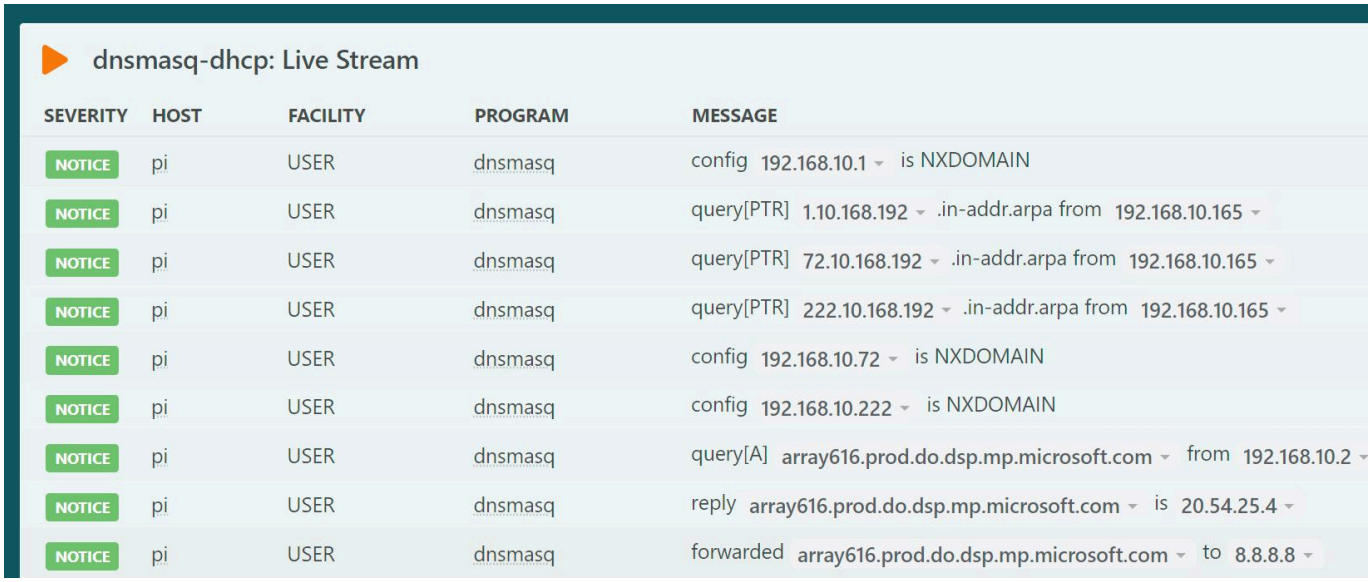
```
logzilla dashboards remove "Linux: dnsmasq Events"
```

```
These dashboards will be removed:
id: 270, title: Linux: dnsmasq Events, public: False, widgets: 4
Do you want to remove all selected dashboards [Y/n]
```

After confirming removal, import the modified file:

```
logzilla dashboards import -I linux-dnsmasq-events.dashboard.json
```

The absence of output indicates a successful import. Refreshing the LogZilla UI shows the updated dashboard:



The widget now displays only the desired event types.

CHAPTER 03

Receiving Data

Ingest logs into LogZilla from network devices, Windows, Linux, Java apps, SNMP, NGINX, cloud services, and custom HTTP integrations

© LogZilla Corporation · logzilla.ai

Receiving Data in LogZilla

LogZilla accepts log data from diverse sources including network devices, servers, applications, and cloud services. Multiple ingestion methods support real-time log collection, processing, and analysis.

Supported Data Sources

Network Infrastructure

- **Syslog Events**: Standard syslog over UDP/TCP from network devices, servers, and applications
- **Cisco IOS**: Configuration for Cisco routers, switches, and security appliances
- **Juniper SRX**: Security platform logging with structured syslog
- **SNMP Traps**: Network monitoring and alerting via SNMP trap reception

Servers and Applications

- **Windows Events**: Forward Windows Event Log entries using the LogZilla Windows Agent
- **Java Applications**: Log4j and Log4j2 integration for enterprise applications
- **NGINX**: Web server access and error logs with key-value parsing
- **BIND DNS**: DNS server query and response logging
- **Avaya Communication Manager**: Unified communications platform logs

Cloud and HTTP Sources

- **AWS CloudWatch**: Forward CloudWatch logs via Kinesis Data Firehose
- **HTTP Event Receiver**: Direct HTTP/HTTPS ingestion with JSON payloads for custom integrations and modern applications

Getting Started

Configure LogZilla listeners: Verify syslog ports and HTTP receiver settings in the LogZilla interface

Choose ingestion method: Select syslog, HTTP, or cloud-specific integration based on the data source

Configure the sender: Follow device or application-specific setup guides in this section

Verify reception: Confirm log delivery and troubleshoot connectivity if needed

For network troubleshooting and packet-level verification, see Syslog Troubleshooting.

Related Topics

- Syslog Settings
- HTTP Receiver Settings
- Data Transforms

- Event Correlation

Receiving Syslog Events

Configure LogZilla to receive standard syslog events over UDP and TCP, including listener ports, config.yaml locations, and conf.d customization

Receiving Syslog Events

LogZilla receives standard syslog events by default and typically requires no changes. To view or change listener ports and runtime options, use the UI: `Settings` → `System Settings` → `Syslog Daemon`. For detailed field descriptions, see Syslog Settings.

When advanced customization is required (custom sources, `conf.d/`, or pipeline rules), see Syslog pipeline customization. Apply changes cautiously because they can affect event ingest and performance.

Configuration is managed primarily via the UI. For background material and defaults, see Network Communications and Syslog Basics.

Configuration locations

- `/etc/logzilla/syslog-ng/config.yaml`: main YAML used to render the syslog-ng configuration inside the container.
- `/etc/logzilla/syslog-ng/conf.d/`: directory for additional `*.conf` files included by the main template. The path is controlled by the `custom_conf_dir` key in `config.yaml`.

Avoid creating custom top-level `log` statements. Use `extra_log_rules` to insert filters and rewrites into the main pipeline.

When to customize

- Add a dedicated listener for a specific source or transport.
- Tag a source using `source_type` for dedicated rule processing.
- Forward or archive events using the Forwarder module (do not configure forwarding in syslog-ng). See Downstream Syslog Receivers.

Most values in `config.yaml` are generated from LogZilla settings. To inject lightweight filters or rewrites into the main pipeline, use the `extra_log_rules` string.

Important

Do not configure destinations in syslog-ng for forwarding or archival. Use the Forwarder module so downstream systems receive parsed and enriched data. See Downstream Syslog Receivers.

Forwarding and destinations

Forwarding or archival should be configured in the Forwarder module, not in syslog-ng. For raw troubleshooting captures, use the procedures in Syslog Troubleshooting. For forwarding options (including file outputs), see the Forwarder module: Downstream Syslog Receivers.

Sources

Custom sources can be defined with dedicated ports. A `source_type` can be used to tag events from a source for specialized parsing.

Standard sources provided by default configuration (do not change unless necessary):

- `bsd` - TCP on port 514 (or the value set by SYSLOG_BSD_TCP_PORT), for BSD-style syslog messages
- `bsd_udp` - UDP on port 514 (or the value set by SYSLOG_BSD_UDP_PORT), for BSD-style syslog messages using UDP
- `rfc5424` - TCP on port 601 (or the value set by SYSLOG_RFC5424_PORT), for RFC 5424 style syslog messages
- `json` - TCP on port 515 (or the value set by SYSLOG_JSON_PORT), for sending raw JSON messages (newline separated) over a TCP connection
- `tls` - TCP on port 6514 (or the value set by SYSLOG_TLS_PORT). TLS-encrypted RFC 5424 reception
- `raw` - TCP on port 516 (or the value set by SYSLOG_RAW_PORT), for sources not complying with the syslog standard; no parsing is performed and the raw message is sent to LogZilla as is
- `raw_udp` - UDP on port 516 (or the value set by SYSLOG_RAW_UDP_PORT), same as raw, without parsing; the message is sent to LogZilla as is

To add a custom source, define an entry in the `sources` array with:

- `name` : unique source name.
- `enabled` : boolean toggle.
- `type` : `network` or `syslog` .
- `port` : listener port.
- `transport` : `tcp` , `udp` , or `tls` (for TLS-encrypted TCP).
- `tls_cert_file` / `tls_key_file` : paths to TLS certificate and key when `transport` is `tls` .
- `flags` : list of syslog-ng flags.
- `program_override` : override the `program` field value.
- `extra_fields` : key-value map added to the event `extra_fields` .
- `source_type` : tag string added to events from this source (in `extra_fields._source_type`) for dedicated parsing workflows.

Dedicated sources (`source_type`)

Dedicated parsing can be enabled by tagging events from a specific source and loading rules that target that tag:

Set `source_type` on the syslog-ng source in `config.yaml`.

In the relevant Lua rule, set `SOURCE_FILTER = "<tag>"`.

Only events with the matching `source_type` are processed by rules that declare the corresponding `SOURCE_FILTER`.

Examples: config.yaml customization

The following examples show minimal, safe changes to `config.yaml`. Add new entries to the existing `sources` list.

Example 1: Add a TLS source with a dedicated tag

Adds a TLS listener on port 6514 with certificate files and a `source_type` for dedicated rule routing. Optionally sets flags and a program name.

```
sources:
  - name: tls_west
    enabled: true
    type: network
    port: 6514
    transport: tls
    tls_cert_file: /etc/logzilla/server.crt
    tls_key_file: /etc/logzilla/server.key
    flags: ["syslog-protocol"]
    program_override: "tls-wf"
    extra_fields:
      site: "west-dc"
    source_type: "west"
```

Example 2: Add a raw UDP source for unparsed logs

Adds a UDP listener on port 1516 that bypasses syslog parsing and tags events for dedicated handling.

```
sources:
  - name: raw_udp_1516
    enabled: true
    type: network
    port: 1516
    transport: udp
    flags: ["no-parse"]
    program_override: "raw-udp"
    extra_fields:
      log_type: "raw"
    source_type: "devices"
```

After editing `config.yaml`, restart the module as shown below.

Adding extra files in `/etc/logzilla/syslog-ng/conf.d` directory

For more complex cases, additional `*.conf` files can be added in this directory, and they will be included in the main config. This can be used to add syslog-ng sources, filters, or rewrite rules.

Important

Do not create destinations here for forwarding or archival. Use the Forwarder module instead. See Downstream Syslog Receivers. Create sources as in `config.yaml` as described above.

To accomplish this:

Create a `xxx.conf` file (where `xxx` is the desired name) in the `/etc/logzilla/syslog-ng/conf.d` directory. (More than one of these files can be created, as desired, and they can all take effect.)
Add configuration directives appropriate for a source, filter, or rewrite rule to the new `xxx.conf` file. These should follow standard syslog-ng syntax (see the syslog-ng Open Source Edition Administration Guide).
Important: Custom `log` entries should **not** be created or configured. It is required that the `log` section be modified only by LogZilla, or LogZilla may cease receiving events.
If `log` customization is desired, such as adding new *filters* or *rewrites*, then see below for detailed instructions.

For many cases, adding a file in `conf.d` is enough. Sources and destinations defined in these files are implicitly added to the main config. Restart the module after changes.

For some advanced cases, like when you want to add some extra filters, then `/etc/logzilla/syslog-ng/config.yaml` should be modified. In particular, if extra *syslog-ng* configuration directives are needed, they should be added to the `extra_log_rules` entry in this file.

Example: apply a filter via `conf.d` and `extra_log_rules`

This example filters events from a specific host using a small `conf.d` file and the `extra_log_rules` hook. This avoids custom top-level `log` statements and keeps the main pipeline intact.

Create `/etc/logzilla/syslog-ng/conf.d/select_host.conf` with:

```
filter f_only_host {
    host("1.2.3.4");
};
```

Edit `/etc/logzilla/syslog-ng/config.yaml` and set:

```
extra_log_rules: "filter(f_only_host);"
```

The filter is injected into the main pipeline. Any file destinations defined in `config.yaml` or `conf.d` are included automatically. For built-in troubleshooting options, see the Debugging page below.

Restarting syslog-ng after changes

After changes to the syslog-ng configuration, restart the module:

```
logzilla restart -c syslog
```

Verification

For verification and troubleshooting steps, see Syslog Troubleshooting.

Cisco Ios Configuration

Configure Cisco IOS and IOS-XE devices to send syslog to LogZilla with correct timestamping, buffered logging, NTP sources, and loopback source-interface

Cisco IOS configuration

Minimal, safe Cisco IOS/IOS-XE configuration for sending syslog to LogZilla.

For default listener ports and how to change them, see Network Communications. For an overview of syslog formats and transports, see Syslog Basics. For troubleshooting, see Syslog Troubleshooting.

Event timestamping in LogZilla

For syslog inputs, LogZilla stores the event time as the server receive time. The syslog-ng ingestion template sets the timestamp, which avoids issues from devices with incorrect clocks.

Minimal IOS configuration

```
service timestamps log datetime localtime show-timezone
service timestamps debug datetime localtime show-timezone
logging source-interface Loopback0
logging buffered 65536
logging trap informational
logging host <LOGZILLA_IP>
ntp server <NTP_IP_1>
ntp server <NTP_IP_2>
```

Timestamps

- Use `service timestamps log datetime localtime show-timezone` and `service timestamps debug datetime localtime show-timezone` to include date/time and a timezone label in messages.
- Avoid `uptime` timestamping; include actual date/time for auditability.

Logging

- `logging source-interface <IFACE>` ensures a stable source IP/hostname (use a Loopback or management interface) so events attribute consistently.
- `logging buffered 65536` configures a local buffer. This does not affect sending to remote receivers.
- `logging host <LOGZILLA_IP>` sets the receiver. Devices commonly support multiple destinations; configure conservatively.

- `logging trap informational` (severity 0-6) is recommended for normal operation. Use 0-7 only for device-local troubleshooting.

Windows Event Forwarding

Forward Windows Event Log entries to LogZilla over HTTP and HTTPS using the LogZilla Windows Agent, with installation, access tokens, and failover servers

Windows Event Forwarding (Windows Agent)

Windows does not natively send syslog. The LogZilla Windows Agent forwards Windows Event Log entries to LogZilla over HTTP/HTTPS. The agent runs as a Windows service and requires a LogZilla access token.

Download

- Agent and README: logzilla/extras · winagent

Prerequisites

- Configuration tool `SyslogAgentConfig.exe` requires .NET Framework 4.7.2 or later.
- The service `SyslogAgent.exe` has no prerequisites.
- A LogZilla access token for authentication.

Installation and first run

Download and run the `.msi` installer from the GitHub repository.

The installer creates `C:\Program Files\LogZilla\SyslogAgent` and places the binaries and manual in that directory.

Launch the configuration tool `SyslogAgentConfig.exe` (run as Administrator).

Configuration summary

The configuration tool provides the following key settings. Refer to the README in the GitHub repository for full UI screenshots and field descriptions.

- Primary LogZilla Server
  - HTTP/HTTPS address of the LogZilla server, optionally including a port.
  - Example: `https://logzilla.example.com:8443`
- Primary / Secondary API Key
  - Use a LogZilla access token. Tokens can be created via the CLI; see Getting Started.
- Secondary LogZilla Server
  - Optional HTTP/HTTPS address for dual-forwarding.
- Primary/Secondary Use TLS and Select Primary/Secondary Cert

    - When using HTTPS, select the `.pfx` certificate files as required. The configuration tool imports these into the agent directory.

- Event Logs
    - Choose which Windows Event Logs to forward.
- Look up Account IDs
    - Look up account names for logged user IDs; disable to reduce overhead.
- Ignore / Include Event IDs
    - Reduce volume by specifying event IDs to ignore or include.
- Catch-up / Only while running
    - Choose whether to send missed events after restarts (catch-up) or only send while the agent is running.
- Facility and Severity
    - Select default facility; set severity to Dynamic or Fixed.
- Extra Key/Values
    - Add custom key-value pairs to aid parsing in LogZilla rules.
- Max Batch Size / Max Batch Age
    - Control batching size and time; set batch size to 0 for immediate send.
- File Watcher (tail)
    - Optionally tail a text file and forward each new line as an event.
    - File Name: path to the file that will be tailed.
    - Program Name: value shown in LogZilla's `program` field for tailed events.
- Log Level and Log File Name
    - Configure agent logging for local diagnostics.

Delivery endpoint and authentication

The agent sends HTTP/HTTPS POST requests to LogZilla's HTTP Receiver at the `/incoming` path and uses a token for authentication.

- Example endpoint: `https://logzilla.example.com/incoming`
- Token header examples are documented in HTTP Event Receiver.

Verification

- In LogZilla, verify events from the agent are visible. The default extra fields include:

- `_source_type: windows_agent`
- `log_type: eventlog` (or `file` when using File Watcher)
- `event_id`, `event_log`

- For network-level checks and captures, see Syslog Troubleshooting.
- For receiver endpoint reference and minimal tests, see HTTP Event Receiver.

LogZilla Windows App

Install the "MS Windows" app from the LogZilla app store (Settings → App store) so messages from the agent are parsed and labeled appropriately.

Receiving Snmp Traps

Enable the LogZilla SNMP trap daemon from the System Settings UI or CLI, set UDP port 162, and verify trap reception with program and facility filters

Receiving SNMP Traps

LogZilla can receive SNMP traps via the built-in SNMP trap daemon. The module can be enabled and configured in the web interface.

Enable in the UI

Open `Settings` → `System Settings` → `SNMPTrapd`.

Set the following fields as required:

- SNMPTrapd Enabled → On
- SNMPTrapd Port → `162` (UDP)
- SNMPTrapd Severity → default `INFO`
- SNMPTrapd Facility → default `LOCAL0`
- SNMPTrapd Program → default `SNMPTrap`
- SNMPTrapd Format → default format compatible with `snmptrapd(8)`

Click `Save changes`.

Notes:

- Defaults and field meanings are defined by the SNMPTrapd settings model: `SNMPTRAPD_ENABLED`, `SNMPTRAPD_PORT`, `SNMPTRAPD_SEVERITY`, `SNMPTRAPD_FACILITY`, `SNMPTRAPD_PROGRAM`, and `SNMPTRAPD_FORMAT`.
- If the module does not start automatically after enabling, restart services.

Verify reception

From a device or a test sender, deliver a trap to the LogZilla host on UDP port `162`.

In LogZilla, search for traps using:

- `program:"SNMPTrap"` (or the configured program value), and/or
- `facility:LOCAL0` (or the configured facility), and/or
- text from the configured `SNMPTRAPD_FORMAT`.

For packet-level checks, use the procedures in Syslog Troubleshooting.

Optional CLI Configuration

The `logzilla settings` command can be used to configure the SNMP trap daemon from the command line.

```
# Enable and set the UDP port
logzilla settings update SNMPTRAPD_ENABLED=true SNMPTRAPD_PORT=162

# Set event fields
logzilla settings update \
  SNMPTRAPD_SEVERITY=INFO \
  SNMPTRAPD_FACILITY=LOCAL0 \
  SNMPTRAPD_PROGRAM=SNMPTrap

# Set a custom snmptrapd(8) message format when needed
logzilla settings update \
  SNMPTRAPD_FORMAT='Enterprise OID: %N, Trap Type: %W, Trap Sub-Type: %q, Uptime: %T,
Description: %W, PDU Attribute/Value Pair Array: %v'

# Apply if a restart is required
logzilla restart
```

Related topics

- Server Settings
- Syslog Troubleshooting

Receiving Java Events

Forward events from Java applications to LogZilla using Log4j 2 syslog appenders over TCP with RFC5424 formatting, facility, and program name

Receiving Java Events

Java applications can forward events to LogZilla using syslog (recommended) or HTTP/HTTPS. The examples below show minimal, production-safe settings.

Prerequisites

- Confirm syslog listener ports in LogZilla (see Syslog Settings).
- Choose the facility and a clear program name for the app (for example, `LOCAL0` and `MyJavaApp`).

Option A: Log4j 2 (syslog over TCP, RFC5424)

Create or edit `log4j2.xml` and define a `Syslog` appender. Replace `LOGZILLA_HOST` with the LogZilla server address and adjust the program name and facility as needed.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Syslog name="LZSyslog"
            host="LOGZILLA_HOST"
            port="514"
            protocol="TCP"
            format="RFC5424"
            appName="MyJavaApp"
            facility="LOCAL0"/>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="LZSyslog"/>
    </Root>
  </Loggers>
  </Configuration>
```

Restart the Java application after saving changes.

Option B: Log4j 1.x (legacy)

For Log4j 1.x, add a `SyslogAppender` to `log4j.properties`. Note that the legacy appender commonly uses UDP. Upgrade to Log4j 2, or forward locally to a relay when TCP is required.

```
log4j.rootLogger=INFO, SYSLOG
log4j.appender.SYSLOG=org.apache.log4j.net.SyslogAppender
log4j.appender.SYSLOG.syslogHost=LOGZILLA_HOST
log4j.appender.SYSLOG.Facility=LOCAL0
log4j.appender.SYSLOG.layout=org.apache.log4j.EnhancedPatternLayout
log4j.appender.SYSLOG.layout.ConversionPattern=%m - thread=%t class=%C{1} method=%M%n
```

Alternative: HTTP/HTTPS

Applications that prefer HTTP can send structured JSON to the HTTP Receiver at `/incoming`. See HTTP Event Receiver for the JSON envelope, token headers, and minimal tests.

Verification

In LogZilla, confirm reception by searching for the configured program name or message text. For packet-level checks, see Syslog Troubleshooting.

Optional: Transform fields in LogZilla

If messages contain class, method, or thread markers, those values can be extracted into tags or used to adjust the `program` field with rewrite rules. See Rewrite Rules for techniques and safe deployment steps.

Juniper Srx Configuration

Configure Juniper SRX devices to forward security logs to LogZilla in RFC5424 sd-syslog structured format using stream mode and a source address

Juniper SRX Configuration

Juniper SRX devices should send logs in RFC5424 structured format (key-value pairs) rather than legacy RFC3164. The SRX `sd-syslog` format is recommended.

Prerequisites

- Confirm syslog listener ports on the LogZilla server (see Syslog Settings).
- Choose a stable source IP/interface on the SRX for logging.
- Ensure network policy allows the SRX to reach LogZilla on the selected syslog port.

Configure SRX security logging (structured syslog)

Enter configuration mode and apply settings similar to the following. Replace the sample addresses with the correct SRX source and LogZilla destination.

```
edit
set security log mode stream
set security log format sd-syslog
set security log source-address 1.1.1.1
set security log stream logzilla host 10.1.1.2
show | compare
commit check
commit
```

Notes:

- `format sd-syslog` enables RFC5424 structured data.
- `source-address` is the SRX interface address used as the syslog source.
- The `stream logzilla host` command sets the LogZilla destination address.

Verification

Generate traffic or events that produce logs on the SRX.

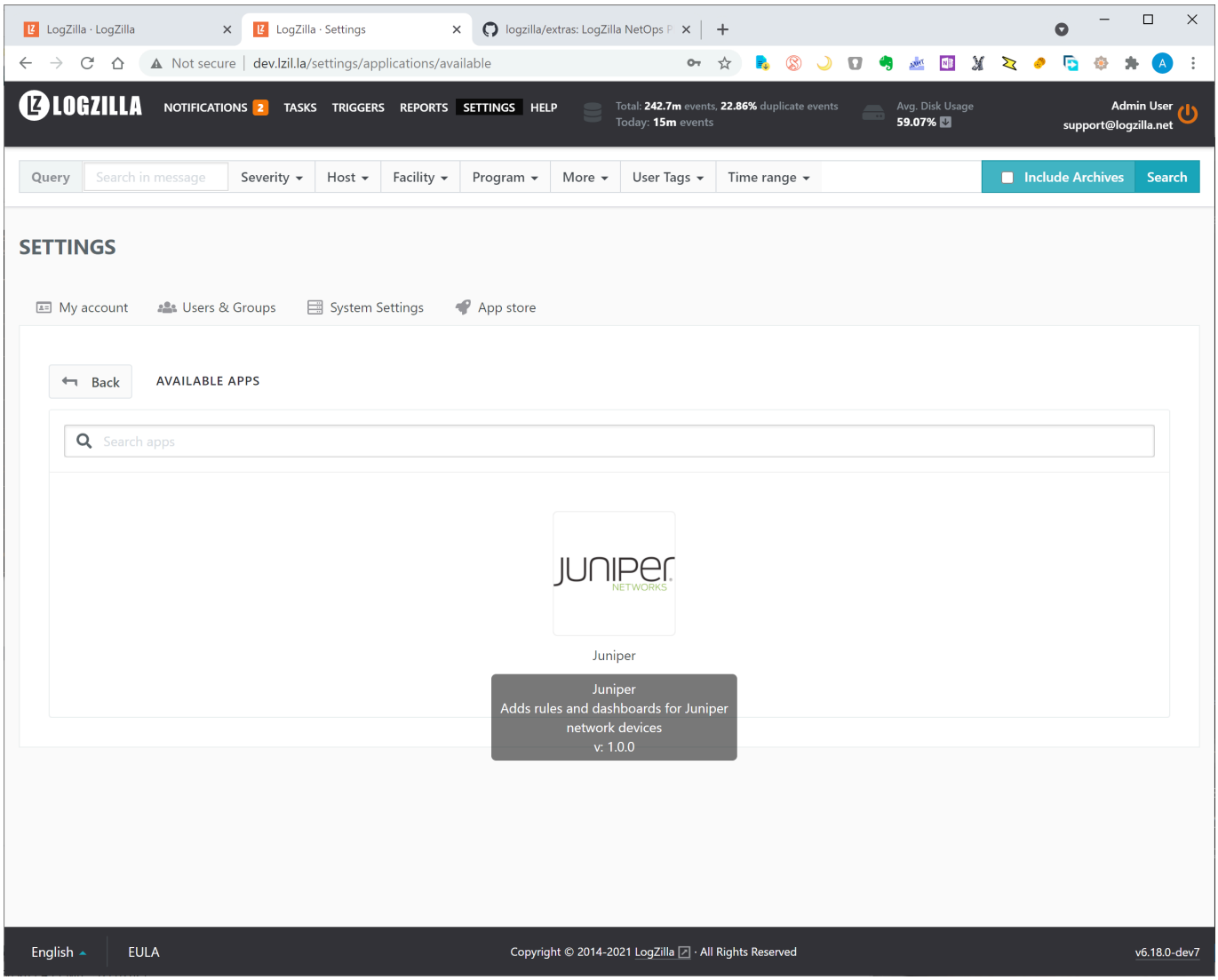
In LogZilla, search by host or program to confirm reception. For example:

- `host:"1.1.1.1"` (the configured SRX source address)
- Text from expected messages

For packet-level checks and capture examples, see Syslog Troubleshooting.

Juniper appstore rule (optional)

The Juniper appstore rule improves readability and adds user tags for key fields. Install from the `Settings → App store` page.



Related topics

- Syslog Relays

Nginx

Forward NGINX access and error logs to LogZilla using syslog with a key=value log_format that captures client IP, status, bytes, and request details

NGINX Version

Requires NGINX `v1.7.1+` (open source) or `v1.5.3+` (commercial) for syslog logging.

Prerequisites

- Confirm syslog listener ports in LogZilla (see Syslog Settings).
- Ensure the NGINX host can reach LogZilla on the chosen syslog port.

Configure logging

Verify the include is present under the `http {` block in `/etc/nginx/nginx.conf`:

```
include /etc/nginx/conf.d/*.conf;
```

Create `/etc/nginx/conf.d/logging.conf` with a key=value log format and syslog destinations. Replace `LOGZILLA_HOST` with the server hostname/IP.

LogZilla custom log format (key=value pairs)

```
log_format logzilla_kv
 'Site="$server_name" Server="$host" DstPort="$server_port" '
 'DstIP="$server_addr" Src="$remote_addr" SrcIP="$realip_remote_addr" '
 'User="$remote_user" Time_Local="$time_local" Protocol="$server_protocol" '
 'Status="$status" Bytes_Out="$bytes_sent" '
 'Bytes_In="$upstream_bytes_received" HTTP_Referrer="$http_referer" '
 'User_Agent="$http_user_agent" Nginx_Version="$nginx_version" '
 'HTTP_X_Forwarded_For="$http_x_forwarded_for" '
 'HTTP_X_Header="$http_x_header" URI_Query="$query_string" URI="$uri" '
 'HTTP_Method="$request_method" Response_Time="$upstream_response_time" '
 'Cookie="$http_cookie" Request_Time="$request_time" ';

access_log syslog:server=LOGZILLA_HOST:514,facility=local0,severity=info,tag=nginx_access
logzilla_kv;
error_log  syslog:server=LOGZILLA_HOST:514,facility=local0,severity=notice,tag=nginx_error
notice;
```

NGINX writes in key=value pairs with the `log_format` above. LogZilla extracts these pairs automatically for tagging and rewrites (see Data Transforms). Alternatively, a JSON `log_format` can be used; LogZilla extracts fields from both key=value and JSON bodies.

Reload and verify

Test and reload NGINX:

```
sudo nginx -t
sudo systemctl reload nginx
```

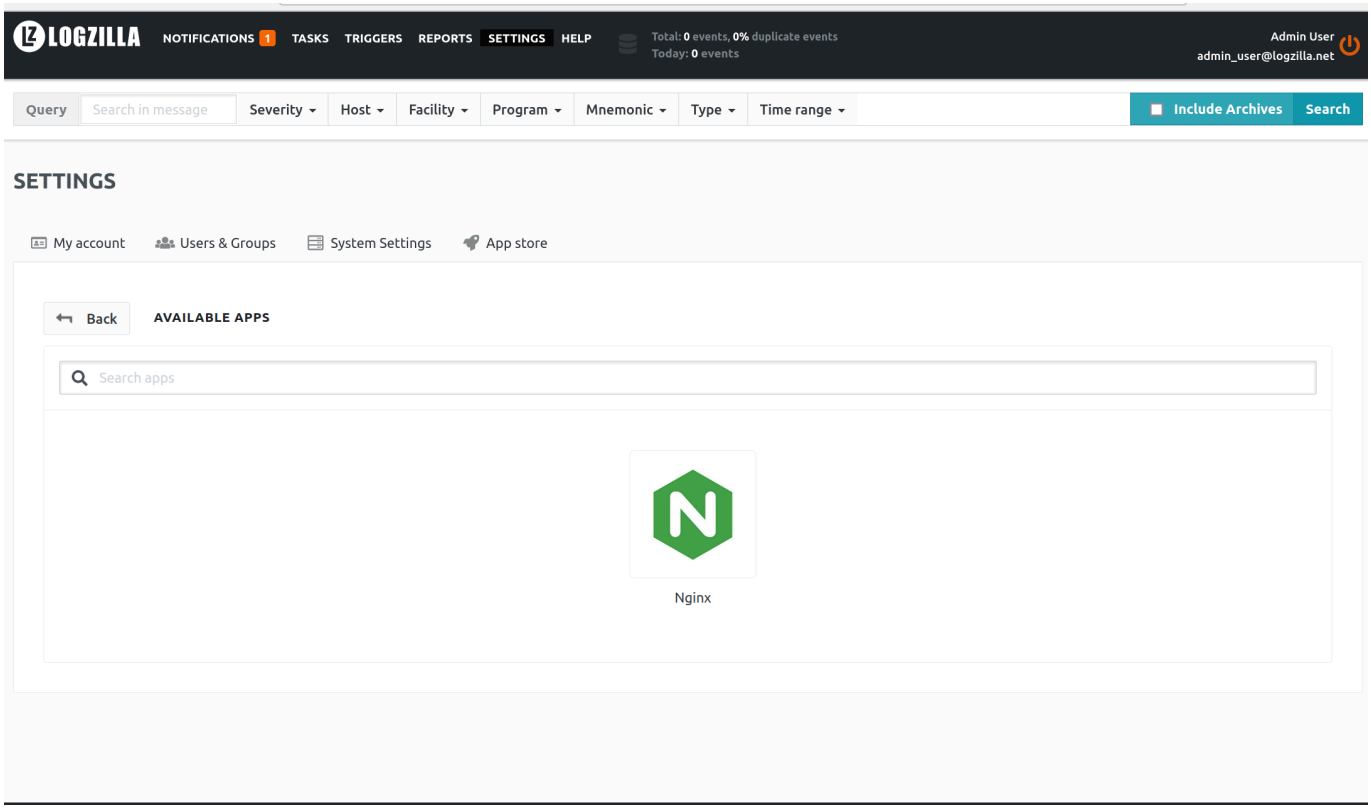
In LogZilla, search for the configured tags/programs to confirm reception:

- `program:"nginx_access"` or `program:"nginx_error"`
- Specific key=value fields (for example, `Status:"200"`)

For packet-level checks, use Syslog Troubleshooting. For encrypted or WAN paths, consider sending to a local relay and forwarding securely to LogZilla. See Syslog Relays.

Next, the nginx LogZilla rule must be installed. This rule is available from the LogZilla *appstore*. The rule is installed by going to `Settings -> App store` in the LogZilla UI.

Add the *Nginx* app to enable the rule.



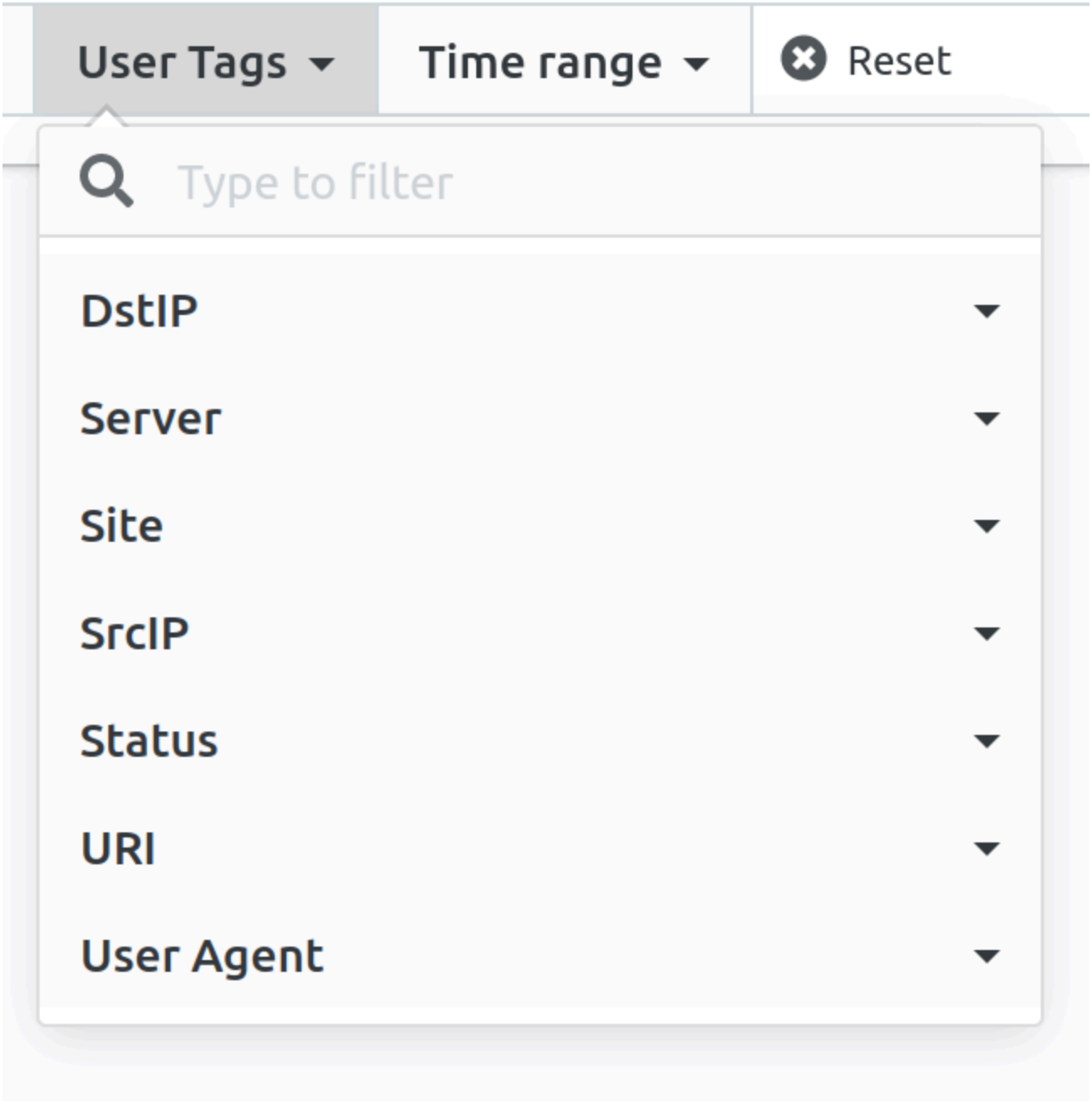
The LogZilla server should now have entries similar to the following:

```
Site="localhost" Server="192.168.250.112" DstPort="80" DstIP="192.168.250.112"
Src="192.168.250.2" SrcIP="192.168.250.2" User="-"
Time_Local="17/Nov/2021:17:45:07 +0000" Protocol="HTTP/1.1" Status="304"
Bytes_Out="189" Bytes_In="-" HTTP_Referrer="-" User_Agent="Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:94.0) Gecko/20100101 Firefox/94.0" Nginx_Version="1.18.0"
HTTP_X_Forwarded_For="-" HTTP_X_Header="-" URI_Query="-" URI="/main.html"
HTTP_Method="GET" Response_Time="-" Cookie="-" Request_Time="0.000"
```

If logs are not being sent or received, check the nginx log. For LogZilla-side troubleshooting steps, see Syslog Troubleshooting.

NGINX Dashboard Widgets

Widgets will now contain tags similar to:


User Tags
Time range
Reset
Type to filter
DstIP
Server
Site
SrcIP
Status
URI
User Agent

Paloalto Pan-Os Configuration

Configure Palo Alto PAN-OS syslog server profiles, custom log formats for traffic and threat logs, and log forwarding to LogZilla

Configure PAN-OS Syslog

Prerequisites

- Include the device IPv4 address in the syslog header: `Panorama/Device > Setup > Management` → `Logging and Reporting Settings` → `Log Export and Reporting` → set `Syslog HOSTNAME Format` to `ipv4-address`.
- Confirm syslog listener ports in LogZilla (see Syslog Settings).

Create a syslog server profile

Opens `Server Profiles > Syslog` and select `Add`.

A profile Name and Location is specified (Location refers to the virtual system when VSYS is enabled).

On the `Servers` tab, an entry is added with:

- Syslog Server (LogZilla IP or hostname)
- Transport (UDP/TCP as required)
- Port (default 514 for UDP)
- Facility (for example, `LOG_USER`)

Custom log formats

Threat logs

```
PaloAlto_Threat type="$type" src="$src" dst="$dst" rule="$rule" srcuser="$srcuser"
sessionid="$sessionid" action="$action" misc="$misc" dstloc="$dstloc" referer="$referer"
http_method="$http_method" http_headers="$http_headers"
```

Traffic logs

```
PaloAlto_Traffic type="$type" src="$src" dst="$dst" natsrc="$natsrc" natdst="$natdst"
rule="$rule" srcuser="$srcuser" from="$from" to="$to" sessionid="$sessionid"
sport="$sport" dport="$dport" natsport="$natsport" natdport="$natdport" proto="$proto"
action="$action" bytes="$bytes" packets="$packets" dstloc="$dstloc"
action_source="$action_source"
```

Attach log forwarding

A `Log Forwarding` profile is created or updated to reference the syslog server profile (Objects → Log Forwarding).

The forwarding profile is applied to the required security policies: Policies → Security → select a rule → Actions → set `Log Forwarding` to the created profile. `Log at Session End` can be enabled as required.

Commit changes

Commit the configuration (Changes → Commit).

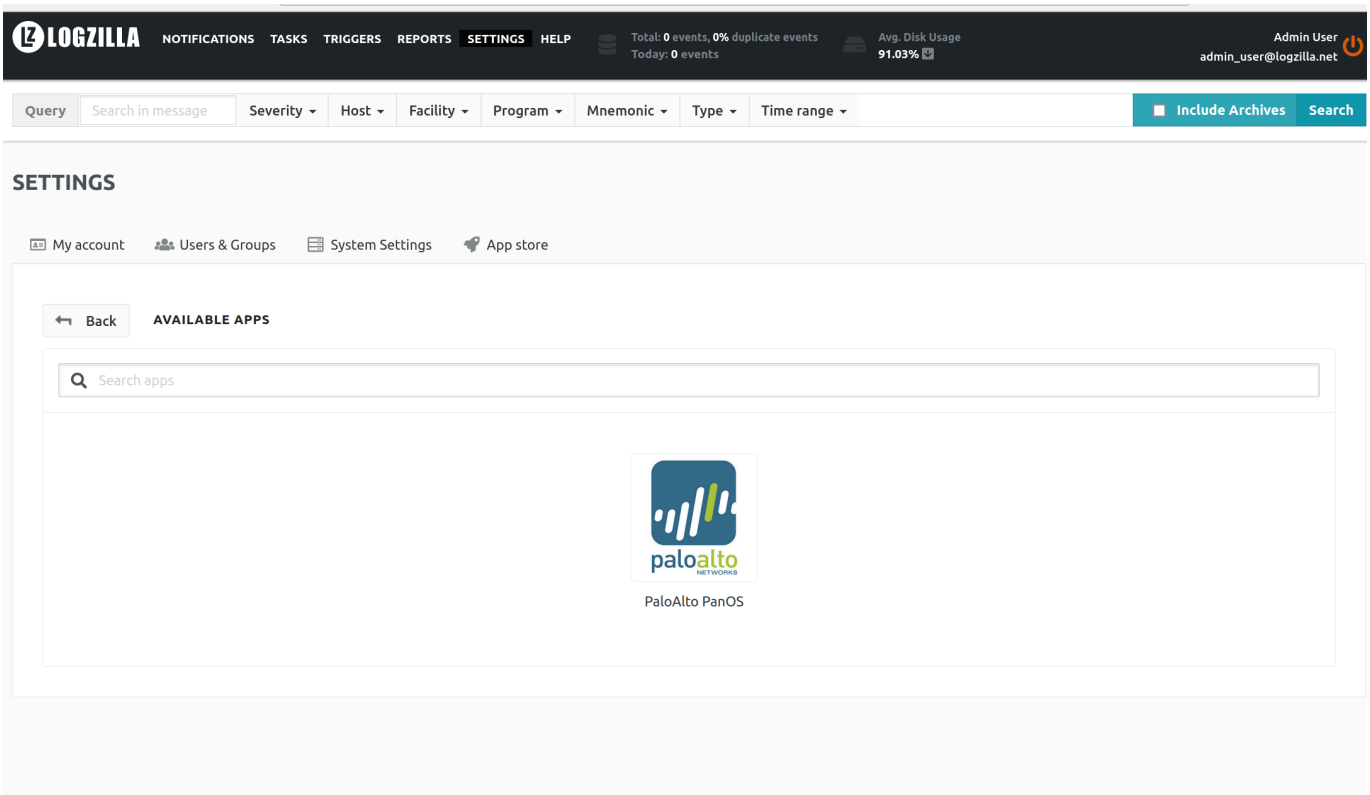
Verify in LogZilla

- Reception can be confirmed by searching for the device host or expected text in LogZilla.
- For packet-level checks, use Syslog Troubleshooting.

LogZilla Rules and Dashboards

Rules and dashboards for Palo Alto are available in the LogZilla appstore (`Settings` → `App store`).

Install the Palo Alto app to enable the rule.



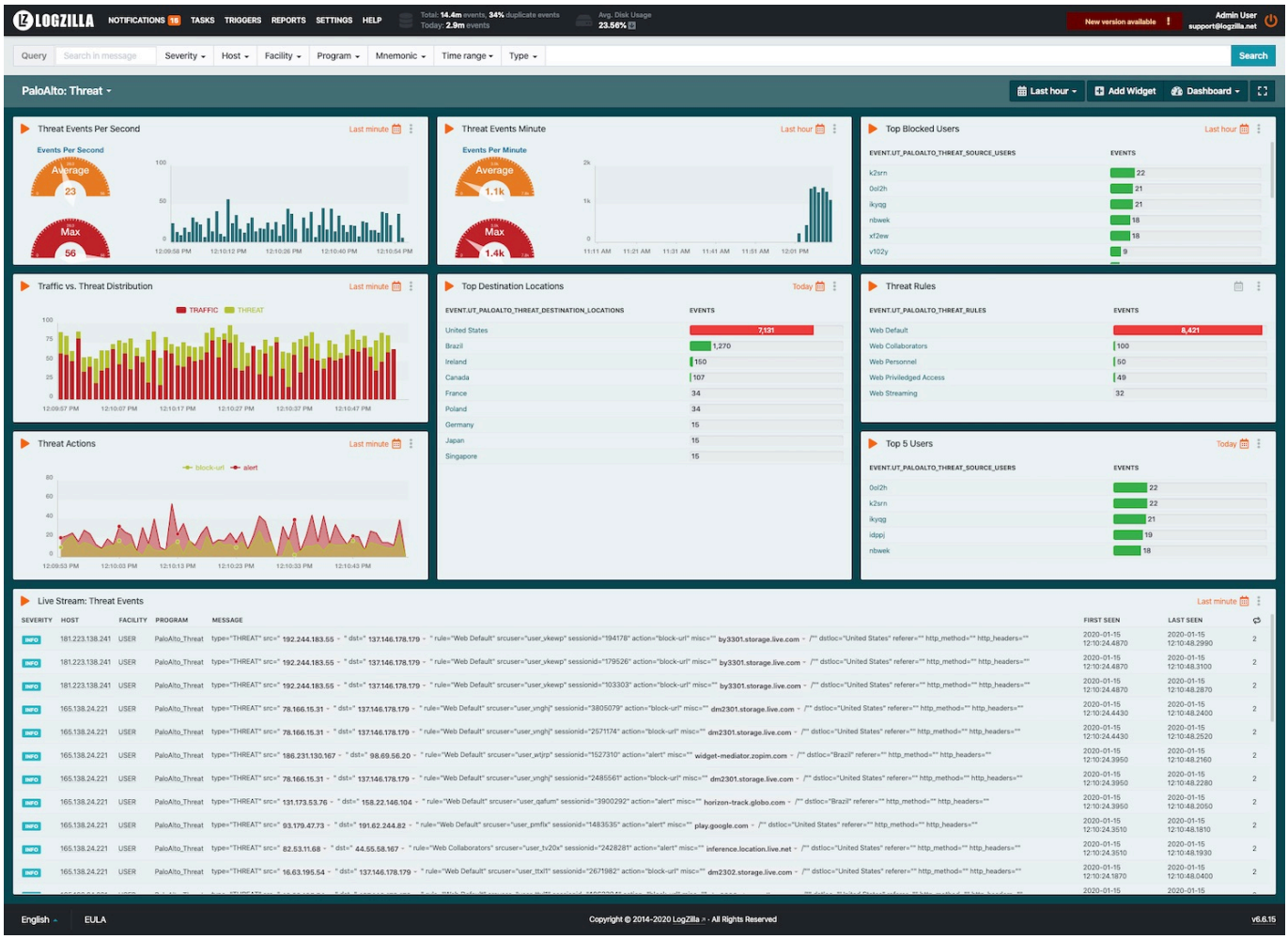
Related topics

- Syslog Settings
- Syslog Troubleshooting

Example dashboards

After installation, the dashboards will look similar to these examples.

Threat dashboard



Traffic dashboard

Aws Cloudwatch Kinesis Setup

Deliver AWS CloudWatch logs to LogZilla using Kinesis Data Firehose over HTTPS to the /firehose endpoint with access token authentication

AWS CloudWatch and Kinesis Data Firehose

AWS CloudWatch logs can be forwarded to LogZilla using Kinesis Data Firehose over HTTP/HTTPS. Firehose delivers logs to LogZilla's `/firehose` endpoint with authentication via access tokens.

Prerequisites

- A LogZilla access token, refer to the LogZilla API documentation.
- AWS account with CloudWatch logs and Kinesis Data Firehose access.
- Network access from AWS to the LogZilla server on HTTP/HTTPS ports.

Configure Kinesis Data Firehose

Create delivery stream

In the AWS Console, navigate to Kinesis Data Firehose.

Select `Create delivery stream`.

Set:

- Source: `Direct PUT` or `Kinesis Data Streams`
- Destination: `HTTP Endpoint`

Enter a delivery stream name (for example, `logzilla-firehose`).

Configure destination settings

Set the HTTP endpoint details:

- HTTP endpoint URL: `https://LOGZILLA_HOST/firehose`
- Access key: the LogZilla access token
- Content encoding: `GZIP` (recommended)
- Retry duration: `3600` seconds (or as required)

Configure backup settings

Select an S3 bucket for failed delivery backup. This is required by Firehose for error handling and replay.

Create the stream

Review settings and create the delivery stream. AWS will validate the endpoint and begin processing.

Connect CloudWatch logs

Create subscription filter

In CloudWatch, select the log group to forward.

Create a subscription filter:

- Destination: the Kinesis Data Firehose delivery stream
- Filter pattern: leave empty to forward all logs, or specify a pattern
- Role: create or select an IAM role with Firehose write permissions

Verify delivery

Check LogZilla reception

Search LogZilla for events with:

- Program: `firehose` (default for Firehose events)
- Recent timestamps matching the CloudWatch log entries

Note: If a LogZilla app is installed for the log type (such as AWS VPC Flow), the app may set different program names and add Vendor/Product tags.

Test with curl

Send a test event to verify the `/firehose` endpoint:

```
url="https://LOGZILLA_HOST/firehose"
apikey="YOUR_TOKEN_HERE"

# Create test payload (CloudWatch log format)
payload='{"logGroup":"/test/logs","logStream":"test-stream","logEvents":[{"message":"Test
log entry from curl"}]}'
test_data=$(echo "$payload" | gzip | base64 -w 0)

curl -X POST "$url" \
  -H "Content-Type: application/json" \
  -H "X-Amz-Firehose-Access-Key: $apikey" \
  -d "{\"requestId\": \"test-123\", \"records\": [{\"data\": \"$test_data\"}]}"
```

The test event should appear in LogZilla with program `firehose`.

Troubleshooting

Common issues

- **403 Forbidden**: Check the access token and ensure it matches the Firehose configuration.
- **404 Not Found**: Verify the endpoint URL uses `/firehose` (not `/incoming`).
- **No events in LogZilla**: Check CloudWatch subscription filter and Firehose delivery metrics in AWS.

Firehose delivery metrics

Monitor Firehose delivery in the AWS Console:

- Delivery success/failure rates
- Error logs for failed deliveries
- S3 backup bucket for failed events

Network troubleshooting

For packet-level checks, see Testing And Verification.

AWS VPC Flow Logs

For AWS VPC Flow Logs, install the **AWS CloudWatch VPC Flow** app from the LogZilla appstore (`Settings` → `App store`). The app provides specialized parsing, dashboards, and security triggers for VPC flow log data.

See the app's documentation for configuration details and features.

Http Event Receiver

Send events to LogZilla over HTTPS at the /incoming endpoint using structured JSON or raw payloads, with interactive API docs and OpenAPI schema

HTTP Event Receiver

LogZilla has a "universal" facility to receive events via HTTP. This is called "universal" because it is not specific to any particular scenario — it is intended to be used with custom integrations.

LogZilla receives events over HTTP(S) on the UI/API port at path `/incoming`. The listener port is configured in Front settings (see Server Settings).

Interactive API docs are available at `/incoming/docs`.

Docs and schema

- Interactive docs: `/incoming/docs`
- OpenAPI schema: `/incoming/openapi.json`

Structured JSON Data Format

Recommended format of incoming data allows for best performance, as multiple events can be sent in single request. The events sent to LogZilla should to be formatted as JSON, with structure:

```
{
  "events": [
    // event1,
    // event2,
    // etc.
  ]
}
```

As the JSON array notation indicates, more than one event message can be sent per transmission, if desired. Then each event should have structure:

```
{
  "ts": 1704063600.1234,
  "host": "testhost.org",
  "program": "testprogram",
  "message": "this is the message",
  "user_tags": {
      "city": "Atlanta",
      "state": "Georgia"
  },
  "extra_fields": {
      "city": "Atlanta",
      "state": "Georgia"
  },
  "json": {
      "int_value": 1,
      "float_value": 1.1,
      "string_value": "foo",
      "object_value": {
        "foo" : "bar",
      },
      "array_value": ["bar", "baz"]
  }
}
```

Data Contents

The event fields that can be sent to LogZilla via HTTP are:

Field	Description
`ts`	epoch timestamp
`host`	the originating host of the log message
`program`	the program that generated the log message
`message`	log message
`priority`	number represents both the RFC-3164 Facility and Severity of the event in the message
`user_tags`	additional string fields that will be available as event attributes in both LogZilla rules and queries
`extra_fields`	additional string fields that will be available as event attributes in LogZilla rules
`json`	a special field that contains any json that will be available as event attribute in LogZilla rules

Unstructured JSON Data Format

If it's not possible to use the structured JSON format, then the raw JSON can be sent, by using `/incoming/raw` path. In this case, the JSON can contain any values, and it will be in the `extra_fields` of the message, and also in the serialized form in the `message` field. The `host` will be set to the IP address of the sender, and the `program` will be set to `http_receiver`.

This case is usually used with cooperation with some rules (usually from an app) that will extract interested fields from `extra_fields` and create appropriate event, depending on the actual content.

You can also use any subpath of `/incoming/raw`, like for example `/incoming/raw/app1`. The subpath will be available in the `extra_fields._url_path` field - in this example it will be `/app1`. This can be used in the rules to recognize events from different sources.

Authentication

When sending events (structured or unstructured), provide an access token using one of the supported methods:

- `Authorization: token YOUR_GENERATED_TOKEN`
- `X-LZ-Access-Key: YOUR_GENERATED_TOKEN`
- `AUTHTOKEN=YOUR_GENERATED_TOKEN` as a query parameter

See Obtaining an Auth Token for token management. If token changes are not picked up automatically, restart the module:

```
logzilla restart -c httpreceiver
```

Upon successful receipt of a JSON `events` data element, the HTTP receiver will respond with HTTP status code `202` and message:

```
{"detail": "ok"}
```

Examples

An example curl command using structured JSON:

```
curl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: token YOUR_GENERATED_TOKEN' \
  -X POST -d  '{
      "events": [ {
        "message": "Test Message",
        "host": "curl.test",
        "program": "myapp",
        "extra_fields": { "city": "Atlanta", "state": "Georgia" },
        "json": { "int_value": 1, "string_value": "foo", "array_value": ["foo"] }
      } ] }' \
  'http://lzserver.company.com/incoming'
```

An example of using unstructured JSON:

```
curl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: token YOUR_GENERATED_TOKEN' \
  -X POST -d  '{"foo": "bar"}' \
  'http://lzserver.company.com/incoming/raw/testapp'
```

In the latter case, the event will be created with `host` set to the IP address of the sender, `program` set to `http_receiver`, and `message` set to the `{"foo": "bar"}` string. Also the `extra_fields.foo` will contain `bar` and `extra_fields._url_path` will contain `/testapp`.

Verification

To verify end-to-end reception:

- Send a minimal test payload and include `-i` to show HTTP status:

```
curl -i \
  -H 'Content-Type: application/json' \
  -H 'Authorization: token YOUR_GENERATED_TOKEN' \
  -X POST -d '{"events":[{"message":"health
check","host":"curl.test","program":"verify"}]}' \
  'http://lzserver.company.com/incoming'
```

Expected response contains `HTTP/1.1 202 Accepted` and body:

```
{"detail": "ok"}
```

Settings reference

For worker count, access logs, and invalid-request logging, see HTTP Receiver Settings.

Compatibility endpoints

For backward compatibility, the receiver supports:

- `Content-Type: application/x-ndjson` to `/incoming` (rewritten to `/incoming/syslog-ndjson`).
- `/incoming/syslog-lines` for raw lines processing.

These endpoints are deprecated; use structured JSON whenever possible.

Health endpoints

- `/incoming/ping`
  - Lightweight check. Always returns `200` with body `{"status":"ok"}`.
  - Example:

```
curl -iS http://HOST:PORT/incoming/ping
```

- `/incoming/readiness`
  - Ready when the ingest pipeline is available. Returns `200` with `{"status":"ready"}`; otherwise returns `503` with `{"message":"Service temporarily unavailable"}`.
  - Example:

```
curl -iS http://HOST:PORT/incoming/readiness
```

- `/incoming/liveness`
  - Liveness probe. Always returns `200` with `{"status":"alive"}`.
  - Example:

```
curl -iS http://HOST:PORT/incoming/liveness
```

Avaya Communications Manager

Forward Avaya Communication Manager system, security, kernel, and CM IP logs to LogZilla via syslog using the System Management Interface

Configure Avaya Communication Manager Syslog

Avaya Communication Manager (CM) can forward system logs to LogZilla via syslog over TCP or UDP. The configuration is performed through the System Management Interface (SMI) web interface.

Prerequisites

- Administrative access to Avaya Communication Manager System Management Interface
- Confirm syslog listener ports in LogZilla (see Syslog Settings)
- Network connectivity between Communication Manager and LogZilla on the selected syslog port

Configure syslog server

Access the System Management Interface and navigate to the Administration menu.

Select `Server (Maintenance)` from the menu options.

In the left navigation pane under `Security`, select `Server Log Files`.

Locate an available log server entry in the table where `Enabled` is set to `No` (typically the first row unless other syslog servers are already configured).

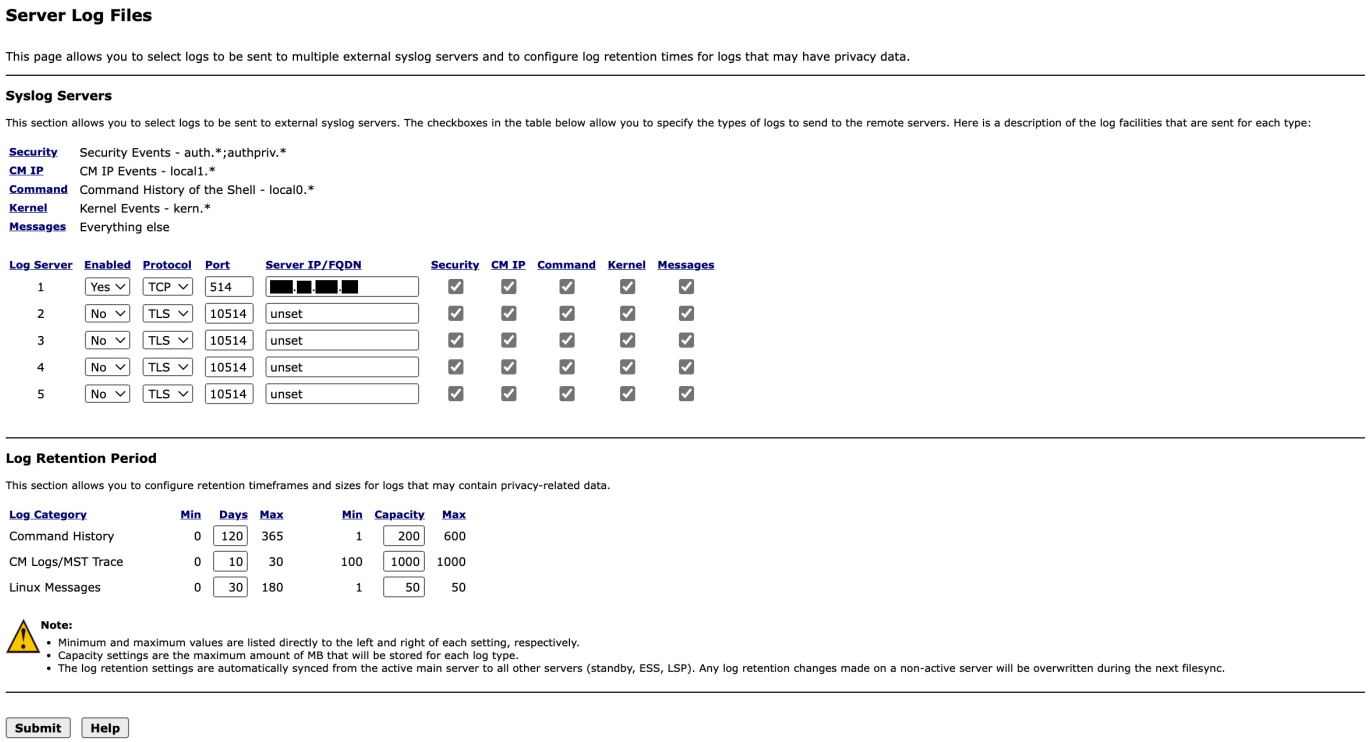
Configure the following settings:

- Enabled: `Yes`
- Protocol: `TCP` (recommended) or `UDP`
- Port: `514` (or the configured LogZilla syslog port)
- Server IP/FQDN: LogZilla server hostname or IP address

Select the log types to forward by checking the appropriate columns:

- Security: System security events
- CM IP: Communication Manager IP-related events
- Command: Administrative commands
- Kernel: Operating system kernel messages
- Messages: General system messages

Apply the configuration by clicking `Submit`.



Server Log Files

This page allows you to select logs to be sent to multiple external syslog servers and to configure log retention times for logs that may have privacy data.

Syslog Servers

This section allows you to select logs to be sent to external syslog servers. The checkboxes in the table below allow you to specify the types of logs to send to the remote servers. Here is a description of the log facilities that are sent for each type:

Security	Security Events - auth.*;authpriv.*
CM IP	CM IP Events - local1.*
Command	Command History of the Shell - local0.*
Kernel	Kernel Events - kern.*
Messages	Everything else

Log Server	Enabled	Protocol	Port	Server IP/FQDN	Security	CM IP	Command	Kernel	Messages
1	Yes	TCP	514	[redacted]	☑	☑	☑	☑	☑
2	No	TLS	10514	unset	☑	☑	☑	☑	☑
3	No	TLS	10514	unset	☑	☑	☑	☑	☑
4	No	TLS	10514	unset	☑	☑	☑	☑	☑
5	No	TLS	10514	unset	☑	☑	☑	☑	☑

Log Retention Period

This section allows you to configure retention timeframes and sizes for logs that may contain privacy-related data.

Log Category	Min	Days	Max	Min	Capacity	Max
Command History	0	120	365	1	200	600
CM Logs/MST Trace	0	10	30	100	1000	1000
Linux Messages	0	30	180	1	50	50

Note:
- Minimum and maximum values are listed directly to the left and right of each setting, respectively.
- Capacity settings are the maximum amount of MB that will be stored for each log type.
- The log retention settings are automatically synced from the active main server to all other servers (standby, ESS, LSP). Any log retention changes made on a non-active server will be overwritten during the next filesync.

Submit Help

Verify in LogZilla

- Confirm reception by searching for events from the Communication Manager host in LogZilla.
- For network-level troubleshooting, see Syslog Troubleshooting.

Related topics

- Syslog Settings
- Syslog Troubleshooting

Linux Bind

Route BIND DNS queries, responses, and server events to LogZilla by configuring named.conf logging channels and syslog-ng or rsyslog forwarding

Configure BIND DNS Logging

BIND (Berkeley Internet Name Domain) is the most widely used DNS server software. BIND logs DNS queries, responses, and server events to local syslog, which can then be forwarded to LogZilla via syslog-ng or rsyslog.

Prerequisites

- BIND DNS server installed and running on Linux
- Administrative access to the BIND server
- Syslog-ng or rsyslog installed for log forwarding
- Confirm syslog listener ports in LogZilla (see Syslog Settings)
- Network connectivity between BIND server and LogZilla on syslog ports

Configure BIND logging

Modify `/etc/bind/named.conf.options` to enable syslog output:

```
logging {
    channel syslog {
        syslog local0;
        severity info;
        print-severity yes;
        print-category yes;
    };
    category lame-servers { null; };
    category default { syslog; };
    category queries { syslog; };
};
```

This configuration:

- Sends logs to syslog facility `local0`
- Includes severity and category information
- Logs DNS queries and general server events
- Discards lame-server notifications

Configure syslog-ng forwarding

Verify the main source exists in `/etc/syslog-ng/syslog-ng.conf`:

```
source s_src {
    system();
    internal();
};
```

Create `/etc/syslog-ng/conf.d/bind.conf` to forward BIND logs:

```
filter f_bind { facility(local0); };

destination d_logzilla {
    tcp("LOGZILLA_HOST" port(514));
};

log {
    source(s_src);
    filter(f_bind);
    destination(d_logzilla);
};
```

Replace `LOGZILLA_HOST` with the LogZilla server address.

Configure rsyslog forwarding (alternative)

For rsyslog, create `/etc/rsyslog.d/bind.conf`:

```
# Forward BIND logs (local0 facility) to LogZilla
local0.* @@LOGZILLA_HOST:514
```

Replace `LOGZILLA_HOST` with the LogZilla server address. The `@@` syntax uses TCP; use `@` for UDP.

Apply configuration

Restart the services to apply changes:

```
# Restart BIND
sudo systemctl restart named

# Restart syslog service (choose one)
sudo systemctl restart syslog-ng
# OR
sudo systemctl restart rsyslog
```

Verify in LogZilla

- Confirm reception by searching for events with facility `local0` or program containing DNS-related terms.
- Generate test DNS queries to produce log entries.
- For network-level troubleshooting, see Syslog Troubleshooting.

Troubleshooting

- Verify BIND configuration syntax: `named-checkconf`
- Check service status: `systemctl status named syslog-ng`
- Test local syslog: `logger -p local0.info "BIND test message"`
- Monitor syslog files: `/var/log/syslog` or `/var/log/messages`

Related topics

- Syslog Settings
- Syslog Troubleshooting
- Syslog Relays

CHAPTER 04

Data Transforms

Overview of LogZilla data transforms: rewrite rules, Lua rules, and custom apps for normalizing event fields and adding searchable tags

© LogZilla Corporation · logzilla.ai

Data transforms

Data transforms normalize, enrich, and route events so downstream search, widgets, dashboards, and alerts operate on clean, consistent fields. This section explains when to transform data, how rewrite rules and Lua rules work, and how LogZilla Apps package transformation logic.

Topics in this section

- Transforming event streams
- Rewrite rules
- Lua rules
- Creating custom apps
- Testing and verification
- Walkthrough: rewrite rule
- Walkthrough: Lua app

Related reading

- LogZilla Apps
- Syslog pipeline customization
- HTTP Event Receiver
- Command Line Tools —- Data Commands

Transforming Event Streams

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

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
- Syslog pipeline customization
- HTTP Event Receiver
- Command Line Tools —- Data Commands

Rewrite Rules

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

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.

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.
- 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
- Lua rules
- LogZilla Apps
- Command Line Tools —- Data Commands

Lua Rules

Lua rules for advanced LogZilla event parsing and enrichment using the event object API, key-value helpers, and conditional field extraction

Lua rules

Lua rules provide advanced parsing and enrichment capabilities for complex data transformation scenarios. They execute custom logic to extract fields, apply conditional processing, and enrich events with contextual information.

When to use Lua rules

- Vendor-specific formats that require multiple extractions.
- Conditional logic to interpret message variants.
- Field mappings, lookups, or normalization across sources.
- Message reformatting to create consistent, readable content.
- Context enrichment that adds durable, searchable tags.

Event object API

Lua rules operate on an `event` object with these key properties:

- `event.message` - The original log message text
- `event.host` - Source hostname or IP address
- `event.program` - Program or service name
- `event.severity` - Numeric severity level (0-7)
- `event.facility` - Syslog facility code
- `event.user_tags` - Table for custom searchable tags
- `event.extra_fields` - Table for additional structured data

Available helper functions

LogZilla provides helper functions for common parsing tasks:

- `get_kv_parser(sep, delim, quote)` - Creates key-value parser
- `starts_with(str, prefix)` - Check string prefix
- `ends_with(str, suffix)` - Check string suffix
- `get_port_name(port)` - Convert port number to service name
- `push_event(event)` - Generate extra event from current processing context

Format examples

Lua rules are stored as `.lua` files in the rule directory. The following example shows a basic structure for parsing key-value pairs:

```
-- Parse key-value pairs from messages like: srcip="192.168.1.10" dstip="8.8.8.8"
action="accept"
local kv_parser = get_kv_parser(' ', '=', '"')

function process(event)
    -- Check if this looks like a key-value format
    if string.find(event.message, '="') then
        -- Set program name for identification
        event.program = "CustomApp"

        -- Parse all key-value pairs from the message
        local kvpairs = kv_parser:match(event.message)

        -- Convert parsed pairs to searchable user tags
        for key, value in pairs(kvpairs) do
            if key == "srcip" then
                event.user_tags["SrcIP"] = value
            elseif key == "dstip" then
                event.user_tags["DstIP"] = value
            elseif key == "action" then
                event.user_tags["Action"] = value
            end
        end
    end
end
```

For conditional tagging based on message content:

```
function process(event)
    -- Only process events from specific sources
    if event.program == "MyApp" then
        -- Add severity tags based on message content
        if string.find(event.message, "ERROR") then
            event.user_tags["Severity"] = "High"
            event.user_tags["Alert"] = "Error Detected"
        elseif string.find(event.message, "WARN") then
            event.user_tags["Severity"] = "Medium"
            event.user_tags["Alert"] = "Warning"
        end

        -- Extract and tag IP addresses
        local ip_pattern = "(%d+%.%d+%.%d+%.%d+)"
        local ip = string.match(event.message, ip_pattern)
        if ip then
            event.user_tags["IP_Address"] = ip
        end
    end
end
```

How Lua rules work

- The parsing engine evaluates installed rules in a defined order.
- Each rule inspects the event and decides whether to act.
- Rules can enrich events by writing into structured fields and tags.
- Rules can perform controlled rewrites for consistency and clarity.
- Rules can skip events that do not meet criteria without overhead.
- Rules can generate new events if needed for complex scenarios.

The result is a normalized event with useful fields for search, dashboards, and triggers.

Common enrichment patterns

- Add normalized tags, such as device role, location, or application.
- Extract identifiers (for example, user, session, interface) into dedicated fields.
- Convert vendor codes into readable labels.
- Reconstruct terse messages into human-readable summaries when needed.

Creating new events from rules

Since version 6.39, Lua rules can create additional events derived from the original event or created completely from scratch. This capability enables complex processing scenarios such as splitting message into multiple events.

The `push_event(new_event)` helper function creates new events. The `new_event` object passed to this function will be processed and indexed as any other event.

The `Event:new()` constructor creates new, empty events. All fields must be populated with appropriate values:

```
local new_event = Event:new()

new_event:set_timestamp_now() -- this updates event timestamp to current ts
new_event.message = "Sample log message"
new_event.host = "localhost"
new_event.program = "MyApp"
new_event.user_tags["CustomTag"] = "Value"

push_event(new_event)
```

Cloning existing events and modifying specific fields provides a more convenient approach. The `clone()` method supports this pattern:

```
local cloned_event = event:clone()

cloned_event.message = "Modified log message"

push_event(cloned_event)
```

Generated events are not subject to rule processing to prevent infinite loops.

The event passed as an argument to the `process()` function will be processed and indexed as usual after the rule completes. Rules that intend to create new events and skip the original should return `ProcessResult.DELETE`. Modifying the original event is more efficient than creating a new one in most cases.

Authoring guidelines

- Keep rule intent focused; split distinct concerns into separate rules.
- Prefer stable, low-cardinality tags for dashboards and filters.
- Use consistent field names across apps to improve reuse.
- Gate rules to the intended sources using dedicated inputs where applicable; see Syslog pipeline customization.

Managing rules via CLI

Refer to the CLI section for full syntax and options: Command Line Tools —- Data Commands.

Typical operations include:

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

# Validate or test a rule file before enabling
logzilla rules validate /path/to/rule.yaml
logzilla rules test --path /path/to/rule.yaml

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

# Review recent runtime errors for rules
logzilla rules errors
```

Performance and reliability

- Narrow rule scope early to minimize unnecessary processing.
- Avoid creating many high-cardinality tags; prefer normalized keys.
- Monitor parser metrics and rule errors during rollouts.
- Validate and test rules prior to enabling in production.

```
# Parser overview and throughput
logzilla events parser-stats

# Field and tag insights
logzilla events values --scope fields --limit 50
logzilla events values --scope tags --limit 50
```

Safety and governance

- Treat rules as configuration-as-code; review changes and track history.
- Favor readability and stable behavior over clever parsing tricks.
- Ensure sensitive data is not emitted as tags or message content.

Related reading

- Overview and decision guide
- Rewrite rules
- LogZilla Apps
- Syslog pipeline customization
- Command Line Tools —- Data Commands

Creating Custom Apps

Package LogZilla transformation rules, dashboards, triggers, and configuration schema as a custom app with a standard meta.yaml directory layout

Creating custom apps

A LogZilla App packages transformation logic together with dashboards and triggers so parsing, enrichment, and visualization remain consistent. Apps are the preferred way to distribute vendor-specific parsing and to standardize field names and tags across environments.

Why create an app

- Ensure consistent parsing and enrichment across systems.
- Ship ready-to-use dashboards and widgets alongside rules.
- Include common triggers for alerting on important conditions.

What an app contains

- Transformation rules
  - Lua rules implement complex parsing and enrichment.
  - Rewrite rules supplement simple match-and-modify changes.
- Dashboards
  - Panels and widgets aligned to the normalized fields and tags.
- Triggers
  - Alert conditions that fire on important events or patterns.
- Configuration files
  - User-configurable settings that modify rule behavior without code changes.

App directory structure

A LogZilla App follows this standard directory structure:

```
my_custom_app/
├── meta.yaml                    # App metadata and description
├── config/
│   ├── config.yaml              # Default configuration values
│   └── config_schema.yaml       # Configuration validation schema
├── rules/
│   ├── 100-preprocessing.lua    # Lua transformation rules
│   └── 200-enrichment.lua       # (numbered for execution order)
├── dashboards/
│   └── main_dashboard.yaml      # Dashboard definitions
└── triggers/
    └── alert_rules.yaml         # Trigger/alert definitions
```

Required files

meta.yaml

Every app must include a `meta.yaml` file with basic metadata:

```yaml
---
name: My Custom App
description: Parses and enriches events from custom application
version: '1.0'
author: Your Organization
```

config/config_schema.yaml

Define the schema for user-configurable options:

```
---
$id: my-custom-app-config
$schema: https://json-schema.org/draft/2020-12/schema
title: My Custom App Configuration
type: object
properties:
  enabled:
    type: boolean
    description: Enable event processing
    default: true
  severity_levels:
    type: object
    description: Map status codes to severity levels
    properties:
      error:
        type: string
        enum: ["High", "Medium", "Low"]
        default: "High"
```

Field and tag design

- Use clear, self-explanatory names (for example, `device_role`, `site`, `interface`).

Distribution and installation

Apps can be installed through the UI or command line:

Via UI (Settings → Applications)

- The App Store shows available and installed apps, details, and actions.
- Installing an app enables its rules, dashboards, and triggers.
- Uninstalling removes installed content; dependent apps may require prerequisites.

Via Command Line

```
# Create a new custom app (server will print the directory path as app_dir)
logzilla apps create my_custom_app

# Add your files (rules/, config/, dashboards/, triggers/) into the printed app_dir

# Validate the app without installing
logzilla apps test my_custom_app

# Install the app by code
logzilla apps install my_custom_app

# List installed apps
logzilla apps list

# Uninstall the app by code
logzilla apps uninstall my_custom_app
```

Configuration strategy

- Favor configuration-driven behavior for mappings and feature toggles.
- Validate configuration changes and support safe reload where available.
- Keep configuration small, documented, and scoped to the app's purpose.

Versioning and lifecycle

- Track changes with semantic versioning.
- Provide pre-install checks in documentation, including parent dependencies where applicable.
- Remove or archive deprecated dashboards and triggers as part of the upgrade steps.

Testing before rollout

- Validate and test rules using the CLI; see Testing and verification.
- Use representative samples via device sources or the HTTP Event Receiver.
- Review parser metrics, rule errors, and field/tag values after enabling.
- Confirm dashboards render as intended and triggers fire on expected conditions.

Governance and security

- Treat the app as configuration-as-code; review and approve changes.
- Avoid emitting sensitive data in tags, messages, or dashboards.
- Align with organizational RBAC so dashboards and alerts reflect permitted data access.

Related reading

- Overview and decision guide
- Lua rules
- Rewrite rules
- LogZilla Apps
- Command Line Tools —- Data Commands

Testing And Verification

Validate LogZilla rewrite and Lua rules with representative inputs, logzilla rules validate and test commands, rule reload, and runtime error review

Testing and verification

Testing ensures transformation logic behaves as expected before and after rollout. The following practices apply to rewrite rules, Lua rules, and Apps that include transformation logic.

Prepare representative inputs

- Use device-specific sources when available to generate realistic traffic.
- For custom pipelines or ad hoc tests, send events via the HTTP Event Receiver.
  - Structured JSON enables explicit control over `host`, `program`, and fields.
  - Unstructured JSON under `/incoming/raw` is suitable for quick experiments that rely on rules to extract values.

Manage rules during tests

- Keep test changes in a distinct rule set or app build.
- Validate syntax and structure prior to enabling.
- Reload rules after updates and monitor for errors.

```
# Validate a rule file before enabling
logzilla rules validate /path/to/rule.yaml

# Test a rule against sample data (where applicable)
logzilla rules test --path /path/to/rule.yaml

# Enable or disable rules and reload after changes
logzilla rules enable "Test Rule"
logzilla rules disable "Test Rule"
logzilla rules reload

# Inspect recent runtime errors for rules
logzilla rules errors
```

Observe parser health and field outcomes

- Confirm that the parser processes events without errors or stalls.
- Inspect field values and tag distributions to verify normalization and enrichment.

```
# Parser performance and health overview
logzilla events parser-stats

# Review values seen for fields and tags
logzilla events values --scope fields --limit 50
logzilla events values --scope tags --limit 50

# Review field and tag cardinality (optional)
logzilla events cardinality --scope all
```

Verify in the UI

- Run focused searches to confirm expected tags and normalized fields are present.
- Check pre-built or custom dashboards for visual confirmation and KPI impact.
- Confirm triggers that depend on newly enriched fields behave as intended.

Use traffic capture and replay as needed

- Capture short windows of traffic to reproduce an issue or verify a fix consistently.

```
# Start a short capture and store decoded events
logzilla sniffer start --filename sniff-{t}.out --time 60 --live

# Stop capture or check status
logzilla sniffer status
logzilla sniffer stop

# Replay captured traffic for regression testing
logzilla sniffer replay sniff-2025-01-01.out
```

Rollout and fallback

- Stage changes in a lower environment when possible.
- Monitor parser metrics and rule errors immediately after enabling.
- Maintain the ability to disable a problematic rule quickly if an unexpected outcome is observed.

Common pitfalls to avoid

- Overly broad matches that unintentionally rewrite many events.
- High-cardinality tags created by free-form data (for example, unique identifiers or full message fragments).
- Inconsistent field names that reduce dashboard and alert reuse.
- Silent failures due to unvalidated configuration changes.

Related reading

- Overview and decision guide
- Rewrite rules
- Lua rules
- Creating custom apps
- Command Line Tools —- Data Commands

Rewrite Rule Walkthrough

Worked example of a LogZilla rewrite rule that drops keepalive noise and tags surviving events with site and device_role for filtering

Rewrite Rule Walkthrough

This use case shows how to reduce event noise and add business context with a focused rewrite rule. The example remains vendor-neutral and applies to common keepalive or heartbeat patterns.

Scenario

- Devices emit periodic keepalives that clutter dashboards.
- Operations needs those dropped.
- Remaining events need searchable context tags such as `site` and `device_role` for filtering and reporting.

Goal

- Drop well-defined noise events early in the pipeline.
- Add stable, low-cardinality tags to useful events.

Prerequisites

- Permission to manage rules and view parser statistics.

Plan

Define a clear match criterion for noise (for example, "keepalive").

Identify stable tags to add (for example, `site`, `device_role`).

Author a small rewrite rule (YAML recommended; JSON is also supported).

Validate, reload, and verify behavior.

Procedure

Prepare representative events for testing.

- For ad hoc tests, post two events via the HTTP Event Receiver:

```
# Noise event candidate (should be dropped)
curl -H 'Content-Type: application/json' \
     -H 'Authorization: token YOUR_GENERATED_TOKEN' \
     -X POST -d '{
       "events": [ {
         "host": "lab-router-01",
         "program": "netd",
         "message": "keepalive: OK",
         "user_tags": {"site": "west-dc", "device_role": "edge"}
       } ] }' \
     'http://lzserver.company.com/incoming'

# Useful event candidate (should be retained and tagged)
curl -H 'Content-Type: application/json' \
     -H 'Authorization: token YOUR_GENERATED_TOKEN' \
     -X POST -d '{
       "events": [ {
         "host": "lab-router-01",
         "program": "netd",
         "message": "interface ge-0/0/1 up",
         "user_tags": {"site": "west-dc", "device_role": "edge"}
       } ] }' \
     'http://lzserver.company.com/incoming'
```

The following YAML rewrite rule example matches the noise pattern and drops those events. The rule ensures stable tags are present on retained events.

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

The first rule drops events whose `message` contains "keepalive". The second rule adds `user_tags` when `program` equals `netd`.

Validate and Load the Rule

```
# Validate rule file
logzilla rules validate /path/to/noise-reduction.yaml

# Add the rule and assign a recognizable name
logzilla rules add /path/to/noise-reduction.yaml --name "Noise Reduction"

# Reload rules so changes take effect
logzilla rules reload
```

Troubleshooting

- If events are not dropped, verify the match operator and value: use `op: "=*"` with `"*keepalive*"` to match wildcard contains.
- Confirm the event fields being matched (for example, `message`, `program`) by inspecting values:

  ```
  logzilla events values --scope fields --limit 50
  ```

- Check for rule errors:

  ```
  logzilla rules errors
  ```

- Ensure rules were reloaded after changes:

  ```
  logzilla rules reload
  ```

Notes

- Rewrite rules are best for simple, deterministic actions:
  - Normalize a core field, add a tag, replace controlled text, parse a simple key=value pair, or drop low-value events.
- Escalate to Lua when multiple extractions, conditional logic, external lookups, or message reformatting are required.

Related reading

- Rewrite rules
- Lua rules
- Testing and verification
- Command Line Tools —- Data Commands

Lua App Walkthrough

End-to-end LogZilla App example that parses SaaS webhook JSON with a Lua rule and ships dashboards and triggers for the normalized fields

Lua Engine Walkthrough

This use case shows how to package parsing and enrichment logic together with dashboards and triggers as a LogZilla App. The example remains vendor-neutral and focuses on events posted via the HTTP Event Receiver.

Scenario

- A SaaS system posts webhook events to `/incoming`.
- The organization needs normalized fields, searchable tags, and a dashboard and alerts relevant to these events.
- The solution should be packaged for repeatable deployment.

Goal

- Normalize incoming events and add durable, searchable tags.
- Provide dashboards and triggers aligned to the normalized fields.
- Package everything as a LogZilla App and manage it via the Applications UI.

Prerequisites

- Permission to manage applications in Settings → Applications.
- Ability to post test data through the HTTP Event Receiver.
- Access to rule management commands under Command Line Tools —- Data Commands for testing.

Plan

Define inputs and outputs

- Inputs: representative sample payloads from the SaaS webhook.
- Outputs: normalized fields and tags the organization will search on and chart in dashboards.

Create the Lua rule that processes webhook JSON payloads.

Extract key fields and add searchable tags.

Package the rule with a dashboard and trigger as a LogZilla App.

Install and verify the app via the Applications UI.

Implementation

Step 1: Create the Lua Rule

Create the file `rules/500-webhook-parser.lua`:

```
-- Webhook Parser Rule
-- Processes JSON payloads from HTTP Event Receiver
-- Sample webhook: {"event_type": "user_login", "user_id": "john.doe", "ip_address":
"192.168.1.100", "status": "success"}

function process(event)
    -- Only process events from the HTTP receiver for our webhook app
    if event.program ~= "http_receiver" or event.extra_fields._url_app ~=
"webhook_parser" then
        return
    end

    -- Skip if already processed by an upstream rule
    if event.extra_fields._source_type ~= nil then
        return
    end

    -- Set basic event identification
    event.program = "WebhookApp"

    -- Extract and normalize key fields from the JSON payload
    for k, v in pairs(event.extra_fields) do
        -- Skip internal LogZilla fields
        if not starts_with(k, "_") and not starts_with(k, "tags[") then
            -- Map specific fields to user tags
            if k == "event_type" then
                event.user_tags["EventType"] = v
            elseif k == "user_id" then
                event.user_tags["UserID"] = v
            elseif k == "ip_address" then
                event.user_tags["SourceIP"] = v
            elseif k == "status" then
                event.user_tags["Status"] = v

                -- Add severity based on status
                if v == "failed" or v == "error" then
                    event.user_tags["Severity"] = "High"
                elseif v == "warning" then
                    event.user_tags["Severity"] = "Medium"
                else
                    event.user_tags["Severity"] = "Low"
                end
            end
        end
    end

    -- Create a readable message from the extracted data
    local event_type = event.user_tags["EventType"] or "unknown"
    local user_id = event.user_tags["UserID"] or "unknown"
```

```
    local status = event.user_tags["Status"] or "unknown"

    event.message = string.format("Webhook Event: %s for user %s - Status: %s",
                                  event_type, user_id, status)

    -- Mark event as processed by this app to avoid re-processing
    event.extra_fields._source_type = "webhook"
end
```

Best practice: Guard rules to avoid re-processing. Check an internal marker (for example, `event.extra_fields._source_type`) at the start of the rule and set it after successful processing. This reduces unnecessary work when events are revisited or flow through multiple rules.

Step 2: Create App Configuration

Create `config/config.yaml`:

```
---
# Webhook Parser Configuration
enabled: true
severity_mapping:
  failed: "High"
  error: "High"
  warning: "Medium"
  success: "Low"
```

Step 3: Create App Metadata

Create `meta.yaml`:

```
---
name: Webhook Parser
description: Parses and enriches webhook events from external systems
version: '1.0'
author: Your Organization
```

Step 4: Test the Rule

Test the Lua rule with sample data:

```
# Validate the rule syntax
logzilla rules validate rules/500-webhook-parser.lua

# Test with sample webhook data
curl -X POST http://your-logzilla-server/incoming/raw/webhook_parser \
  -H "Authorization: token YOUR_GENERATED_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"event_type": "user_login", "user_id": "john.doe", "ip_address": "192.168.1.100",
"status": "success"}'

# Check if events are being processed correctly
logzilla events search --limit 10 --filter program=WebhookApp
```

Step 5: Package and Deploy the App

Create the complete app directory structure:

```
webhook_parser/
├── meta.yaml
├── config/
│   └── config.yaml
└── rules/
    └── 500-webhook-parser.lua
```

Install the app via the Applications UI or CLI:

```
# Create the app and note the printed app_dir
logzilla apps create webhook_parser

# Validate without installing
logzilla apps test webhook_parser

# Install by code
logzilla apps install webhook_parser

# Verify installation
logzilla apps list
```

Test the complete workflow:

```
# Send test webhook
curl -X POST http://your-logzilla-server/incoming/raw/webhook_parser \
  -H "Authorization: token YOUR_GENERATED_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"event_type": "user_logout", "user_id": "jane.smith", "status": "failed"}'

# Verify tags were created
logzilla events values --scope tags --limit 20
```

Verify App

After installation, verify the app is working by checking:

Events are being parsed: Search for `program=WebhookApp`

Tags are being created: Look for `EventType`, `UserID`, `SourceIP`, `Status`, `Severity` tags

Messages are readable: Verify the formatted message content

Sample Test Data

Use this sample webhook payload for testing:

```
curl -H 'Content-Type: application/json' \
     -H 'Authorization: token YOUR_GENERATED_TOKEN' \
     -X POST -d '{
       "events": [ {
         "host": "webhook.source.local",
         "program": "http_receiver",
         "message": "saas webhook",
         "extra_fields": {
           "source": "saas",
           "tenant": "acme",
           "action": "user_login",
           "status": "success"
         }
       } ] }' \
     'http://lzserver.company.com/incoming'
```

Design normalized fields and tags

- Define a concise set of fields and tags used by dashboards and triggers (for example, `tenant`, `action`, `resource`, `status`, `severity`, and `site`).
- Avoid high-cardinality values where possible.

Implement transformation logic (high level)

- Lua rules: extract multiple values from the payload, apply conditional branches for message variants, normalize severity, and build a human-readable message when necessary.
- Rewrite rules (optional): apply simple match-and-modify normalization such as standardizing `program` or adding a static `device_role`.

Prepare dashboards and triggers

- Dashboards: present trends and breakdowns across the normalized fields (for example, actions over time, top tenants, failures by resource).
- Triggers: alert on important conditions, such as repeated failures or high-severity events.

Install and manage the app

- Use Settings → Applications to install the packaged app.
- The App Store shows available and installed apps, details, and actions.
- Installing enables the app's rules, dashboards, and triggers.

Verification

- Post representative events via the HTTP Event Receiver and confirm:
  - Normalized fields and tags are present in search results.
  - Dashboards populate as expected.
  - Triggers evaluate correctly and produce the intended actions.

```
# Review recent rule errors
logzilla rules errors

# Parser performance and health
logzilla events parser-stats

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

Troubleshooting

- Verify payload ingestion and parser health:

  ```
  logzilla events parser-stats
  ```

- Check for rule errors:

```
logzilla rules errors
```

- Inspect fields and tags used by dashboards and triggers:

```
logzilla events values --scope fields --limit 50
logzilla events values --scope tags --limit 50
```

- Reload rules after changes:

```
logzilla rules reload
```

Rollout and maintenance

- Version the app and record notable changes and deprecations.
- Keep normalized fields stable to protect dashboards and alerts.
- Review rule errors and parser metrics immediately after updates. {{ ... }}

Notes

- Apps are the preferred vehicle for distributing transformation rules with dashboards and triggers for consistent outcomes.
- Use Lua rules for complex extraction and enrichment; supplement with rewrite rules for simple normalization where appropriate.
- Avoid exposing sensitive values in tags or messages.

Related reading

- Creating custom apps
- Lua rules
- Rewrite rules
- Testing and verification
- LogZilla Apps
- Command Line Tools —- Data Commands

CHAPTER 05

Creating Triggers

LogZilla triggers watch live event streams and fire predefined responses, from notifications to custom scripts, when matching conditions occur

At the heart of LogZilla's responsive capabilities lie the 'triggers'. These are predefined criteria that, when met, initiate a specific response. Triggers act as watchful guardians, continuously monitoring event streams. Upon detecting a specified event, the trigger 'fires', setting in motion a defined reaction. This could range from recording the event's occurrence to initiating automated solutions.

The following pages walk through creating and fine-tuning triggers, from the dashboard where monitoring begins to the detailed steps of defining a trigger.

Trigger Page

Create and order LogZilla triggers from the Triggers page, matching on user tags, severity, and message content to drive downstream actions

Trigger Firing Order

Note that the order in which triggers are listed are the same order they will be matched upon (from top to bottom of the page). Once a match is made and stop flag is enabled for the trigger, no other triggers are processed. Thus, it is important that users start with the most finite matches and prioritize wider ranging matches further down the list.

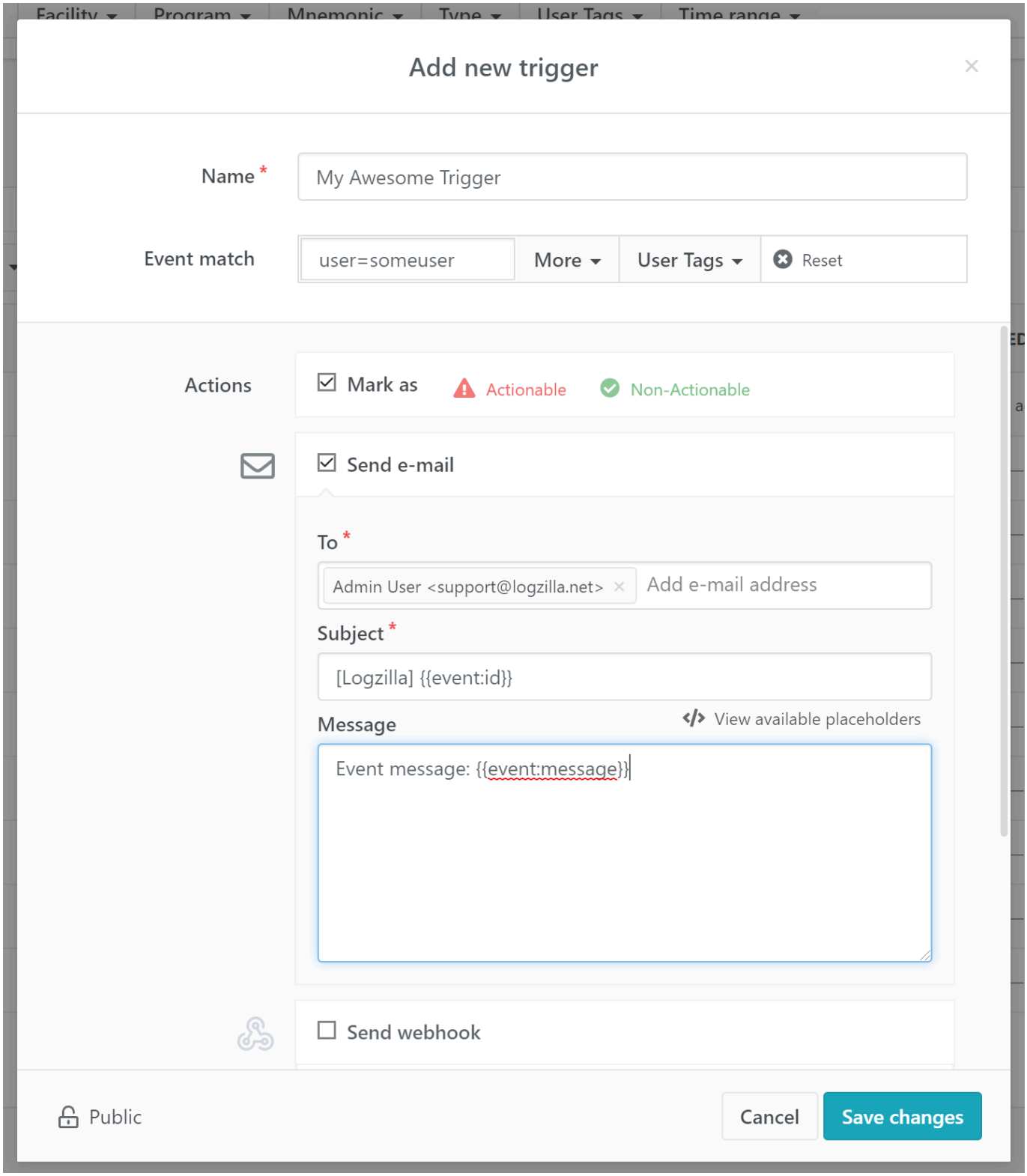
For example, a match on `interface` would match `interface GigabitEthernet1/0/1`, `interface GigabitEthernet1/0/2`, etc., then stop processing further rules.

Instead, users may want a more finite match such as `GigabitEthernet1/0/1` to be ordered higher (or lower depending on the intent).

Creating a Trigger

In the LogZilla UI, click the 'Triggers' link in the top menu. There, users will see a button near the top of the page 'Add new trigger', and below that a list of any triggers already created on the server. Clicking the button will allow users to create a trigger with no pre-set information selected. This is the easiest way to create triggers that will apply to the widest range of conditions.

If users would like to monitor failed logins for all servers, this is the best place to do it. Simply click the button, give the new trigger a name, and enter the search criteria, 'failed login' in the 'Event match' section. By default, 'Issue Notification' is already selected, so for a system wide rule, that's all that is needed. Just click 'Save changes' and the trigger will be active.

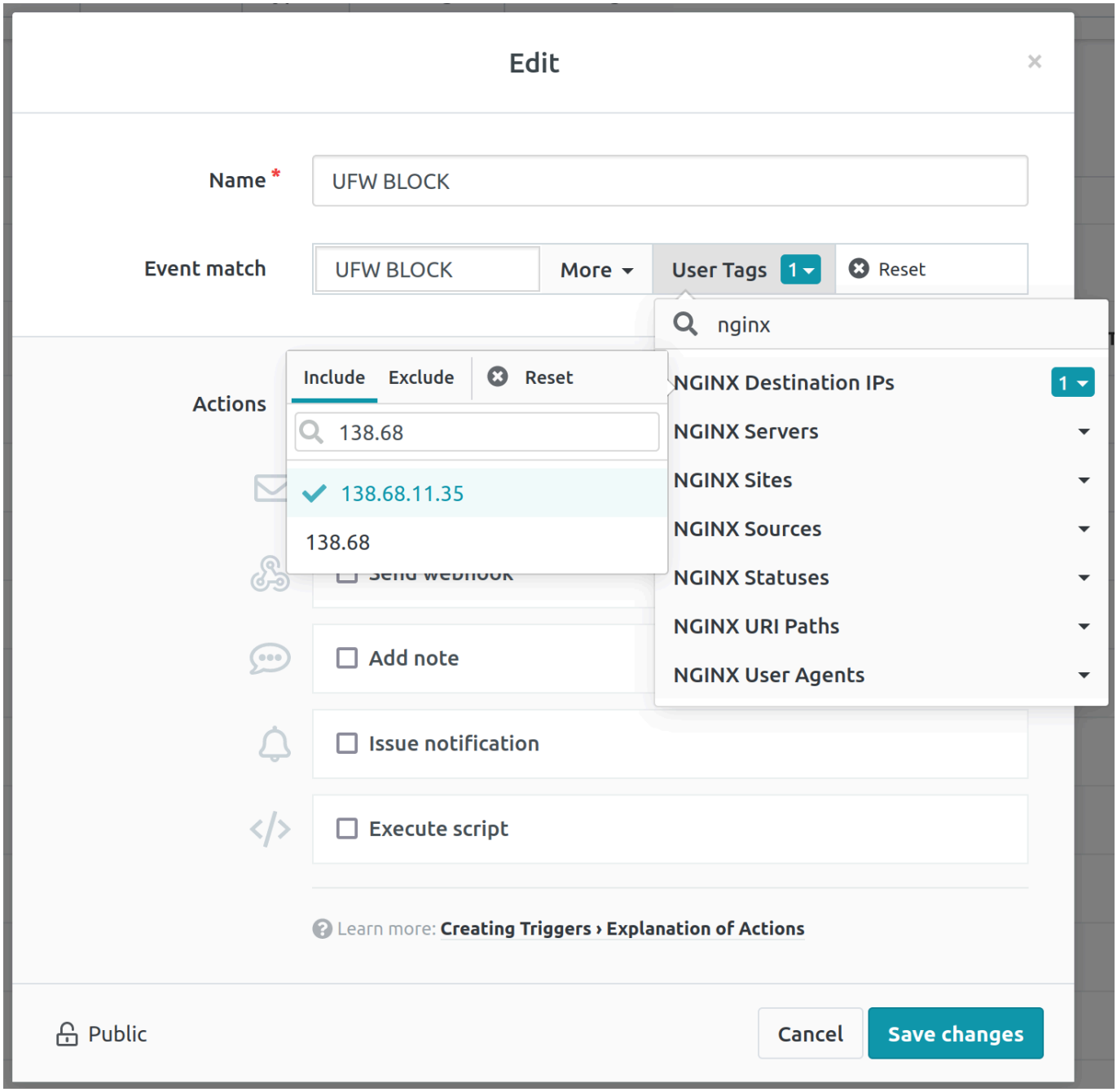


User tags can be used in the filter. User tags are special key/value pairs associated with each individual event. The LogZilla rules can parse the data in each event message and then set specific named (configurable) tags to values from the event data. For example, some common tags are `DstIP` and `DstPort`, respectively representing the destination IP address and the destination IP port for the given event. User tag `DstIP` could for example have value `192.168.0.2`.

Triggering events can be filtered based on user tags. If the "User Tag" dropdown is selected, optionally at the top of the dropdown a filter for the desired user tag name can be entered (such as if user tag `DstPort` is desired then "Dst" can be entered in the search field at the top of the dropdown, and each user tag with a name containing "Dst", such as `DstPort` will be listed).

Once the desired user tag is shown it can be clicked to open the values dropdown. The values dropdown allows choosing the particular values for the given user tag either to be included or excluded, such that only those events with the chosen values for the designated user tag will cause the trigger, or those with the chosen values will be specifically excluded from causing the trigger. The top of this dropdown as well contains the search box to find particular values of interest. Multiple user tag values can be chosen by clicking on each and a check mark will be shown next to those so designated as an indicator, or the checked ones can be clicked once more to deselect them.

A special value of `*` can be typed in, then selected. This value has special meaning: it selects only those events that have *some* value for the designated user tag. This is useful because not every event may contain every user tag. For example there may be events that have no SrcPort, and those events are not desired to be included. In order to select only those events that have a value for SrcPort, without distinction of what that value is, the `*` filter value should be used.

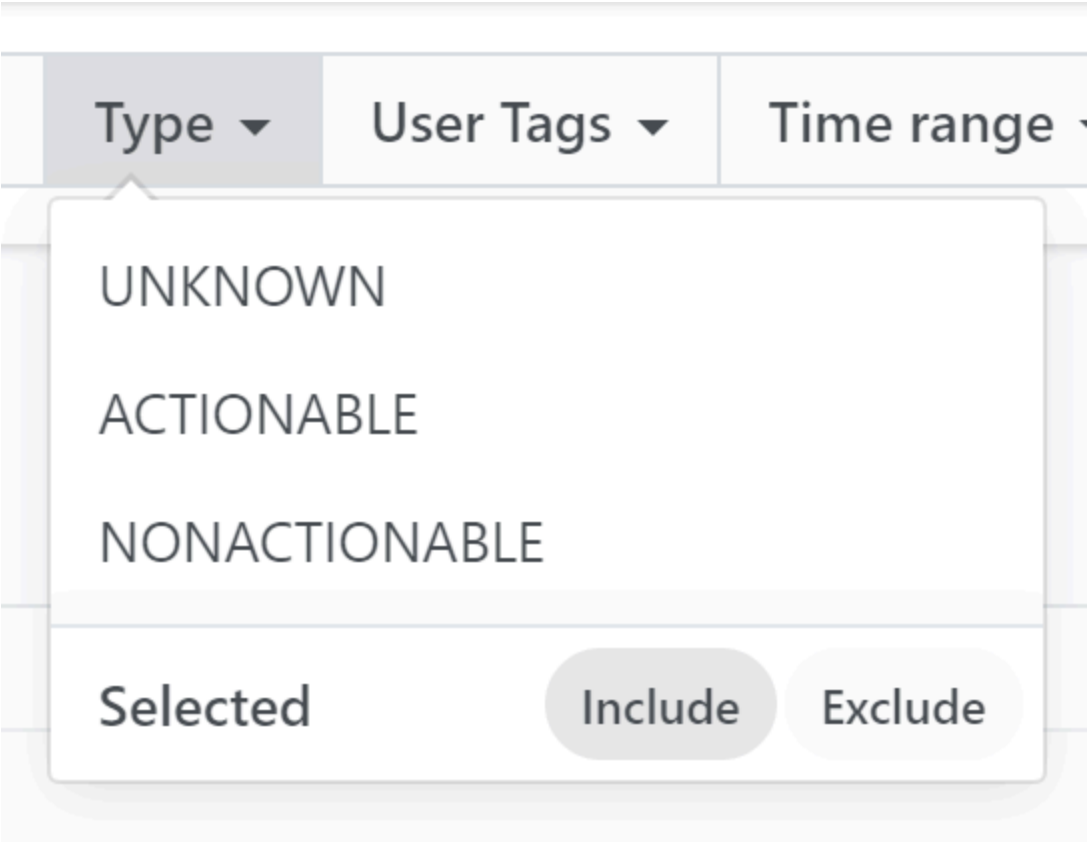

Edit
Name *
UFW BLOCK
Event match
UFW BLOCK
More
User Tags 1
Reset
nginx
NGINX Destination IPs 1
NGINX Servers
NGINX Sites
NGINX Sources
NGINX Statuses
NGINX URI Paths
NGINX User Agents
Actions
Include
Exclude
Reset
138.68
138.68.11.35
138.68
Add note
Issue notification
Execute script
Learn more: Creating Triggers › Explanation of Actions
Public
Cancel
Save changes

Explanation Of Actions

LogZilla trigger actions covering Mark As, Send E-mail, Webhook, and Run Script, including event variable substitution and regex capture syntax

Mark As

This allows users to mark incoming events as Actionable or Non-actionable. This simplifies future searches when using these options from the 'Type' drop down in the search bar.

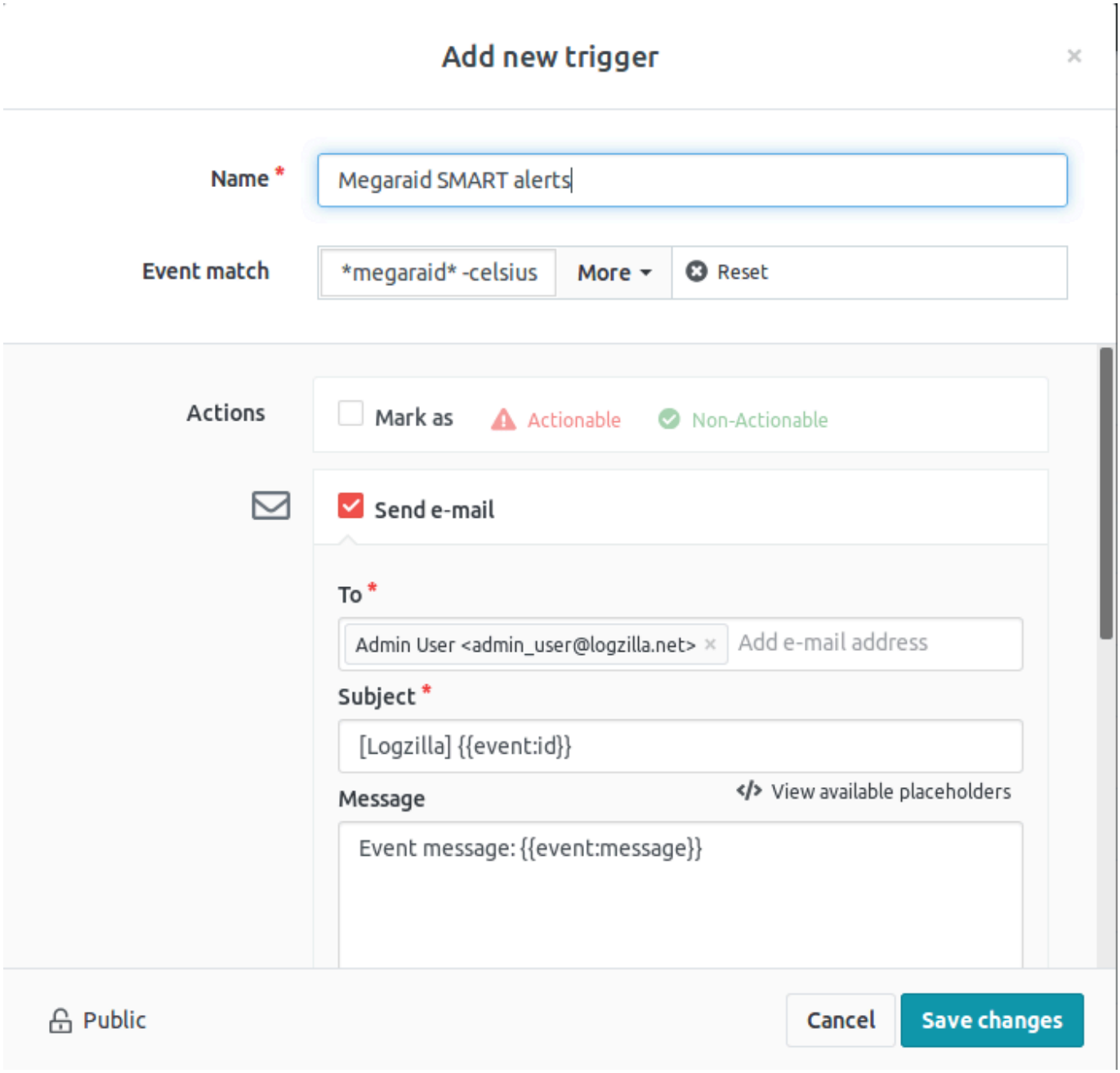


The value of this is that everyday events that administrators don't need cluttering search results can be marked as Non-actionable, while events like 'low disk space', 'fan failure', or 'CPU over-utilization' can be marked as Actionable.

When searching, events that are not marked with either can be found by selecting the 'Unknown' type.

Send E-mail

For high priority events, administrators may need immediate notification of occurrence. Selecting this option allows users to enter the address of the person or team responsible.



Users can also add a Subject and message content for this trigger. Variables that can be used are:

- `{{event:host}}`
- `{{event:severity}}`
- `{{event:facility}}`
- `{{event:first_occurrence}}`
- `{{event:last_occurrence}}`
- `{{event:program}}`
- `{{event:cisco_mnemonic}}`

- `{{event:snareid}}`
- `{{event:message}}`
- `{{event:ut:abc}}` (the meaning of this is "user tag named abc")
- `{{regexp:message:abc:n}}` (see explanation below)

`Match-Message` can be used to match portions of the event message based on regular expressions. Define one or more patterns in the email template header using lines of the form: `Match-Message-<name>: <regex>`

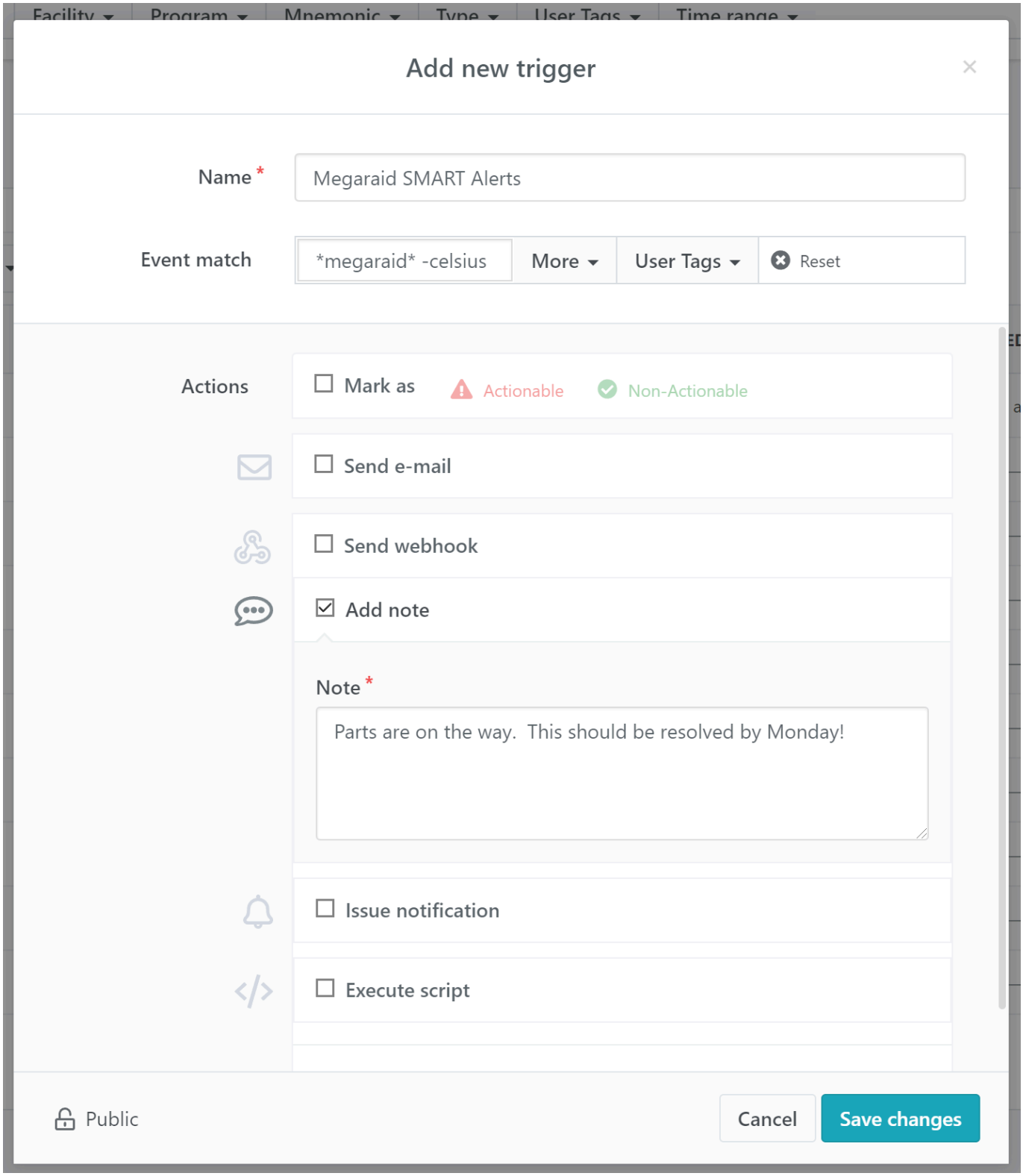
For example, to capture an endpoint IP address and MAC address: `Match-Message-EndpointIPAddress: EndpointIPAddress="(\d+\.\d+\.\d+\.\d+)"` `Match-Message-EndpointMacAddress: EndpointMacAddress="((?:\w\w:){5}\w\w)"`

Then use the extracted values as `{{regexp:message:<name>:n}}`, where `n` is 0 for the whole match, or 1, 2, and so on for content of the nth parenthesized group in the regular expression. Using the examples above: `{{regexp:message:EndpointIPAddress:1}}`

See the Settings sections of the documentation for information on setting SMTP options for email alerts.

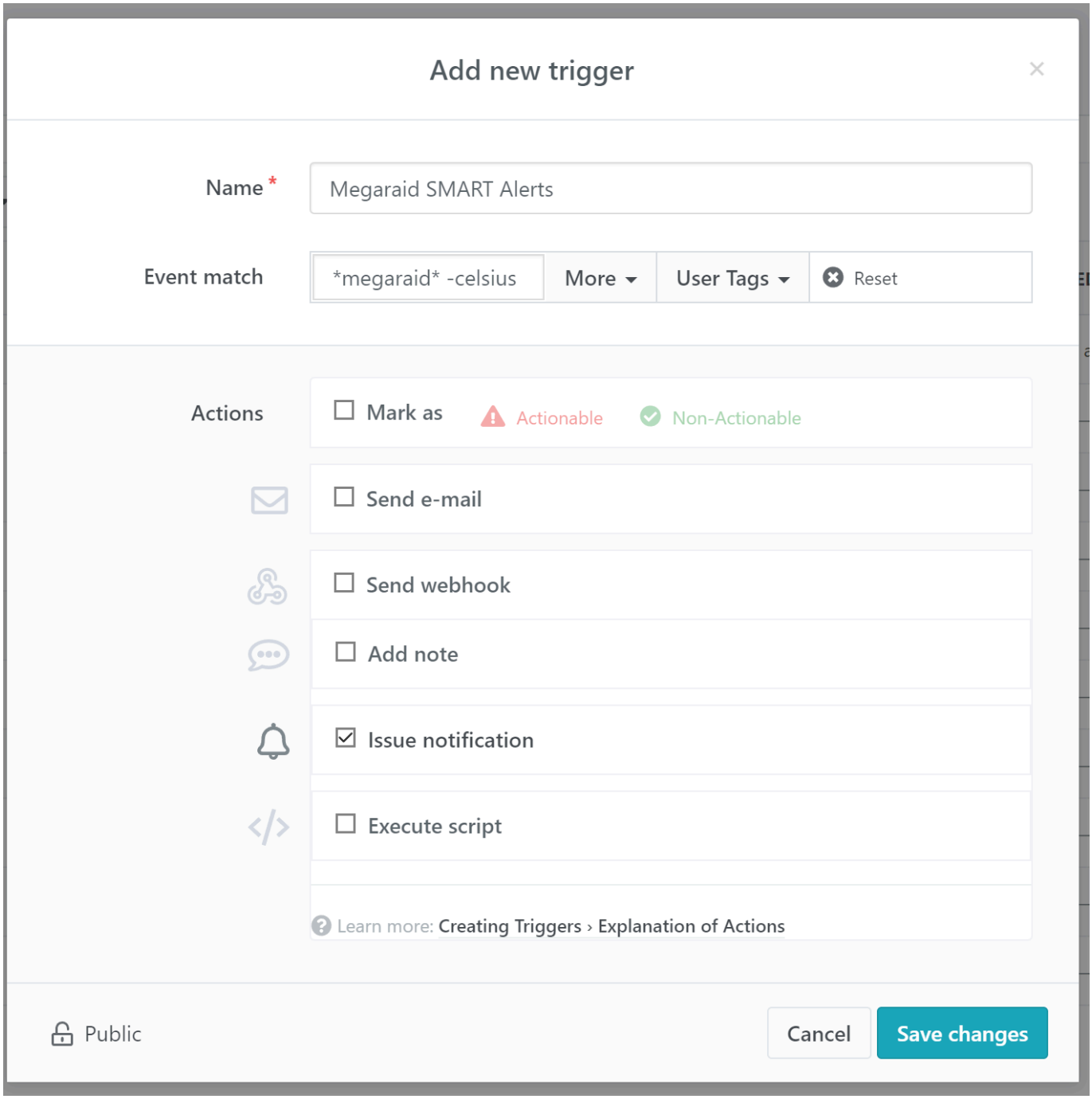
Add note

When an event occurs, other users may need to be given more information to reduce duplication of effort.



Issue Notification

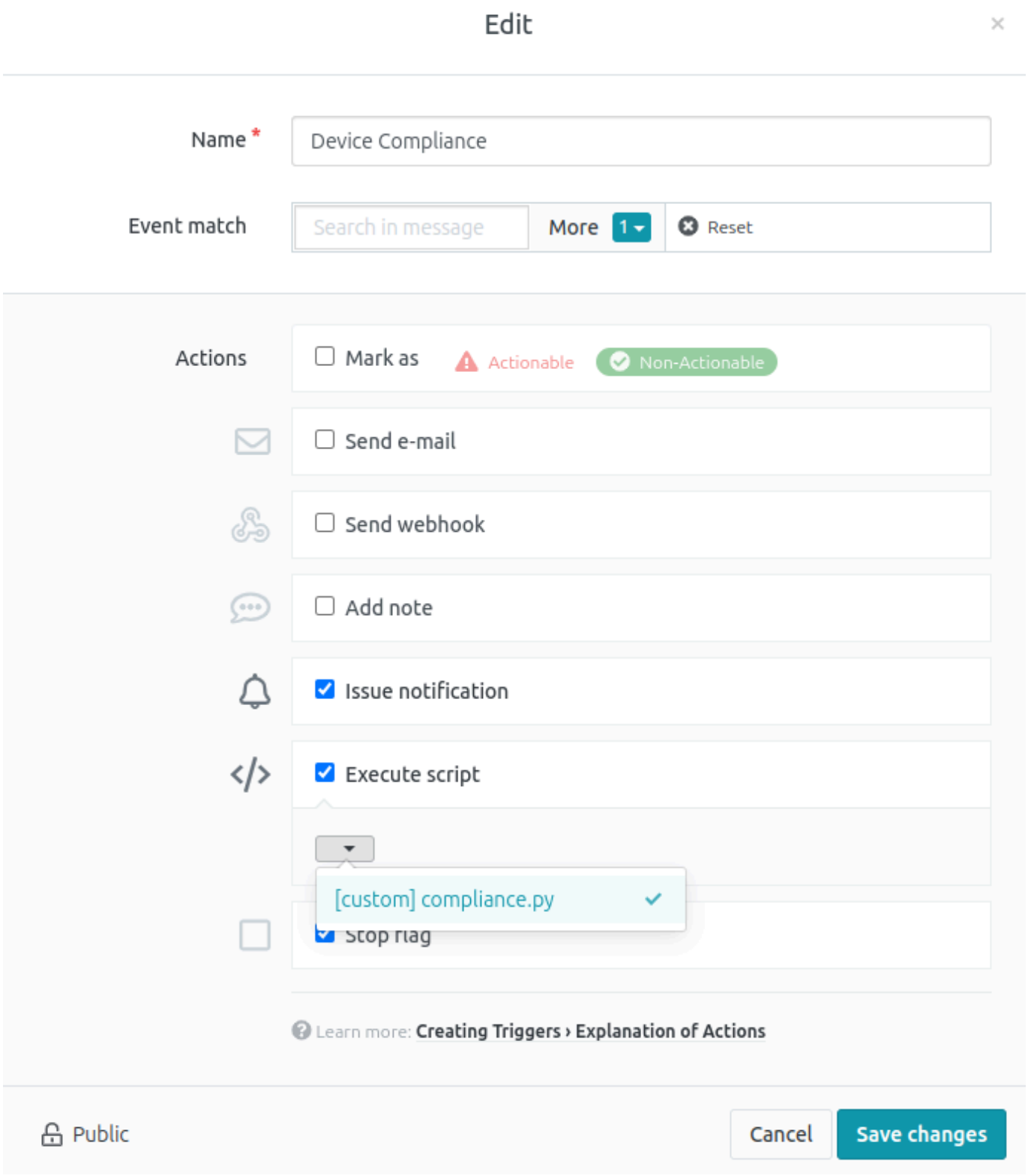
Selecting this option will produce a notification that will increment in the page header, and show up on the notifications page.



From the notifications page, users can Search, View, Edit, and Delete notifications. More information on this can be found in the Notifications section of the documentation.

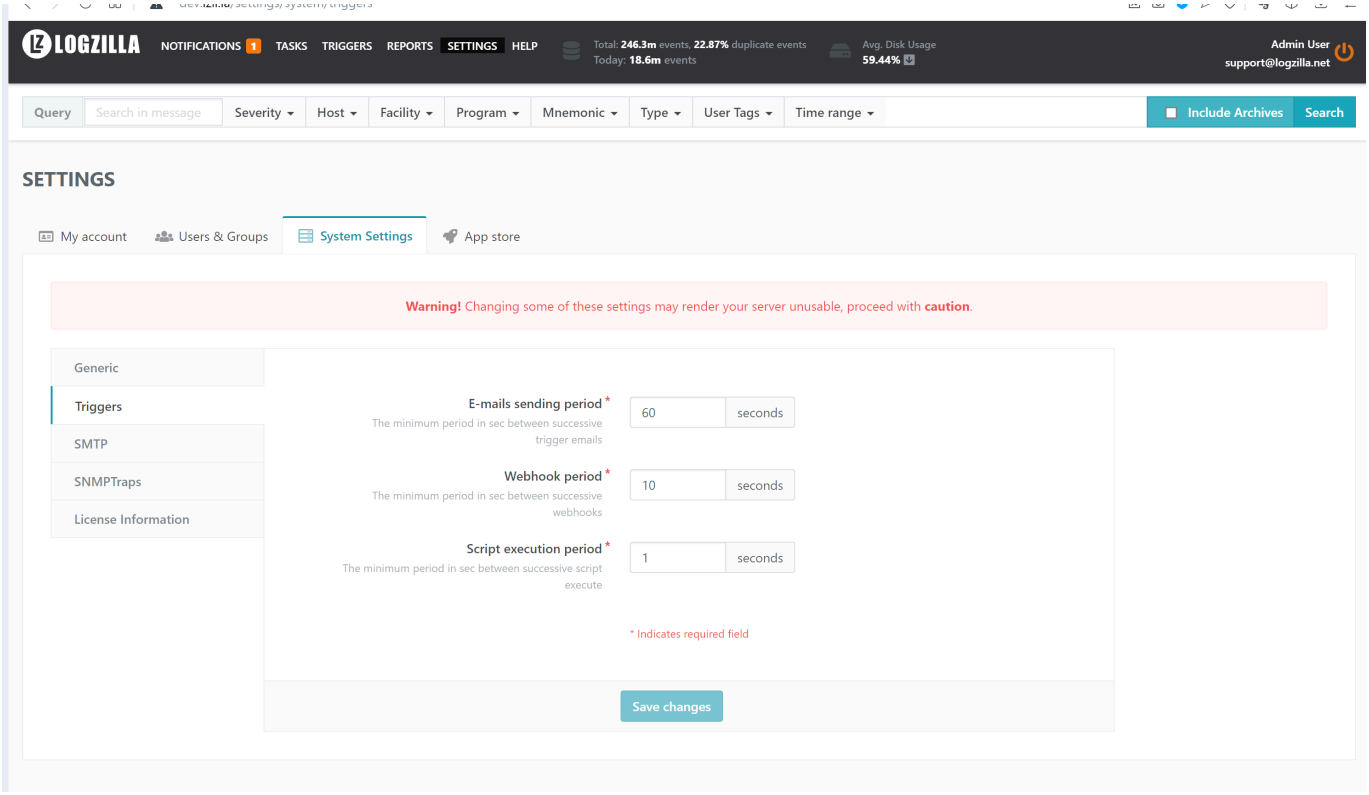
Execute Script

This option lets users write and execute their own scripts and trigger them whenever an event occurs. Just enter the name of the script to run in the box, and it will run whenever the event recurs. The Trigger Scripts section of the documentation provides more information on this feature.



Trigger Settings

Default Trigger settings can be changed in the Setting menu under System Settings, then Triggers.


LOGZILLA
NOTIFICATIONS 1
TASKS
TRIGGERS
REPORTS
SETTINGS
HELP
Total: 246.3m events, 22.87% duplicate events
Today: 18.6m events
Avg. Disk Usage
59.44%
Admin User
support@logzilla.net
Query
Search in message
Severity
Host
Facility
Program
Mnemonic
Type
User Tags
Time range
Include Archives
Search
SETTINGS
My account
Users & Groups
System Settings
App store
Warning! Changing some of these settings may render your server unusable, proceed with caution.
Generic
Triggers
SMTP
SNMPTraps
License Information
E-mails sending period *
The minimum period in sec between successive trigger emails
60
seconds
Webhook period *
The minimum period in sec between successive webhooks
10
seconds
Script execution period *
The minimum period in sec between successive script execute
1
seconds
* Indicates required field
Save changes

Trigger Scripts

Execute Python, Perl, shell, or compiled scripts from LogZilla triggers using EVENT_ and TRIGGER_ environment variables to access full event context

Script Types

LogZilla can execute various types of scripts, including:

- Python
- Perl
- sh, bash, zsh, csh, etc.
- Compiled Executables

Script Environment

All triggers passed to a script contain the matched message information as environment variables. To manipulate any of the data, simply reference the corresponding environment variable.

The following list of variables is automatically passed into each script:

```
# EVENT_ID                      =   <integer>
# EVENT_SEVERITY                =   <integer>
# EVENT_FACILITY                =   <integer>
# EVENT_MESSAGE                 =   <string>
# EVENT_HOST                    =   <string>
# EVENT_PROGRAM                 =   <string>
# EVENT_CISCO_MNEMONIC          =   <string>
# EVENT_USER_TAGS               =   <string>
# EVENT_STATUS                  =   <integer>
# EVENT_FIRST_OCCURRENCE        =   <float>
# EVENT_LAST_OCCURRENCE         =   <float>
# EVENT_COUNTER                 =   <integer>
# TRIGGER_ID                    =   <integer>
# TRIGGER_AUTHOR                =   <string>
# TRIGGER_AUTHOR_EMAIL          =   <string>
# TRIGGER_HITS_COUNT            =   <integer>
```

Parsing User Tags

The `EVENT_USER_TAGS` environment variable contains all user tags as a single comma-separated string in the format:

`key1=value1,key2=value2,key3=value3`

For example, if an event has user tags for `Username`, `SrcIP`, and `Auth Success`, the environment variable would contain:

```
EVENT_USER_TAGS="Username=admin,SrcIP=192.168.1.100,Auth Success=false"
```

Bash Example: Parsing User Tags

```
#!/bin/bash

# Parse user tags from EVENT_USER_TAGS environment variable
parse_user_tags() {
    local tags="$1"
    local IFS=','
    for pair in $tags; do
        local key="${pair%%=*}"
        local value="${pair#*=}"
        case "$key" in
            Username) USERNAME="$value" ;;
            SrcIP) SRC_IP="$value" ;;
            "Auth Success") AUTH_SUCCESS="$value" ;;
            # Add more tags as needed
        esac
    done
}

# Use the function
parse_user_tags "$EVENT_USER_TAGS"

# Now you can use the extracted values
echo "Username: $USERNAME"
echo "Source IP: $SRC_IP"
echo "Auth Success: $AUTH_SUCCESS"
```

Python Example: Parsing User Tags

```
#!/usr/bin/env python3
import os

# Parse user tags from EVENT_USER_TAGS environment variable
def parse_user_tags(tags_string):
    """Parse comma-separated key=value pairs into a dictionary."""
    tags = {}
    if tags_string:
        for pair in tags_string.split(','):
            if '=' in pair:
                key, value = pair.split('=', 1)
                tags[key] = value
    return tags

# Get environment variable
event_user_tags = os.environ.get('EVENT_USER_TAGS', '')

# Parse into dictionary
user_tags = parse_user_tags(event_user_tags)

# Access individual tags
username = user_tags.get('Username', '')
src_ip = user_tags.get('SrcIP', '')
auth_success = user_tags.get('Auth Success', '')

print(f"Username: {username}")
print(f"Source IP: {src_ip}")
print(f"Auth Success: {auth_success}")
```

Script Execution

Scripts may be executed directly or within dedicated Docker containers, depending on the script's requirements:

Simple Scripts

For simple scripts that do not require anything beyond what is available in a standard Linux install, simply place the script in the `/etc/logzilla/scripts` directory and select it in the UI when creating a trigger.

Here's an example of a simple shell script that logs the environment variables to the `logzilla.log`:

Create a `test.sh` file in `/etc/logzilla/scripts/`:

```
cat << EOF > /etc/logzilla/scripts/test.sh
#!/bin/bash
# Print all environment variables matching '^EVENT_' to the log
echo "Test script env vars" >> /var/log/logzilla/logzilla.log
env | grep '^EVENT_' >> /var/log/logzilla/logzilla.log
EOF
```

Make sure the script is executable:

```
chmod 755 /etc/logzilla/scripts/test.sh
```

Reload script-server:

```
logzilla restart -c scriptserver
```

Once the script is in place and executable, users can select it from the LogZilla UI when creating a trigger.

Custom Scripts

For scripts that require additional libraries or programs, such as Python packages, users may use their own Docker image containing all required modules.

Working Example: Custom Docker Container

In this example, a container will be created that brings up an interface on a Cisco device after it is shut down, then send a notification to Slack. The script uses Python and Netmiko to SSH into the device and apply the necessary configuration changes.

Note: All of the files below are also available on the GitHub Repo

Prepare custom image

Create a work directory used for Dockerfile and scripts

Python Script

NOTE: The following sample code is user-contributed and should be reviewed prior to using it verbatim in production.

- Download or create `compliance.py` using the example from the GitHub repo
- make the script executable

Yaml and Slack Key

Create a `compliance.yaml` file and update the Slack webhook key. Edit the YAML configuration to fit the environment by updating the following variables:

```
# Cisco credentials
ciscoUsername: "cisco"
ciscoPassword: "cisco"

# Slack settings
# Replace the value below with your actual post URL
posturl: "https://hooks.slack.com/services/XXXXXXXXX/XXXXXXXXX/XXXXXXXXXXXXXXXXXXXXXXXX"
default_channel: "#demo"
slack_user: "logzilla-bot"

# Logging and debug settings
log_file: "/var/log/logzilla/logzilla.log"

# Change to 0 in production:
debug_level: 2  # 0, 1, or 2

bring_interface_up: true

# Execution timeout for device connection and Slack:
timeout: 10
```

Dockerfile

Create a new file named `Dockerfile` with the following content:

```
# Use a logzilla script-server base image
FROM logzilla/script-server:latest

# Copy the requirements.txt file to the container
COPY requirements.txt /tmp/requirements.txt

# Install Python dependencies
RUN pip install -r /tmp/requirements.txt \
    --no-cache-dir --break-system-packages --root-user-action=ignore

# Copy script content to the container
RUN mkdir -p /scripts
COPY compliance.py /scripts
COPY compliance.yaml /scripts
RUN chmod +x /scripts/compliance.py
```

Requirements.txt

Create a `requirements.txt` file with the following content:

```
netmiko
paramiko
pyyaml
requests
```

Docker compose file

Create a `compose.yaml` file with the following content:

```
services:
  api:
    build:
      context: .
    container_name: compliance-script-server
    environment:
      SCRIPTS_ENABLED: "1"
      SCRIPTS_DIR: /scripts
      SCRIPTS_LOGS_DIR: /var/log/script-logs
    volumes:
        - logs:/var/log/script-logs/
    networks:
      - lz_network
volumes:
  logs:

networks:
  lz_network:
    name: lz_main
    external: true
```

Directory Contents

The work directory should contain the following files:

- Dockerfile
- compliance.py
- compliance.yaml
- compose.yaml
- requirements.txt

Run custom script container using docker compose

```
docker compose up --build -d
```

Check containers is running

```
# docker ps -a -f name=compliance-script-server
CONTAINER ID   IMAGE                              COMMAND                  CREATED
STATUS         PORTS     NAMES
e55547cfb505   custom-trigger-cisco-compliance   "fastapi run /usr/li..."   7 seconds ago
Up 7 seconds             compliance-script-server
```

Register custom script container

Create or edit `/etc/logzilla/settings/script_server.yaml`:

```
---
SERVERS:
  - name: custom
    url: http://compliance-script-server:8000/scripts
```

Reload LogZilla settings:

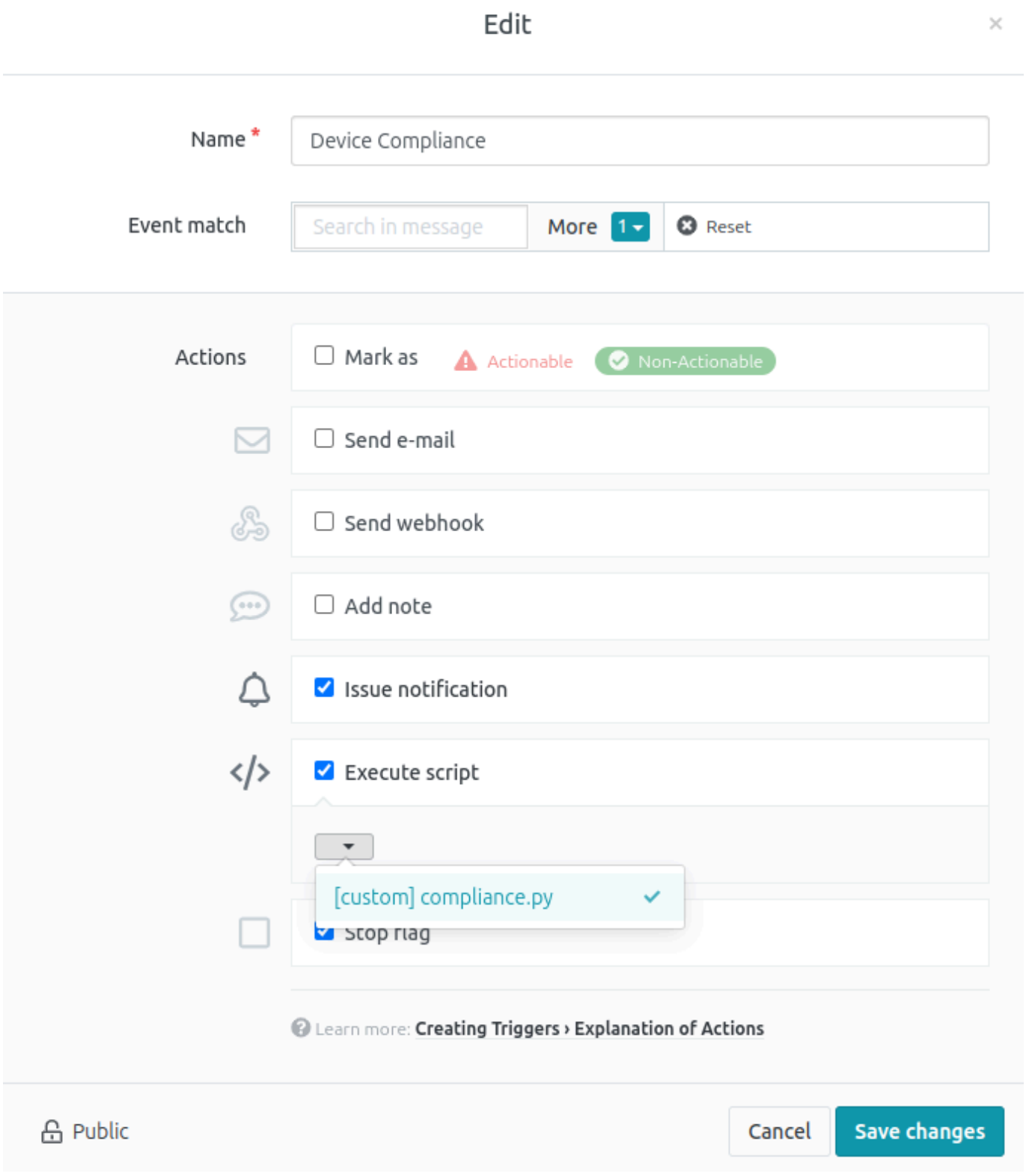
```
logzilla settings reload script_server
```

LogZilla UI

Log into the LogZilla Web Interface and:

- Create a new trigger from the trigger menu.
- Select the `execute script` option.
- From the dropdown menu, select `[custom] compliance.py`.

Any patterns matching this trigger will now execute the script.


Edit
Name *
Device Compliance
Event match
Search in message
More 1
Reset
Actions
Mark as
Actionable
Non-Actionable
Send e-mail
Send webhook
Add note
Issue notification
Execute script
[custom] compliance.py
Stop flag
Learn more: Creating Triggers › Explanation of Actions
Public
Cancel
Save changes

CHAPTER 06

Event Correlation

LogZilla Event Correlation combines trigger-based stateless matching with Simple Event Correlator for stateful multi-event pattern detection across time windows

© LogZilla Corporation · logzilla.ai

Event Correlation in LogZilla

Event correlation transforms individual log events into meaningful insights by identifying patterns, relationships, and anomalies across multiple events over time. LogZilla provides both simple trigger-based correlation and advanced stateful correlation through SEC (Simple Event Correlator).

Correlation Methods

Stateless Correlation (Triggers)

LogZilla Triggers provide immediate, single-event correlation:

- Real-time event matching against defined filters
- Instant action execution when conditions are met
- Simple alerting and notification scenarios
- Managed through the web interface or API

Stateful Correlation (SEC)

Simple Event Correlator enables complex, multi-event correlation:

- Maintains state across events over time
- Supports event pairs, thresholds, and time windows
- Handles scenarios requiring memory of past events
- Processes events through the Script Server architecture

Common Correlation Scenarios

Network Infrastructure

- **Interface flapping**: Detect rapid link up/down cycles
- **Device reload monitoring**: Correlate planned vs. unplanned restarts
- **Service dependency tracking**: Monitor cascading service failures
- **Security incident detection**: Identify attack patterns across devices

System Monitoring

- **Threshold-based alerting**: Count events within time windows
- **Service availability**: Track service start/stop sequences
- **Performance degradation**: Correlate resource exhaustion indicators
- **Capacity planning**: Identify usage pattern trends

Security Analysis

- **Brute force detection**: Count failed login attempts
- **Privilege escalation**: Correlate authentication and authorization events
- **Data exfiltration**: Monitor unusual file access patterns
- **Audit trail integrity**: Detect log tampering attempts

Windows Environment

- **Event ID correlation**: Leverage Windows Event ID patterns
- **Security event monitoring**: Track critical Windows security events
- **Service monitoring**: Correlate Windows service state changes
- **System integrity**: Monitor audit log and system file changes

Getting Started

Basic Correlation

Identify patterns: Determine which events need correlation

Choose method: Select triggers for simple cases, SEC for complex scenarios

Create rules: Define correlation logic and actions

Test thoroughly: Verify rules work with sample events

Monitor performance: Ensure correlation processing scales appropriately

Advanced Correlation

Design SEC instances: Organize rules by function or system type

Configure forwarders: Route specific events to appropriate SEC instances

Implement time windows: Define appropriate correlation timeframes

Set up alerting: Configure actions for correlation matches

Monitor and tune: Adjust rules based on operational experience

Architecture Overview

Event correlation in LogZilla follows this processing flow:

Event ingestion: Syslog-ng receives and forwards events

Parsing and enrichment: Events are normalized and tagged

Trigger evaluation: Stateless triggers match individual events

SEC processing: Complex correlation via Script Server and SEC instances

Action execution: Correlated events trigger alerts and responses

Best Practices

Rule Design

- Start with simple correlation scenarios
- Use specific patterns to avoid false positives
- Set appropriate time windows for event relationships
- Test rules thoroughly before production deployment

Performance Considerations

- Filter events before sending to SEC instances
- Use multiple SEC instances for different correlation types
- Monitor SEC process resource usage
- Implement correlation rule lifecycle management

Operational Management

- Document correlation rules and their purposes
- Implement version control for SEC rule files
- Monitor correlation effectiveness and tune as needed
- Plan for correlation rule maintenance and updates

Additional Capabilities

LogZilla's event correlation extends beyond basic pattern matching to provide enterprise-grade operational intelligence:

Business Process Monitoring

- **Revenue pipeline tracking**: Monitor e-commerce workflows end-to-end
- **Service level correlation**: Correlate technical metrics with business KPIs
- **Compliance automation**: Automate SOX, PCI, and regulatory monitoring

Advanced Security Detection

- **APT correlation**: Detect multi-stage advanced persistent threats
- **Lateral movement tracking**: Identify sophisticated attack progressions
- **Behavioral analysis**: Correlate user and system behavior patterns

Infrastructure Intelligence

- **Cascading failure prediction**: Prevent widespread outages through early detection
- **Capacity planning automation**: Predict resource exhaustion before impact
- **Multi-cloud correlation**: Unified monitoring across cloud providers

DevOps Integration

- **CI/CD pipeline correlation**: Track deployment success across environments
- **Performance regression detection**: Correlate code changes with performance impact
- **Automated remediation**: Trigger corrective actions based on correlation results

Related Topics

- Advanced Event Correlation Walkthrough
- SOAR Security Orchestration
- Creating Triggers
- Data Transforms
- Windows Event Forwarding
- Syslog Settings

Event Correlation Overview

Enable LogZilla Event Correlation to detect multi-event patterns across time using stateless triggers and stateful SEC rules with forwarder routing

Event Correlation Overview

Event correlation identifies patterns, relationships, and anomalies across multiple log events over time. LogZilla provides both stateless and stateful correlation methods to detect complex scenarios that single-event analysis cannot identify.

Before you begin

- Enable EC (required):

```
sudo logzilla settings update SEC_ENABLED=true
```

- Enable Forwarder (required):

```
sudo logzilla settings update FORWARDER_ENABLED=true
```

- File locations:
  - EC rules: `/etc/logzilla/sec/<instance>/rules/`
  - Forwarder configs: `/etc/logzilla/forwarder.d/`
- Apply changes:
  - Forwarder configs: `logzilla settings reload forwardermodule`
  - EC rule changes: `logzilla settings reload sec`
  - Both together: `logzilla settings reload sec forwardermodule`
- Validate forwarder configuration:

```
logzilla forwarder print
```

  This command displays the parsed forwarder configuration and reports any syntax errors.

- Verify EC container:

```
sudo docker ps | grep lz_sec
```

Container Logging

- EC rules use environment variables `$SYSLOG_HOSTNAME` and `$SYSLOG_BSD_TCP_PORT` to target the LogZilla syslog receiver
- The `-T` flag enables TCP transport; omit for UDP if TCP is not supported
- EC runs in a container with limited utilities; use inline Perl for complex operations rather than external commands
- DNS resolution occurs within the EC container; ensure proper DNS configuration for hostname lookups

Core Correlation Concepts

- **Event triggers**: Conditions that initiate correlation analysis
- **Event filters**: Criteria that determine which events to correlate
- **Event pairing**: Associations between multiple related events
- **Time windows**: Duration constraints for event relationships
- **Thresholds**: Frequency or count-based correlation triggers
- **Suppression**: Logic to reduce noise and prevent alert storms

LogZilla Correlation Architecture

LogZilla implements a two-tier correlation system:

Stateless Correlation (Triggers)

LogZilla Triggers provide immediate, single-event correlation:

- Events are matched against trigger filters in real-time
- Actions execute immediately when conditions are met
- Suitable for alerting and immediate responses
- Managed through the LogZilla web interface or API

Stateful Correlation

Event Correlation enables advanced, multi-event correlation:

- Maintains state across multiple events over time
- Supports complex patterns like event pairs, thresholds, and sequences
- Handles scenarios requiring memory of past events
- Integrated via the Script Server for scalable processing

Event Processing Flow

Event Ingestion: Incoming events (syslog, snmp, httpx, webhooks, etc.) are received and forwarded to ParserModule

Parsing: Events are normalized and enriched with user tags via Parser Rules (YAML), Lua scripts, or Apps.

Trigger Evaluation: QueryModule matches events against active triggers

Action Execution: Matched triggers execute configured actions

Event Correlation Processing: Complex correlation rules process events via Script Server

Response: Correlated events inject a new event back into LogZilla for output triggering (Email, Webhook, Scripts, etc.)

Event Correlation Process Flow

Scalable Correlation Design

The architecture addresses traditional EC limitations:

- **Selective forwarding**: Only relevant events are sent to EC for processing
- **Distributed processing**: Script Server manages multiple EC instances
- **Event filtering**: Triggers pre-filter events before correlation analysis
- **Resource optimization**: Correlation load is distributed and controlled

Use Cases

Network Infrastructure

- **Interface flapping**: Detect rapid up/down state changes
- **Device failures**: Correlate multiple error conditions
- **Security incidents**: Identify attack patterns across devices

System Monitoring

- **Service failures**: Detect cascading service dependencies
- **Performance degradation**: Correlate resource exhaustion indicators
- **Unauthorized access**: Track suspicious login patterns

Application Monitoring

- **Error clustering**: Group related application errors

- **Transaction failures**: Correlate multi-step process failures
- **Capacity planning**: Identify usage pattern trends

Getting Started

Event correlation in LogZilla involves:

Configure triggers: Create stateless correlation rules for immediate responses

Define EC rules: Develop stateful correlation patterns for complex scenarios

Test correlation: Verify rules work with sample events

Monitor performance: Ensure correlation processing scales appropriately

LogZilla Pre-Processing Performance Advantages

LogZilla's architecture provides significant performance benefits for event correlation:

Traditional EC Approach Limitations

- **Complex regex parsing**: EC must parse raw log messages with CPU-intensive regex
- **Duplicate parsing logic**: Same parsing repeated across multiple EC rules
- **Limited context**: Only raw message text available for correlation
- **Performance bottlenecks**: EC spends most CPU time on text parsing, not correlation
- **Alert fatigue**: Thousands of individual event alerts daily
- **Missed attack patterns**: Single events don't show attack progression
- **Manual correlation**: Analysts must manually connect related events

LogZilla + EC Architecture Benefits

- **Pre-extracted fields**: LogZilla's Lua engine extracts fields once, EC accesses via environment variables
- **5x-10x performance improvement**: EC focuses on correlation logic instead of parsing
- **Rich context**: EC receives structured data like `EVENT_USER_TAGS_PAM_USER_TRACKING`, `EVENT_USER_TAGS_SRCIP`
- **Centralized parsing**: One optimized parsing engine serves all LogZilla features
- **Simplified rules**: EC rules use simple environment variable checks instead of complex regex
- **Attack pattern detection**: Multi-stage attack identification
- **Intelligent response**: Context-aware automation based on external data
- **Reduced noise**: Correlation replaces individual event alerts
- **Separation of concerns**: EC focuses on correlation, triggers handle actions

Example Performance Comparison

Traditional EC (inefficient):

```
pattern=.*sshd.*Failed password for (\w+) from ([\d.]+) port (\d+)
```

LogZilla Pre-Processing (efficient):

```
pattern=AUTH_FAILED
# Access: $ENV{EVENT_USER_TAGS_PAM_USER_TRACKING}, $ENV{EVENT_USER_TAGS_SRCIP}
```

Script Execution Context

EC vs LogZilla Trigger Scripts

Distinction: Scripts called by EC and LogZilla Triggers receive data in different ways:

EC `shellcmd` Scripts

- **Data Access**: Command-line arguments (`$1` , `$2` , `$3` , etc.)
- **Usage**: `shellcmd (/path/to/script.sh "%variable1" "%variable2")`
- **Script**: `USERNAME="$1"; SRC_IP="$2"`

LogZilla Trigger Scripts

- **Data Access**: Environment variables (`$EVENT_*`)
- **Usage**: `exec_script: true` in trigger configuration
- **Script**: `USERNAME="$EVENT_USER_TAGS_PAM_USER_TRACKING"`

Important: Do not mix these execution contexts. EC scripts cannot access `$EVENT_*` environment variables.

Quick reference

- You must use a LogZilla forwarder configuration to send events to SEC.
- The forwarder can use a rewrite rule to alter the message contents to be sent to SEC, for example to send user tags.
- The forwarder should use an appropriate `match` configuration to limit the rate at which events are sent to SEC, to only what is necessary.
- SEC `shellcmd` scripts: receive values as positional arguments (`$1` , `$2` , ...). Pass required variables explicitly from the rule, for example: `shellcmd (/path/to/script.sh "%username" "%src_ip")` .

Prerequisites

Event Correlation implementation requires:

- LogZilla server with administrative access
- EC feature enabled in LogZilla settings

- Basic understanding of regular expressions
- SSH access to the LogZilla server

Enabling EC

Check EC Status

Verify if EC is enabled on the LogZilla server:

```
sudo logzilla settings list | grep SEC_ENABLED
```

Enable EC

If EC is not enabled, activate it:

```
sudo logzilla settings update SEC_ENABLED=true
```

Once EC is enabled, the container starts automatically. This process takes approximately one minute.

Enable Event Forwarding

Event Correlation relies on the Forwarder module to send selected events to SEC instances.

Enable the forwarder module:

```
sudo logzilla settings update FORWARDER_ENABLED=true
```

For detailed forwarder configuration and examples, see:

- Forwarder and Deduplication Overview
- Downstream Syslog Receivers
- Forwarding to Event Correlation

Verify EC Container

Verify the container has started:

```
sudo docker ps | grep lz_sec
```

The output should show the EC container is running:

```
lz_sec    logzilla/script-server:v6.37.4   "/usr/bin/run_script..."   About a minute ago
Up About a minute
```

EC Configuration Management

Rule Loading

EC automatically loads new rules from `/etc/logzilla/sec/` when started. Verify EC is running:

```
sudo docker ps | grep lz_sec
```

Verify Rule Processing

Check EC logs to confirm rules are loaded:

```
sudo docker logs lz_sec | tail -20
```

Look for messages indicating successful rule loading.

Apply Configuration Changes

After making changes to EC rules, reload the SEC configuration:

```
sudo logzilla settings reload sec
```

If a reload does not pick up changes (rare), then restart the SEC container:

```
sudo logzilla restart -c sec
```

Troubleshooting

EC Not Processing Events

Check EC container status:

```
 sudo docker ps | grep lz_sec
```

Review EC logs:

```
sudo docker logs lz_sec
```

Verify rule syntax:

```
sudo docker exec -it lz_sec sec --testonly --conf=/etc/logzilla/sec/rule-file.sec
```

Events Not Correlating

Verify test events are reaching LogZilla:

- Check Events page for original test events
- Ensure expected program names appear

Check pattern matching:

- Ensure event format matches EC rule patterns
- Use exact string matching when possible for better performance

No Correlation Output

Verify EC can send events back to LogZilla:

```
sudo docker exec -it lz_sec printenv | grep SYSLOG
```

Test logger connectivity from EC container:

```
sudo docker exec -it lz_sec /usr/bin/logger -T -n $SYSLOG_HOSTNAME -P
$SYSLOG_BSD_TCP_PORT -t EC-TEST "Test message"
```

Technical Notes

- EC rules use environment variables `$SYSLOG_HOSTNAME` and `$SYSLOG_BSD_TCP_PORT` to target the LogZilla syslog receiver
- The `-T` flag enables TCP transport; omit for UDP if TCP is not supported
- EC runs in a container with limited utilities; use inline Perl for complex operations rather than external commands
- DNS resolution occurs within the EC container; ensure proper DNS configuration for hostname lookups

Related Topics

- Event Correlation Rule Types
- Windows Event Correlation

- Advanced Event Correlation Walkthrough
- Creating Triggers

Event Correlation Types

Reference for LogZilla Event Correlation rule types including Single, Pair, EventGroup, and SingleWithThreshold, with SubStr and RegExp pattern matching

Event Correlation Rule Types

The following Event Correlation (EC) rule types are used in LogZilla solution examples. Each entry lists purpose, required keys, and a verified example from existing documentation.

Pattern type (ptype)

- `SubStr` : substring matching (preferred for performance)
- `RegExp` : regular expression matching when substring is not sufficient

---

Single

Guidance: Single-event conditions should use LogZilla Triggers for performance and simplicity. Use an EC Single rule only when it is a required step inside a larger, stateful correlation chain. See Creating Triggers.

Match a single event that meets the condition.

- Keys: `ptype` , `pattern` , optional `context` , `desc` , `action`

Example (Windows successful login after brute force): (This example requires the following forwarder configuration:)

```
# /etc/logzilla/forwarder.d/windows-security.yaml
window_size: 0
type: sec
sec_name: windows-security
match:
  - field: MSWin EventID
    value: ["4624", "4625", "4732", "1102"]
rules:
- rewrite:
    message: >-
      [WINDOWS_AUTH] host=${:host} MSWin_EventID=${:MSWin EventID} SrcIP=${:SrcIP}
      Subject_User_Name=${:Subject User Name} Target_Domain_Name=${:Target Domain Name}
      Target_Group_Name=${:Target Group Name} Target_User_Name=${:Target User Name}
      Username=${:Username} User_Domain_Name=${:User Domain Name} $MESSAGE
```

```
# Detect successful login after brute force (correlate on user)
type=Single
ptype=RegExp
pattern=(?s)\[WINDOWS_AUTH\].*?host=([^\s]+).*?MSWin_EventID=4624\b.*?Username=([^\s]+)
context=BRUTE_FORCE_ACTIVE_\L$2\E
desc=Successful login after brute force user=$2 host=$1
action=write /var/log/logzilla/sec/windows-security-messages.log \
         "WINDOWS_COMPROMISE_SUSPECTED %t user=$2 host=$1 (4624 after brute force)"; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
         "WINDOWS_COMPROMISE_SUSPECTED user=\"$2\" host=\"$1\"
correlation=\"4624_after_bruteforce\"")
```

SingleWithThreshold

Count matching events within a time window and fire when the threshold is reached.

- Keys: `ptype`, `pattern`, optional `context`, `desc`, `action`, `thresh`, `window`

Example (Windows brute force): (This example requires the forwarder configuration from the previous example.)

```
# Count failed auth attempts (threshold by user, case-insensitive)
type=SingleWithThreshold
ptype=RegExp
pattern=(?s)\[WINDOWS_AUTH\].*?host=([^\s]+).*?MSWin_EventID=4625\b.*?SrcIP=([^\s]+).*?
Username=([^\s]+)
context=\L$3\E
desc=Windows failed logon burst (4625) user=$3 src=$2 host=$1
action=create BRUTE_FORCE_ACTIVE_\L$3\E 1800; \
       write /var/log/logzilla/sec/windows-security-messages.log \
         "BRUTE_FORCE_DETECTED %t user=\L$3\E src=$2 host=$1 attempts=$thresh"; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.alert \
         "BRUTE_FORCE_DETECTED user=\"\L$3\E\" src_ip=\"$2\" target=\"$1\"
attempts=\"$thresh\"")
thresh=10
window=300
```

Pair

Detect a two-step sequence: an initial event followed by a second event.

- Keys: `ptype`, `pattern`, `context`, `desc`, `action`, plus `ptype2`, `pattern2`, `context2`, `desc2`, `action2`

Example (BGP down then up with outage duration): (This example requires the following forwarder configuration:)

```
# /etc/logzilla/forwarder.d/cisco-network-health.yaml
window_size: 0
type: sec
sec_name: cisco-network-health
match:
  - field: cisco_mnemonic
    value:
      - "BGP-5-ADJCHANGE"
      - "BGP-4-MAXPFX"
      - "BGP-3-NOTIFICATION"
      - "OSPF-5-ADJCHG"
rules:
- rewrite:
    message: >-
      [CISCO_NETWORK_HEALTH]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      srcip="${:SrcIP}"
      dstip="${:DstIP}"
      srcport="${:SrcPort}"
      dstport="${:DstPort}"
      protocol="${:protocol}"
      $MESSAGE
```

```
# BGP adjacency outage tracker (host + neighbor)

type=Pair

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Down\b
context=BGP_ADJ_$1_$2
desc=BGP Neighbor $2 Down on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       eval %down_time ( time() ); \
       shellcmd (/usr/bin/host %neighbor_ip > /tmp/bgp-lookup-%neighbor_ip 2>/dev/null); \
       eval %hostname ( readfile("/tmp/bgp-lookup-%neighbor_ip") ); \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_ADJ_DOWN %t router=%router_host neighbor=%neighbor_ip hostname=%hostname";
\
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_ADJ_DOWN router=\"%router_host\" neighbor=\"%neighbor_ip\"
hostname=\"%hostname\"")

# ---- END (same host + neighbor, Up) ----
ptype2=RegExp
pattern2=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Up\b
context2=BGP_ADJ_$1_$2
desc2=BGP Neighbor $2 Up on $1
action2=eval %up_time ( time() ); \
        eval %outage_duration ( %up_time - %down_time ); \
        write /var/log/logzilla/sec/network-health.log \
          "BGP_ADJ_UP %t router=%router_host neighbor=%neighbor_ip
downtime=%outage_duration hostname=%hostname"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH \
          "BGP_OUTAGE_RESOLVED router=\"%router_host\" neighbor=\"%neighbor_ip\"
downtime=\"%outage_duration\" hostname=\"%hostname\""); \
        delete /tmp/bgp-lookup-%neighbor_ip

# Pairing window (how long to wait for the Up after Down)
window=86400
```

PairWithWindow

Like `Pair`, but requires the second event to arrive within a specified window.

- Keys: same as `Pair`, plus `window`

Example (BGP adjacency down/up within 4 hours): (This rule requires the forwarder configuration from the previous example.)

```
# BGP adjacency flapping detector (down/up within 4 hours)

type=PairWithWindow

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+.*\bneighbor\s+\S+\s+Down\b
context=BGP_FLAP_$1_$2
desc=BGP neighbor outage detected on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       eval %outage_start ( time() )

# ---- END (same host + neighbor, Up within 4 hours) ----
ptype2=RegExp
pattern2=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+.*\bneighbor\s+\S+\s+Up\b
context2=BGP_FLAP_$1_$2
desc2=BGP neighbor %neighbor_ip flapped on %router_host
action2=eval %outage_duration ( time() - %outage_start ); \
        write /var/log/logzilla/sec/network-health.log \
          "BGP_FLAP_4H %t router=%router_host neighbor=%neighbor_ip
duration=%outage_duration window=14400s"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH -p local0.warning \
          "BGP_FLAP_4H host=\"%router_host\" neighbor=\"%neighbor_ip\"
duration=\"%outage_duration\"")

# Pairing window (4 hours = 14400 seconds)
window=14400
```

SingleWithScript

Run an external script to decide or compute a condition.

- Keys: `ptype`, `pattern`, `script`, `desc`, `action`, optional `action2`

Example (Memory trend analysis): (This example requires the following forwarder configuration):

```
# /etc/logzilla/forwarder.d/cisco-device-health.yaml
window_size: 0
type: sec
sec_name: cisco-device-health
match:
  - field: cisco_mnemonic
    value: ["SYS-1-CPURISINGTHRESHOLD", "SYS-4-FREEMEMLOW", "LINK-3-UPDOWN"]
rules:
- rewrite:
    message: >-
      [CISCO_DEVICE_HEALTH]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      $MESSAGE
```

SEC Rule:

```
# Analyze memory trend in external script
# Cisco memory threshold tracking

type=SingleWithScript

ptype=RegExp
pattern=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="SYS-4-FREEMEMLOW"\s+host="([^"]+)"\s+

script=/usr/local/bin/cisco-memory-threshold-tracking.sh $1
desc=Memory alert counter for $1
action=eval %hostname ( "$1" ); \
       write /var/log/logzilla/sec/device-health.log \
         "MEMORY_ALERT %t host=%hostname count=$2"
```

EventGroup3

Correlate three related conditions (with thresholds) to indicate a larger issue.

- Keys: `ptype`, `pattern`, `context`, `thresh` for each group segment

Example (Network device health): (This example requires the forwarder configuration from the previous example.)

```
# Cisco Device Health (CPU-%¥80%, Mem low, Iface down)

type=EventGroup3
# Group by device host (keep $1 as host in all patterns)
context=HOST_$1
window=300

# -------- CPU HIGH (>=80%) --------
# Matches %SYS-1-CPURISINGTHRESHOLD lines that mention 80-100%
ptype=RegExp
pattern=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="SYS-1-CPURISINGTHRESHOLD"\s+host="
([^"]+)"\b.*\b(8\d|9\d|100)%\b
thresh=3

# -------- MEMORY LOW (-%^high usage) --------
# Matches Cisco low-watermark memory events (no % necessarily present)
ptype2=RegExp
pattern2=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="SYS-4-FREEMEMLOW"\s+host="([^"]+)"\b.*
thresh2=2

# -------- INTERFACE DOWN --------
# Matches interface down events; capture interface name if present (not used in final
group alert)
ptype3=RegExp
pattern3=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="(?:LINK-3-UPDOWN|LINEPROTO-5-
UPDOWN)"\s+host="([^"]+)"\b.*?\bInterface\s+([^,]+),\s+changed state to down\b
thresh3=1

desc=Device health degradation on host=$1 (CPU>=80% x3, Mem low x2, IfDown x1 within 5m)
action=write /var/log/logzilla/sec/network-health.log \
         "DEVICE_HEALTH_ALERT %t host=$1 symptoms=cpu_ge80x3,mem_lowx2,ifdownx1"; \
       shellcmd (logger -t SEC-CRITICAL -p local0.crit \
         "DEVICE_HEALTH_ALERT host=\"$1\" correlation=\"cpu>=80x3,mem_lowx2,ifdownx1\"")
```

Notes

- Prefer `ptype=SubStr` and exact string matching when possible for better performance.
- Use `thresh` and `window` to control noise and define meaningful time bounds for correlation.
- When calling external scripts from EC rules (`shellcmd`), pass values as positional arguments and read them as `$1` , `$2` , etc., in the script.
- Prefer LogZilla Triggers for single-event logic whenever possible; reserve EC for multi-event/stateful correlation.

Bgp Network Monitoring Correlation

Track BGP adjacency changes, calculate outage duration, and detect chronic instability in Cisco routers using LogZilla Event Correlation rules

BGP Network Monitoring and Correlation

Purpose

BGP (Border Gateway Protocol) monitoring requires correlating adjacency changes, outage duration tracking, and network stability analysis. Pre- processing extracts BGP-specific fields, and SEC handles time-based correlation patterns that triggers do not cover.

Prerequisites

- Event Correlation must be enabled.

BGP Adjacency Outage Tracking

Required App: *Cisco IOS*

Forwarder Configuration for Cisco Network Health Rules

```
# /etc/logzilla/forwarder.d/cisco-network-health.yaml
window_size: 0
type: sec
sec_name: cisco-network-health
match:
  - field: cisco_mnemonic
    value:
      - "BGP-5-ADJCHANGE"
      - "BGP-4-MAXPFX"
      - "BGP-3-NOTIFICATION"
      - "OSPF-5-ADJCHG"
rules:
- rewrite:
    program: sec-cisco-network-health
    message: >-
      [CISCO_NETWORK_HEALTH]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      srcip="${:SrcIP}"
      dstip="${:DstIP}"
      srcport="${:SrcPort}"
      dstport="${:DstPort}"
      protocol="${:protocol}"
      $MESSAGE
```

SEC Rule: BGP Outage Duration Calculation

```
# BGP adjacency outage tracker (host + neighbor)

type=Pair

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Down\b
context=BGP_ADJ_$1_$2
desc=BGP Neighbor $2 Down on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       eval %down_time ( time() ); \
       shellcmd (/usr/bin/host %neighbor_ip > /tmp/bgp-lookup-%neighbor_ip 2>/dev/null); \
       eval %hostname ( readfile("/tmp/bgp-lookup-%neighbor_ip") ); \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_ADJ_DOWN %t router=%router_host neighbor=%neighbor_ip hostname=%hostname";
\
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_ADJ_DOWN router=\"%router_host\" neighbor=\"%neighbor_ip\"
hostname=\"%hostname\"")

# ---- END (same host + neighbor, Up) ----
ptype2=RegExp
pattern2=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Up\b
context2=BGP_ADJ_$1_$2
desc2=BGP Neighbor $2 Up on $1
action2=eval %up_time ( time() ); \
        eval %outage_duration ( %up_time - %down_time ); \
        write /var/log/logzilla/sec/network-health.log \
          "BGP_ADJ_UP %t router=%router_host neighbor=%neighbor_ip
downtime=%outage_duration hostname=%hostname"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH \
          "BGP_OUTAGE_RESOLVED router=\"%router_host\" neighbor=\"%neighbor_ip\"
downtime=\"%outage_duration\" hostname=\"%hostname\""); \
        delete /tmp/bgp-lookup-%neighbor_ip

# Pairing window (how long to wait for the Up after Down)
window=86400
```

LogZilla Trigger: BGP Outage Response

```
name: "BGP Outage Business Impact"
filter:
  - field: program
    op: eq
    value: sec-cisco-network-health
  - field: message
    op: "=~"
    value: "BGP_OUTAGE_RESOLVED"
actions:
  exec_script: true
  script_path: "/usr/local/bin/bgp-outage-analysis.sh"
  send_webhook: true
  send_webhook_template: |
    {
      "event_type": "bgp_outage_resolved",
      "host": "{{:host}}",
      "IP": "{{:SrcIP}}",
      "severity": "warning"
    }
```

Business Intelligence Script

For this script, the goal is to derive data from the log message that is not present in the message itself, in order to send that information to the script. This example uses the LogZilla **Event Enrichment** appstore app.

That app allows administrators to associate additional data with each log message, such as the store type and tunnel ID, based on the IP address of the device in question.

Detailed information about Event Enrichment is available in the **Event Enrichment** appstore entry, which is listed under *Settings*, *App store*, then *Add*. The *Event Enrichment* app icon displays the documentation for the app and how to use it.

For this script Event Enrichment uses **SrcIP** to look up the store name, store number, store type, and tunnel ID.

In `/etc/logzilla/apps/event_enrichment/config/config.yaml` add the following:

```
---
- name: Simple Host Lookup
  description: |
    Used to add store name, store number, store type, and tunnel ID.
    Referenced by SrcIP (the router address for that location).
  metadata_file: StoreInformation.yaml
  lookup_field: SrcIP
```

Next, in `/etc/logzilla/apps/event_enrichment/config/StoreInformation.yaml` add the following:

```
---
"246.219.157.165":
  Store Name: Store AAA
  Store Number: 1001
  Tunnel ID: 123
"107.122.210.185":
  Store Name: Store BBB
  Store Number: 1002
  Tunnel ID: 234
"CE5-G":
  Store Name: Store CDD
  Store Number: 1003
  Tunnel ID: 345
"PE4":
  Store Name: Store DDD
  Store Number: 1004
  Tunnel ID: 456
```

With the *Event Enrichment* app running using that information, the information will be available to the LogZilla forwarder, and thus to SEC, as user tags.

The forwarder configuration is as follows:

```
# /etc/logzilla/forwarder.d/cisco-network-health.yaml
window_size: 0
type: sec
sec_name: store-network-event
match:
  - field: cisco_mnemonic
    value:
      - "BGP-5-ADJCHANGE"
      - "BGP-4-MAXPFX"
      - "BGP-3-NOTIFICATION"
      - "OSPF-5-ADJCHG"
rules:
- rewrite:
    message: >-
      [STORE_NETWORK_EVENT]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      srcip="${:SrcIP}"
      dstip="${:DstIP}"
      srcport="${:SrcPort}"
      dstport="${:DstPort}"
      protocol="${:protocol}"
      store_name="${:Store Name}"
      store_number="${:Store Number}"
      tunnel_id="${:Tunnel ID}"
      $MESSAGE
```

The SEC rule would be as follows:

```
# ==========================================
# BGP adjacency outage tracker (host + neighbor)
# Uses Event Enrichment app to provide store metadata
# ==========================================

type=Pair

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[STORE_NETWORK_EVENT\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="([^"]*)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+store_name="([^"]*)"\s+store_number="([^"]*)"\s+tunnel_id="
([^"]*)"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Down\b
context=BGP_ADJ_$1_$6
desc=BGP Neighbor $6 Down on $1
action=eval %router_host ( "$1" ); \
       eval %store_ip ( "$2" ); \
       eval %store_name ( "$3" ); \
       eval %store_number ( "$4" ); \
       eval %tunnel_id ( "$5" ); \
       eval %neighbor_ip ( "$6" ); \
       eval %down_time ( time() ); \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_ADJ_DOWN %t router=%router_host neighbor=%neighbor_ip store_ip=%store_ip
store_name=%store_name store_number=%store_number tunnel_id=%tunnel_id"; \
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_ADJ_DOWN router=\"%router_host\" neighbor=\"%neighbor_ip\"
store_ip=\"%store_ip\" store_name=\"%store_name\" store_number=\"%store_number\"
tunnel_id=\"%tunnel_id\"")

# ---- END (same host + neighbor, Up) ----
ptype2=RegExp
pattern2=\[STORE_NETWORK_EVENT\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="([^"]*)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+store_name="([^"]*)"\s+store_number="([^"]*)"\s+tunnel_id="
([^"]*)"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Up\b
context2=BGP_ADJ_$1_$6
desc2=BGP Neighbor $6 Up on $1
action2=eval %up_time ( time() ); \
        eval %outage_duration ( %up_time - %down_time ); \
        write /var/log/logzilla/sec/network-health.log \
          "BGP_ADJ_UP %t router=%router_host neighbor=%neighbor_ip
downtime=%outage_duration store_name=%store_name store_number=%store_number
tunnel_id=%tunnel_id"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH \
          "BGP_OUTAGE_RESOLVED router=\"%router_host\" neighbor=\"%neighbor_ip\"
downtime=\"%outage_duration\" store_name=\"%store_name\" store_number=\"%store_number\"
tunnel_id=\"%tunnel_id\""); \
        shellcmd (/usr/local/bin/bgp-outage-analysis.sh "%outage_duration" "%store_name"
```

```
"%store_number" "%tunnel_id")

window=86400
```

```
#!/bin/bash
# /usr/local/bin/bgp-outage-analysis.sh

DOWNTIME=$1
STORE_NAME=$2
STORE_NUMBER=$3
TUNNEL_ID=$4

# Validate inputs
if [ -z "$DOWNTIME" ] || [ -z "$STORE_NUMBER" ] || [ -z "$TUNNEL_ID" ]; then
    logger -t BGP-BUSINESS "ERROR: Missing required parameters"
    exit 1
fi

# Convert seconds to human readable
HOURS=$((DOWNTIME / 3600))
MINUTES=$(((DOWNTIME % 3600) / 60))
DURATION_TEXT="${HOURS}h ${MINUTES}m"
BUSINESS_IMPACT="standard"

# Tunnel 241 gets special handling (critical business locations)
if [[ "$TUNNEL_ID" == "241" ]]; then
    BUSINESS_IMPACT="high"

    # Alert management for critical tunnel outages > 4 hours
    if [[ "$DOWNTIME" -gt 14400 ]]; then
        curl -s -X POST "https://slack.company.com/api/webhooks/management" \
             -H "Content-Type: application/json" \
             -d "{\"text\":\"CRITICAL: Store $STORE_NUMBER (Tunnel 241) was down for
$DURATION_TEXT\"}" \
             2>/dev/null || logger -t BGP-BUSINESS "ERROR: Failed to send Slack alert"
    fi
fi

# Log business metrics
logger -t BGP-BUSINESS "Store: $STORE_NUMBER, Name: $STORE_NAME, Duration:
$DURATION_TEXT, Impact: $BUSINESS_IMPACT, Tunnel: $TUNNEL_ID"

# Update network operations dashboard
curl -s -X POST "https://dashboard.company.com/api/bgp-events" \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d
"store=$STORE_NUMBER&store_name=$STORE_NAME&duration=$DOWNTIME&impact=$BUSINESS_IMPACT&tu
nnel=$TUNNEL_ID" \
     2>/dev/null || logger -t BGP-BUSINESS "ERROR: Failed to update dashboard"

exit 0
```

BGP Flapping Detection

Multi-Event BGP State Correlation

Detect BGP neighbors that change state multiple times within a short period.

SEC Rule: BGP Flapping Detection

```
# Track BGP-5-ADJCHANGE state changes per neighbor
type=SingleWithThreshold
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+(Up|Down)
context=BGP_FLAP_$2
desc=BGP neighbor flapping detected
action=eval %hostname ( "$1" ); \
       eval %neighbor ( "$2" ); \
       eval %state ( "$3" ); \
       create BGP_FLAP_%neighbor 3600; \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_FLAP_DETECTED %t neighbor=%neighbor host=%hostname state=%state
flap_count=$thresh window=600s"; \
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_FLAP_DETECTED neighbor=\"%neighbor\" host=\"%hostname\" state=\"%state\"
flap_count=\"$thresh\" window=\"600s\"")
thresh=5
window=600
```

BGP Route Advertisement Monitoring

Route Withdrawal/Advertisement Correlation

Monitor BGP route advertisements and detect routing instability.

SEC Rule: Route Instability Detection

```
# Route Instability Detection
# Detects excessive BGP route-related events per neighbor

type=SingleWithThreshold

ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-[34]-
(?:MAXPFX|NOTIFICATION)"\s+host="([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="
[^"]*"\s+dstport="[^"]*"\s+protocol="[^"]*"\s+(.*)

context=ROUTE_INSTABILITY_%neighbor_ip
desc=Route instability from BGP neighbor $2 on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       eval %message_detail ( "$3" ); \
       eval %route_prefix ( (%message_detail =~ /Prefix\s+([\d.\/]+)/) ? "$1" : "N/A" ); \
       eval %addr_family ( (%message_detail =~ /\((IPv[46]|VPNv[46])[^)]*\)/) ? "$1" :
"N/A" ); \
       eval %vrf ( (%message_detail =~ /VRF\s+(\S+)/) ? "$1" : "default" ); \
       create ROUTE_INSTABILITY_%neighbor_ip 3600; \
       write /var/log/logzilla/sec/network-health.log \
         "ROUTE_INSTABILITY %t router=%router_host neighbor=%neighbor_ip
prefix=%route_prefix addr_family=%addr_family vrf=%vrf event_count=$thresh window=300s"; \
       shellcmd (logger -t SEC-NETWORK-HEALTH -p local0.warning \
         "ROUTE_INSTABILITY router=\"%router_host\" neighbor=\"%neighbor_ip\"
prefix=\"%route_prefix\" addr_family=\"%addr_family\" vrf=\"%vrf\"
event_count=\"$thresh\"")

thresh=10
window=300
```

Multi-Router BGP Correlation

Network-Wide BGP Event Correlation

Detect BGP events affecting multiple routers simultaneously, indicating upstream provider issues.

SEC Rule: Network-Wide BGP Correlation

```
# Multi-Router BGP Impact Detection
# Detects when same BGP neighbor affects 3+ routers within 5 minutes

type=SingleWithThreshold

ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-[345]-
(?:ADJCHANGE|NOTIFICATION|MAXPFX)"\s+host="([^"]+)"\s+srcip="([^"]+)"\s+dstip="
[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="[^"]*"\s+

context=BGP_MULTI_ROUTER_IMPACT_%neighbor_ip_HOST_%router_host
desc=BGP events from neighbor $2 affecting multiple routers
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       create BGP_NEIGHBOR_IMPACT_%neighbor_ip 300 \
         (report BGP_NEIGHBOR_IMPACT_%neighbor_ip \
           /bin/bash -c 'echo "%neighbor_ip affecting multiple routers" | \
             logger -t SEC-NETWORK-HEALTH -p local0.crit \
               "BGP_NETWORK_OUTAGE neighbor=\"%neighbor_ip\" affected_routers=\"$thresh\"
window=\"300s\" - possible upstream provider issue"'); \
       add BGP_NEIGHBOR_IMPACT_%neighbor_ip %router_host; \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_NETWORK_OUTAGE %t neighbor=%neighbor_ip router=%router_host
total_routers=$thresh"

thresh=3
window=300
```

LogZilla Trigger: Network-Wide BGP Outage Response

```
name: "Network-Wide BGP Outage"
filter:
  - field: program
    op: eq
    value: SEC-NETWORK-HEALTH
  - field: message
    op: "=~"
    value: "BGP_NETWORK_OUTAGE"
actions:
  send_email: true
  send_email_template: |
    Subject: CRITICAL: Network-Wide BGP Outage

    Network-wide BGP outage detected affecting multiple routers.

    Host: {{:host}}
    Message: {{:message}}

    This indicates a potential upstream provider issue.
    Escalate to network operations immediately.
  issue_notification: true
```

BGP Security Monitoring

BGP Hijack Detection

Monitor for unexpected BGP route advertisements that could indicate hijacking.

SEC Rule: BGP Hijack Detection

```
# BGP Route Anomaly Detection
# Detects suspicious BGP activity via malformed updates and prefix flooding

type=Single

ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-[34]-
(NOTIFICATION|MAXPFX)"\s+host="([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="
[^"]*"\s+dstport="[^"]*"\s+protocol="[^"]*"\s+(.*)

desc=Suspicious BGP activity detected from neighbor $3 on $2
action=eval %event_type ( "$1" ); \
       eval %router_host ( "$2" ); \
       eval %neighbor_ip ( "$3" ); \
       eval %message_detail ( "$4" ); \
       eval %error_code ( (%message_detail =~ /(\d+\/\d+)/) ? "$1" : "N/A" ); \
       eval %error_desc ( (%message_detail =~ /\(([^)]+)\)/) ? "$1" : "N/A" ); \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_ANOMALY_DETECTED %t event=%event_type router=%router_host
neighbor=%neighbor_ip error_code=%error_code error_desc=%error_desc"; \
       shellcmd (logger -t SEC-SECURITY -p local0.warning \
         "BGP_ANOMALY_DETECTED event=\"%event_type\" router=\"%router_host\"
neighbor=\"%neighbor_ip\" error=\"%error_desc\" - REVIEW REQUIRED")
```

References

- Brute Force Attack Detection
- Network Infrastructure Correlation
- Advanced Event Correlation Walkthrough

Brute Force Attack Detection

Detect SSH, FTP, RDP, and web brute-force attacks in LogZilla by correlating failed authentication attempts across PAM, sshd, and web server logs

Brute Force Attack Detection and Response

Brute force attack detection requires correlating multiple failed authentication attempts over time, followed by automated responses. The architecture separates correlation logic (SEC) from response automation (triggers), enabling structured response workflows.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview.

Basic Brute Force Detection

LogZilla Forwarder Configuration

Required App: `linux` app for basic authentication failure detection and `linux__pam` app (for `PAM User Tracking` and `PAM Remote Host` user tags).

```
# /etc/logzilla/forwarder.d/linux-auth.yaml
window_size: 0
type: sec
sec_name: linux-auth
match:
  - field: program
    value:
      - "sshd"
      - "vsftpd"
      - "proftpd"
      - "pure-ftpd"
      - "httpd"
      - "apache2"
      - "nginx"
      - "xrdp"
rules:
- rewrite:
    message: >-
      [LINUX_AUTH]
      program="${:program}"
      host="${:host}"
      srcip="${:SrcIP}"
      username="${:Username}"
      auth_success="${:Auth Success}"
      $MESSAGE
```

SEC Rule: Failed Login Threshold Detection

```
# /etc/logzilla/sec/linux-auth/failed-login-detection.sec
# ============================================
# Failed Login Detection
# Detects multiple failed login attempts indicating brute force attack
# ============================================

type=SingleWithThreshold
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="false"
desc=Failed login for $4 from $3
thresh=5
window=1800
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       create FAILED_LOGIN_ACTIVE_%username_%srcip 1800; \
       write /var/log/logzilla/sec/security.log "BRUTE_FORCE_DETECTED %t program=%program
host=%host srcip=%srcip username=%username"; \
       shellcmd logger -t SEC-SECURITY -p local0.warning "BRUTE_FORCE_DETECTED
program=\"%program\" host=\"%host\" srcip=\"%srcip\" username=\"%username\" - possible
brute force attack"
```

LogZilla Trigger: Brute Force Response

```
name: "Brute Force Attack Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "BRUTE_FORCE_DETECTED"
actions:
  exec_script: true
  script_path: "/usr/local/bin/brute-force-response.sh"
  issue_notification: true
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "brute_force_attack",
      "username": "{{:Username}}",
      "source_ip": "{{:SrcIP}}",
      "target_host": "{{:host}}",
      "service": "{{:program}}",
      "attempts": "5"
    }
```

Note: The trigger filter now correctly matches `BRUTE_FORCE_DETECTED` from the SEC rule output.

Intelligent Response Script

```
#!/bin/bash
# /usr/local/bin/brute-force-response.sh
# Called by LogZilla trigger - receives data via environment variables

# Parse user tags from EVENT_USER_TAGS (format: key1=value1,key2=value2)
parse_user_tags() {
    local tags="$1"
    local IFS=','
    for pair in $tags; do
        local key="${pair%%=*}"
        local value="${pair#*=}"
        case "$key" in
            Username) USERNAME="$value" ;;
            SrcIP) SRC_IP="$value" ;;
        esac
    done
}

# Validate required environment variables
if [[ -z "$EVENT_USER_TAGS" ]] || [[ -z "$EVENT_MESSAGE" ]]; then
    logger -t SECURITY-RESPONSE "ERROR: Missing required environment variables"
    exit 1
fi

# Extract data from environment variables
parse_user_tags "$EVENT_USER_TAGS"
TARGET_HOST="$EVENT_HOST"

# Parse attempt count and program from SEC message
# Format: BRUTE_FORCE_DETECTED ... program="sshd" host="server1" srcip="1.2.3.4"
username="admin" attempts="5" window="300s"
PROGRAM=$(echo "$EVENT_MESSAGE" | grep -oP 'program="\K[^"]+')
ATTEMPT_COUNT=$(echo "$EVENT_MESSAGE" | grep -oP 'attempts="\K[^"]+')

# Validate we got required data
if [[ -z "$USERNAME" ]] || [[ -z "$SRC_IP" ]]; then
    logger -t SECURITY-RESPONSE "ERROR: Could not parse Username or SrcIP from user tags:
$EVENT_USER_TAGS"
    exit 1
fi

# Query threat intelligence for IP reputation
IP_REPUTATION=$(curl -s "https://threat-intel.company.com/ip/$SRC_IP" 2>/dev/null || echo
"unknown")

# Check if user account is service account or human
ACCOUNT_TYPE=$(curl -s "https://identity.company.com/api/account-type/$USERNAME"
```

```
2>/dev/null || echo "unknown")

# Make intelligent blocking decision
if [[ "$IP_REPUTATION" == "known_malicious" ]]; then
    # Block immediately for known bad IPs
    iptables -I INPUT -s "$SRC_IP" -j DROP 2>/dev/null
    logger -t SECURITY-RESPONSE "Blocked known malicious IP: $SRC_IP attacking
$USERNAME@$TARGET_HOST via $PROGRAM"

    # Create high-priority incident
    curl -s -X POST "https://servicedesk.company.com/api/incidents" \
         -H "Content-Type: application/json" \
         -d "{\"priority\":\"high\",\"subject\":\"Malicious IP Brute
Force\",\"details\":\"IP: $SRC_IP, Target: $USERNAME@$TARGET_HOST, Service: $PROGRAM,
Attempts: $ATTEMPT_COUNT\"}" \
         2>/dev/null

elif [[ "$ACCOUNT_TYPE" == "service_account" ]]; then
    # Service account attacks are high priority
    logger -t SECURITY-RESPONSE "Service account brute force: $USERNAME from $SRC_IP
($ATTEMPT_COUNT attempts via $PROGRAM)"

    # Temporarily disable service account
    curl -s -X POST "https://identity.company.com/api/disable-account" \
         -H "Content-Type: application/json" \
         -d "
{\"username\":\"$USERNAME\",\"reason\":\"brute_force\",\"source_ip\":\"$SRC_IP\"}" \
         2>/dev/null

    # Alert operations team immediately
    curl -s -X POST "https://slack.company.com/api/webhooks/ops-alerts" \
         -H "Content-Type: application/json" \
         -d "{\"text\":\"🚨 URGENT: Service account $USERNAME under brute force attack
from $SRC_IP ($ATTEMPT_COUNT attempts via $PROGRAM on $TARGET_HOST)\"}" \
         2>/dev/null

else
    # Standard user account - rate limit and monitor
    # Implement temporary IP blocking with auto-expiry
    iptables -I INPUT -s "$SRC_IP" -j DROP 2>/dev/null
    echo "iptables -D INPUT -s $SRC_IP -j DROP 2>/dev/null" | at now + 1 hour 2>/dev/null

    logger -t SECURITY-RESPONSE "Temporarily blocked $SRC_IP for brute force against
$USERNAME@$TARGET_HOST ($ATTEMPT_COUNT attempts via $PROGRAM)"

    # Create standard priority ticket
    curl -s -X POST "https://servicedesk.company.com/api/tickets" \
         -H "Content-Type: application/json" \
         -d "{\"priority\":\"medium\",\"subject\":\"Brute Force
Attack\",\"details\":\"User: $USERNAME, IP: $SRC_IP, Host: $TARGET_HOST, Service:
```

```
$PROGRAM, Attempts: $ATTEMPT_COUNT\"}" \
         2>/dev/null
fi

# Log to SEC tracking file for auditing
echo "$(date -
Iseconds)|$USERNAME|$SRC_IP|$TARGET_HOST|$PROGRAM|$ATTEMPT_COUNT|$IP_REPUTATION|$ACCOUNT_
TYPE" >> /var/log/logzilla/sec/auth-failure-responses.log

exit 0
```

Advanced Brute Force Patterns

Distributed Brute Force Detection

Detect coordinated attacks from multiple source IPs targeting the same user.

SEC Rule: Distributed Attack Detection

```
# /etc/logzilla/sec/linux-auth/distributed-brute-force-detection.sec
# ============================================
# Distributed Brute Force Detection
# Tracks failed login attempts per username from multiple source IPs
# Detects coordinated attacks - alerts only, no blocking
# ============================================

type=SingleWithThreshold
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="false"
desc=Distributed brute force attack detected against user $4
thresh=20
window=600
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       write /var/log/logzilla/sec/security.log "DISTRIBUTED_BRUTE_FORCE %t
program=%program host=%host username=%username latest_srcip=%srcip total_attempts=%thresh
window=600s"; \
       shellcmd logger -t SEC-SECURITY -p local0.warning "DISTRIBUTED_BRUTE_FORCE
username=\"%username\" host=\"%host\" latest_srcip=\"%srcip\" total_attempts=\"%thresh\"
window=\"600s\" - distributed attack from multiple IPs detected"
```

Password Spray Detection

Detect password spray attacks (same password against multiple users).

SEC Rule: Password Spray Detection

```
# /etc/logzilla/sec/linux-auth/password-spray-detection.sec
# Password Spray Attack Detection
# Detects when single IP attempts authentication as multiple different users
# Indicates password spray attack

type=SingleWithThreshold
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="false"
desc=Password spray attack detected from $3
thresh=50
window=1800
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       write /var/log/logzilla/sec/security.log "PASSWORD_SPRAY_DETECTED %t srcip=%srcip
total_attempts=50 window=1800s - single IP attacking multiple accounts"; \
       shellcmd logger -t SEC-SECURITY -p local0.alert "PASSWORD_SPRAY_DETECTED
srcip=\"%srcip\" username=\"%username\" host=\"%host\" program=\"%program\"
total_attempts=50 window=1800s - single IP attacking multiple accounts"
```

Successful Login After Brute Force

Compromise Detection Pattern

Detect successful logins that occur shortly after brute force attempts, indicating potential credential compromise.

SEC Rule: Post-Brute Force Success Detection

You must also use the SEC rule indicated in *Failed Login Threshold Detection* above, because it tracks the repeated failed logins before success.

```
# /etc/logzilla/sec/linux-auth/brute-force-success.sec
# ============================================
# Successful Login After Failed Attempts
# Detects when user successfully logs in after multiple failures
# Indicates potential successful brute force / credential compromise
# ============================================

type=Single
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="true"
desc=Successful login for $4 from $3 after previous brute force attempts
context=FAILED_LOGIN_ACTIVE_$4_$3
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       write /var/log/logzilla/sec/security.log "CREDENTIAL_COMPROMISE_SUSPECTED %t
program=%program host=%host srcip=%srcip username=%username - successful login after brute
force"; \
       shellcmd (logger -t SEC-SECURITY -p local0.crit "CREDENTIAL_COMPROMISE_SUSPECTED
program=\"%program\" host=\"%host\" srcip=\"%srcip\" username=\"%username\" - POSSIBLE
ACCOUNT COMPROMISE - IMMEDIATE ACTION REQUIRED"); \
       delete FAILED_LOGIN_ACTIVE_%username_%srcip
```

LogZilla Trigger: Compromise Response

```
name: "Credential Compromise Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "CREDENTIAL_COMPROMISE_SUSPECTED"
actions:
  exec_script: true
  script_path: "/usr/local/bin/credential-compromise-response.sh"
  send_email: true
  send_email_template: |
    Subject: CRITICAL: Potential Credential Compromise

    User: {{:Username}}
    Source IP: {{:SrcIP}}
    Target Host: {{:host}}

    Successful login detected after brute force attack.
    Immediate investigation required.
```

Cross-Service Brute Force Detection

Multi-Service Attack Correlation

Detect attackers attempting brute force across multiple services (SSH, RDP, HTTP).

SEC Rule: Cross-Service Correlation

```
# /etc/logzilla/sec/linux-auth/cross-service-correlation.sec
# ==========================================
# Cross-Service Brute Force Detection
# Tracks failed login attempts across different services from same IP
# Detects when attacker tries multiple services (SSH, RDP, HTTP, etc.)
# ==========================================

type=SingleWithThreshold
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="false"
desc=Cross-service attack detected from $3
thresh=15
window=900
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       write /var/log/logzilla/sec/security.log "CROSS_SERVICE_ATTACK %t srcip=%srcip
program=%program host=%host username=%username total_attempts=15 window=900s - attacker
targeting multiple services"; \
       shellcmd logger -t SEC-SECURITY -p local0.alert "CROSS_SERVICE_ATTACK
srcip=\"%srcip\" program=\"%program\" host=\"%host\" username=\"%username\"
total_attempts=15 window=900s - attacker targeting multiple services"
```

Time-Based Attack Pattern Detection

Off-Hours Attack Detection

Detect brute force attacks occurring during unusual hours for additional context.

SEC Rule: Time-Based Correlation

```
# /etc/logzilla/sec/linux-auth/time-based-correlation.sec
# ==========================================
# Time-Based Brute Force Detection
# Enhanced brute force detection with time context
# Identifies attacks during off-hours or weekends as higher priority
# ==========================================

type=SingleWithThreshold
ptype=RegExp
pattern=.*\[LINUX_AUTH\]\s+program="([^"]+)"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+username="([^"]+)"\s+auth_success="false"
desc=Brute force attack with time analysis for $4 from $3
thresh=8
window=300
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %srcip ( "$3" ); \
       eval %username ( "$4" ); \
       eval %hour ( ( localtime( time() ) )[2] ); \
       eval %dow_raw ( ( localtime( time() ) )[6] ); \
       eval %day_of_week ( ( %dow_raw == 0 ) ? 7 : %dow_raw ); \
       eval %time_context ( ( ( %hour < 6 ) || ( %hour > 22 ) || ( %day_of_week > 5 ) ) ?
"suspicious" : "normal" ); \
       write /var/log/logzilla/sec/security.log "TIME_BASED_BRUTE_FORCE %t
program=%program host=%host srcip=%srcip username=%username attempts=8
time_context=%time_context hour=%hour day=%day_of_week"; \
       shellcmd logger -t SEC-SECURITY -p local0.alert "BRUTE_FORCE_DETECTED
program=\"%program\" host=\"%host\" srcip=\"%srcip\" username=\"%username\" attempts=8
time_context=\"%time_context\" hour=\"%hour\" day_of_week=\"%day_of_week\""
```

Related Topics

- SOAR Security Orchestration
- Event Correlation Rule Types
- Creating Triggers
- Advanced Event Correlation Walkthrough

Application Performance Monitoring

Correlate response-time degradation, error clustering, and service dependency failures in LogZilla using SEC threshold and trend analysis rules

Application Performance Monitoring Correlation

Application performance correlation detects degradation patterns, error clustering, and service dependency issues that span multiple events over time. LogZilla's pre-processing extracts performance metrics, while SEC handles complex threshold analysis and trend correlation.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview.

Response Time Degradation Detection

LogZilla Forwarder Configuration

```
# /etc/logzilla/forwarder.d/linux-performance.yaml
window_size: 0
type: sec
sec_name: linux-performance
match:
  - field: extra:_is_performance
    value:
      - "true"
rules:
- rewrite:
    message: >-
      [LINUX_PERFORMANCE]
      program="${:program}"
      host="${:host}"
      srcip="${:SrcIP}"
      username="${:Username}"
      service="${:service}"
      endpoint="${:endpoint}"
      status="${:HTTP Status Code}"
      response_time_ms="${:response_time_ms}"
      response_time_sec="${:response_time_sec}"
      performance="${:performance}"
      error="${:error}"
      severity="${:severity}"
      memory_usage="${:memory_usage}"
      memory_usage_pct="${:memory_usage_pct}"
      process_count="${:process_count}"
      $MESSAGE
```

SEC Rule: Response Time Degradation with Recovery

```
# /etc/logzilla/sec/linux-performance/response-time-degradation.sec
# ==========================================
# Response Time Degradation Detection (Linux)
# Detects repeated slow responses for monitored services/endpoints
# Uses linux-performance message rewrite output
# ==========================================

type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*host="([^\"]+)".*service="([^\"]+)".*endpoint="
([^\"]*)".*response_time_ms="([^\"]*)".*performance="(WARNING|CRITICAL)".*severity="
([^\"]*)"
desc=Response time degradation detected for $1 on $3
thresh=5
window=300
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create PERF_DEGRADATION_ACTIVE_%host_%endpoint 900; \
       write /var/log/logzilla/sec/performance.log "RESPONSE_TIME_DEGRADATION %t
host=%host service=%service endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms occurrences=%thresh window=300s"; \
       shellcmd (logger -t SEC-PERFORMANCE -p local0.warning "RESPONSE_TIME_DEGRADATION
host=\"%host\" service=\"%service\" endpoint=\"%endpoint\" severity=\"%severity\"
response_time_ms=\"%response_time_ms\" occurrences=\"%thresh\" window=\"300s\"")
```

LogZilla Trigger: Performance Degradation Response

```
name: "Application Performance Degradation"
filter:
  - field: program
    op: eq
    value: SEC-PERFORMANCE
  - field: message
    op: "=~"
    value: "RESPONSE_TIME_DEGRADATION"
actions:
  exec_script: true
  script_path: "/usr/local/bin/performance-degradation-response.sh"
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "performance_degradation",
      "service": "{{event:ut:service}}",
      "endpoint": "{{event:ut:endpoint}}",
      "host": "{{event:ut:host}}",
      "severity": "{{event:ut:performance}}"
    }
```

Intelligent Performance Response Script

```
#!/bin/bash
# /usr/local/bin/performance-degradation-response.sh

SERVICE="$EVENT_USER_TAGS_SERVICE"
ENDPOINT="$EVENT_USER_TAGS_ENDPOINT"
HOST="$EVENT_USER_TAGS_HOST"
AVG_RESPONSE="$EVENT_USER_TAGS_AVG_RESPONSE"

# Query application metrics for additional context
CPU_USAGE=$(curl -s "http://$HOST:9090/metrics" | grep "cpu_usage" | awk '{print $2}')
MEMORY_USAGE=$(curl -s "http://$HOST:9090/metrics" | grep "memory_usage" | awk '{print
$2}')
ACTIVE_CONNECTIONS=$(curl -s "http://$HOST:9090/metrics" | grep "active_connections" |
awk '{print $2}')

# Determine root cause and appropriate response
if [[ $(echo "$CPU_USAGE > 80" | bc) -eq 1 ]]; then
    # High CPU - scale horizontally if possible
    logger -t PERF-RESPONSE "High CPU detected on $HOST - attempting auto-scale"
    curl -X POST "https://orchestrator.company.com/api/scale-service" \
         -d "service=$SERVICE&action=scale_out&reason=high_cpu"

elif [[ $(echo "$MEMORY_USAGE > 90" | bc) -eq 1 ]]; then
    # Memory pressure - restart service
    logger -t PERF-RESPONSE "Memory pressure on $HOST - scheduling service restart"
    curl -X POST "https://orchestrator.company.com/api/restart-service" \
         -d "service=$SERVICE&host=$HOST&reason=memory_pressure"

elif [[ $(echo "$ACTIVE_CONNECTIONS > 1000" | bc) -eq 1 ]]; then
    # Connection overload - enable rate limiting
    logger -t PERF-RESPONSE "Connection overload on $HOST - enabling rate limiting"
    curl -X POST "https://loadbalancer.company.com/api/rate-limit" \
         -d "service=$SERVICE&limit=100&duration=300"

else
    # Unknown cause - create investigation ticket
    logger -t PERF-RESPONSE "Unknown performance issue on $HOST - creating ticket"
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=medium&subject=Performance
Investigation&service=$SERVICE&host=$HOST"
fi

# Update performance dashboard
curl -X POST "https://dashboard.company.com/api/performance-events" \
     -d
```

```
"service=$SERVICE&endpoint=$ENDPOINT&host=$HOST&response_time=$AVG_RESPONSE&status=degrad
ed"
```

Error Rate Correlation

HTTP Error Clustering Detection

Correlate HTTP error patterns to detect application issues.

SEC Rule: Error Rate Spike Detection

```
# /etc/logzilla/sec/linux-performance/error-rate-spike-detection.sec
##########################################
# HTTP Error Rate Spike Detection (Linux)
# Detects spikes of HTTP 4xx/5xx responses for monitored services
# Relies on linux-performance rewrite message format
##########################################

type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="([^"]*)".*endpoint="
([^"]*)".*status="([45]\d{2})"
desc=Spike in HTTP error responses for $1 on $3
thresh=50
window=300
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %status ( "$4" ); \
       create HTTP_ERROR_RATE_ACTIVE_%host_%endpoint 600; \
       write /var/log/logzilla/sec/performance.log "HTTP_ERROR_RATE_SPIKE %t host=%host
service=%service endpoint=%endpoint status_code=%status occurrences=%thresh window=300s"
```

Service Dependency Correlation

Cascading Service Failure Detection

Detect when upstream service failures cause downstream performance issues.

SEC Rule: Service Dependency Correlation

```
# /etc/logzilla/sec/linux-performance/service-dependency-correlation.sec
# ==========================================
# Service Dependency Correlation (Linux)
# Detects upstream service degradation and tracks downstream impact
# Uses linux-performance message rewrite output
# Tracks dependencies: upstream (auth, payment, inventory) -> downstream (web, api,
mobile)
# ==========================================

# Rule 1: Detect upstream service degradation
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="
(auth|payment|inventory)".*endpoint="([^"]*)".*response_time_ms="([^"]+)".*performance="
(WARNING|CRITICAL)".*severity="([^"]*)"
desc=Upstream service degradation detected for $2
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create UPSTREAM_DEGRADED_%service 1800; \
       write /var/log/logzilla/sec/performance.log "UPSTREAM_SERVICE_DEGRADATION %t
upstream_service=%service host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
       shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "UPSTREAM_ISSUE
service=\"%service\" host=\"%host\" endpoint=\"%endpoint\" severity=\"%severity\"
response_time_ms=\"%response_time_ms\"")

# Rule 2: Detect downstream impact when auth service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_auth
desc=Downstream service impact detected from auth degradation on $2
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %endpoint ( "$3" ); \
       eval %response_time_ms ( "$4" ); \
       eval %performance ( "$5" ); \
       eval %severity ( "$6" ); \
       create CASCADING_FAILURE_%service_auth 900; \
       write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t
downstream=%service upstream_cause=auth host=%host endpoint=%endpoint severity=%severity
response_time_ms=%response_time_ms"; \
```

```
        shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"auth\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")

# Rule 3: Detect downstream impact when payment service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_payment
desc=Downstream service impact detected from payment degradation on $2
action=eval %host ( "$1" ); \
        eval %service ( "$2" ); \
        eval %endpoint ( "$3" ); \
        eval %response_time_ms ( "$4" ); \
        eval %performance ( "$5" ); \
        eval %severity ( "$6" ); \
        create CASCADING_FAILURE_%service_payment 900; \
        write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t
downstream=%service upstream_cause=payment host=%host endpoint=%endpoint
severity=%severity response_time_ms=%response_time_ms"; \
        shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"payment\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")

# Rule 4: Detect downstream impact when inventory service is degraded
type=Single
ptype=RegExp
pattern=.*\[LINUX_PERFORMANCE\].*host="([^"]*)".*service="(web|api|mobile)".*endpoint="
([^"]*)".*response_time_ms="([^"]+)".*performance="(WARNING|CRITICAL)".*severity="([^"]*)"
context=UPSTREAM_DEGRADED_inventory
desc=Downstream service impact detected from inventory degradation on $2
action=eval %host ( "$1" ); \
        eval %service ( "$2" ); \
        eval %endpoint ( "$3" ); \
        eval %response_time_ms ( "$4" ); \
        eval %performance ( "$5" ); \
        eval %severity ( "$6" ); \
        create CASCADING_FAILURE_%service_inventory 900; \
        write /var/log/logzilla/sec/performance.log "CASCADING_FAILURE %t
downstream=%service upstream_cause=inventory host=%host endpoint=%endpoint
severity=%severity response_time_ms=%response_time_ms"; \
        shellcmd (logger -t SEC-DEPENDENCY -p local0.warning "CASCADING_FAILURE
downstream=\"%service\" upstream_cause=\"inventory\" host=\"%host\" endpoint=\"%endpoint\"
severity=\"%severity\" response_time_ms=\"%response_time_ms\"")
```

Application Resource Correlation

Memory Leak Detection

Correlate memory usage patterns with application performance over time.

SEC Rule: Memory Leak Detection

```
# /etc/logzilla/sec/linux-performance/memory-leak-detection.sec
# Memory Leak Detection SEC Rule for LogZilla
#
# Prerequisites:
# 1. Deploy memory-usage-logging.sh to /usr/local/bin/ on monitored hosts
# 2. Deploy check-memory-trend.sh to /usr/local/bin/ on LogZilla server
# 3. Make both scripts executable: chmod +x /usr/local/bin/*.sh
# 4. Add memory-usage-logging.sh to crontab on monitored hosts:
#    */5 * * * * /usr/local/bin/memory-usage-logging.sh
# 5. Ensure LogZilla rewrite rule (linux-performance.yaml) forwards a single-line message
#    that contains program="MEMORY_USAGE" and fields host, service, memory_usage,
memory_usage_pct
#
# This rule analyzes memory usage trends over time to detect memory leaks.
# It requires at least 10 samples (50 minutes) to establish a trend.
#
# Exit code from check-memory-trend.sh:
#   0 = Memory leak detected (triggers action)
#   1 = Normal memory usage (triggers action2)
# Track memory usage trends over time per service
# Match the [LINUX_PERFORMANCE] header for host, then the original message at the end for
actual data
# Pass captured values to the trend checker: host, service, memory_usage

type=SingleWithScript
ptype=RegExp
pattern=\[LINUX_PERFORMANCE\].*program="MEMORY_USAGE".*host="([^"]+)".*service=
(\S+)\s+memory_usage=([\d.]+)\s+memory_usage_pct=([\d.]+)\s+process_count=(\d+)
script=/usr/local/bin/check-memory-trend.sh %1 %2 %3
desc=Memory leak detected: %2 on %1
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %memory_usage ( "$3" ); \
       eval %memory_usage_pct ( "$4" ); \
       eval %process_count ( "$5" ); \
       shellcmd (logger -t SEC-PERFORMANCE -p local0.alert "MEMORY_LEAK_DETECTED
host=%host service=%service memory_usage_mb=%memory_usage
memory_usage_pct=%memory_usage_pct process_count=%process_count trend=increasing")
action2=logonly  # Normal memory usage (no leak detected)
```

Memory Trend Analysis Script

```
#!/bin/bash
# /usr/local/bin/check-memory-trend.sh
# Memory Trend Analysis Script for SEC
# Analyzes memory usage trends to detect memory leaks
#
# Usage: check-memory-trend.sh <host> <service> <current_memory_mb>
# Exit 0 = Memory leak detected (triggers SEC action)
# Exit 1 = Normal memory usage (triggers SEC action2)

HOST="$1"
SERVICE="$2"
CURRENT_MEMORY="$3"

# Validate inputs
if [ -z "$HOST" ] || [ -z "$SERVICE" ] || [ -z "$CURRENT_MEMORY" ]; then
    echo "Usage: $0 <host> <service> <current_memory_mb>" >&2
    exit 1
fi

# Create history directory if it doesn't exist
HISTORY_DIR="/var/tmp/lz_memory_history"
mkdir -p "$HISTORY_DIR" 2>/dev/null

# Track memory history per service per host
MEMORY_HISTORY="${HISTORY_DIR}/memory_${HOST}_${SERVICE}"
echo "$(date +%s) $CURRENT_MEMORY" >> "$MEMORY_HISTORY"

# Keep only last hour of data (3600 seconds)
CUTOFF=$(($(date +%s) - 3600))
if [ -f "$MEMORY_HISTORY" ]; then
    awk -v cutoff="$CUTOFF" '$1 >= cutoff' "$MEMORY_HISTORY" > "${MEMORY_HISTORY}.tmp"
    mv "${MEMORY_HISTORY}.tmp" "$MEMORY_HISTORY"
fi

# Count number of samples
MEMORY_SAMPLES=$(wc -l < "$MEMORY_HISTORY" 2>/dev/null || echo "0")

# Need at least 10 samples (50 minutes of data at 5-minute intervals) for trend analysis
if [ "$MEMORY_SAMPLES" -lt 10 ]; then
    # Not enough data yet, consider normal
    exit 1
fi

# Calculate linear regression slope to detect memory growth trend
# Slope = (n * sum(xy) - sum(x) * sum(y)) / (n * sum(x^2) - sum(x)^2)
SLOPE=$(awk '{
    sum_x += NR
```

```
    sum_y += $2
    sum_xy += NR * $2
    sum_x2 += NR * NR
    n++
} END {
    if (n * sum_x2 - sum_x * sum_x == 0) {
        print 0
    } else {
        slope = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x * sum_x)
        print slope
    }
}' "$MEMORY_HISTORY")

# Memory leak thresholds (configurable)
# LEAK_THRESHOLD_MB_PER_SAMPLE: MB growth per sample interval
# At 5-minute intervals, 0.5 MB/sample = 6 MB/hour growth
LEAK_THRESHOLD_MB_PER_SAMPLE=0.5

# Check if slope exceeds leak threshold
# Use bc for floating point comparison
if command -v bc >/dev/null 2>&1; then
    IS_LEAK=$(echo "$SLOPE > $LEAK_THRESHOLD_MB_PER_SAMPLE" | bc -l)
    if [ "$IS_LEAK" -eq 1 ]; then
        # Memory leak detected
        echo "Memory leak detected: service=$SERVICE host=$HOST slope=${SLOPE}MB/sample"
>&2
        exit 0
    fi
else
    # Fallback to integer comparison if bc not available
    SLOPE_INT=$(printf "%.0f" "$SLOPE")
    if [ "$SLOPE_INT" -ge 1 ]; then
        echo "Memory leak detected: service=$SERVICE host=$HOST slope=${SLOPE}MB/sample"
>&2
        exit 0
    fi
fi

# Normal memory usage
exit 1
```

Generic Application Error Correlation

Note: This example uses message pattern matching since no database parsing apps exist in LogZilla by default. To parse specific database fields, you would need to create custom parsing rules.

Forwarder Configuration

```
# /etc/logzilla/forwarder.d/app-error-correlation.yaml
type: sec
sec_name: app-errors
rules:
  - match:
      - field: program
        op: "eq"
        value: ["webapp", "api-server", "microservice"]
      - field: message
        op: "=~"
        value: "(ERROR|FATAL|Exception|timeout|slow.*query)"
    rewrite:
      message: "[APP_ERROR] host=$HOST service=$PROGRAM severity=${:severity}"
```

SEC Rule: Application Error Correlation

```
# /etc/logzilla/sec/app-errors/app-error-correlation.sec
##############################################
# Application Error Correlation
# Detects error spikes across application services
# Uses app-error-correlation rewrite message format
##############################################

type=SingleWithThreshold
ptype=RegExp
pattern=\[APP_ERROR\] host=([^\s]+) service=([^\s]+) severity=([^\s]+)
desc=Application error spike detected on $1
thresh=10
window=300
action=eval %host ( "$1" ); \
       eval %service ( "$2" ); \
       eval %severity ( "$3" ); \
       create APP_ERROR_SPIKE_%host_%service 600; \
       write /var/log/logzilla/sec/performance.log "ERROR_SPIKE_DETECTED %t host=%host
service=%service severity=%severity occurrences=%thresh window=300s"; \
       shellcmd (logger -t SEC-ALERT -p local0.alert "ERROR_SPIKE_DETECTED host=\"%host\"
service=\"%service\" severity=\"%severity\" occurrences=\"10\" window=\"300s\""); \
       shellcmd (/usr/local/bin/app-error-response.sh "%host" "%service")
```

Related Topics

- Network Infrastructure Correlation
- Advanced Event Correlation Walkthrough

- Event Correlation Rule Types

Network Infrastructure Correlation

Detect interface flapping, BGP adjacency loss, and cascading network failures in LogZilla by correlating Cisco and OSPF events with SEC rules

Network Infrastructure Event Correlation

Network infrastructure correlation detects patterns in network device behavior, from simple interface flapping to complex BGP adjacency issues. LogZilla's pre-processing capabilities enable highly efficient SEC rules by extracting network-specific fields before correlation.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview.

BGP Adjacency Correlation

Business Value

Monitor BGP neighbor relationships to detect network connectivity issues, calculate outage duration, and identify chronic instability patterns.

LogZilla Forwarder Configuration

```
# /etc/logzilla/forwarder.d/cisco-network-health.yaml
window_size: 0
type: sec
sec_name: cisco-network-health
match:
  - field: cisco_mnemonic
    value:
      - "BGP-5-ADJCHANGE"
      - "BGP-4-MAXPFX"
      - "BGP-3-NOTIFICATION"
      - "OSPF-5-ADJCHG"
rules:
- rewrite:
    message: >-
      [CISCO_NETWORK_HEALTH]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      srcip="${:SrcIP}"
      dstip="${:DstIP}"
      srcport="${:SrcPort}"
      dstport="${:DstPort}"
      protocol="${:protocol}"
      $MESSAGE
```

SEC Rule: BGP Neighbor Down/Up Correlation

```
# BGP adjacency outage tracker (host + neighbor)

type=Pair

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Down\b
context=BGP_ADJ_$1_$2
desc=BGP Neighbor $2 Down on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_ip ( "$2" ); \
       eval %down_time ( time() ); \
       shellcmd (/usr/bin/host %neighbor_ip > /tmp/bgp-lookup-%neighbor_ip 2>/dev/null); \
       eval %hostname ( readfile("/tmp/bgp-lookup-%neighbor_ip") ); \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_ADJ_DOWN %t router=%router_host neighbor=%neighbor_ip hostname=%hostname";
\
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_ADJ_DOWN router=\"%router_host\" neighbor=\"%neighbor_ip\"
hostname=\"%hostname\"")

# ---- END (same host + neighbor, Up) ----
ptype2=RegExp
pattern2=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+srcip="[^"]*"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="[^"]*"\s+protocol="
[^"]*"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+Up\b
context2=BGP_ADJ_$1_$2
desc2=BGP Neighbor $2 Up on $1
action2=eval %up_time ( time() ); \
        eval %outage_duration ( %up_time - %down_time ); \
        write /var/log/logzilla/sec/network-health.log \
          "BGP_ADJ_UP %t router=%router_host neighbor=%neighbor_ip
downtime=%outage_duration hostname=%hostname"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH \
          "BGP_OUTAGE_RESOLVED router=\"%router_host\" neighbor=\"%neighbor_ip\"
downtime=\"%outage_duration\" hostname=\"%hostname\""); \
        delete /tmp/bgp-lookup-%neighbor_ip

# Pairing window (how long to wait for the Up after Down)
window=86400
```

SEC Rule: BGP Flapping Detection

```
# Detect BGP neighbor flapping (multiple up/down cycles)
# 6 state changes within 30 minutes
type=SingleWithThreshold
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="BGP-5-ADJCHANGE"\s+host="
([^"]+)"\s+.*\bneighbor\s+([0-9A-Fa-f:.]+)\s+(Up|Down)
context=BGP_FLAP_$2
desc=BGP neighbor flapping detected
action=eval %hostname ( "$1" ); \
       eval %neighbor ( "$2" ); \
       eval %state ( "$3" ); \
       create BGP_FLAP_%neighbor 1800; \
       write /var/log/logzilla/sec/network-health.log \
         "BGP_FLAP_DETECTED %t neighbor=%neighbor host=%hostname state=%state
flap_count=$thresh window=1800s"; \
       shellcmd (logger -t SEC-NETWORK-HEALTH \
         "BGP_FLAP_DETECTED neighbor=\"%neighbor\" host=\"%hostname\" state=\"%state\"
flap_count=\"$thresh\" window=\"1800s\"")
thresh=5
window=1800
```

Interface Monitoring Correlation

Interface Up/Down Correlation

LogZilla extracts interface names and states from Cisco LINK messages.

Forwarder Configuration

```
# /etc/logzilla/forwarder.d/cisco-interface-health.yaml
window_size: 0
type: sec
sec_name: cisco-interface-health
match:
  - field: cisco_mnemonic
    value:
      - "LINK-3-UPDOWN"
      - "LINK-5-CHANGED"
      - "LINEPROTO-5-UPDOWN"
rules:
- rewrite:
    program: sec-cisco-interface-health
    message: >-
      [CISCO_INTERFACE_HEALTH]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      $MESSAGE
```

SEC Rule: Interface Flapping Detection

```
# Detect interface flapping (multiple up/down state changes)
# 5 state changes within 10 minutes indicates flapping

type=SingleWithThreshold

ptype=RegExp
pattern=\[CISCO_INTERFACE_HEALTH\]\s+cisco_mnemonic="(?:LINK-3-UPDOWN|LINEPROTO-5-
UPDOWN)"\s+host="([^"]+)"\s+.*\bInterface\s+([^,\s]+).*\bchanged state to (up|down)

context=IFACE_FLAP_$1_$2
desc=Interface flapping detected on $2 at $1
action=eval %hostname ( "$1" ); \
       eval %interface ( "$2" ); \
       eval %state ( "$3" ); \
       create IFACE_FLAP_%hostname_%interface 3600; \
       write /var/log/logzilla/sec/network-health.log \
         "INTERFACE_FLAP_DETECTED %t host=%hostname interface=%interface state=%state
flap_count=$thresh window=600s"; \
       shellcmd (logger -t SEC-INTERFACE-HEALTH \
         "INTERFACE_FLAP_DETECTED host=\"%hostname\" interface=\"%interface\"
state=\"%state\" flap_count=\"$thresh\" window=\"600s\"")

thresh=5
window=600
```

Network Device Health Correlation

Multi-Metric Device Health

Correlate CPU, memory, and interface errors to assess overall device health.

Device Health Forwarder Configuration

```
# /etc/logzilla/forwarder.d/cisco-device-health.yaml
window_size: 0
type: sec
sec_name: cisco-device-health
match:
  - field: cisco_mnemonic
    value:
      - "SYS-1-CPURISINGTHRESHOLD"
      - "SYS-4-FREEMEMLOW"
      - "SYS-4-THRESHOLD_TK"
      - "LINK-3-UPDOWN"
      - "LINEPROTO-5-UPDOWN"
rules:
  - rewrite:
      message: >-
        [CISCO_DEVICE_HEALTH]
        cisco_mnemonic="${:cisco_mnemonic}"
        host="${:host}"
      $MESSAGE
```

SEC Rule: Device Health Correlation

```
# ============================================
# Cisco Device Health (CPU≥80%, Mem low, Iface down)
# ============================================

type=EventGroup3
context=HOST_$1
window=300

# -------- CPU HIGH (>=80%) --------
ptype=RegExp
pattern=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="SYS-1-CPURISINGTHRESHOLD"\s+host="
([^"]+)"\b.*\b(8\d|9\d|100)%\b
thresh=3

# -------- MEMORY LOW --------
ptype2=RegExp
pattern2=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="(?:SYS-4-FREEMEMLOW|SYS-4-
THRESHOLD_TK)"\s+host="([^"]+)"\b.*
thresh2=2

# -------- INTERFACE DOWN --------
ptype3=RegExp
pattern3=\[CISCO_DEVICE_HEALTH\]\s+cisco_mnemonic="(?:LINK-3-UPDOWN|LINEPROTO-5-
UPDOWN)"\s+host="([^"]+)"\b.*?\bInterface\s+([^,]+),\s+changed state to down\b
thresh3=1

desc=Device health degradation on host=$1 (CPU>=80% x3, Mem low x2, IfDown x1 within 5m)
action=write /var/log/logzilla/sec/network-health.log \
         "DEVICE_HEALTH_ALERT %t host=$1 symptoms=cpu_ge80x3,mem_lowx2,ifdownx1"; \
       shellcmd (logger -t SEC-CRITICAL -p local0.crit \
         "DEVICE_HEALTH_ALERT host=\"$1\" correlation=\"cpu>=80x3,mem_lowx2,ifdownx1\"")
```

OSPF Neighbor Correlation

OSPF Adjacency Monitoring

Track OSPF neighbor state changes and detect routing instability.

SEC Rule: OSPF Neighbor Correlation

```
# OSPF Neighbor State Change Tracker
# Tracks OSPF neighbor down/up events and detects instability

type=Pair

# ---- START (neighbor Down) ----
ptype=RegExp
pattern=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="OSPF-5-ADJCHG"\s+host="
([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+.*\bto DOWN\b
context=OSPF_DOWN_$1_$2
desc=OSPF neighbor $2 down on $1
action=eval %router_host ( "$1" ); \
       eval %neighbor_id ( "$2" ); \
       eval %down_time ( time() ); \
       create OSPF_DOWN_%router_host_%neighbor_id 7200; \
       write /var/log/logzilla/sec/network-health.log \
         "OSPF_NEIGHBOR_DOWN %t router=%router_host neighbor=%neighbor_id"; \
       shellcmd (logger -t SEC-NETWORK-HEALTH -p local0.notice \
         "OSPF_NEIGHBOR_DOWN router=\"%router_host\" neighbor=\"%neighbor_id\"")

# ---- END (same neighbor to Full) ----
ptype2=RegExp
pattern2=\[CISCO_NETWORK_HEALTH\]\s+cisco_mnemonic="OSPF-5-ADJCHG"\s+host="
([^"]+)"\s+srcip="([^"]+)"\s+dstip="[^"]*"\s+srcport="[^"]*"\s+dstport="
[^"]*"\s+protocol="[^"]*"\s+.*\bto FULL\b
context2=OSPF_DOWN_$1_$2
desc2=OSPF neighbor $2 restored on $1
action2=eval %up_time ( time() ); \
        eval %outage_duration ( %up_time - %down_time ); \
        write /var/log/logzilla/sec/network-health.log \
          "OSPF_NEIGHBOR_RESTORED %t router=%router_host neighbor=%neighbor_id
downtime=%outage_duration"; \
        shellcmd (logger -t SEC-NETWORK-HEALTH -p local0.info \
          "OSPF_NEIGHBOR_RESTORED router=\"%router_host\" neighbor=\"%neighbor_id\"
downtime=\"%outage_duration\""); \
        delete OSPF_DOWN_%router_host_%neighbor_id

# Pairing window (2 hours = 7200 seconds)
window=7200
```

Network Security Correlation

Firewall Connection Correlation

Monitor firewall connection patterns for security analysis.

Firewall Monitoring Configuration

Required Apps: `cisco__asa` app (for `SrcIP` and `DstIP` user tags), `sonicwall` app (for `fw_action` user tag)

```
# /etc/logzilla/forwarder.d/cisco-firewall.yaml
window_size: 0
type: sec
sec_name: cisco-firewall
match:
  - field: cisco_mnemonic
    value:
      - "ASA-2-106001"
rules:
- rewrite:
    message: >-
      [CISCO_FIREWALL]
      cisco_mnemonic="${:cisco_mnemonic}"
      host="${:host}"
      srcip="${:SrcIP}"
      dstip="${:DstIP}"
      srcport="${:SrcPort}"
      dstport="${:DstPort}"
      protocol="${:protocol}"
      $MESSAGE
```

SEC Rule: Suspicious Connection Patterns

```
# Port Scan Detection (Cisco ASA)
# Detects when source IP attempts connections to 20+ different destination ports
# within 5 minutes - indicates port scanning activity

type=SingleWithThreshold

ptype=RegExp
pattern=\[CISCO_FIREWALL\]\s+cisco_mnemonic="ASA-2-106001"\s+host="([^"]+)"\s+srcip="
([^"]+)"\s+dstip="([^"]+)"\s+srcport="[^"]*"\s+dstport="([^"]+)"\s+protocol="[^"]*"\s+

context=PORT_SCAN_$2_$4
desc=Port scan detected from $2
action=eval %firewall_host ( "$1" ); \
       eval %src_ip ( "$2" ); \
       eval %dst_ip ( "$3" ); \
       eval %dst_port ( "$4" ); \
       create PORT_SCAN_DETECTED_%src_ip 300 \
         (report PORT_SCAN_DETECTED_%src_ip \
           /bin/bash -c 'logger -t SEC-SECURITY -p local0.alert "PORT_SCAN_DETECTED
src_ip=\"%src_ip\" unique_ports=\"$thresh\" window=\"300s\" - BLOCKING"; \
             iptables -I INPUT -s %src_ip -j DROP 2>/dev/null || true; \
             echo "%src_ip" >> /var/log/logzilla/sec/blocked-ips.log'); \
       add PORT_SCAN_DETECTED_%src_ip %dst_port

thresh=20
window=300
```

Related Topics

- Event Correlation Rule Types
- Advanced Event Correlation Walkthrough
- SOAR Security Orchestration

Soar Security Orchestration

Orchestrate LogZilla security responses using SEC correlation for impossible-travel, credential abuse, and multi-stage attack patterns with GeoIP and PAM data

SOAR Security Orchestration

Security Orchestration, Automation, and Response (SOAR) capabilities detect sophisticated attack patterns that span multiple events, systems, and time periods. LogZilla's pre-processing extracts security fields, while SEC handles complex correlation logic. SEC generates synthetic events back to LogZilla, where triggers perform intelligent automated responses.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview.

Geographic Anomaly Detection

Impossible Travel Detection

Detect when the same user logs in from geographically distant locations within an impossible timeframe.

Required Apps and Configuration

Required Apps:

- `linux__pam` app (for `PAM User Tracking` and `PAM Remote Host` user tags)
- `geoip` app (for `SrcIP Country`, `SrcIP City`, `SrcIP State` user tags)

IP Address Mapping Rule

The GeoIP app requires `SrcIP` user tags, but PAM creates `PAM Remote Host`. Create a mapping rule:

File: `/etc/logzilla/rules/enabled/601-pam-mapping.lua`

```
-- Map PAM Remote Host to SrcIP for GeoIP processing
function process(event)
    if event.user_tags["PAM Remote Host"] then
        event.user_tags["SrcIP"] = event.user_tags["PAM Remote Host"]
    end
end
```

LogZilla Forwarder Configuration

```
# /etc/logzilla/forwarder.d/linux-security.yaml
window_size: 0
type: sec
sec_name: linux-security
match:
  - field: extra:_is_security
    value:
      - "true"
rules:
- rewrite:
    message: >-
      [LINUX_SECURITY]
      program="${:program}"
      host="${:host}"
      srcip="${:SrcIP}"
      srcport="${:SrcPort}"
      dstip="${:DstIP}"
      dstport="${:DstPort}"
      username="${:Username}"
      src_country="${:SrcIP Country}"
      src_city="${:SrcIP City}"
      src_state="${:SrcIP State}"
      pam_user_tracking="${:PAM User Tracking}"
      pam_action="${:PAM Action}"
      pam_remote_user="${:PAM Remote User}"
      pam_remote_host="${:PAM Remote Host}"
      bytes="${:bytes}"
      bytes_mb="${:BytesMB}"
      $MESSAGE
```

SEC Rule: Impossible Travel Detection

```
# /etc/logzilla/sec/linux-security/detect-impossible-travel.sec
# ============================================
# Detect impossible travel (Linux security)
# Same user authenticating from two different countries within 1 hour
# Uses [LINUX_SECURITY] rewritten message from linux-security.yaml
# ============================================
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="([^"]+)".*srcip="([^"]+)".*src_country="
([^"]+)".*pam_user_tracking="([^"]+)"
desc=Impossible travel check for user $4 on host $1
action=eval %host ( "$1" ); \
       eval %srcip ( "$2" ); \
       eval %src_country ( "$3" ); \
       eval %pam_user ( "$4" ); \
       shellcmd (mkdir -p /var/log/sec /var/log/logzilla/sec 2>/dev/null; \
                 statefile="/var/log/sec/impossible-travel-%pam_user"; \
                 old_country=""; old_ip=""; \
                 if [ -f "$statefile" ]; then \
                   old_country=$(sed -n '1p' "$statefile"); \
                   old_ip=$(sed -n '2p' "$statefile"); \
                 fi; \
                 if [ -n "$old_country" ] && [ "%src_country" != "$old_country" ]; then \
                   echo "IMPOSSIBLE_TRAVEL %t host=%host user=%pam_user old_ip=$old_ip
old_country=$old_country new_ip=%srcip new_country=%src_country" >>
/var/log/logzilla/sec/security.log; \
                   logger -t SEC-SECURITY -p local0.alert "IMPOSSIBLE_TRAVEL
user=\"%pam_user\" old_ip=\"$old_ip\" old_country=\"$old_country\" new_ip=\"%srcip\"
new_country=\"%src_country\""; \
                 fi; \
                 echo "%src_country" > "$statefile"; \
                 echo "%srcip" >> "$statefile")
```

Processing Chain

The complete geographic correlation works through this processing chain:

PAM Rule (`600-linux-pam.lua`) extracts authentication data:

- Creates `PAM User Tracking` (username)
- Creates `PAM Remote Host` (source IP)

Mapping Rule (`601-pam-mapping.lua`) prepares for GeoIP:

- Copies `PAM Remote Host` → `SrcIP`

GeoIP Rule (`999-add-geodata.lua`) adds geographic data:

- Creates `SrcIP Country`, `SrcIP City`, `SrcIP State`

Forwarder sends enriched event to SEC with all user tags appended

SEC Rule parses user tags from message and detects impossible travel

LogZilla Trigger: Impossible Travel Response

```
# Trigger responds to SEC-generated impossible travel events
name: "Impossible Travel Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "IMPOSSIBLE_TRAVEL"
actions:
  send_email: true
  send_email_template: |
    Subject: SECURITY ALERT: Impossible Travel Detected

    User: {{event:ut:Username}}
    Impossible Country: {{event:ut:SrcIP Country}}
    Impossible State: {{event:ut:SrcIP State}}
    Impossible City: {{event:ut:SrcIP City}}

    This indicates potential credential compromise.
  exec_script: true
  script_path: "/usr/local/bin/impossible-travel-response.sh"
  issue_notification: true
```

Intelligent Response Script

```
#!/bin/bash
# /usr/local/bin/impossible-travel-response.sh
# Called by SEC shellcmd - receives data via command-line arguments

USERNAME="$1"
DETAILS="$2"

# Query HR system for user travel status
TRAVEL_STATUS=$(curl -s "https://hr-api.company.com/travel-status/$USERNAME")

# Query threat intelligence for IP reputation
# Note: IP would need to be extracted from DETAILS or passed as separate argument
IP_REPUTATION=$(curl -s "https://threat-intel.company.com/ip-reputation/$(echo $DETAILS |
grep -oE '[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+')")

# Make intelligent decision based on context
if [[ "$TRAVEL_STATUS" == "approved_travel" ]]; then
    # User has approved travel - lower priority
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - approved travel
detected"
    # Create low-priority ticket
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=low&subject=Travel Login Verification&user=$USERNAME"
elif [[ "$IP_REPUTATION" == "malicious" ]]; then
    # High-risk scenario - immediate response
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - malicious IP detected"
    # Disable account immediately
    curl -X POST "https://identity.company.com/api/disable-account" -d
"username=$USERNAME"
    # Create high-priority incident
    curl -X POST "https://servicedesk.company.com/api/incidents" \
         -d "priority=critical&subject=Account
Compromise&user=$USERNAME&details=$DETAILS"
    # Alert SOC team
    curl -X POST "https://slack.company.com/api/webhooks/soc-alerts" \
         -d "text=CRITICAL: Account compromise detected for $USERNAME"
else
    # Standard response - investigate
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - investigation
required"
    # Temporarily restrict account
    curl -X POST "https://identity.company.com/api/restrict-account" -d
"username=$USERNAME"
    # Create medium-priority ticket
    curl -X POST "https://servicedesk.company.com/api/tickets" \
```

```
          -d "priority=medium&subject=Impossible Travel Investigation&user=$USERNAME"
fi
```

Lateral Movement Detection

Cross-System Access Correlation

Detect lateral movement by correlating edge system logins with internal network access.

SEC Rule: Lateral Movement Correlation

```
# /etc/logzilla/sec/linux-security/detect-lateral-movement.sec
# ============================================
# Lateral Movement Detection (Linux Security)
# Detects when users log into edge systems and then access internal networks
# Uses [LINUX_SECURITY] rewritten message from linux-security.yaml
# ============================================

# Create context for edge system logins (matches forwarder field order)
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="((?:web|dmz|edge|vpn)[^"]*)".*pam_user_tracking="
([^"]+)".*pam_action="session opened"
desc=Edge system access by $2
action=eval %host ( "$1" ); \
       eval %pam_user ( "$2" ); \
       create EDGE_ACCESS_%pam_user_%host 3600; \
       write /var/log/logzilla/sec/security.log "EDGE_LOGIN %t username=%pam_user
host=%host"; \
       shellcmd (logger -t SEC-AUDIT -p local0.info "EDGE_LOGIN user=\"%pam_user\"
host=\"%host\"")

# Detect internal network access from edge systems
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="((?:web|dmz|edge|vpn)[^"]*)".*dstip="([^"]+)".*dstport="
([^"]+)".*username="([^"]+)"
context=EDGE_ACCESS_$4_$1
desc=Potential lateral movement by $4 from $1 to $2:$3
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
       eval %username ( "$4" ); \
       shellcmd (echo "%dstip" | grep -Eq '^(10\.|172\.(1[6-9]|2[0-
9]|3[01])\.|192\.168\.)' && { \
                   echo "LATERAL_MOVEMENT %t username=%username src_host=%host
dst_ip=%dstip dst_port=%dstport" >> /var/log/logzilla/sec/security.log; \
                   logger -t SEC-SECURITY -p local0.crit "LATERAL_MOVEMENT
user=\"%username\" src_host=\"%host\" dst_ip=\"%dstip\" dst_port=\"%dstport\""; \
                 } || true)
```

LogZilla Trigger: Lateral Movement Response

```
name: "Lateral Movement Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "LATERAL_MOVEMENT"
actions:
  exec_script: true
  script_path: "/usr/local/bin/lateral-movement-response.sh"
  issue_notification: true
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "lateral_movement",
      "user": "{{event:ut:Username}}",
      "source_host": "{{event:host}}",
      "destination": "{{event:ut:DstIP}}:{{event:ut:DstPort}}",
      "severity": "critical"
    }
```

Advanced Persistent Threat (APT) Detection

Multi-Stage APT Correlation

Detect sophisticated attack campaigns across multiple stages and systems. This demonstrative example shows how SEC can correlate events to identify multi-stage attacks.

Prerequisites:

- Deploy `bandwidth-monitor.sh` script (see below)
- Add to crontab: `*/5 * * * * /path/to/bandwidth-monitor.sh`

Bandwidth Monitor Script

```
#!/bin/bash
# Simple bandwidth monitor for APT detection
# Logs large outbound transfers to syslog
THRESHOLD_MB=100
LOG_TAG="bandwidth-monitor"

ss -tn state established | awk 'NR>1 {
    split($4, src, ":")
    split($5, dst, ":")
    src_ip = src[1]
    dst_ip = dst[1]
    dst_port = dst[length(dst)]

    # Check for external connections (demo)
    if (dst_ip !~ /^(10\.|172\.(1[6-9]|2[0-9]|3[01])\.|192\.168\.)/) {
        bytes = int(rand() * 400000000) + 100000000
        printf "%s %s %s %s\n", src_ip, dst_ip, dst_port, bytes
    }
}' | while read src_ip dst_ip dst_port bytes; do
    bytes_mb=$((bytes / 1048576))
    if [ "$bytes_mb" -ge "$THRESHOLD_MB" ]; then
        logger -t "$LOG_TAG" \
            "LARGE_TRANSFER srcip=${src_ip} dstip=${dst_ip} dstport=${dst_port}
bytes=${bytes} bytes_mb=${bytes_mb}"
    fi
done
```

SEC Rule: APT Campaign Detection

```
# /etc/logzilla/sec/linux-security/detect-apt-campaign.sec
# ============================================
# APT Campaign Detection - Demonstrative Example
# Three-stage attack detection showing SEC correlation capabilities
# ============================================

# Stage 1: Suspicious outbound connection to external IP
# Detects connections to known-bad or suspicious IPs from kernel/firewall logs
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="kernel".*host="([^"]+)".*dstip="
((?:185\.220\.|45\.153\.|23\.129\.)[^"]+)".*dstport="([^"]+)"
desc=Suspicious outbound connection from $1 to $2:$3
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
       create APT_INDICATOR_%host 86400; \
       write /var/log/logzilla/sec/security.log "APT_STAGE1 %t host=%host
destination=%dstip:%dstport"; \
       shellcmd logger -t SEC-APT -p local0.warning "APT_STAGE1 host=\"%host\"
dst=\"%dstip:%dstport\" - suspicious outbound detected"

# Stage 2: Suspicious process execution on flagged host
# Looks for credential dumping or hacking tools in program field
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="
(mimikatz|pwdump|procdump|hashcat|hydra|metasploit)".*host="([^"]+)"
context=APT_INDICATOR_$2
desc=Suspicious process $1 on compromised host $2
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       write /var/log/logzilla/sec/security.log "APT_STAGE2 %t host=%host
process=%program"; \
       shellcmd logger -t SEC-APT -p local0.alert "APT_STAGE2 host=\"%host\"
process=\"%program\" - credential tool detected"

# Stage 3 Use Perl for numeric comparison
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="bandwidth-monitor".*host="([^"]+)".*dstip="
([^"]+)".*dstport="([^"]+)".*bytes="([0-9]+)"
context=APT_INDICATOR_$1
desc=Large transfer from compromised host $1 to $2:$3 ($4 bytes)
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
```

```
       eval %bytes ( "$4" ); \
       eval %large ( %bytes > 104857600 ); \
       if %large ( write /var/log/logzilla/sec/security.log "APT_CAMPAIGN %t host=%host
dst=%dstip:%dstport bytes=%bytes - FULL ATTACK CHAIN DETECTED"; \
                   shellcmd logger -t SEC-APT -p local0.crit "APT_CAMPAIGN host=\"%host\"
dst=\"%dstip:%dstport\" bytes=\"%bytes\" - FULL ATTACK CHAIN DETECTED" )
```

LogZilla Trigger: APT Campaign Response

```
name: "APT Campaign Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-APT
  - field: message
    op: "=~"
    value: "APT_CAMPAIGN"
actions:
  send_email: true
  send_email_template: |
    Subject: CRITICAL: APT Campaign Detected

    A multi-stage attack has been detected and correlated by SEC.

    Full attack chain: Suspicious connection → Credential tool → Data exfiltration

    Immediate investigation and containment required.
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "apt_campaign",
      "severity": "critical",
      "message": "{{event:message}}"
    }
```

Privilege Escalation Detection

Account Privilege Changes Correlation

Monitor for suspicious privilege escalations and administrative access patterns.

SEC Rule: Privilege Escalation Correlation

```
# ============================================
# Privilege Escalation Detection
# Two-stage detection: failed admin attempts → successful escalation
# Uses [LINUX_SECURITY] rewritten messages from linux-security.yaml
# ============================================

# Stage 1: Track multiple failed administrative access attempts (sudo/su)
# Looks for failed authentication in PAM logs
type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="(sudo|su)".*host="([^"]+)".*pam_user_tracking="
([^"]+)".*authentication failure
desc=Multiple failed admin access attempts by $3 on $2
thresh=5
window=300
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %pam_user ( "$3" ); \
       create PRIV_ESC_ATTEMPT_%pam_user_%host 600; \
       write /var/log/logzilla/sec/security.log "PRIVILEGE_ESCALATION_ATTEMPT %t
program=%program user=%pam_user host=%host count=5"; \
       shellcmd (logger -t SEC-SECURITY -p local0.warning "PRIVILEGE_ESCALATION_ATTEMPT
program=\"%program\" user=\"%pam_user\" target=\"%host\" count=5")

# Stage 2: Detect successful admin access after failed attempts
# Looks for successful PAM session opened after failures
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="(sudo|su)".*host="([^"]+)".*pam_user_tracking="
([^"]+)".*pam_action="(session opened|command executed)"
context=PRIV_ESC_ATTEMPT_$3_$2
desc=Successful privilege escalation by $3 on $2 after failed attempts
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %pam_user ( "$3" ); \
       eval %pam_action ( "$4" ); \
       delete PRIV_ESC_ATTEMPT_%pam_user_%host; \
       write /var/log/logzilla/sec/security.log "PRIVILEGE_ESCALATION_SUCCESS %t
program=%program user=%pam_user host=%host action=%pam_action"; \
       shellcmd (logger -t SEC-SECURITY -p local0.alert "PRIVILEGE_ESCALATION_SUCCESS
program=\"%program\" user=\"%pam_user\" target=\"%host\" action=\"%pam_action\" -
successful after failures")
```

Related Topics

- Brute Force Attack Detection

- BGP Network Monitoring Correlation
- Event Correlation Rule Types
- Creating Triggers

Windows Event Correlation

Correlate Windows Security events 4624, 4625, 4732, and 1102 in LogZilla to detect successful brute-force attacks, privilege escalation, and log tampering

Windows Event Correlation for Administrators

Windows administrators face unique challenges monitoring enterprise environments with thousands of events daily. LogZilla's Windows correlation capabilities combine pre-built triggers for immediate alerts with SEC correlation for complex attack pattern detection.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview.

Brute Force Attack Detection

Business Problem

Failed login alerts (Event ID 4625) generate thousands of events daily in enterprise environments. Simple triggers create alert fatigue, while sophisticated attacks go undetected.

Correlation Solution

Detect successful brute force attacks by correlating failed attempts with subsequent successful logins.

LogZilla Forwarder Configuration

Required App: `ms_windows` app (for `MSWin EventID`, `MSWin Failed Login User`, and `MSWin Failed Login Source Network` user tags)

File: `/etc/logzilla/forwarder.d/windows-brute-force.yaml`

```
# /etc/logzilla/forwarder.d/windows-brute-force.yaml
window_size: 0
type: sec
sec_name: windows-security
match:
  - field: MSWin EventID
    value: ["4624", "4625", "4732", "1102"]
rules:
- rewrite:
    message: >-
      [WINDOWS_AUTH] host=${:host} MSWin_EventID=${:MSWin EventID} SrcIP=${:SrcIP}
      Subject_User_Name=${:Subject User Name} Target_Domain_Name=${:Target Domain Name}
      Target_Group_Name=${:Target Group Name} Target_User_Name=${:Target User Name}
      Username=${:Username} User_Domain_Name=${:User Domain Name} $MESSAGE
```

SEC Rule: Brute Force Detection

File: `/etc/logzilla/sec/windows-security/brute-force.sec`

New:

```
# ==========================
# Windows brute-force (per user) - EventGroup variant
# ==========================

# 1) Track failed logins per (host, srcip, user)
type=EventGroup
ptype=RegExp
# Allow newlines between fields
pattern=.*\[WINDOWS_AUTH\][\s\S]*host=([^\s"]+)[\s\S]*MSWin_EventID=4625[\s\S]*SrcIP=
([^\s"]+)[\s\S]*Username=([^\s"]+)
thresh=10
window=300
multact=no
# Group key is the desc (one group per unique values)
desc=Windows failed logon burst for user $3 from $2 to $1

# Fire when threshold is reached
action=eval %target_host ( "$1" ); \
       eval %client_ip ( "$2" ); \
       eval %user_lc ( lc("$3") ); \
       create BRUTE_FORCE_ACTIVE_%user_lc 1800; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
         "WINDOWS_BRUTE_FORCE user=\"%user_lc\" host=\"%target_host\"
src=\"%client_ip\"")

# Optional: set vars when the group starts; clean up when it expires
init=eval %target_host ( "$1" ); eval %client_ip ( "$2" ); eval %user_lc ( lc("$3") )
end=delete BRUTE_FORCE_ACTIVE_%user_lc

# 2) Detect successful login after failures
type=Single
ptype=RegExp
pattern=.*\[WINDOWS_AUTH\][\s\S]*host=([^\s"]+)[\s\S]*MSWin_EventID=4624[\s\S]*SrcIP=
([^\s"]+)[\s\S]*Username=([^\s"]+)
desc=Windows successful logon after failures (same user) $3
action=eval %target_host ( "$1" ); \
       eval %client_ip ( "$2" ); \
       eval %user_lc ( lc("$3") ); \
       if (BRUTE_FORCE_ACTIVE_%user_lc) \
         shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
           "WINDOWS_COMPROMISE_SUSPECTED user=\"%user_lc\" host=\"%target_host\"
src=\"%client_ip\""); \
       delete BRUTE_FORCE_ACTIVE_%user_lc
```

Old:

```
# =========================
# Windows brute-force (per user, case-insensitive) - ALTERNATIVE SOLUTION
# =========================

# 1) Track failed logins per user using EventGroup
type=EventGroup
ptype=RegExp
pattern=.*\[WINDOWS_AUTH\]\s+host=([^\s"]+)\s+MSWin_EventID=4625\s+SrcIP=
([^\s"]+)\s+Username=([^\s"]+).*
thresh=10
desc=Windows failed logon burst for user $3 from $2 to $1
action=eval %target_host ( "$1" ); \
       eval %client_ip ( "$2" ); \
       eval %user_lc ( lc("$3") ); \
       create BRUTE_FORCE_ACTIVE_%user_lc 1800; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
         "WINDOWS_COMPROMISE_SUSPECTED target=\"%target_host\" src_ip=\"%src_ip\"
username=\"%user_lc\""); \

init=eval %target_host ( "$1" ); eval %client_ip ( "$2" ); eval %user_lc ( lc("$3") )
end=delete BRUTE_FORCE_ACTIVE_%user_lc
window=300
multact=no

# 2) Detect successful login after failures
type=Single
ptype=RegExp
pattern=.*\[WINDOWS_AUTH\]\s+host=([^\s"]+)\s+MSWin_EventID=4624\s+SrcIP=
([^\s"]+)\s+Username=([^\s"]+).*
desc=Windows successful logon after failures (same user) $3
action=eval %target_host ( "$1" ); \
       eval %client_ip ( "$2" ); \
       eval %user_lc ( lc("$3") ); \
       if (BRUTE_FORCE_ACTIVE_%user_lc) \
         shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
           "WINDOWS_COMPROMISE_SUSPECTED user=\"%user_lc\" host=\"%target_host\"
src=\"%client_ip\""); \
       delete BRUTE_FORCE_ACTIVE_%user_lc; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit \
         "WINDOWS_COMPROMISE_SUSPECTED target=\"%target_host\" src_ip=\"%src_ip\"
username=\"%user_lc\""); \
```

LogZilla Trigger: Compromise Response

```
name: "Windows Account Compromise Response"
filter:
  - field: program
    op: eq
    value: SEC-WINDOWS-SECURITY
  - field: message
    op: "=~"
    value: "WINDOWS_COMPROMISE_SUSPECTED"
actions:
  exec_script: true
  script_path: "/usr/local/bin/windows-compromise-response.sh"
  send_email: true
  send_email_template: |
    Subject: CRITICAL: Windows Account Compromise

    User: {{event:ut:username}}
    Source IP: {{event:ut:SrcIP}}
    Target Host: {{event:ut:target}}

    Successful login detected after brute force attack.
    Account may be compromised - immediate investigation required.
```

Intelligent Response Script

```
#!/bin/bash
# /usr/local/bin/windows-compromise-response.sh
# Called by SEC shellcmd - receives data via command-line arguments

USERNAME="$1"
SRC_IP="$2"
TARGET_HOST="$3"

# Query Active Directory for account details
ACCOUNT_TYPE=$(ldapsearch -x -h dc.company.com -b "dc=company,dc=com" \
               "(sAMAccountName=$USERNAME)" memberOf | grep -c "Domain Admins")

# Check IP reputation
IP_REPUTATION=$(curl -s "https://threat-intel.company.com/ip/$SRC_IP")

if [[ "$ACCOUNT_TYPE" -gt 0 ]]; then
    # Domain admin account compromised - immediate lockdown
    logger -t SECURITY-RESPONSE "Domain admin compromise: $USERNAME"

    # Disable account immediately
    net user "$USERNAME" /active:no /domain

    # Create critical incident
    curl -X POST "https://servicedesk.company.com/api/incidents" \
         -d "priority=critical&subject=Domain Admin Compromise&user=$USERNAME"

    # Alert CISO immediately
    curl -X POST "https://slack.company.com/api/webhooks/ciso-alerts" \
         -d "text=CRITICAL: Domain admin $USERNAME compromised from $SRC_IP"
else
    # Standard user account
    logger -t SECURITY-RESPONSE "User account compromise: $USERNAME"

    # Force password reset
    net user "$USERNAME" /passwordreq:yes /domain

    # Create high-priority ticket
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=high&subject=Account Compromise&user=$USERNAME"
fi
```

Privilege Escalation Detection

Attack Pattern

Attackers often add compromised accounts to privileged groups, then clear audit logs to hide evidence.

SEC Rule: Privilege Escalation Campaign

```
# Track additions to local groups (parse from message)
type=Single
ptype=RegExp
pattern=(?s)\[WINDOWS_AUTH\].*?host=([^\s]+).*?MSWin_EventID=4732.*?Target_User_Name=
([^\s]+).*?Username=([^\s]+)
desc=User added to local group user=$3 group=$2 host=$1
action=create PRIV_ESC_HOST_$1 3600; \
       write /var/log/logzilla/sec/windows-security-messages.log \
         "PRIVILEGE_GRANTED %t user=$3 group=$2 host=$1"; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.warning "PRIVILEGE_GRANTED
user=\"$3\" group=\"$2\" host=\"$1\"")

# Detect audit log clearing after privilege escalation (correlate on host)
type=Single
ptype=RegExp
pattern=(?s)\[WINDOWS_AUTH\].*?host=([^\s]+).*?MSWin_EventID=1102\b
context=PRIV_ESC_HOST_$1
desc=Audit log cleared after privilege escalation host=$1
action=write /var/log/logzilla/sec/windows-security-messages.log \
         "PRIVILEGE_ESCALATION_ATTACK %t host=$1 (1102 after 4732)"; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit
"PRIVILEGE_ESCALATION_ATTACK host=\"$1\" correlation=\"1102_after_4732\""); \
       delete PRIV_ESC_HOST_$1
```

Service Flapping Detection

Operational Problem

Critical Windows services that repeatedly crash indicate system instability but generate too many individual alerts to be useful.

SEC Rule: Service Stability Monitoring

```
# Detect repeated service failures (7031) using regex-parsed service name
type=SingleWithThreshold
ptype=RegExp
pattern=(?s)\[WINDOWS_SERVICE\].*?MSWin_EventID=7031.*?Service_Name=\"([^\"]+)\".*?
terminated unexpectedly
desc=Service flapping detected service="$1"
thresh=5
window=600
action=write /var/log/logzilla/sec-windows-service-messages.log \
  SWT-HIT %t service="$1"; \
       write /var/log/logzilla/sec-windows-service-messages.log \
         SERVICE-FLAPPING 7031 %t service="$1" failures="5"
  shellcmd (logger -t SEC-WINDOWS-OPERATIONS -p local0.warning "SERVICE_FLAPPING
service=\"$1\" failures=\"5\"")
```

Advanced Windows Correlation Patterns

Malware Persistence Detection

Correlate suspicious process creation with service installation and network connections to detect malware establishing persistence.

```
# Track suspicious process execution from user directories (4688 comes first)
type=Single
ptype=RegExp
pattern=\[WINDOWS_PROCESS\].*?MSWin_EventID=4688.*?Process_Name="([^"]+)".*?
Process_Path="C:\\Users\\.*?\\([^"\\]+)".*?Username="([^"]*)".*?User_Domain_Name="([^"]*)"
desc=Suspicious process from user directory filename=$2 account=$4\\$3
action=eval %proc_user ( "$3" ); \
       eval %proc_domain ( "$4" ); \
       eval %proc_name ( "$1" ); \
       create SUSPICIOUS_PROC_$2 1800; \
       write /var/log/logzilla/sec/windows-process.log \
         SUSPICIOUS-PROCESS %t filename="$2" process_name="$1" account="$4\\$3"; \
       shellcmd (logger -t SEC-WINDOWS-MALWARE "SUSPICIOUS_PROCESS filename=\"$2\"
process_name=\"$1\" account=\"$4\\\\$3\"")

# Correlate with service installation (7045) to detect malware persistence
type=Single
ptype=RegExp
pattern=\[WINDOWS_PROCESS\].*?MSWin_EventID=7045.*?Service_Account="([^"]*)".*?
Service_Filename="C:\\Users\\.*?\\([^"\\]+)".*?Service_Name="([^"]+)"
context=SUSPICIOUS_PROC_$2
desc=Malware persistence detected - process to service service=$3 filename=$2
svc_account=$1
action=write /var/log/logzilla/sec/windows-process.log \
         MALWARE-PERSISTENCE %t service="$3" filename="$2" service_account="$1"
process_account="%proc_domain\\%proc_user" process_name="%proc_name"; \
       shellcmd (logger -t SEC-WINDOWS-SECURITY -p local0.crit "MALWARE_PERSISTENCE
service=\"$3\" filename=\"$2\" service_account=\"$1\"
process_account=\"%proc_domain\\\\%proc_user\"")
```

PowerShell Attack Chain Detection

Detect PowerShell-based attacks by correlating execution with credential access and network activity.

```
# Track PowerShell execution
type=Single
ptype=RegExp
pattern=\[WINDOWS_PROCESS\][\s\S]*?MSWin_EventID=4104[\s\S]*?Script_Path="([^"]*)"[\s\S]*?
User_Domain_Name="[^"]*"\s*([\s\S]*)
desc=PowerShell script execution detected
action=eval %ps_script_path ( "$1" ); \
       eval %ps_message ( "$2" ); \
       create POWERSHELL_ACTIVE_%ps_script_path 900; \
       eval %cred_access ( ( %ps_message =~ /(Invoke-Mimikatz|Get-Credential|ConvertTo-
SecureString)/i ) ? 1 : 0 ); \
       shellcmd ([ %cred_access -eq 1 ] && logger -t SEC-WINDOWS-SECURITY -p local0.alert
"POWERSHELL_CREDENTIAL_ACCESS script_path=\"%ps_script_path\"" || true)
```

Related Topics

- SOAR Security Orchestration
- Brute Force Attack Detection
- Event Correlation Rule Types
- Windows Event Forwarding
- Creating Triggers

Advanced Event Correlation Walkthrough

Step-by-step walkthrough for building a LogZilla SEC Pair rule that tracks interface up/down events, calculates downtime, and alerts on extended outages

Overview

Event Correlation implementation using network interface monitoring provides administrators with practical experience creating correlation rules. The process involves tracking interface up/down events, calculating downtime durations, and generating critical alerts for extended outages.

Prerequisites: Before starting this walkthrough, review Event Correlation Overview for setup requirements and enable Event Correlation on the LogZilla server.

Create a Test Correlation Rule

The following steps create a simple correlation rule that tracks network interface up/down events and calculates downtime.

Create the Correlation Rule File

Create the SEC instance directory structure and rule file:

```
sudo mkdir -p /etc/logzilla/sec/interface-monitoring/rules
sudo cat << 'EOF' > /etc/logzilla/sec/interface-monitoring/rules/interface-
correlation.sec
# Interface Up/Down Correlation Rule
# Tracks interface state changes and calculates downtime
# Note: continue=TakeNext allows the same event to be processed by multiple rules

type=Pair
ptype=RegExp
continue=TakeNext
pattern=.*Interface (\S+) is down$
desc=Interface $1 Down - Starting Timer
action=eval %interface_name ( "$1" ); \
    eval %down_time ( time() ); \
    shellcmd (/usr/bin/logger -T -n $SYSLOG_HOSTNAME -P $SYSLOG_BSD_TCP_PORT \
    -t EC-INTERFACE -p local0.info \
    "Interface $1 outage started at $(date)")
ptype2=RegExp
pattern2=.*Interface ($1) is up$
desc2=Interface $1 Up - Calculating Downtime
action2=eval %up_time ( time() ); \
    eval %downtime_seconds ( %up_time - %down_time ); \
    eval %downtime_minutes ( int( %downtime_seconds / 60 ) ); \
    shellcmd (/usr/bin/logger -T -n $SYSLOG_HOSTNAME -P $SYSLOG_BSD_TCP_PORT \
    -t EC-INTERFACE -p local0.notice \
    "Interface %interface_name restored after %downtime_seconds seconds
(%downtime_minutes minutes)")
EOF
```

Configure Event Forwarding to the correlation instance

Create a forwarder configuration to route interface events to the SEC instance:

Create Forwarder Configuration

Note: ensure the forwarding engine is enabled as noted in Event Correlation Overview

```
sudo cat << 'EOF' > /etc/logzilla/forwarder.d/interface-monitoring.yaml
match:
  - field: message
    op: "=~"
    value: "Interface .* is .*"
type: sec
name: "Interface Monitoring Forwarder"
sec_name: interface-monitoring
EOF
```

Configuration Fields:

- `name`: Human-readable description for the forwarder (for debugging/identification)
- `sec_name`: Must match the SEC instance directory name (`/etc/logzilla/sec/interface-monitoring/`)

Verify Forwarder Rule

```
logzilla forwarder print
```

Apply Forwarder Changes

```
logzilla settings reload forwardermodule
```

Create a Critical Alert Rule

Add a second rule that generates critical alerts for extended outages:

Create Critical Alert Rule File

```
sudo cat << 'EOF' > /etc/logzilla/sec/interface-monitoring/rules/critical-alerts.sec
# Critical Alert for Extended Outages
# Alerts if interface stays down for more than 5 minutes

type=PairWithWindow
ptype=RegExp
pattern=.*Interface (\S+) is down$
desc=Interface $1 Down - Monitoring for Extended Outage
action=shellcmd (/usr/bin/logger -T -n $SYSLOG_HOSTNAME -P $SYSLOG_BSD_TCP_PORT \
        -t EC-CRITICAL -p local0.alert \
        "CRITICAL: Interface $1 has been down for over 60 seconds")
ptype2=RegExp
pattern2=.*Interface ($1) is up$
desc2=Interface $1 Up - Canceling Critical Alert
action2=shellcmd (/usr/bin/logger -T -n $SYSLOG_HOSTNAME -P $SYSLOG_BSD_TCP_PORT \
        -t EC-INTERFACE -p local0.info \
        "Interface $1 restored before critical threshold")
window=300
EOF
```

Create Test Event Generator

Create a script to generate test events that will trigger the correlation rules.

Create Test Script

```
sudo cat << 'EOF' > /tmp/test-interface-events.sh
#!/bin/bash

# Test Event Generator for Interface Correlation
# Generates interface up/down events to test correlation rules

INTERFACE_NAME="eth0"
LOGZILLA_HOST="127.0.0.1"  # Change to LogZilla server IP if running remotely
LOGZILLA_PORT="514"
LOGGER_CMD="/usr/bin/logger -T -n $LOGZILLA_HOST -P $LOGZILLA_PORT -t TestDevice -p
local0.info"

echo "Starting interface correlation test..."
echo "This will generate interface down/up events with a 30-second gap"
echo "Watch LogZilla UI for correlation results"
echo ""

# Generate interface down event
echo "$(date): Sending interface down event"
$LOGGER_CMD "Interface $INTERFACE_NAME is down"

# Wait 30 seconds
echo "Waiting 30 seconds before sending up event..."
sleep 30

# Generate interface up event
echo "$(date): Sending interface up event"
$LOGGER_CMD "Interface $INTERFACE_NAME is up"

echo "Test complete. Check LogZilla for correlation results."
echo "Look for events with Program = EC-INTERFACE"
EOF
```

Make Script Executable

```
sudo chmod +x /tmp/test-interface-events.sh
```

Create Extended Outage Test

Create a second test for the critical alert rule:

Create Extended Test Script

```
sudo cat << 'EOF' > /tmp/test-critical-alert.sh
#!/bin/bash

# Extended Outage Test - Triggers Critical Alert
# Generates interface down event and waits 6 minutes

INTERFACE_NAME="eth1"
LOGZILLA_HOST="127.0.0.1"  # Change to LogZilla server IP if running remotely
LOGZILLA_PORT="514"
LOGGER_CMD="/usr/bin/logger -T -n $LOGZILLA_HOST -P $LOGZILLA_PORT -t TestDevice -p
local0.info"

echo "Starting critical alert test..."
echo "This will generate an interface down event and wait 6 minutes"
echo "A critical alert should appear after 5 minutes"
echo ""

# Generate interface down event
echo "$(date): Sending interface down event for $INTERFACE_NAME"
$LOGGER_CMD "Interface $INTERFACE_NAME is down"

echo "Waiting 6 minutes to trigger critical alert..."
echo "Watch LogZilla UI - critical alert should appear after 5 minutes"
sleep 360

# Generate interface up event
echo "$(date): Sending interface up event"
$LOGGER_CMD "Interface $INTERFACE_NAME is up"

echo "Extended test complete."
EOF
```

Make Extended Script Executable

```
sudo chmod +x /tmp/test-critical-alert.sh
```

Restart Event Correlation Service

```
logzilla settings reload sec
```

Execute Tests and Verify Results

Run Basic Correlation Test

Execute the basic interface correlation test:

```
/tmp/test-interface-events.sh
```

Monitor Results in LogZilla UI

Open the LogZilla web interface

Navigate to the Events page

Apply one of these filters:

- Program = `TestDevice` (raw input events)
- Program = `EC-INTERFACE` (correlated output events)

Expected results:

- Initial down event with timestamp
- Correlation result showing downtime calculation

Run Critical Alert Test

In a separate terminal, execute the extended outage test:

```
/tmp/test-critical-alert.sh
```

Verify Critical Alert

In LogZilla UI, filter by Program = `EC-CRITICAL`

After 5 minutes, a critical alert should appear

The alert should show the interface name and duration

Understanding the Results

Expected Output

For the basic test, administrators should see events similar to:

```
Program: EC-INTERFACE
Message: Interface eth0 down at 1640995200

Program: EC-INTERFACE
Message: Interface eth0 restored after 30 seconds (0 minutes)
```

For the critical test:

```
Program: EC-CRITICAL
Message: CRITICAL: Interface eth1 has been down for 5 minutes
```

How the Correlation Works

Pair Rule: Matches interface down events and waits for corresponding up events

Time Calculation: Event Correlation calculates the difference between down and up timestamps

Event Generation: New enriched events are sent back to LogZilla with calculated downtime

Window Rule: Monitors for events that don't have a matching pair within the specified time window

Advanced Configuration

Modify Time Windows

To change the critical alert threshold, edit the rule file:

```
sudo nano /etc/logzilla/sec/interface-monitoring/rules/critical-alerts.sec
```

Change `window=300` to desired seconds (e.g., `window=600` for 10 minutes).

Add More Complex Patterns

Administrators can extend rules to match different event formats:

```
# Match multiple interface formats
pattern=(Interface|Port|Link) (\S+) (is down|down|failed)
```

Apply Configuration Changes

After making changes to correlation rules, restart the Event Correlation service to load the new configuration:

```
sudo logzilla restart -c sec
```

Troubleshooting

Verify Basic System Health

First, ensure the SEC container is running and accessible:

```
# Check SEC container status
sudo docker ps | grep lz_sec

# Review SEC logs for errors
sudo docker logs lz_sec
```

Validate Configuration

Verify Forwarder Configuration

Verify the forwarder configuration is syntactically correct:

```
logzilla forwarder print
```

This command displays the parsed configuration and reports any syntax errors.

Verify SEC Rule Syntax

Validate the SEC rule files for syntax errors:

```
sudo docker exec -it lz_sec sec --testonly --conf=/etc/logzilla/sec/interface-
monitoring/rules/interface-correlation.sec
```

This command parses the SEC rule file and reports any syntax errors without starting the correlation engine.

Test Event Flow

1. Verify Events Reach LogZilla

Check that the test events are being received:

- Navigate to the Events page in LogZilla UI
- Filter by Program = `TestDevice`
- Confirm interface up/down events appear

2. Test SEC Connectivity

Verify SEC can send events back to LogZilla:

```
# Check SEC environment variables
sudo docker exec -it lz_sec printenv | grep SYSLOG

# Test TCP connectivity (preferred)
sudo docker exec -it lz_sec sh -c \
  '/usr/bin/logger -T -n "$SYSLOG_HOSTNAME" -P "$SYSLOG_BSD_TCP_PORT" \
  -t EC-TEST "SEC connectivity test"'
```

If the test event doesn't appear in LogZilla, try UDP as a fallback:

```
sudo docker exec -it lz_sec sh -c \
  '/usr/bin/logger -n "$SYSLOG_HOSTNAME" -P "$SYSLOG_BSD_UDP_PORT" \
  -t EC-TEST "SEC UDP connectivity test"'
```

Note: Using `sh -c` ensures environment variables are expanded inside the container.

Common Issues

Events Not Correlating

- **Pattern mismatch**: Ensure the SEC rule patterns match the exact event format. Test events may include additional fields (hostname, program) that need to be accounted for in patterns.
- **Timing issues**: SEC rules have time windows - events outside the window won't correlate
- **Case sensitivity**: SEC pattern matching is case-sensitive
- **Events not reaching SEC**: Check forwarder configuration and ensure events match the forwarder's filter criteria

No Correlation Output

- **SEC rules not loading**: Check `docker logs lz_sec` for rule syntax errors
- **Forwarder not routing**: Verify forwarder configuration with `logzilla forwarder print`
- **Network connectivity**: Test SEC → LogZilla connectivity using the test commands above

Pattern Debugging Example

If SEC receives events like:

```
clitests TestDevice Interface eth0 is down
```

But the rule pattern is:

```
pattern=Interface (\S+) is down$
```

The pattern won't match. Fix it with:

```
pattern=.*Interface (\S+) is down$
```

Related Documentation

- Event Correlation Rule Types
- Creating Triggers

CHAPTER 07

Alerts

LogZilla alerts deliver real-time notifications, script automation, trigger import and export, and outgoing webhook integrations with platforms like Slack

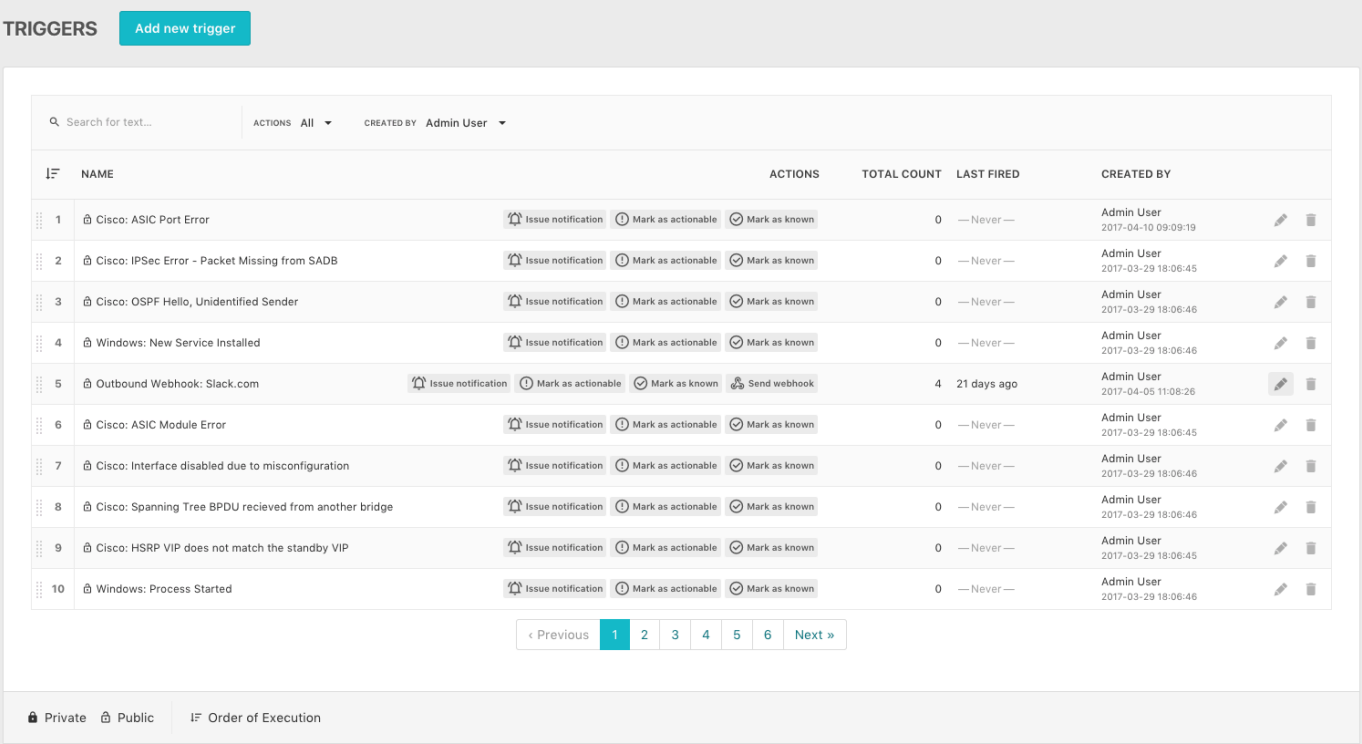
LogZilla alerts play an integral role within the platform. Users will find a concise, step-by-step exploration designed to enhance understanding and usage of LogZilla's alert features.

- **Alerts Overview**: An exploration of the alert mechanism of LogZilla. Users can learn how to customize, notify, and respond to system events in real-time.
- **LogZilla Automation**: Information about LogZilla's built-in automation capabilities. Covers utilizing user-contributed scripts and manipulating data using environment variables.
- **Trigger Import and Export**: Hands-on experience managing triggers. Covers the export and import functionalities, ensuring a seamless setup process.
- **Outgoing Webhooks**: Integration of LogZilla alerts with platforms like Slack. Covers the process of setting up notifications for external services.

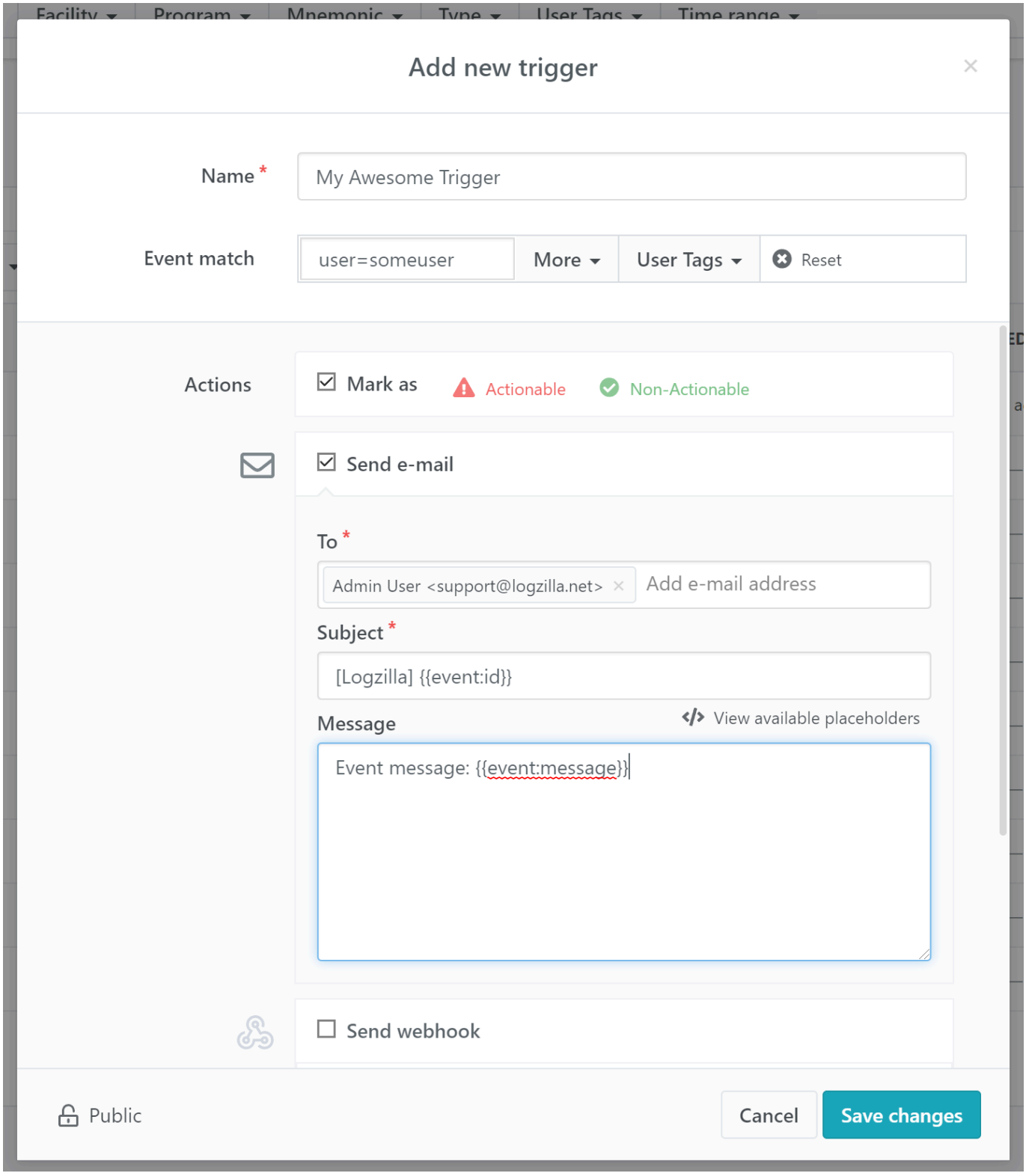
Alerts Overview

LogZilla triggered alerts provide northbound notification, automation, and correlation hooks that fire on filters for severity, hostname, or search terms

Triggered alerts provide various northbound mechanisms to allow the system to notify, automate, and even correlate events which allow users to stay on top of potential problems, track system configuration changes, automate repetitive tasks, etc.

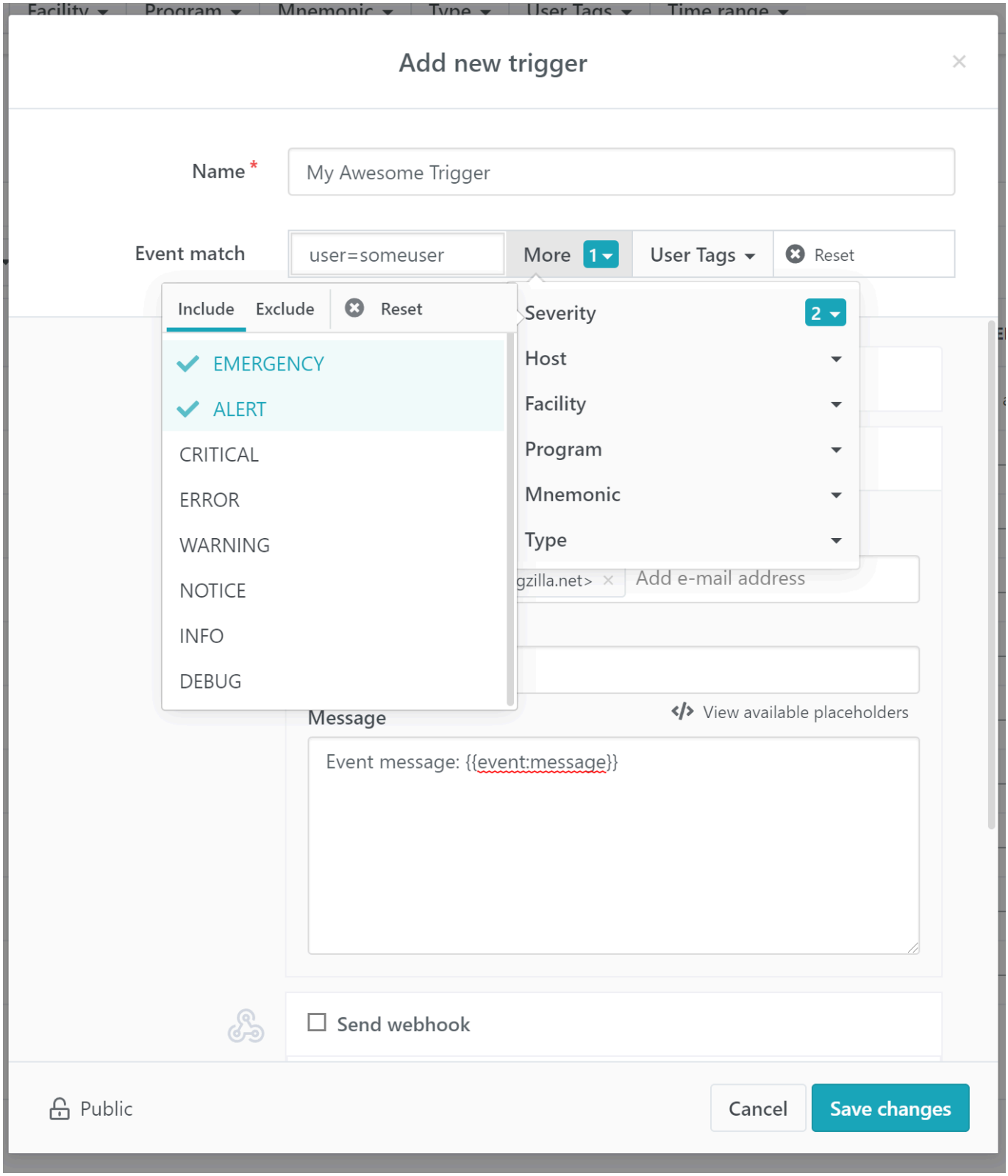


Clicking the 'Add New Trigger' button from the Triggers menu will allow users to create a new trigger and set the required filters to match on.

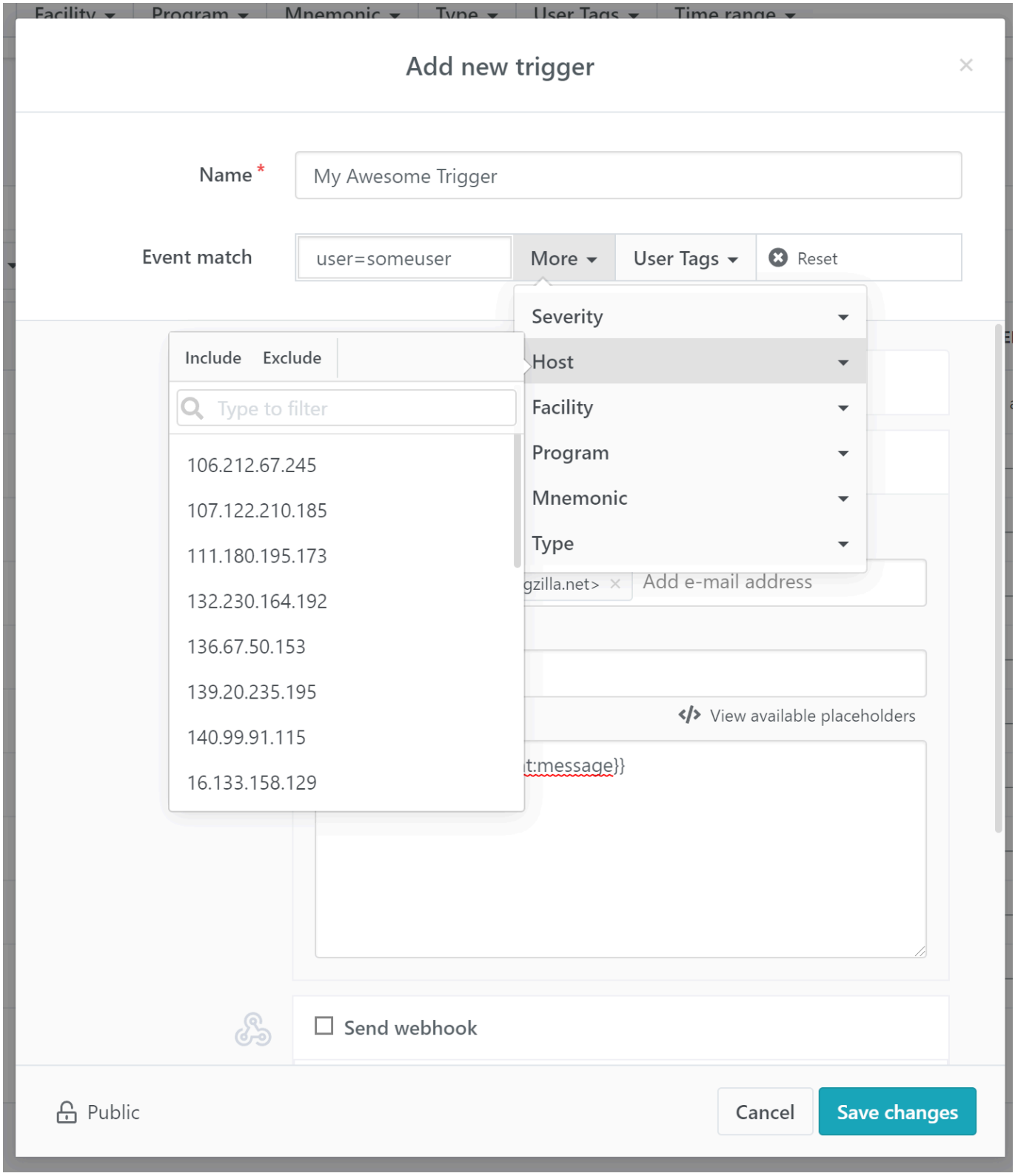


Notifications can be created based on search terms or any of the other fields listed such as `Severity`, `Hostname`, etc.

Severity Filter



Host Filter


Facility
Program
Mnemonic
Type
User Tags
Time range
Add new trigger
Name *
My Awesome Trigger
Event match
user=someuser
More
User Tags
Reset
Severity
Host
Facility
Program
Mnemonic
Type
Include
Exclude
Type to filter
106.212.67.245
107.122.210.185
111.180.195.173
132.230.164.192
136.67.50.153
139.20.235.195
140.99.91.115
16.133.158.129
gzilla.net>
Add e-mail address
View available placeholders
t:message}}
Send webhook
Public
Cancel
Save changes

Automations

Automate LogZilla responses with SOAR-style trigger actions that run scripts, post webhooks, send email, and remediate events like Cisco interface outages

LogZilla's automation capabilities provide Security Orchestration, Automation, and Response (SOAR) functionality that enables organizations to respond to events automatically, reducing manual intervention and improving response times. Users can create automated workflows that execute scripts, send webhooks, generate notifications, and orchestrate responses across multiple systems.

How Automation Works

LogZilla automation operates through triggers that execute actions when specific events occur. When a trigger matches an event, it can:

- Execute custom scripts with full event context
- Send HTTP webhooks to external systems
- Generate email notifications with event details
- Create internal notifications and alerts
- Mark events for classification and filtering

All automation actions receive complete event information through environment variables, enabling context-aware responses.

Practical Automation Examples

The following examples demonstrate real-world automation scenarios users can implement:

Network Interface Recovery

Scenario: A Cisco interface goes down unexpectedly.

Automation: A trigger detects the `%LINK-3-UPDOWN` event and executes a Python script that:

Uses python to SSH into the device
Runs `no shutdown` on the affected interface
Sends a Slack notification confirming the remediation
Implementation: The script receives the device hostname via the `EVENT_HOST` environment variable and interface details parsed from `EVENT_MESSAGE`.

Duplex Mismatch Correction

Scenario: A CDP duplex mismatch is detected on a network link.

Automation: When LogZilla receives a `%CDP-4-DUPLEX_MISMATCH` event, a trigger executes a script that:

Connects to both devices involved in the mismatch
Configures both interfaces to use the same duplex setting
Logs the corrective action taken

Service Restart with Threshold

Scenario: A critical service fails and needs automatic recovery with escalation limits.

Automation: Using event correlation rules, the system:

Detects service failure events
Attempts to restart the service automatically
Tracks restart attempts (maximum 3 times in 5 minutes)
Escalates to human operators if automatic recovery fails

Flapping Interface Detection

Scenario: A network interface is unstable, repeatedly going up and down.

Automation: Event correlation identifies the pattern:

Correlates interface down/up event pairs within a time window
Counts "short outage" occurrences
Generates an alert when flapping exceeds threshold (e.g., 10 times in 6 hours)
Notifies network team of unstable interface requiring investigation

Security Incident Response

Scenario: Multiple failed login attempts detected from a single source.

Automation: A trigger identifies the pattern and:

Automatically disables the affected user account
Creates a security incident ticket in ServiceNow
Sends alert to security team with user and source IP details
Blocks the source IP at the firewall if attempts continue

Malware Communication Detection

Scenario: A host makes numerous outbound connections to suspicious destinations.

Automation: When firewall logs show high-volume connection attempts:

Cross-reference destination IPs with threat intelligence feeds
Automatically block confirmed malicious IPs at the firewall

Quarantine the source host via NAC integration

Create high-priority security incident for investigation

ServiceNow Integration

Scenario: Automatically create and update incident tickets for critical events.

Automation: A trigger sends HTTP POST requests to ServiceNow's REST API:

Creates incident tickets with event details

Uses correlation IDs to prevent duplicate tickets

Updates existing tickets when related events occur

Includes event context, severity, and affected systems

Implementation: The webhook payload includes LogZilla variables like `{{event:host}}`, `{{event:message}}`, and `{{event:severity}}` to populate ServiceNow fields automatically.

Interactive Slack Automation

Scenario: Network interface down with human-in-the-loop remediation.

Automation: When an interface goes down:

Sends Slack message with interface details and "Remediate" button

Operator clicks button to authorize automatic remediation

LogZilla receives postback and executes interface recovery script

Sends confirmation message with remediation results

Implementation Methods

Script Execution

Scripts receive complete event context through environment variables:

```
EVENT_HOST=router01.company.com
EVENT_MESSAGE=%LINK-3-UPDOWN: Interface GigabitEthernet0/1, changed state to down
EVENT_SEVERITY=3
EVENT_FACILITY=23
EVENT_PROGRAM=LINK
```

Scripts can use this information to make intelligent, context-aware decisions about remediation actions.

Event Correlation

Complex automation scenarios use the Simple Event Correlator (SEC) to:

- **PairWithWindow**: Match related events within time windows

- **SingleWithThreshold**: Count occurrences and trigger on thresholds
- **EventGroup**: Correlate multiple event types for complex patterns

Webhook Integration

Send structured data to external APIs using HTTP POST requests with JSON payloads containing event variables and custom fields.

LogZilla SOAR Advantages

LogZilla has provided SOAR capabilities for years before the term "Security Orchestration, Automation, and Response" became widely adopted. This established foundation offers several advantages:

- **Proven Reliability**: Battle-tested automation engine with years of production deployment
- **High Performance**: Purpose-built for high-volume event processing and real-time response
- **Flexible Integration**: Extensive API and webhook support for security tool orchestration
- **Cost Effective**: SOAR functionality without the complexity and licensing costs of dedicated SOAR platforms
- **Unified Platform**: Combines log management, SIEM capabilities, and SOAR automation in a single solution

Getting Started

1. Identify Automation Opportunities

Look for repetitive manual tasks triggered by specific events:

- Service restarts after failures
- Network device configuration issues
- Security incidents requiring immediate response
- Ticket creation for critical alerts

2. Start Simple

Begin with basic automations before building complex workflows:

- Email notifications for critical events
- Slack alerts with event details
- Simple script execution for common fixes

3. Build Context-Aware Scripts

Use environment variables to create intelligent automations:

- `EVENT_HOST` for device-specific actions
- `EVENT_MESSAGE` for parsing specific details
- `EVENT_SEVERITY` for response prioritization

- `EVENT_USER_TAGS` for enriched context

4. Implement Safeguards

Include error handling and limits in automation scripts:

- Maximum retry attempts with backoff
- Validation before making changes
- Logging of all automated actions
- Escalation paths when automation fails

Advanced Scenarios

Domain Trust Enumeration Detection

Detect reconnaissance activity by correlating events from multiple domain controllers when a single source attempts to enumerate domain trusts across several systems within a short timeframe.

Windows Event Log Service Monitoring

Monitor for Windows Event Log service failures and automatically restart the service, escalating to administrators if the service fails to stay running.

Technical Implementation

For detailed technical information on implementing automation scripts, including environment variables, script requirements, and Docker container setup, see the Trigger Scripts documentation.

For complex event correlation scenarios, see the Event Correlation documentation.

Trigger Import Export

Import and export LogZilla triggers as JSON from the Tools menu or command line, with duplicate detection to prevent overwriting existing rules

Trigger Import and Export

LogZilla Triggers are stored in standard JSON format and may be imported and exported from both the UI and the command line.

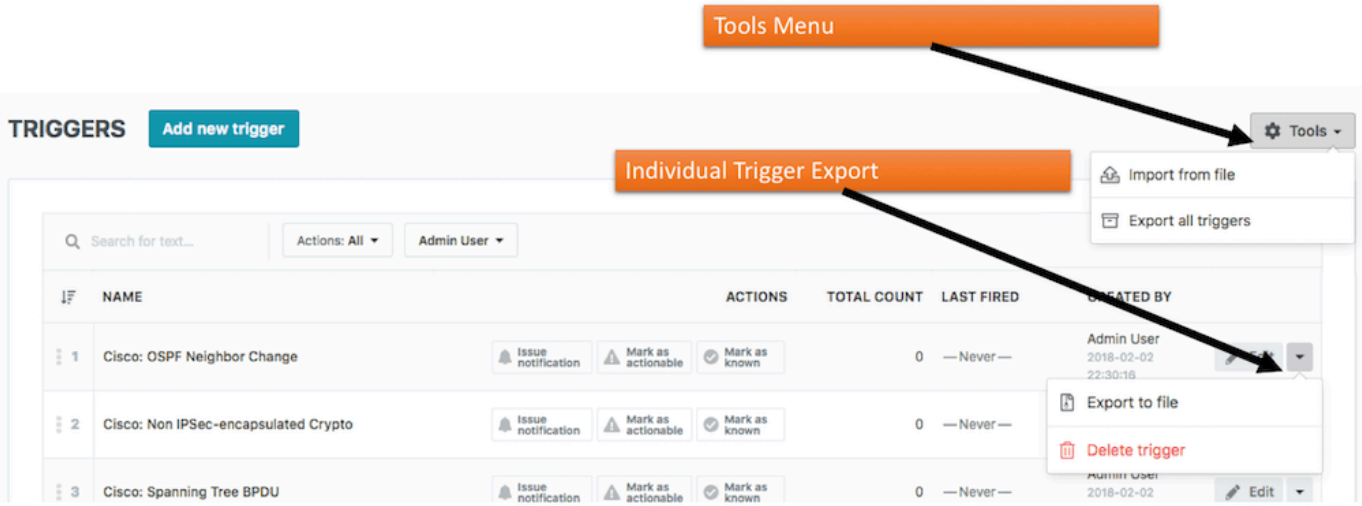
Import/Export From UI

Exporting Triggers

Users may export all triggers or individual triggers by selecting either the **Tools** menu or an individual trigger's **edit** menu dropdown.

In either case, selecting the **export** option will prompt for the filename and location to be saved to.

Trigger Import/Export Menus

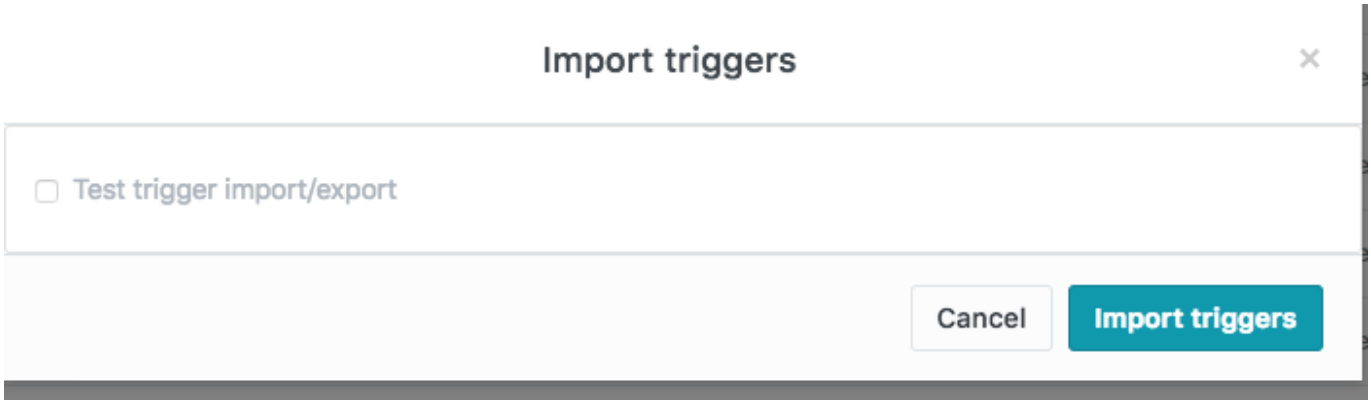


Importing Triggers

The **Tools** menu also includes an option to import triggers.

During individual trigger import, a check is made to ensure that the trigger being imported is not a duplicate of an existing trigger. If the import is a duplicate, the option to click the checkbox for that trigger will not be available.

Trigger Import/Export - Unable to select due to existing trigger



Trigger Import/Export - Trigger import passes test, select to proceed

Command Line

Import

The output below shows the syntax for importing triggers from the command line.

Available options are:

- `-I` or `--input-file` : the name of the file to import
- `--owner` : an optional username to assign as the owner/creator of that trigger

```
# logzilla triggers import -h
usage: triggers import [-h] [-I INPUT_FILE] [--owner OWNER] [name]

positional arguments:
  name                  name filter. To use wildcard put word in quotation
                        marks e.g.: "*cisco*"

optional arguments:
  -h, --help            show this help message and exit
  -I INPUT_FILE, --input-file INPUT_FILE
                        import triggers from file
  --owner OWNER         set owner for imported triggers. Default "admin"
```

Export

The output below shows the syntax for exporting triggers from the command line.

Available options are:

- List all available triggers (`-l`)
- `-O` or `--output-file` : the name of the file the triggers will be exported to
- `-F yaml` or `-F json` : the format of the export file
- `--owner` : only export triggers belonging to the specified owner
- `--trigger-id` : only export the specified (by id) trigger
- `--with-built-in` : include built-in triggers in the export (by default they are not included)

```
# logzilla triggers export -h
usage: triggers export [-h] [-O OUTPUT_FILE] [-F {yaml,json}] [--owner OWNER]
                       [--trigger-id TRIGGER_ID] [--with-built-in]
                       [name]

positional arguments:
  name                  name filter. To use wildcard put word in quotation
                        marks e.g.: "*cisco*"

optional arguments:
  -h, --help            show this help message and exit
  -O OUTPUT_FILE, --output-file OUTPUT_FILE
                        file to write triggers to
  -F {yaml,json}, --format {yaml,json}
                        export format
  --owner OWNER         limit triggers to those owned by given user
  --trigger-id TRIGGER_ID
                        trigger-id filter
  --with-built-in       show built-in triggers. By default built-in triggers
                        are hidden
```

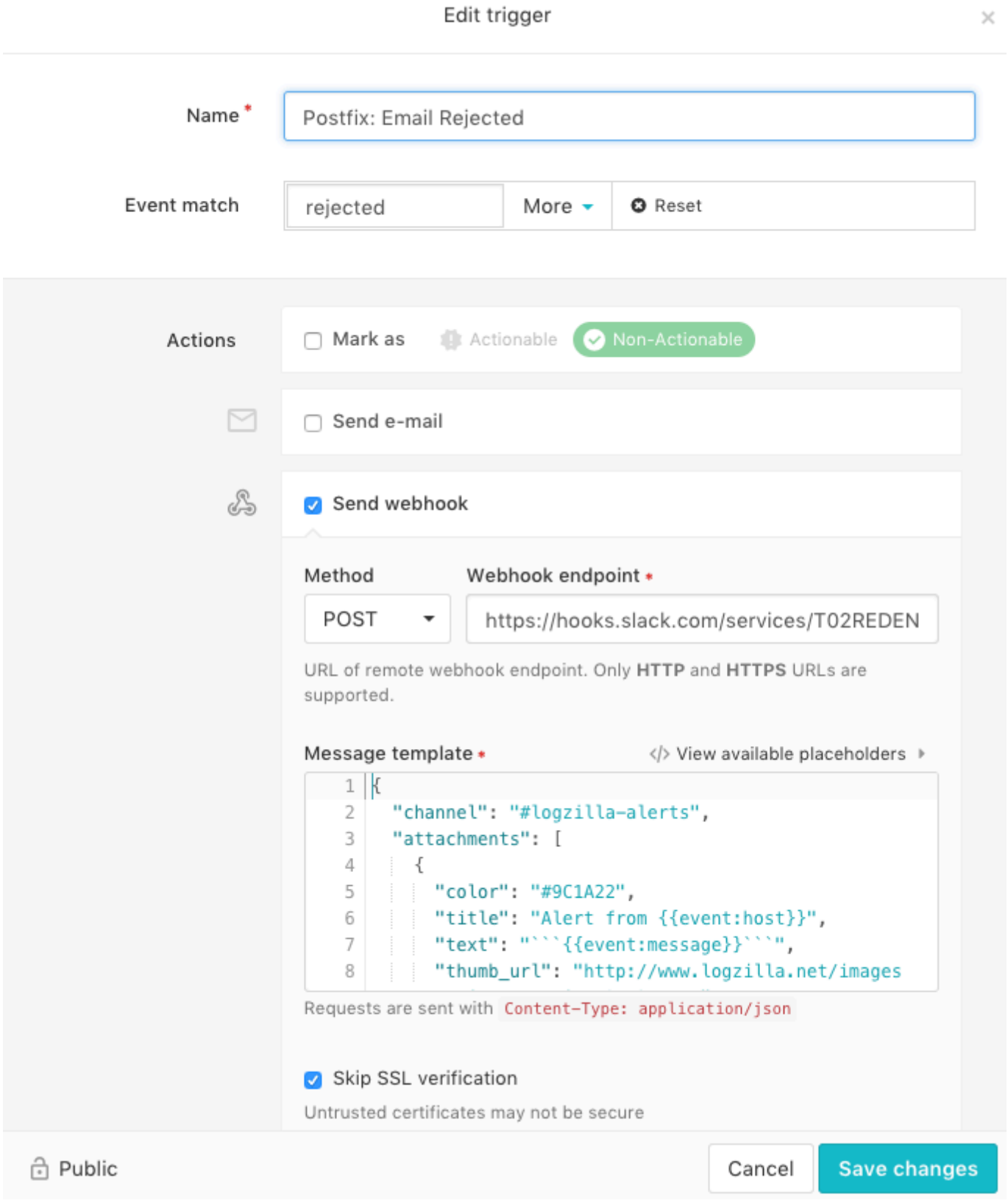
Outgoing Webhooks

Send LogZilla trigger events to Slack, PagerDuty, or any HTTP endpoint using GET or POST webhooks with JSON payload templates and event variable substitution

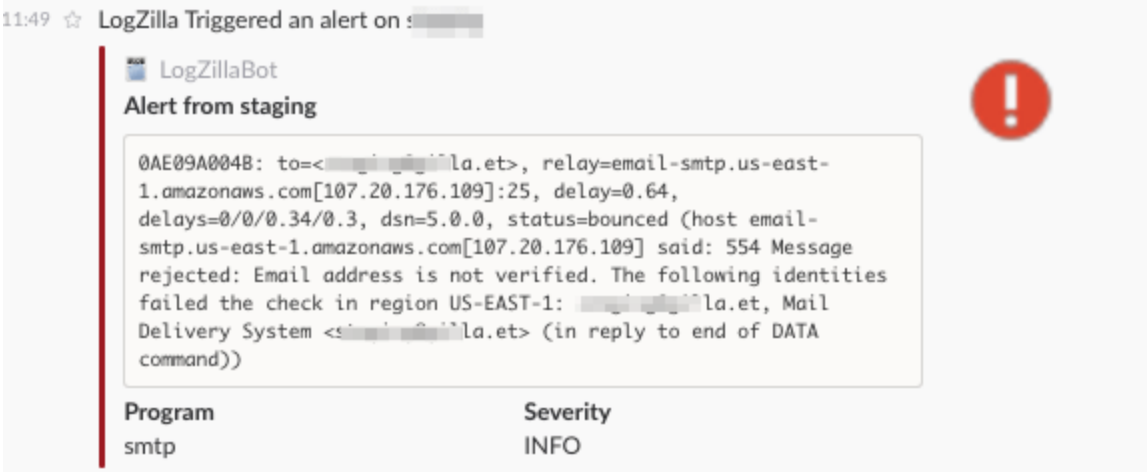
The Webhook option in the trigger menu allows users to send a `GET` or `POST` webhook command to a northbound server such as slack.com

The example below shows a trigger set to match `rejected` messages from Postfix. When the trigger matches, it will post to a Slack channel automatically.

Webhook Post to Slack



Alert Shown in Slack Channel



Post Data Used

The Trigger above uses the following JSON:

```
{
  "channel": "#logzilla-alerts",
  "attachments": [
    {
      "color": "#9C1A22",
      "title": "Alert from {{event:host}}",
      "text": "```{{event:message}}```",
      "thumb_url": "http://www.logzilla.net/images/icon_warning_25x25.png",
      "fallback": "Alert from {{event:host}}",
      "author_icon": "http://www.logzilla.net/images/log_file_icon_25x25.png",
      "pretext": "LogZilla Triggered an alert on {{event:host}}",
      "author_link": "mailto:support@logzilla.net",
      "fields": [
        {
          "value": "{{event:program}}",
          "title": "Program",
          "short": "true"
        },
        {
          "value": "{{event:severity}}",
          "short": "true",
          "title": "Severity"
        }
      ],
      "mrkdwn_in": [
        "text",
        "fields"
      ],
      "author_name": "LogZillaBot"
    }
  ],
  "username": "logzilla-bot",
  "icon_url": "http://www.logzilla.net/images/logo_orange_png_cropped_40x40.png"
}
```

CHAPTER 08

Forwarding Module

LogZilla Forwarding Module deduplicates events at ingest and delivers the compressed stream to syslog, SNMP, Splunk, file, and Event Correlation destinations

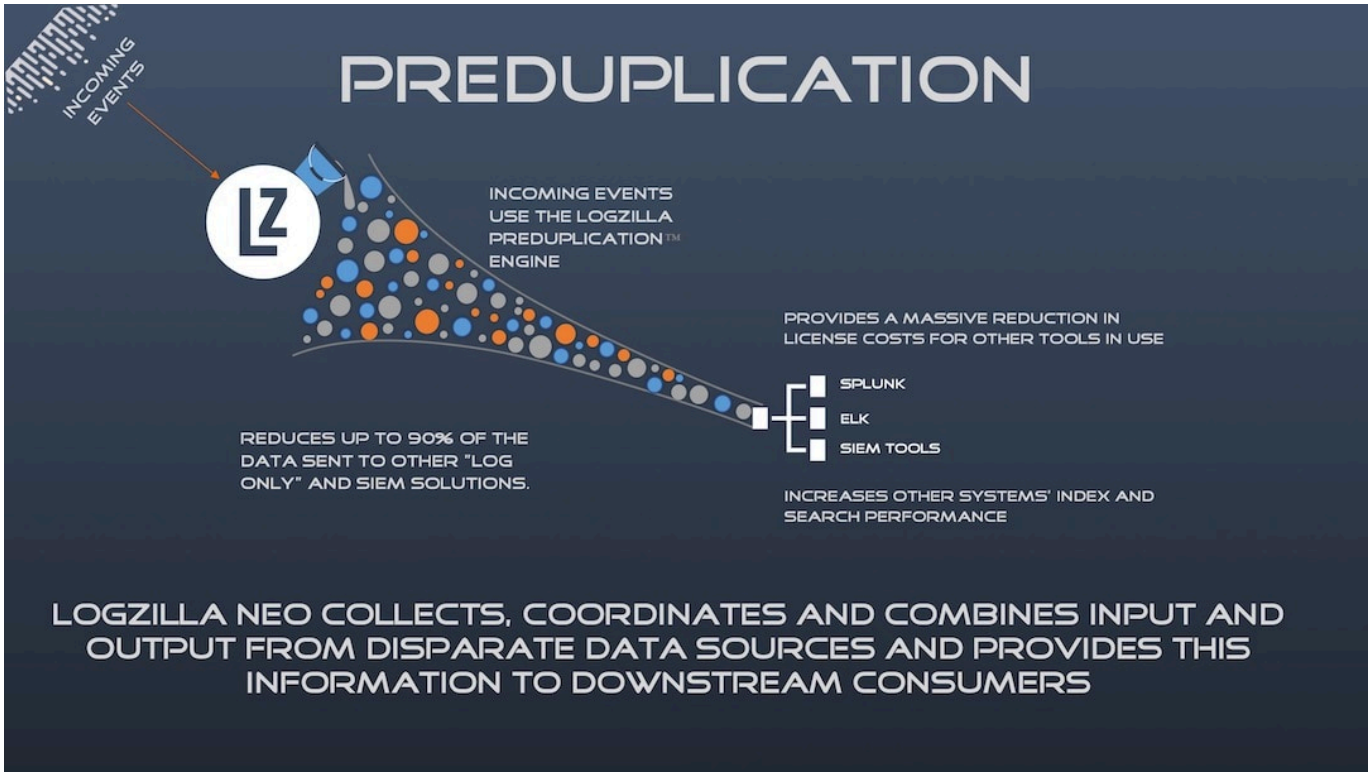
Dedup Forwarder Introduction

LogZilla collapses repeated events at ingest through deduplication windows, then forwards the reduced stream to syslog, file, Splunk HEC, or SNMP receivers

Deduplication and Forwarder Overview

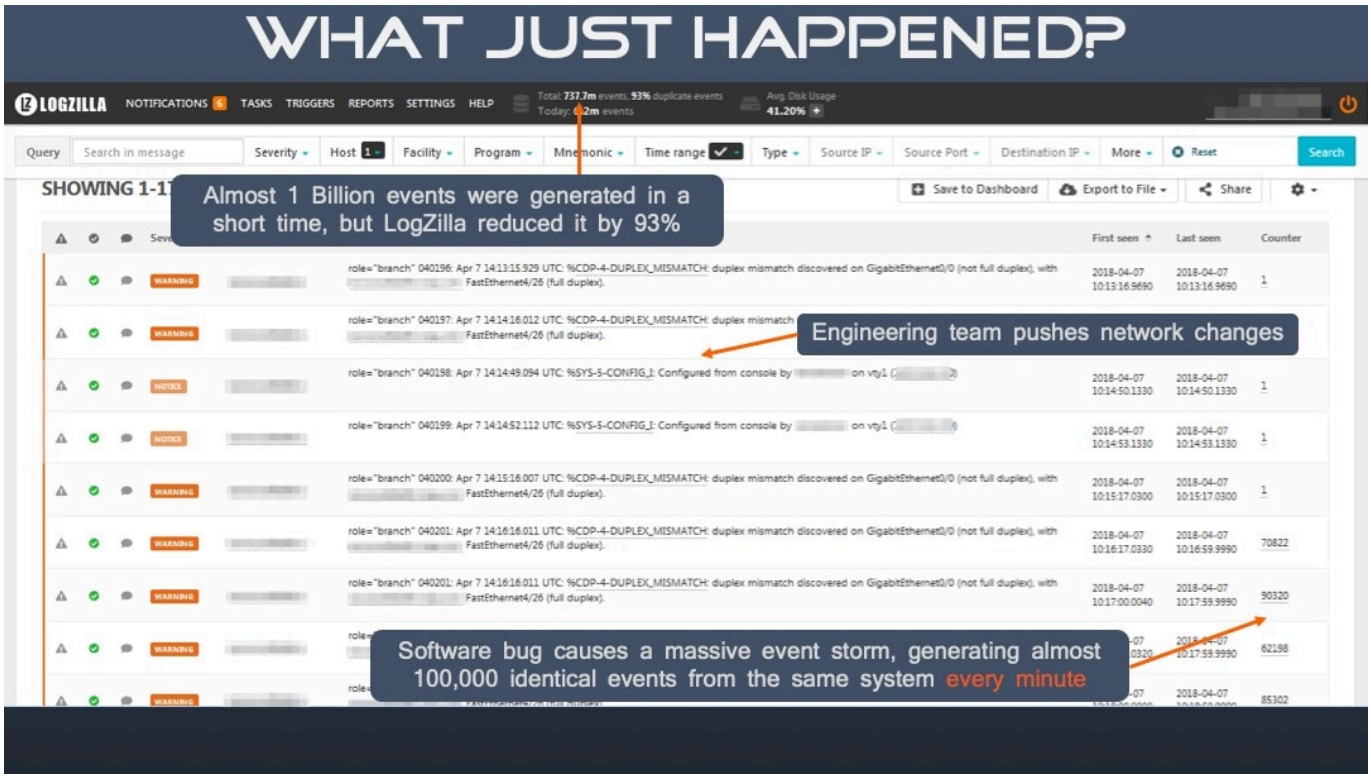
LogZilla performs deduplication at ingest to collapse repeated events into a single forwarded event per window, with a count of occurrences. This reduces downstream CPU, storage, and licensing impact during event storms, while preserving the signal needed for operations.

Deduplication process



In large environments, devices may emit the same event repeatedly (for example, during a link flap or authentication failure). Deduplication at ingest ensures downstream receivers handle a small, meaningful set of events rather than an overwhelming flood.

Event storm example



In the scenario above, hundreds of thousands of duplicate events could be reduced to just a handful, with a counter indicating how many times each unique event occurred.

Forwarding options

LogZilla can forward events as a log or a trap, regardless of how the event was received (syslog, HTTP receiver, traps, or app inserts). The module applies deduplication first, then applies any forwarder rules, and finally delivers to the configured destination(s).

Forwarder matching and rules

Forwarders use the same `match` structure as rewrite rules to decide which events to send to a destination. Each forwarder can include an optional `match` list; only events that satisfy all match entries are forwarded.

For a full description of `match` semantics, common fields, and supported operators, see Rewrite Rules.

Regex matching example

Use regular expressions for complex matching against the message text:

```
# /etc/logzilla/forwarder.d/interface-monitoring.yaml
match:
  - field: message
    op: "=~"
    value: "Interface .* is (up|down)"
type: syslog
target: central-log-collector:514
name: "Interface State Change Forwarder"
```

This forwards events where the `message` indicates an interface state change, regardless of whether it transitioned up or down.

Configuration locations and workflow

- `/etc/logzilla/forwarder.yaml` (or `.json`): Global forwarder file. It is created automatically on first use with sane defaults (`window_size: 60`, `fast_forward_first: true`). Most users should not modify this file directly.
- `/etc/logzilla/forwarder.d/`: Recommended location. Place individual forwarders here, one forwarder per file (YAML or JSON). This keeps configurations modular and easier to manage.

After adding or changing forwarders:

```
# Inspect the merged forwarder configuration (useful for syntax checks)
logzilla forwarder print

# Reload the forwarder to apply changes
logzilla forwarder reload
```

To enable the Forwarder module feature flag:

```
logzilla settings update FORWARDER_ENABLED=true
```

For full CLI usage, refer to:

- System commands
- Data commands (forwarder)

Downstream Syslog Receivers

Configure LogZilla to forward deduplicated events to downstream syslog, file, Splunk HEC, or SNMP destinations using per-forwarder YAML files in forwarder.d

Syslog Module

The Forwarder module allows forwarding all or specific matched events to a downstream log receiver. Deduplication occurs at ingest to reduce repeated events before forwarding. The downstream receiver is not limited to *syslog*; additional destination types are supported in addition to *syslog*: *file*, *splunk-hec*, and *snmp*.

Enable The Module

To enable the Forwarder Module, run the following command on the LogZilla server. See also System Commands.

```
sudo logzilla settings update FORWARDER_ENABLED=true
```

1. Configure rule(s)

LogZilla creates a main forwarder configuration file automatically when the forwarder starts. While it can contain global options and forwarders, the recommended practice is to keep the global file minimal and place individual forwarders as separate files under `/etc/logzilla/forwarder.d/`. Use one forwarder per file for simpler administration.

Examples of the Main Forwarder Configuration File

Here are some sample configurations for the main forwarder configuration file:

Forward to host A

```
---
window_size: 60
forwarders:
- type: syslog
  target: 192.168.0.114:514
  transport: tcp
  unsent_buffer_limit: 250000
  protocol: rfc5424
  rules:
  - match:
      field: counter
      op: gt
      value: 1
    rewrite:
      message: $MESSAGE ORIGIN="$HOST" LZ_Dedup_Count="$COUNTER"
  - match:
      field: counter
      op: le
      value: 1
    rewrite:
      message: $MESSAGE ORIGIN="$HOST"
fast_forward_first: true
```

Forward to host B

```
---
window_size: 60
forwarders:
- type: syslog
  target: 192.168.0.117:514
  transport: udp
  protocol: bsd
  rules:
  - match:
      field: counter
      op: gt
      value: 1
    rewrite:
      message: $MESSAGE ORIGIN="$HOST" LZ_Dedup_Count="$COUNTER"
  - match:
      field: counter
      op: le
      value: 1
    rewrite:
      message: $MESSAGE ORIGIN="$HOST"
fast_forward_first: true
```

Forward to file

```
---
window_size: 1
fast_forward_first: true
forwarders:
- match:
    field: cisco_mnemonic
    value: BGP-5-ADJCHANGE
  type: file
  target: "/var/log/logzilla/sec/simple.log"
  format: tsv
  separator: "\t"
  fields:
  - last_occurrence
  - host
  - message
```

Examples of Individual Forwarder Configuration Files

Individual forwarder configuration files can be used. The files can consist of JSON (`.json`) or YAML (`.yaml`). Each file defines a particular forwarder configuration, one forwarder per file, using the same syntax and options as specified in the `forwarders` configuration element as explained above.

For example, `non-dedup.yaml` might contain the following:

```
window_size: 0
type: file
path: /var/log/logzilla/non-dedup.log
```

IMPORTANT: Source host marker

Downstream receivers such as Splunk need to know which host the event originated from. Append a key/value pair in the rewrite rule to carry the original host. The recommended key name is `ORIGIN` (examples above). See Forwarding to Splunk for example transforms.

Forwarder Main Configuration

format is also supported, in which case end it in `.json`):

`match` This is a filter that defines which events should be forwarded

- its syntax is exactly the same like the one used in rewrite rules Match Conditions in Rewrite Rules. This is a global filter, affecting all forwarders; it can also be set in individual forwarders.

`window_size` This is the default value of window_size, if not given for a particular forwarder. It is the time in seconds of keeping every message while looking for its duplicates. The higher number set here, the better deduplication will work, but also the longer delay will be introduced (every message is kept for as much seconds before being forwarded to defined target). Setting this to 0 disables deduplication completely.

`fast_forward_first` The default value of fast_forward_first, if not given for particular forwarder. This defines the behavior for the first unique occurrence in the window. When true (default), the first occurrence will be forwarded without delay, while all following duplicates will be collected and forwarded at the end of the window. When false, first occurrence will be deduped alongside all the duplicates that follow. This setting delays forwarding of every event by `window_size` seconds to allow prospective deduplication.

`forwarders` This section defines forwarders. Multiple forwarders and mixing Syslog and SNMP trap destinations may be used. Every element of the `forwarders` table has a mandatory field `type` which defines what type of forwarder it is - currently `snmp` , `syslog` , `file` , and `splunk-hec` are supported. Other fields depend on the forwarder type.

For example, the following would forward to both an SNMP Trap receiver and a Syslog receiver:

```
---
forwarders:
- oid_map:
  - oid: ".1.2.0"
    src: facility
    type: s
  - oid: ".1.3.0"
    src: severity
    type: i
  - oid: ".1.4.0"
    src: cisco_mnemonic
    type: s
  - oid: ".1.5.0"
    src: message
    type: s
  - oid: ".1.99.0"
    src: counter
    type: i
  oid_prefix: 1.3.6.1.4.1.9.9.41.1.2.3
  target: snmp-server:162
  trap_oid: 1.3.6.1.4.1.2021.991
  type: snmp
- protocol: bsd
  rules:
  - match:
      field: counter
      op: gt
      value: 1
    rewrite:
      message: "$MESSAGE LZ_dedupCount=$COUNTER"
  target: central-log-collector:514
  transport: tcp
  type: syslog
```

Configuration for Each Forwarder

Every element of forwarders array has two mandatory fields: `type` and `target`. Each type might have more mandatory fields. Supported types: splunk-hec, snmp, syslog, file

Common Options

The following options can be used in every forwarder type:

`match` As explained above, the filter. Only events matching this filter will be forwarded

`window_size` As explained above, this is the time in seconds of keeping every message while looking for its duplicates. The higher number set here, the better deduplication will work, but also the longer delay will be introduced (every message is kept for as much seconds before being forwarded to defined target). Setting this to 0 disables deduplication completely.

`fast_forward_first` As explained above, this defines the behavior for the first unique occurrence in the window. When true (default), the first occurrence will be forwarded without delay, while all following duplicates will be collected and forwarded at the end of the window. When false, first occurrence will be deduped alongside all the duplicates that follow.

`rules` This allows you to use rules in the same way as for rewrite rules. You can specify any list of rules, that will be applied to the event in order before it is forwarded - and which can update its fields. See Rewrite Rules for details on rewrite rules.

Forwarder Options for Particular Forwarder Types

Syslog

`target` This is host and port of the target syslog server.

`transport` Either `tcp` or `udp`. The `tcp` transport can operate in either blocking or non-blocking mode depending on the configuration.

`unsent_buffer_limit` The maximum number of events (post predup) that will be buffered in case the destination is down. If the destination comes back up before the buffer overflows, events will be forwarded in the original order. If the destination remains down and the buffer exceeds the limit, additional events will not be buffered. Defaults to 25000. Applies only to `tcp` transport. Note that buffering is enabled **after** the forwarder realizes that the destination is down, which might be significantly later depending on network communication.

`protocol` Either `bsd` for the classic (RFC3164) protocol or the newer `rfc5424` protocol

`octet_count` Use the octet counting framing method for sending messages.

File

This forwarder saves all forwarded events in a file, in json or TSV format, one line per event.

`target` The path to the file where events are to be saved. This is a path in the container `lz_forwarder_module`, so this file can be accessed with `logzilla shell` into the forwarder container. If desired, this file can be saved directly on the host file system if the file is put in a path inside the `/var/log/logzilla/` directory, because that directory (and subdirectories) is shared between the host and the LogZilla docker container.

`format` Defaults to `json`, in which case it always save whole event. Another option is `tsv` which uses tab separated values, but other separators can be specified (defaults to TAB); with TSV format a list of fields that are written to output file can be provided.

`separator` For the TSV format this is the string used to separate fields (defaults to TAB).

`fields` The array of fields to be written in TSV format (defaults to `["host", "program", "message"]`).

`rotate_period` The time in seconds after which log file will be renamed with `.0` appended (so if it's `fwd.log` it will become `fwd.log.0`), and then the original path will be reopened as an empty file. As appropriate, each previous `.0` file will overwritten so there is always no more than just two log files - the previous and the current one. The default value of 0 disables rotation completely.

splunk-hec
This forwarder sends events in JSON format to Splunk HTTP Event Collector. The receiving splunk instance should be configured to: have a Splunk HEC source enabled; have a HEC token; globally enable the HTTP Event Collector. Both HTTP and HTTPS transports are supported (HTTPS is recommended for production deployments). Documentation for these Splunk settings is available in the Splunk HTTP Event Collector documentation. (This has been successfuly tested on Splunk 8.0.1).

`target` The address in the format `HOST`, `HOST:PORT`, `http://HOST:PORT`, or `https://HOST:PORT`, where `HOST` and `PORT` are replaced with the actual values of the host name and TCP/UDP port. If `PORT` is omitted, the default Splunk value of `8088` is used. For clarity, example targets include `https://splunk.example.com:8088`.

`token` The HEC token as specified in Splunk.

SNMP
This forwarder sends an SNMP Trap for each matching event. A list of variables that will be added to the trap can be defined, with values copied from the particular fields of event.

`target` This is the host and port of the SNMP server.

`trap_oid` This will be set as the type of outgoing SNMP trap.

`oid_prefix` Whenever `oid` in the map starts with a dot, it will be prefixed with this prefix.

`oid_map` This is the list of variables that will be added to the trap. For every variable you define:

`type` For now only `i` (32 bit integer) and `s` (string) are supported.

`oid` The object id of this variable; if it starts with a dot then it is prefixed with `oid_prefix`.

`src` The name of the event field variable in which the value will be set.

`value` if no `src` is defined, a constant can be configured here that will be copied for this value.

2. Add the Forwarder Configuration(s)

Recommended: place individual forwarder configuration files in `/etc/logzilla/forwarder.d/` (one forwarder per file). After changes, verify and reload:

```
logzilla forwarder print
logzilla forwarder reload
```

Advanced users may choose to manage the global file at `/etc/logzilla/forwarder.yaml` or `/etc/logzilla/forwarder.json`. In most cases, prefer the `forwarder.d` layout. See Data Commands for CLI details.

3. Apply forwarder configuration

Reload the forwarder after creating or changing configuration files:

```
sudo logzilla forwarder reload
```

Downstream Snmp Receivers

Forward LogZilla events as SNMP traps to downstream trap receivers with custom OID mapping, trap_oid selection, and severity-based pre-match filters

SNMP Trap

The SNMP TRAP forwarder allows forwarding of all or specific matched events to a downstream trap receiver. Deduplication occurs at ingest to reduce repeated events before forwarding.

For configuration detail on each section other than the actual `forwarders` section below, see Downstream Syslog Receivers.

Sample Forwarder Config File

```
window_size: 60
pre_match:
- field: message
  op: "~*"
  value: duplex mismatch discovered on
- field: severity
  value:
  - 1
  - 2
  - 5
forwarders:
- type: snmp
  target: 10.10.1.200:162
  trap_oid: 1.3.6.1.4.1.2021.991
  oid_prefix: 1.3.6.1.4.1.9.9.41.1.2.3
  oid_map:
  - type: s
    oid: ".1.2.0"
    src: facility
  - type: i
    oid: ".1.3.0"
    src: severity
  - type: s
    oid: ".1.4.0"
    src: cisco_mnemonic
  - type: s
    oid: ".1.5.0"
    src: message
  - type: i
    oid: ".1.99.0"
    src: counter
```

This forwarder sends the specified SNMP Trap for every matching event after deduplication.

Place the configuration file (for example, `fwd-snmp.yaml`) in `/etc/logzilla/forwarder.d/`, then verify and reload:

```
logzilla forwarder print
logzilla forwarder reload
```

NOTE: OIDs can defined here based on the needs, LogZilla does not limit which OIDs you are permitted to send.

trap_oid

Used to set the type of outgoing SNMP trap. In the case of `1.3.6.1.4.1.2021.991`, it specifies that it is from the `UCD-SNMP-MIB`, Specifically, `NOTIFICATION-TEST-MIB`.

oid_prefix

The base OID for all subsequent fields in the `oid_map`.

oid_map

The list of variables to be added to the trap:

- type: Only `i` (32 bit integer) and `s` (string) are supported
- oid: Object id of this variable; if it starts with dot then it's prefixed with `oid_prefix`
- src: Name of event field from LogZilla placed in this variable
- value: If no `src` macro is defined, you may use this to add extra information to each outgoing event.

For example, in a Service Provider Network, this would allow the addition of a customer's BGP AS Number:

```
{ "type" : "i", "oid" : ".1.70.0", "value" : "64512" },
{ "type" : "s", "oid" : ".1.70.1", "value" : "BGP_AS_ID" }
```

Forwarding To Splunk

Reduce Splunk ingest costs by forwarding LogZilla deduplicated events with an ORIGIN marker, then mapping source hosts via Splunk transforms.conf and props.conf

LogZilla NEO may also be used to reduce the amount of data sent to Splunk systems while, at the same time, generating more value in that data.

LogZilla's deduplication reduces repeated events at ingest so Splunk does not have to process large bursts during event storms. The forwarding module works the same way in terms of configuration on the LogZilla side. On the Splunk side, a transform can be used to indicate the original sending host so that Splunk does not attribute all events to the LogZilla system itself.

Source host marker

To help Splunk determine the correct source host, the forwarder rule should append a key/value pair to the forwarded message. The recommended key name is `ORIGIN` (for example: `message: $MESSAGE ORIGIN="$HOST"`). See the forwarder rule examples in Downstream Syslog Receivers.

Splunk Setup

On the Splunk server, create or edit `$SPLUNK_HOME/etc/system/local/transforms.conf` and add:

Splunk Transforms

```
[logzilla_forwarder]
REGEX = ORIGIN=(\S+)
FORMAT = host::$1
DEST_KEY = MetaData:Host
```

Next, create or edit the file `$SPLUNK_HOME/etc/system/local/props.conf` and associate the transform to the source. In the case of this example, this guide is sending everything via TCP port 514, so the source used in Splunk's `props.conf` will be that.

Splunk Props

```
[source::tcp:514]
TRANSFORMS-lz_neo=logzilla_forwarder
```

For options on Splunk's transforms and props files, please reference Splunk's Documentation for further help.

Verify and reload

After editing forwarder rules and reloading Splunk configs:

```
# Inspect the merged forwarder configuration from LogZilla
logzilla forwarder print

# Reload the forwarder to apply rule changes
logzilla forwarder reload
```

Forwarding To Event Correlation

Route selected LogZilla events to SEC instances using type sec forwarders and sec_name, keeping stateful correlation focused on relevant traffic

Forwarding to Event Correlation (SEC)

Purpose

Forwarders control which events LogZilla sends to Event Correlation (SEC) for stateful processing. The sections below cover how to use `type: sec` forwarders, how `sec_name` maps to SEC instances under `/etc/logzilla/sec/`, and how to verify that events reach SEC.

Prerequisites

- Event Correlation enabled in LogZilla settings
  (see Event Correlation Overview).
- Forwarder module enabled:

```
sudo logzilla settings update FORWARDER_ENABLED=true
```

- Basic familiarity with forwarder configuration
  (see Forwarder and Deduplication Overview).

How SEC Forwarders Work

LogZilla uses selective event forwarding to optimize SEC performance:

Event ingestion: LogZilla receives and parses incoming events.

Forwarder evaluation: Events are evaluated against forwarder rules.

Selective routing: Only matching events are sent to SEC instances.

SEC processing: SEC rules correlate the forwarded events.

Response injection: SEC results are sent back into LogZilla as new events.

This approach ensures that SEC focuses only on relevant events and avoids unnecessary load from unrelated traffic.

Forwarder Type `sec`

The `sec` forwarder type sends events from LogZilla to a specific SEC instance.

Basic SEC Forwarder Example

Forward all events from a specific program to SEC:

```
# /etc/logzilla/forwarder.d/test-events.yaml
match:
  - field: program
    op: "eq"
    value: "TEST_APP"
type: sec
name: "Test Application Forwarder"
sec_name: example
```

Configuration breakdown:

- `match` : Defines filtering criteria for events.
- `field: program` : Matches the syslog program field.
- `op: "eq"` : Uses the equality operator.
- `value: "TEST_APP"` : Matches events where `program` equals `TEST_APP` .
- `type: sec` : Sends events to a SEC instance instead of an external receiver.
- `name` : Human-readable description for troubleshooting.
- `sec_name: example` : Must match a SEC instance directory name under `/etc/logzilla/sec/` .

Message Rewriting Before SEC

Transform events before sending them to SEC to simplify correlation rules:

```
# /etc/logzilla/forwarder.d/auth-events.yaml
match:
  - field: program
    op: "eq"
    value: ["sshd", "sudo", "su"]
type: sec
name: "Authentication Events"
sec_name: security-monitoring
rules:
  - match:
      - field: message
        op: "=~"
        value: "Failed password"
    rewrite:
      message: "AUTH_FAILED $MESSAGE"
  - match:
      - field: message
        op: "=~"
        value: "Accepted password"
    rewrite:
      message: "AUTH_SUCCESS $MESSAGE"
```

This prepends `AUTH_FAILED` or `AUTH_SUCCESS` to messages, allowing SEC rules to use simple substring matching (`pattern=AUTH_FAILED`) instead of complex regular expressions.

Multiple SEC Instances

Route different event types to specialized SEC instances:

```
# /etc/logzilla/forwarder.d/network-forwarding.yaml
match:
  - field: program
    op: "eq"
    value: "cisco_ios"
  - field: cisco_mnemonic
    op: "eq"
    value: "BGP-5-ADJCHANGE"
type: sec
name: "BGP Events"
sec_name: bgp-monitoring

---
# Same file, different forwarder
match:
  - field: program
    op: "eq"
    value: "cisco_ios"
  - field: cisco_mnemonic
    op: "=~"
    value: "LINK-.*-UPDOWN"
type: sec
name: "Interface Events"
sec_name: interface-monitoring
```

This separates BGP correlation from interface correlation, allowing independent rule management and scaling.

Forwarder and SEC Instance Relationship

The `sec_name` value in a SEC forwarder configuration must exactly match a directory name under `/etc/logzilla/sec/`.

Example directory structure:

```
/etc/logzilla/sec/
├── example/
│   └── rules/
│       └── test.sec
├── network-monitoring/
│   └── rules/
│       ├── bgp-correlation.sec
│       └── interface-correlation.sec
└── windows-security/
    └── rules/
        └── brute-force.sec
```

Corresponding forwarder `sec_name` values:

- `sec_name: example` → `/etc/logzilla/sec/example/`
- `sec_name: network-monitoring` → `/etc/logzilla/sec/network-monitoring/`
- `sec_name: windows-security` → `/etc/logzilla/sec/windows-security/`

Applying SEC Forwarder Changes

After creating or modifying SEC forwarder configurations:

```
# Validate configuration syntax
logzilla forwarder print

# Apply changes via settings reload
sudo logzilla settings reload forwardermodule

# Verify forwarder is processing
sudo docker logs lz_forwarder | tail -20
```

The `logzilla forwarder print` command displays the merged forwarder configuration and reports syntax errors before applying changes.

Verifying Forwarding to SEC

To confirm that events are being forwarded to SEC:

```
# Monitor SEC logs for incoming events
sudo docker logs lz_sec -f

# Send test event from a client to LogZilla
logger -n <logzilla-ip> -P 514 -t TEST_APP "TEST_EVENT: Sample message"

# SEC logs should show HTTP requests from the forwarder
# Example:
# 2025-10-20 11:15:23 [1] uvicorn.access INFO 172.17.0.1:12345 \
#   "POST /sec/example HTTP/1.1" 200
```

If SEC logs show no activity after sending test events:

Verify the forwarder is enabled:

```
sudo logzilla settings list | grep FORWARDER_ENABLED
```

Check forwarder configuration:

```
logzilla forwarder print
```

Verify the SEC instance exists:

```
ls -la /etc/logzilla/sec/
```

Reload both services:

```
sudo logzilla settings reload sec forwardermodule
```

Complete Working Example

This example demonstrates a complete SEC instance and SEC forwarder for testing.

1. Create SEC Instance and Rule

```
# Create SEC instance directory structure
sudo mkdir -p /etc/logzilla/sec/example/rules

# Create simple SEC rule
sudo cat << 'EOF' > /etc/logzilla/sec/example/rules/test.sec
# Count TEST events and alert at threshold
type=SingleWithThreshold
ptype=SubStr
pattern=TEST_EVENT
desc=SEC Test Rule Triggered
action=shellcmd (logger -T -n $SYSLOG_HOSTNAME -P $SYSLOG_BSD_TCP_PORT \
       -t SEC-ALERT -p local0.alert \
       "SEC_WORKING: Detected $thresh test events in $window seconds")
thresh=3
window=60
EOF
```

2. Create SEC Forwarder Configuration

```
sudo cat << 'EOF' > /etc/logzilla/forwarder.d/test.yaml
match:
  - field: program
    op: "eq"
    value: "TEST_APP"
type: sec
name: "Test SEC Forwarder"
sec_name: example
EOF
```

3. Validate and Apply

```
# Validate configuration
logzilla forwarder print

# Apply changes
sudo logzilla settings reload sec forwardermodule

# Verify SEC container is running
sudo docker ps | grep lz_sec
```

4. Test Correlation

```
# Monitor SEC logs
sudo docker logs lz_sec -f &

# Send three test events
for i in {1..3}; do
  logger -n <logzilla-ip> -P 514 -t TEST_APP "TEST_EVENT: Test message $i"
  sleep 2
done

# Check LogZilla UI for SEC-ALERT message
# Or search logs: sudo docker logs lz_sec | grep "SEC_WORKING"
```

CHAPTER 09

Logzilla Api

LogZilla REST API reference for authentication, event and system queries, WebSocket subscriptions, exports, and custom integration development

LogZilla API

The LogZilla API enables programmatic access to log management capabilities through standard HTTP/HTTPS requests. This RESTful interface supports authentication, query operations, real-time updates, and administrative functions for building custom integrations and automated workflows.

The following chapters provide narrative context and practical examples for using the LogZilla API. For complete API specifications, endpoint details, and interactive documentation, refer to `/api/docs/` on your LogZilla server.

Next Steps

Getting Started - Authentication and basic API usage

Core Concepts - HTTP methods, error handling, pagination, and data structures

Making Queries - Query creation workflow and WebSocket usage

Query Types

- **Event Queries**
- **System/Admin Queries**

Query API Parameters - Comprehensive parameter reference

Sample API Code - Working examples and code snippets

API Endpoint Overview - Common endpoint map referencing `/api/docs`

Query Export Formats - Supported export formats by query type

Getting Started

Generate LogZilla API tokens, authenticate HTTP requests, and verify access with curl against the interactive docs at /api/docs and /incoming/docs

Getting Started

The LogZilla API provides secure, programmatic access to log management and analysis capabilities through standard HTTP/HTTPS requests. Authentication token management, basic API usage patterns, and verification procedures enable integration with LogZilla's RESTful interface.

Note: Interactive documentation endpoints

- Main API (management, queries): `/api/docs`
- HTTP Receiver (data ingestion): `/incoming/docs`

Prerequisites

- Administrative or root access to LogZilla server
- Command-line access for token management
- HTTP client (curl, wget, or application capable of HTTP requests)
- Understanding of REST API concepts

Authentication

All API functions and HTTP event receipt require authentication via an *authorization token*. An auth token is a long alphanumeric sequence that serves as a key associated with a specific user. When provided to LogZilla, the system verifies that the token has been configured to allow API or back-end access. Auth tokens must be kept private as they authorize access to LogZilla data. Tokens persist indefinitely until explicitly revoked.

Two types of auth tokens exist:

- **User tokens** - Full-function tokens with complete API access
- **Ingest-only tokens** - Limited to HTTP Event Receiver data ingestion

Token scope and usage

- User tokens are valid for all management API endpoints under `/api/*` and for ingestion under `/incoming/*`.
- Ingest-only tokens are valid only for the HTTP Receiver under `/incoming/*` and are rejected by the main management API under `/api/*`.

- The HTTP Receiver accepts tokens via `Authorization: token ...`, the `X-LZ-Access-Key` header, or the `AUTHTOKEN` query parameter.

Administrator or root access is required for token management. This can be accomplished through privileged login or `sudo`.

Administrators can manage tokens using the `logzilla authtoken` CLI tool:

```
# logzilla authtoken -h
usage: authtoken [-h] [-d] [-q] {create,revoke,info,list} ...

LogZilla AuthToken manipulation

positional arguments:
  {create,revoke,info,list}
    create              create new token
    revoke              revoke new token
    info                show token info
    list                list all active tokens

optional arguments:
  -h, --help            show this help message and exit
  -d, --debug           debug mode
  -q, --quiet           notify only on warnings and errors (be quiet).
```

Auth Token Management

Auth Token Generation

Administrators can create new full-function user auth tokens using `logzilla authtoken create`:

```
root[~]: # logzilla authtoken create
Creating USER token
user-317526c44e0e04348f3dd084e997cc15950107700ddd7be0
```

The output displays the generated auth token on the last line.

Tokens can be created for specific users by specifying the username:

```
root[~]: # logzilla authtoken create -U john
Creating USER token
user-317526c44e0e04348f3dd084e997cc15950107700ddd7be0
```

Ingest-only tokens are created using the `--ingest-only` option:

```
root[~]: # logzilla authtoken create --ingest-only
Creating INGEST token
ingest-317526c44e0e04348f3dd084e997cc15950107700ddd7be0
```

Auth Token Review

Active auth tokens can be listed using `logzilla authtoken list`:

```
# logzilla authtoken list
Active tokens:
8210276eca565481f66677438ec454025a621e05d7df2a80 created: 2022-05-12
14:37:51.769886+00:00; user: admin
```

Detailed information for a specific auth token can be retrieved using `logzilla authtoken info`:

```
# logzilla authtoken info 8210276eca565481f66677438ec454025a621e05d7df2a80
Token: 8210276eca565481f66677438ec454025a621e05d7df2a80
User: admin
Created: 2022/05/12 14:37:51
```

Auth Token Revocation

Auth tokens can be revoked to permanently delete them and prevent further LogZilla access. Revocation is performed using `logzilla authtoken revoke`:

```
# logzilla authtoken revoke 8210276eca565481f66677438ec454025a621e05d7df2a80
Token 8210276eca565481f66677438ec454025a621e05d7df2a80 revoked.
```

Using the Auth Token

Authorization tokens can be provided to the API in two ways:

- `Authorization` header
- `AUTHTOKEN` parameter in the request URI

Header-Based Authentication

Tokens can be included in the Authorization HTTP header:

```
Authorization: token 701a75372a019fc3b1572454a582a5705bc4e929d305694c
```

URI-Based Authentication

Tokens can be included as a parameter in the request URL:

```
POST /incoming?AUTHTOKEN=701a75372a019fc3b1572454a582a5705bc4e929d305694c
```

Example

Once a token is created, users can connect to the API using standard HTTP methods (POST, GET, PATCH, PUT, etc.).

The following example demonstrates sending a sample event to LogZilla using the standard `events` array JSON structure and header-based authentication:

```
curl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: token 91289817dec1abefd728fab4f43aa58b5e6fa814f' \
  -X POST -d '{"events":[{"host":"web01","program":"sample","message":"Test Message"}]}'
\
  'http://logzilla.mycompany.com/incoming'
```

Verification

To verify API access is working correctly:

Create an auth token using the steps above

Test API connectivity with a simple request:

```
curl -H "Authorization: token YOUR_TOKEN_HERE" \
    "http://your-logzilla-server/api/auth/"
```

Successful authentication returns a JSON response with user information

References

- Interactive API Documentation (`/api/docs` on your LogZilla server) - Full endpoint reference
- Making Queries - Detailed query operations
- HTTP Event Receiver Documentation
    - Event ingestion

Core Concepts

LogZilla API fundamentals covering HTTP methods, JSON conventions, 2xx and 4xx status codes, pagination, and error handling for REST integrations

Core Concepts

Essential API concepts for working with LogZilla's RESTful interface, including HTTP conventions, error handling, pagination mechanisms, and data structures. Understanding these fundamentals enables effective API integration and troubleshooting.

Prerequisites

- Administrative access to LogZilla web interface
- Understanding of REST API concepts and HTTP methods
- JSON data format familiarity
- HTTP client tools (curl, programming language libraries)
- Valid API authentication token (see Getting Started)

HTTP Methods and Conventions

Requests to the API are made using standard HTTP methods. For endpoints that accept or return data in the request/response body, JSON is the standard format, and the user should typically set the `Content-Type: application/json` header for `POST`, `PUT`, and `PATCH` requests.

Error Handling

The LogZilla API uses standard HTTP status codes to indicate the success or failure of an API request.

2xx (Successful)

- `200 OK`: The request was successful.
- `201 Created`: The request was successful, and a resource was created.
- `202 Accepted`: The request has been accepted for processing, but the processing has not been completed (common for asynchronous operations such as queries). The response body usually contains information on checking the status.
- `204 No Content`: The request was successful, but there is no content to return (e.g., for a successful DELETE request).

3xx (Redirection)

- `301 Moved Permanently`: The requested resource has been permanently moved to a new location. The response body usually contains the new location. NOTE: this normally means that the user is trying to access the server using HTTP

and the server is configured for `FORCE_HTTPS==true`. The user should use HTTPS instead.

4xx (Client Errors)

- `400 Bad Request`: The request could not be understood by the server due to malformed syntax or invalid parameters. The response body often contains more specific error details.
- `401 Unauthorized`: Authentication is required and has failed or has not yet been provided. Ensure the auth token is valid and included in the request.
- `403 Forbidden`: Authentication was successful, but the authenticated user does not have permission to perform the requested action.
- `404 Not Found`: The requested resource could not be found.
- `405 Method Not Allowed`: The HTTP method used (e.g., GET, POST) is not supported for the requested resource.
- `408 Request Timeout`: The request timed out (often for async operations).
- `429 Too Many Requests`: The user has sent too many requests in a given amount of time (rate limiting).

5xx (Server Errors)

- `500 Internal Server Error`: An unexpected condition was encountered on the server.
- `503 Service Unavailable`: The server is currently unable to handle the request due to temporary overloading or maintenance.

When an error occurs, the response body will typically be a JSON object with a `detail` key:

```
{"detail": "Error message describing the issue"}
```

For validation errors (400 Bad Request), the response may include field-specific errors:

```
{
  "field_name": ["Error message for this field"],
  "another_field": ["Another error message"]
}
```

For server errors (500), the response may include an `error` key and optionally a `traceback`:

```
{
  "error": "An unexpected error occurred",
  "traceback": "...detailed traceback..."
}
```

Pagination

LogZilla's API uses different pagination mechanisms depending on the endpoint:

Standard List Pagination (Most Endpoints)

Used for endpoints like `/api/users`, `/api/dashboards`, and `/api/triggers`.

Request: Pass `page` and `page_size` as query parameters:

```
GET /api/users?page=2&page_size=50
```

Response structure:

```
{
  "objects": [ ... ],
  "item_count": 157,
  "page_count": 4,
  "page_number": 2
}
```

Query Results Pagination

Used specifically for the `/api/query/{qid}` endpoint for event search results.

Request: Pass pagination parameters in the query body or as URL parameters:

```
{
  "page": 1,
  "page_size": 100,
  "offset": 0
}
```

Response structure: Pagination info is nested within the results:

```
{
  "query_id": "...",
  "status": "SUCCESS",
  "results": {
    "events": {
      "objects": [ ... ],
      "page_number": 1,
      "page_size": 100,
      "offset": 0,
      "item_count": 1234,
      "page_count": 13
    }
  }
}
```

Common Data Structures and Formats

Event Field Names

Events in LogZilla are characterized by several standard fields:

Name	Description
`first_occurrence`	Timestamp of the first occurrence (epoch seconds with microseconds)
`last_occurrence`	Timestamp of the last occurrence (epoch seconds with microseconds)
`counter`	Number of occurrences in the deduplication window
`message`	The event message content
`host`	The originating host
`program`	The process or program name
`cisco_mnemonic`	Cisco mnemonic code (if applicable)
`severity`	Numeric severity (0-7, syslog standard)
`facility`	Numeric facility (0-23, syslog standard)
`status`	Status number (0=unknown, 1=actionable, 2=non-actionable)
`type`	Event type (e.g., `SYSLOG`, `INTERNAL`, `UNKNOWN`)
`User Tags`	User-defined fields

Best Practices

Rate Limiting and Performance

- Use pagination for large result sets
- Cache query IDs for repeated access
- Implement exponential backoff for retries
- Use appropriate `page_size` values (typically 100-1000)

Security

- Store API tokens securely (environment variables, secrets management)
- Use HTTPS for all API communications
- Implement proper error handling
- Regularly rotate API tokens

Query Optimization

- Use specific filters to reduce result sets
- Leverage time ranges to limit data scope

References

- Interactive API Documentation (`/api/docs` on the LogZilla server) - Current and detailed API specifications
- Authentication Guide - Token management and usage
- API Code Examples - Examples of performing API requests with error handling

Api Endpoint Overview

Map of common LogZilla API endpoints for auth, users, dashboards, widgets, triggers, and queries, sourced from lib/logzilla/api/urls.py

API Endpoint Overview

A practical map of commonly used API endpoints, sourced from `lib/logzilla/api/urls.py`. For full request/response schemas, filters, and examples, use the interactive docs at `/api/docs`.

- Auth and session
  - `GET/POST/DELETE /api/auth` —- session info, login, logout
  - `POST /api/reset-password`
  - `GET /api/ping`
- Users and access control
  - `GET/POST /api/users` (and `/{id}`)
  - `GET /api/groups`
  - `POST /api/groups`
  - `GET /api/groups/{id}`
  - `PUT /api/groups/{id}`
  - `PATCH /api/groups/{id}`
  - `DELETE /api/groups/{id}`
  - `GET /api/permissions`
- Dashboards and widgets
  - `GET /api/dashboards`
  - `POST /api/dashboards`
  - `GET /api/dashboards/{id}`
  - `PUT /api/dashboards/{id}`
  - `PATCH /api/dashboards/{id}`
  - `DELETE /api/dashboards/{id}`
  - `POST /api/dashboards/{id}/{report|templates|widgets}`
  - `DELETE /api/dashboards/{id}/widgets`
  - `GET /api/widgets`

  - `POST /api/widgets`
  - `GET /api/widgets/{id}`
  - `PUT /api/widgets/{id}`
  - `PATCH /api/widgets/{id}`
  - `DELETE /api/widgets/{id}`
  - `POST /api/widgets/{id}/report`
  - `GET /api/widget-types`
  - `GET /api/widget-presets`
- Events
  - `GET /api/events/{ev_id}`
  - `GET /api/events/{ev_id}/timestamps`
  - `GET /api/events/{ev_id}/triggers`
  - `GET /api/events/{ev_id}/ai`
  - `POST /api/events/{ev_id}/forward`
- Triggers
  - `GET /api/triggers`
  - `POST /api/triggers`
  - `GET /api/triggers/{id}`
  - `PUT /api/triggers/{id}`
  - `PATCH /api/triggers/{id}`
  - `DELETE /api/triggers/{id}`
  - `GET /api/triggers/{id}/history`
  - `POST /api/triggers-validator` —- validate filters
- Notifications
  - `GET /api/notification-groups`
- Queries (generic)
  - `POST /api/query` —- create
  - `GET /api/query/{qid}` —- results (with paging for Search)
  - `GET /api/query/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/query/{qid}/stop`
  - `GET /api/query-types` —- available query types

  - WebSocket: `/ws/live-updates` (see Making Queries)

- Queries (typed)
  - `POST /api/queries/search`
  - `GET /api/queries/search/{qid}`
  - `GET /api/queries/search/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/search/{qid}/stop`
  - `POST /api/queries/lastn`
  - `GET /api/queries/lastn/{qid}`
  - `GET /api/queries/lastn/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/lastn/{qid}/stop`
  - `POST /api/queries/topn`
  - `GET /api/queries/topn/{qid}`
  - `GET /api/queries/topn/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/topn/{qid}/stop`
  - `POST /api/queries/eventrate`
  - `GET /api/queries/eventrate/{qid}`
  - `GET /api/queries/eventrate/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/eventrate/{qid}/stop`
  - `POST /api/queries/processingstats`
  - `GET /api/queries/processingstats/{qid}`
  - `GET /api/queries/processingstats/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/processingstats/{qid}/stop`
  - `POST /api/queries/storagestats`
  - `GET /api/queries/storagestats/{qid}`
  - `GET /api/queries/storagestats/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/storagestats/{qid}/stop`
  - `POST /api/queries/system_cpu`
  - `GET /api/queries/system_cpu/{qid}`
  - `GET /api/queries/system_cpu/{qid}/{export|recalculate|remove|stop}`
  - `POST /api/queries/system_cpu/{qid}/stop`
  - `POST /api/queries/system_df`
  - `GET /api/queries/system_df/{qid}`
  - `GET /api/queries/system_df/{qid}/{export|recalculate|remove|stop}`

- `POST /api/queries/system_df/{qid}/stop`
- `POST /api/queries/system_iops`
- `GET /api/queries/system_iops/{qid}`
- `GET /api/queries/system_iops/{qid}/{export|recalculate|remove|stop}`
- `POST /api/queries/system_iops/{qid}/stop`
- `POST /api/queries/system_memory`
- `GET /api/queries/system_memory/{qid}`
- `GET /api/queries/system_memory/{qid}/{export|recalculate|remove|stop}`
- `POST /api/queries/system_memory/{qid}/stop`
- `POST /api/queries/system_network`
- `GET /api/queries/system_network/{qid}`
- `GET /api/queries/system_network/{qid}/{export|recalculate|remove|stop}`
- `POST /api/queries/system_network/{qid}/stop`
- `POST /api/queries/system_networkerrors`
- `GET /api/queries/system_networkerrors/{qid}`
- `GET /api/queries/system_networkerrors/{qid}/{export|recalculate|remove|stop}`
- `POST /api/queries/system_networkerrors/{qid}/stop`
- `POST /api/queries/systemstatus`
- `GET /api/queries/systemstatus/{qid}`
- `GET /api/queries/systemstatus/{qid}/{export|recalculate|remove|stop}`
- `POST /api/queries/systemstatus/{qid}/stop`

- Reports
  - `GET /api/reports`
  - `GET /api/reports/{id}` (download via retrieve)
  - `DELETE /api/reports/{id}`
  - `GET /api/reports/{id}/export`
  - `GET /api/reports-templates`
  - `POST /api/reports-templates`
  - `GET /api/reports-templates/{id}`
  - `PUT /api/reports-templates/{id}`
  - `PATCH /api/reports-templates/{id}`
  - `DELETE /api/reports-templates/{id}`
  - `POST /api/reports-templates/{id}/generate`

- `GET /api/reports-schedules`
- `POST /api/reports-schedules`
- `GET /api/reports-schedules/{id}`
- `PUT /api/reports-schedules/{id}`
- `PATCH /api/reports-schedules/{id}`
- `DELETE /api/reports-schedules/{id}`
- `GET /api/reports-schedules/{id}/reports`

- Settings and system
  - `GET /api/license-info`
  - `GET/POST /api/settings` (and extra configs)
  - `POST /api/settings-update-publish`
  - `GET/POST /api/customer-info`
  - `GET /api/monitor`

- Archives
  - `GET /api/archives`
  - `POST /api/archives`
  - `GET /api/archives/{chunk_ts}`
  - `POST /api/archives/migrate`
  - `DELETE /api/archives/range`
  - `POST /api/archives/remove` (range)
  - `GET /api/archive-restore-logs`

- Forwarder counters
  - `GET /api/forwarder-counters`

- Lookup tools
  - `GET /api/lookup/{dns|whois|mac|cisco-mnemonic|geoip|mswin-eventid|mitre-id}/{pk}`

- Terminals (test utilities)
  - `POST /api/terminals` —- create ephemeral shell session

- Mailer and LDAP tester
  - `POST /api/mailer`
  - `POST /api/ldap-tester`

- App store
  - `GET /api/apps` —- available apps
  - `GET /api/installed-apps`
- Misc
  - `GET /api/async-results`
  - `GET /api/docs`, `GET /api/schema`

Notes:

- All endpoints require a valid user token in `Authorization: token <TOKEN>` unless explicitly marked public. See Getting Started for token usage.
- The HTTP Receiver (ingestion) is separate and documented under Receiving Data; its interactive docs are typically at `/incoming/docs`.

Making Queries

Create LogZilla queries through POST /api/query, handle asynchronous 202 responses with query_id polling, and subscribe to WebSocket result streams

Making Queries

The LogZilla Query API provides fundamental operations for data retrieval and analysis, including query creation, asynchronous request handling, and websocket connections for real-time updates.

Prerequisites

- Valid LogZilla API authentication token (see Getting Started)
- Understanding of REST API concepts and HTTP methods
- HTTP client capable of POST requests
- JSON data format familiarity

Creating a New Query

A new query is created through `POST /api/query`, and always includes two parameters (usually with JSON body):

type
Indicates which query you want to perform. See Event Queries and System/Admin Queries for more detail.

params
A JSON object containing the parameters for the query. Every query type has a different list of available parameters.

After creating a query you can get its results either immediately (if it was able to complete in 1 second) with response `200 "OK"`, or (for requests which must be completed asynchronously) a status of `202 "ACCEPTED"` with response body containing a `query_id`.

Asynchronous Requests

NOTE: Although you can query for results at any time with `GET /api/query/<id>` the recommended way of getting query results is to use websockets and subscriptions (see below).

If the initial query returns `202 "ACCEPTED"` check for the query results using the query id value returned from the first query using `GET /api/query/<id>` to get updated results.

Relative Time Queries

Asynchronous queries that use relative time ranges, like "last hour", are perpetually updated to represent changing time, even if they have status `SUCCESS`.

Polling Query Results

To retrieve the current data of an existing query (whether currently processing or not) use `GET /api/query/<id>`.

`GET /api/query/<id>` can return paged results of the data by providing additional parameters of `page_size=<num of results>` and `page=<page number>`. The HTTP result message is always returned immediately but the query status (in the returned JSON) could indicate that the query is incomplete (query status `IN_PROGRESS`) or even as of yet empty (query status `PENDING`).

For example, if the query is not completed immediately the received response would be:

```
{
  "query_id": "72bc846140344b4da3cdcfb831174a3e",
  "status": "IN_PROGRESS",
  "type": "Search",
  "base_time": 1416233863,
  "results": {
    "..."
  },
  "params": {
    "sort": [
      "first_occurrence"
    ],
    "filter": [],
    "page": 1,
    "page_size": 100,
    "time_range": {
      "ts_from": 1000,
      "ts_to": 10000
    }
  },
  "owner_id": 1
}
```

When a query is completed (possibly immediately) the response would be:

```
{
  "query_id": "72bc846140344b4da3cdcfb831174a3e",
  "status": "SUCCESS",
  "type": "Search",
  "base_time": 1416233863,
  "results": {
    "..."
  },
  "params": {
    "sort": [
      "first_occurrence"
    ],
    "filter": [],
    "page": 1,
    "page_size": 100,
    "time_range": {
      "ts_from": 1000,
      "ts_to": 10000
    }
  },
  "owner_id": 1
}
```

Getting Query Results via Websocket

The recommended way of getting query results, especially for widgets, is using websockets. Using the websocket method (vs. `GET /api/query/`) provides initial calculation results, partial results for asynchronous queries, and final results of the query.

Note: You must first create a query with `POST /api/query` and only then use websockets for polling the results.

Websockets for the API are available under `/ws/live-updates` and, after establishing the connection, allows for real-time subscription and unsubscription on events of interest.

Websocket operations should be sent using encoded JSON with an array of commands and parameters, for example:

```
["subscribe", "widget", 2]
```

Subscription to a particular query or widget can be accomplished by providing the appropriate entity id, for which query id is a string and widget id is an integer, subscription to the whole dashboard can be requested in which case websocket updates will then include get updates for all widgets on that dashboard.

Unsubscribing can be accomplished either by providing the same parameters that were used for the subscription, or removal of all subscriptions with:

```
["unsubscribe", "all"]
```

After successful subscription/unsubscription a confirmation result will be returned, which always contains the list of currently subscribed items:

```
["subscription-update", {"query": [], "widget": [2], "dashboard": []}]
```

Once subscribed updates for the requested objects will begin. Each update is a separate message as follows:

```
["widget-update", {
    "widget_id": 2,
    "dashboard_id": null,
    "data": {
        "status": "SUCCESS",
        "query_id": "4f29934c97b1c0857c2341c3cb188371",
        "results": {
            "totals": "...",
            "details": "..."
        }
    }
}]
```

For widgets that are directly subscribed the dashboard_id as shown above will be null. For query subscriptions, both dashboard_id and widget_id will be null. The data field contains exactly the same content that `GET /api/query/<id>` would return, as indicated in the documentation for each particular request type.

Verification

Test query creation with a simple request:

```
curl -X POST -H "Authorization: token YOUR_TOKEN_HERE" \
     -H "Content-Type: application/json" \
     -d '{"type": "Search", "params": {"time_range": {"preset": "last_1_hours"}}}' \
     "http://your-logzilla-server/api/query"
```

Successful query creation returns a JSON response with query_id.

References

- Query API Parameters - Common parameters and filtering options
- Event Queries and System/Admin Queries - Detailed documentation for all query types

Event Query Types

LogZilla event query reference for Search, EventRate, TopN, and LastN, including parameters, sort fields, pagination, and result structures

Query Types —- Event Queries

These queries operate on event data and require the main storage/search services. All parameters and fields are verified against the current implementation in `lib/logzilla/query.py`.

- Search
- EventRate
- TopN
- LastN

Search

Returns counts plus a paginated list of matching events.

Parameters:

- `time_range`: Time period.
- `filter`: Filter expression (see Query API Parameters).
- `sort`: Array of sort fields. Defaults to `["first_occurrence", "-counter"]`.
- `page`, `page_size`, `offset`: Pagination controls.

Result structure (high level):

- `totals.ts_from`, `totals.ts_to`, `totals.count`
- `events.objects` with pagination fields (`page_number`, `page_size`, `item_count`, `page_count`)

Notes:

- Valid sort field name is `first_occurrence` (spelling verified).

EventRate

Returns total count and per-period counts for the given range.

Parameters:

- `time_range`: Time period.

- `filter` : Filter expression (optional).

Result structure (high level):

- `totals.ts_from` , `totals.ts_to` , `totals.count`
- `details[]` with `ts_from` , `ts_to` , `count`

TopN

Returns the top N values for a field in the specified period, with optional subperiod details and subfield breakdowns.

Parameters:

- `time_range`
- `field` (default: `host`)
- `with_subperiods` (bool)
- `top_periods` (bool)
- `filter`
- `limit`
- `show_other` (bool)
- `ignore_empty` (bool, default true)
- `subfields` (list of field names)
- `subfield_limit`

Result structure (high level):

- `totals.ts_from` , `totals.ts_to`
- `totals.values[]` of `{name, count}`
- Optional subfield breakdowns per value when `subfields` provided

LastN

Returns the last N values for a field during the period, sorted by recency.

Parameters:

- `time_range`
- `field` (default: `host`)
- `filter`
- `limit`
- `ignore_empty` (bool)

- `last_seen_threshold` (number)

Result highlights:

- Same overall structure as TopN totals, with `last_seen` information for each value and sorting by recency instead of counts.

Export formats

See Query Export Formats for supported exports by query type.

System Query Types

LogZilla system query reference for SystemStatus, ProcessingStats, StorageStats, and host metrics covering CPU, memory, disk, network, and IOPS

Query Types —- System/Admin Queries

These queries expose system metrics, storage counters, and status. Names and fields are verified against `lib/logzilla/query.py` and API wiring.

- SystemStatus
- ProcessingStats
- StorageStats
- System_CPU
- System_Memory
- System_DF
- System_Network
- System_NetworkErrors
- System_IOPS (alias: System_IOPSQuery)
- Notifications
- Tasks

SystemStatus

High-level system information and disk usage. No time range.

Parameters:

- `show_usage_stats` (bool): include disk usage breakdowns.

ProcessingStats

Internal processing counters for the pipeline.

Parameters:

- `time_range`, `timezone`
- `show_daily_stats` (bool)

Result highlights:

- `totals` and `details` with `new`, `duplicates`, optional `daily_stats`.

StorageStats

Storage counters over a period (rounded to full hours).

Parameters:

- `time_range`, `timezone`
- `show_daily_stats` (bool)
- `show_raw_counters` (bool)

Result highlights:

- `totals` for the period, `all_time` counters.

System_CPU

CPU usage metrics per-CPU or totals.

Parameters:

- `time_range`, `timezone`
- `cpu`: CPU index or `totals`
- Optional filters: `host`

Fields:

- `usage_softirq`, `usage_system`, `usage_idle`, `usage_user`, `usage_irq`, `usage_nice`, `usage_steal`, `usage_iowait`.

System_Memory

Memory utilization metrics.

Parameters:

- `time_range`, `timezone`
- Optional filters: `host`

Fields:

- `buffered`, `cached`, `free`, `used`.

System_DF

Filesystem usage metrics per mount/path.

Parameters:

- `time_range`, `timezone`
- Filters: `host`, `path` (mapped to `fs` internally)

Fields:

- `free`, `used`. (Percentages can be derived by clients.)

System_Network

Network throughput derived from interface counters.

Parameters:

- `time_range`, `timezone`
- Filters: `host`, `interface`

Fields (custom rates):

- `bps_recv` = derivative of `bytes_recv` × 8
- `bps_sent` = derivative of `bytes_sent` × 8

System_NetworkErrors

Network interface error and drop counters.

Parameters:

- `time_range`, `timezone`
- Filters: `host`, `interface`

Fields:

- `drop_in`, `drop_out`, `err_in`, `err_out`.

System_IOPS (System_IOPSQuery)

Disk I/O operations per second (IOPS).

Parameters:

- `time_range`, `timezone`

- Filters: `host`, `disk` (exposed as `name` internally)

Fields (custom rates):

- `reads`, `writes` (derived per second)

Notes

- System queries depend on metrics availability and the `SYSTEM_METRICS_ENABLED` configuration.
- See Making Queries for the request flow.
- For export formats, see Query Export Formats.

Notifications

Notifications group query built on top of system status mechanics.

Parameters:

- `time_range`: Period to consider; values used against `time_range_field`.
- `timezone`: Timezone for interpretation.
- `time_range_field`: One of `updated_at`, `created_at`, `unread_since`, or `read_at`.
- `is_private`: Filter by private flag (bool).
- `read`: Filter by read flag (bool).
- `sort`: List. Defaults to descending by `time_range_field` when not provided.

Tasks

Task list query with filters and sorting.

Parameters:

- `time_range`: Period (default last day).
- `sort`: List of fields (string or list). Default `owner`.
- `is_open`: Filter by open/closed (bool).
- `is_overdue`: Filter by due date (bool).
- `assigned_to`: Filter by user ids (list); empty list means unassigned.
- `owner`: Filter by owner ids (list).

Query Api Parameters

Common LogZilla Query API parameters for time_range, filter expressions, sorting, pagination, and timezone handling across event and system query types

Query API Parameters

LogZilla Query API requests use common parameters for time ranges, filtering options, and result formats. These parameters provide flexible control over data retrieval and analysis operations across different query types.

Prerequisites

- Valid LogZilla API authentication token (see Getting Started)
- Understanding of query creation workflow (see Making Queries)
- JSON data format familiarity
- Boolean logic operators

Common Query Parameters

Although every query type defines its own list of parameters there are some parameters used by most of them:

time_range

For every query, the start and end time period of the desired data must be provided. For some queries, sub-periods within the given period are calculated based on the step parameter - i.e. when requesting event rates, results will contain values for each sub-period, such as all minutes in the last hour, or last 30 days, etc.

The time_range parameter is an object with the following fields:

ts_from
Timestamp (number of seconds from epoch) defining the beginning of the period, for which 0 (zero) can be used to use the current time, or a negative number to specify time relative to the current time

ts_to
Timestamp defining the end of the period. 0 or a negative number can be provided to get time relative to current

step
A single number defining the length of sub-periods in seconds; for example, 60 will create 1-minute periods, 900 will create 15-minute periods, and so on. The default is set heuristically according to ts_from and ts_to - i.e. when a 1 hour time range is requested `step` will be set to 1 minute, for the range of 1 minute or less `step` will be one second, and so on.

preset

As an alternative to ts_from and ts_to, the preset parameter can be used. Based on the timezone, it determines the start of the day and uses appropriate ts_from and ts_to values. Available presets include 'today', 'yesterday', or relative time patterns such as 'last_5_minutes', 'last_1_hour', 'last_7_days', 'last_2_weeks'

timezone

Determines the beginning of the day for preset parameter; by default, GLOBAL_TZ config value is used

Periods are always rounded to the nearest multiple of `step`. Rounding is always up so the last period is often partially in the future, such as if a step of 1 hour is requested and it is now 13:21 then the last period will be 13:00 - 14:00. This then results in results for the current hour being received despite the hour indicated not yet being complete.

For query types that do not use subperiods (such as "LastN") only ts_from and ts_to are important, but `step` and `round_to_step` to round can still be used. (Note that in earlier versions there was an option to provide `round_to_step` and `periods` parameters, which are now unsupported).

filter

By default, every query operates on all data (according to the given time range), but for each, a compound parameter "filter" can be provided which will filter results by selected fields (including as desired message text). This parameter is an array of filter conditions that are always "AND"-ed, meaning that each record must match all of the given conditions to be included in the final results. Filtering is always done before aggregating so if for example the event rate is queried and filtering is specified by hostname then only the number of events with this host name will be reported as the result count.

Every filter condition is an object with following fields:

name	description
`field`	name of the field to filter by, as it appears in results
`value`	actual value to filter by. for fields other than timestamp this can also be a list of possible values (only for "eq" comparison)
`op`	if type is numeric (this includes timestamps) this defines the type of comparison. see immediately below
`ignore_case`	determines whether text comparisons are case sensitive or not. Defaults to True, meaning all comparisons are case insensitive. To force case sensitive mode set ignore_case=False

operator	description
`eq`	value is an exact value to be found. this is the default when no comparison operator is specified. you can also specify list of possible values
`lt`	match only records with field less than the given value
`le`	match only records with field less than or equal to the given value

operator	description
`gt`	match only records with field greater than the given value
`ge`	match only records with field greater than or equal to the given value
`qp`	special operator for message boolean syntax

Examples

Return only events with counter greater than 5:

```
[ { "field": "counter", "op": "gt", "value": 5 } ]
```

Return events from host 'fileserver23' with severity 'ERROR' or higher:

```
[ { "field": "severity", "value": [0, 1, 2, 3] },
  { "field": "host", "value": "fileserver23" } ]
```

Return events from hosts "alpha" and "beta" matching "power failure" in event message text:

```
[ { "field": "message", "value": "power failure" },
  { "field": "host", "value": ["alpha", "beta"] } ]
```

Message Boolean Syntax

Boolean logic expressions enable complex filtering of log message content using logical operators. These expressions allow combining multiple search terms and conditions to create precise filters for message text matching.

Supported Operators

AND

which is also implicitly used between two words/expressions if there is no other operator specified

NOT

shortcut: use either **'!'** or **'-'**

OR

shortcut: **'|'**

Operators are case-insensitive, so: 'AND', 'and', 'AnD' are correct

Examples of forbidden expressions:

```
-Foobar1
Foobar1 | -Foobar2
```

Examples of correct expressions:

Return events containing words 'Foobar1' or 'Foobar2' and not 'Foobar3':

```
[ { "field": "message", "op": "qp", "value": "Foobar1 | Foobar2 !Foobar3" } ]
```

Return events containing words ('Foobar1' or 'Foobar2') and ('Foobar3' or 'Foobar4'):

```
[ { "field": "message", "op": "qp", "value": "(Foobar1 | Foobar2) (Foobar3 | Foobar4)" }
]
```

Common Results Format

The "results" container is always an object with one or several fields, usually containing "totals" and/or "details". The former contains results for the whole period whereas the latter is an array of values for subperiods. Both total and subperiod usually contain "ts_from" and "ts_to" timestamps, to show exact time range for the data retrieved, and then the result "values" or just "count".

See the description of the particular query type in:

- Event Queries
- System/Admin Queries

for details on result formats and examples.

Generic Results Format for System Queries

System queries return data collected by the telegraf system, for different system parameters, and are used for displaying system widgets (that can be used later on for diagnostic monitoring of system performance).

All these queries return "totals" and "details". For details the objects are similar to data for EventRateQuery but there are more keys with different values. An example from *System_CPUQuery*:

```
{
  "details": [
    {
      "ts_from": 1416231300,
      "ts_to": 1416231315,
      "softirq": 0,
      "system": 8.400342,
      "idle": 374.946619,
      "user": 16.067144,
      "interrupt": 0.20001199999999997,
      "nice": 0,
      "steal": 0,
      "wait": 0.20001199999999997
    },
    "..."
  ]
}
```

For totals instead of an array we have a single object with keys like above, but rather than a single value there is a set of values:

```
{
  "system": {
    "count": 236,
    "sum": 1681.6008720000007,
    "min": 5.2671220000000005,
    "max": 9.599976,
    "avg": 7.125427423728817,
    "last": 6.400112999999999,
    "last_ts": 1416234840
  }
}
```

So here there are different kinds of aggregates for the selected time period:

type	description
`count`	number of known values for the given time period
`sum`	total of those values (used for calculating avg)
`min`	minimum value
`max`	maximum value
`avg`	average value (sum / count)
`last`	last known value from given period

type	description
`last_ts`	timestamp when last known value occurred

Related Documentation

- Making Queries - Creating queries and handling asynchronous requests
- Event Queries and System/Admin Queries - Detailed documentation for all query types

Query Export

Export LogZilla query results through GET /api/query/{qid}/export as JSON, CSV, or XLSX, with per-query-type format support for Search, TopN, EventRate, and LastN

Query Export Formats

Exports are requested via `GET /api/query/{qid}/export`.

Parameters:

- `ftype`: `json` (default), `csv`, or `xlsx` (see per-type support below).
- `timezone`: Optional. Timezone name used for date formatting.
- `filename`: Optional. Suggested file name for attachment.

Supported formats by query type (verified in `lib/logzilla/query_export.py`):

- Search —- `json`, `csv`, `xlsx`
- TopN —- `json`, `xlsx`
- EventRate —- `json`
- LastN —- `json`

Notes:

- Microsoft Excel format is `xlsx` (correct spelling). `docx` and `pdf` are not implemented by the exporter classes.
- The export response is a file download with appropriate content type.

Example

```
curl -H "Authorization: token YOUR_TOKEN" \
     -G "http://your-logzilla-server/api/query/QID_HERE/export" \
     --data-urlencode "ftype=xlsx" \
     --data-urlencode "timezone=UTC" \
     --data-urlencode "filename=search_results.xlsx" \
     -o search_results.xlsx
```

See Making Queries for creating a query and retrieving its `qid`, and the interactive docs at `/api/docs` for full endpoint details.

Sample Api Code

Working Python and Bash examples for the LogZilla API, covering token authentication, single-event retrieval, and multi-event search query execution

Sample API Code

There are two examples here, one a Python example that does a simple API query to retrieve an event by ID, and the other is a Bash example that does a search query for multiple events.

Python Example

Prerequisites

- Python 3.6 or higher
- requests library

```
import requests
import json
import time

## Configuration

# Replace with actual LogZilla API URL and authentication token.
BASE_URL = "https://your-logzilla-server/api"
AUTH_TOKEN = "your-api-token-here"

HEADERS = {
    "Authorization": f"token {AUTH_TOKEN}",
    "Content-Type": "application/json"
}

## API Request Function

def make_api_request(url, method="GET", payload=None):
    """
    Reusable function to make API requests with comprehensive error handling.
    """
    try:
        if method == "GET":
            response = requests.get(url, headers=HEADERS, timeout=10)
        elif method == "POST":
            # Apply timeout for consistency
            response = requests.post(url, json=payload, headers=HEADERS, timeout=10)
        else:
            # Unsupported method
            raise ValueError(f"Unsupported HTTP method: {method}")

        response.raise_for_status()
        return response.json()

    except requests.exceptions.HTTPError as e:
        if e.response.status_code == 401:
            print("API Error: 401 Unauthorized. Token refresh required.")
        elif e.response.status_code == 429:
            print("API Error: 429 Too Many Requests. Waiting 60 seconds...")
            time.sleep(60)
        elif e.response.status_code == 404:
            print(f"API Error: 404 Not Found at {url}")
            return None
        else:
            print(f"API Error: {e.response.json().get('detail', str(e))}")
        raise
    except requests.exceptions.RequestException as e:
        print(f"Network error: {e}")
        raise
```

```
## Usage Example

EVENT_ID = "your-event-id-here"
event_url = f"{BASE_URL}/events/{EVENT_ID}"

try:
    print(f"Retrieving event with ID: {EVENT_ID}")

    event_data = make_api_request(event_url, method="GET")

    if event_data is not None:
        print("Retrieved event data:")
        print(json.dumps(event_data, indent=2))
    else:
        print("Event not found.")

except ValueError as e:
    print(f"Request configuration error: {e}")
except requests.exceptions.RequestException:
    print("Request failed. Check error messages for details.")
```

Bash Example

Prerequisites

- Bash
- curl
- jq

```
#!/bin/bash

LOGZILLA_SERVER="https://your-logzilla-server"
API_ENDPOINT="/api/query"
HOST_DICT_ENDPOINT="/api/dictionaries/host?limit=100&show_last_seen=false"
TOKEN="YOUR_TOKEN_HERE"
AUTH_HEADER="Authorization: token $TOKEN"

FOUND_FILE="exists.txt"
MISSING_FILE="missing.txt"

# Clear output files
> "$FOUND_FILE"
> "$MISSING_FILE"

echo "Fetching hosts from $HOST_DICT_ENDPOINT..."

# Pull host list with fallback for both .value and .name formats
HOST_LIST=$(curl -s "$LOGZILLA_SERVER$HOST_DICT_ENDPOINT" \
  -H "$AUTH_HEADER" \
  -H "Accept: application/json" | jq -r '.list[] | .value // .name')

if [[ -z "$HOST_LIST" || "$HOST_LIST" == "null" ]]; then
    echo "Error: No valid hosts returned from API."
    exit 1
fi

for host in $HOST_LIST; do
    [[ -z "$host" || "$host" == "null" ]] && continue

    echo "Querying events for host: $host"

    response=$(curl -s -X POST "$LOGZILLA_SERVER$API_ENDPOINT" \
        -H "Content-Type: application/json" \
        -H "Accept: application/json" \
        -H "$AUTH_HEADER" \
        -d "{
              \"type\": \"Search\",
              \"params\": {
                \"live_query\": false,
                \"limit\": 1,
                \"filter\": [
                  {
                    \"op\": \"eq\",
                    \"field\": \"host\",
                    \"value\": \"$host\",
                    \"ignore_case\": true
                  }
                ],
```

```
                \"time_range\": {
                  \"preset\": \"last_2_days\"
                },
                \"with_events\": true
              }
            }")

    query_id=$(echo "$response" | jq -r '.query_id')

    if [[ "$query_id" == "null" || -z "$query_id" ]]; then
        echo "Error: query_id not found for host $host"
        echo "$host" >> "$MISSING_FILE"
        continue
    fi

    # echo "Submitted query_id: $query_id"

    while true; do
        sleep 2
        result=$(curl -s "$LOGZILLA_SERVER$API_ENDPOINT/$query_id" \
                 -H "$AUTH_HEADER" \
                 -H "Accept: application/json")

        status=$(echo "$result" | jq -r '.status')

        if [[ "$status" == "SUCCESS" ]]; then
            count=$(echo "$result" | jq -r '.results.totals.count // 0')
            if [[ "$count" -gt 0 ]]; then
                echo "✓ Host $host has events ($count)"
                echo "$host" >> "$FOUND_FILE"
            else
                echo "✗ Host $host missing events"
                echo "$host" >> "$MISSING_FILE"
            fi
            break
        elif [[ "$status" == "IN_PROGRESS" || "$status" == "PENDING" ]]; then
            echo "Waiting on host $host..."
        else
            echo "Error or unknown status for host $host"
            echo "$host" >> "$MISSING_FILE"
            break
        fi
    done

    echo "---"
done
```

CHAPTER 10

Administration

LogZilla administration topics covering licensing, security, authentication, archiving, migration, syslog ingest, and performance tuning

LogZilla administration encompasses server management, security configuration, user access control, and system maintenance. Administrators can configure licensing, manage data retention, implement security protocols, and maintain optimal system performance.

Core Administration Topics

- **Server Licensing**: License management and host key configuration
- **Migration and Deployment**: Server migration, offline installations, and upgrades
- **Security Configuration**: HTTPS, TLS tunnels, LDAP authentication, and role-based access control
- **Network Configuration**: Syslog basics, network communications, and custom DNS setup
- **System Maintenance**: Backend configuration, search settings, archiving, and troubleshooting
- **Command Line Tools**: Comprehensive utilities for system management and querying

Software Lifecycle And Release Notes

LogZilla release lifecycle policy and version release notes covering the current release plus the two prior minor versions

Software Lifecycle Policy and Release Notes

LogZilla provides a defined lifecycle for each software release to ensure reliability, performance improvements, and timely security updates.

Support Phases

End of Support (EOS).

A release receives active support, including feature updates, bug fixes, and noncritical enhancements, for one year from its initial release date.

End of Life (EOL).

A release continues to receive security fixes and critical issue patches for two years from its initial release date. After this period, the release reaches EOL and no further updates are provided.

Customer Responsibilities

Customers are encouraged to remain on supported versions to receive the latest improvements, ensure platform stability, and maintain compliance with current security practices. Upgrading regularly helps avoid operating on outdated or unsupported components.

Extended Support

For deployments requiring an extended maintenance window beyond the standard policy, additional support options may be made available upon request.

Release Lifecycle Dates

Version	EOS	EOL
v6.31.0	Dec 15, 2024	Dec 15, 2025
v6.32.0	Apr 8, 2025	Apr 8, 2026
v6.33.0	Jun 27, 2025	Jun 27, 2026
v6.34.0	Feb 6, 2026	Feb 6, 2027

Version	EOS	EOL
v6.35.0	Apr 8, 2026	Apr 8, 2027
v6.36.0	Jul 2, 2026	Jul 2, 2027
v6.37.0	Jul 25, 2026	Jul 25, 2027
v6.38.0	Dec 5, 2026	Dec 5, 2027
v6.39.0	Jan 5, 2027	Jan 5, 2028
v6.40.0	Feb 2, 2027	Feb 2, 2028
v6.41.0	Apr 22, 2027	Apr 22, 2028

Release Notes

This page lists the last three releases. Older notes are available in source control.

Version v6.41

Notable Features

- **App-Schema-Aware AI Queries** - The LogZilla AI Copilot is now aware of every installed application's schema, tag taxonomy, and vendor context. A TopN vendor pre-filter surfaces the most active applications to the query agent automatically, producing substantially more accurate results for vendor-specific natural-language searches.
- **AI Multi-Model Accuracy (LZ-3341)** - Query agent prompts have been restructured for token efficiency and reliability across a wider range of LLM providers, including smaller and locally hosted models, with added decomposition guidance for complex multi-step requests.
- **New AppStore Applications** - Added LogZilla apps for BlueCat DNS (DNSTAP), FortiAnalyzer, FortiWeb, Arista EOS, Cisco IOS-XR, and Dell N-Series switches. Each includes parsing rules, dashboards, triggers, and event generators.

API Updates

- **LZ-3335 - Query Agent Prompt Rewrite** - Restructured the query agent prompt into a token-efficient format that retains clarity on low-end models, improving consistency of generated queries.
- **LZ-3317 - App Schema Injection for AI Queries** - The AI query agent now receives the installed application schema and vendor context from a TopN pre-filter, producing more accurate results for vendor-specific searches.
- **LZ-3316 - Query Decomposition Guidance** - Tightened the main agent prompt with explicit decomposition instructions for multi-step queries.
- **LZ-3328 - Query Timestamp Handling** - Resolved an issue where the query agent occasionally mixed a preset time range with explicit ts_from and ts_to parameters.
- **LZ-3320 - Configurable LLM Request Timeout** - Added a new REQUEST_TIMEOUT setting (default 300 seconds, range 30 to 3600) to the AI configuration, adjustable through ai.yaml or the Settings interface. The default was tuned to

accommodate tool-call buffering on Ollama deployments running on slower GPUs.

- **LZ-3333 - Claude 4.6 Model Support** - Added support for the claude-opus-4-6 and claude-sonnet-4-6 model names in the AI configuration.
- **LZ-3329 - AI Model Type Consolidation** - Consolidated the "vllm" and "ollama" model types into a single "vllm/ollama" option that covers both backends. Existing saved configurations using "vllm" are migrated automatically; configurations still using the old "ollama" value are rejected with a message describing how to update the selection.
- **LZ-3338 - Dynamic Phoenix Tracing** - Phoenix trace export can now be enabled or disabled at runtime without restarting the AI worker process.
- **LZ-3336 - vLLM Streaming Reliability** - Addressed an issue where concurrent vLLM streaming sessions could collide; each streaming request now uses an isolated client instance.
- **LZ-3343 - AI Timeout Guards** - Added timeout protection to the explain_event and ping_llm paths so stalled model calls no longer delay event analysis.
- **LZ-3324 - Typed Query Timeout Errors** - Introduced a typed QueryTimeoutError returned to AI clients, allowing agents to distinguish timeouts from other query failures and retry appropriately.
- **LZ-3323 - AI Module Resource Handling** - Addressed resource handling in the AI module that could leave database connections and threads lingering under sustained load.
- **LZ-3313 - Phoenix Trace Export** - Corrected a tracing configuration issue that prevented spans from reaching Phoenix when enabled.
- **LZ-3311 - FortiGate WAF and Tagging** - Added WAF event support to the FortiGate application and introduced HC_TAGS handling across the Fortinet app family to reduce cardinality on high-volume fields.
- **LZ-3354 - Dashboard and Widget Validation** - Serializers now reject unknown fields in widget and dashboard payloads, surfacing typos earlier instead of silently discarding them.
- **LZ-3354 - IOS-XR Node Extraction** - Corrected the order of IOS-XR parent-rule processing so node identifiers are captured before the generic Cisco prefix cleanup runs.
- **LZ-3360 - SonicWall SonicOS 7+** - Updated the SonicWall application to recognize the customized Syslog ID format introduced in SonicOS 7, and replaced message-based triggers with tag-based filters.
- **LZ-3340 - Dell Switch Receiving Port** - Added a dedicated receiving port for Dell N-Series switches.
- **LZ-3358 - Infoblox NIOS Expanded Coverage (upgrade action required)** - The Infoblox application has been expanded from DNS-query-only coverage to the full NIOS product: DNS, DHCP, administrative audit, Advanced DNS Protection, and Response Policy Zone events. Because some NIOS events use a non-RFC syslog header that collides with standard daemon names, the application now requires a dedicated receiving port. Customers already running the Infoblox app should set **Syslog Infoblox Port** under Settings → System Settings → Application Ports and reconfigure their Infoblox grid members to send to the new port. Until the port is configured and the grid members are repointed, Infoblox events will not be parsed by the application.
- **LZ-3307 - Lifecycle Date Correction** - Adjusted the published end-of-support and end-of-life dates to match the current lifecycle policy.

UI Updates

- **LZ-3353 - License Status Reporting** - The interface now displays the specific reason for a license error (such as exceeding the events per day limit), and on-prem deployments are redirected to a dedicated license error page when a 423 response is returned.
- **LZ-3291 - Upgrade Notification Action** - Replaced an outdated documentation link in the upgrade notification with a clipboard copy action that captures the upgrade command.
- **LZ-3305 - Trigger Filter Initialization** - Addressed an issue affecting some customers where the match-criteria field was missing when creating or editing triggers in the legacy interface.

Version v6.40

Notable Features

- **LogZilla AI Configuration** - AI settings are now managed through Admin → Settings in the UI rather than on-disk configuration files.
- **Event Taxonomies** - All LogZilla apps now follow a shared taxonomy standard and include additional drill-down tags such as Vendor, Product, MITRE, and ~150 compliance mappings (PCI-DSS, HIPAA, SOX, GDPR, and others).

API Updates

- **LZ-3271 - VMware App Update** - Updated the VMware application with refined event generation behavior.
- **LZ-3256 - F5 Dashboard** - Added a new dashboard for F5 device monitoring.
- **LZ-2657 - Dashboard Metric Badges** - Added count-style badge widgets to application dashboards for improved data visualization.
- **LZ-3297 - Watcher CPU Optimization** - Addressed higher-than- expected CPU usage from status checks in the watcher component.
- **LZ-3293 - Taxonomy Consolidation** - Created a unified source for application taxonomies and MITRE ATT&CK mappings.
- **LZ-3277 - Trigger Configuration Regeneration** - Addressed an issue affecting some customers where trigger configurations were not regenerated after app reinstallation.
- **LZ-3276 - Upgrade Version Detection** - Corrected version detection during upgrades so the latest available release is identified.
- **LZ-3274 - App Standards Compliance** - Updated multiple applications to follow new taxonomy standards and removed duplicate rules.
- **LZ-3244 - Tag Naming Consistency** - Standardized IP-related tag names across applications to reduce cardinality.
- **LZ-3230 - User-Defined MCP Servers** - Added support for user-defined MCP servers to enhance AI context for queries.
- **LZ-3227 - Container File Limits** - Adjusted file descriptor limits in containers for improved stability.
- **LZ-3221 - SEC Rule Support** - Modified App Store applications to support Event Correlation example configurations.
- **LZ-3173 - Start Command Timeout Flag** - Corrected the `--no-timeout` switch behavior for the `logzilla start` command.
- **LZ-3157 - SonicWall App Enhancements** - Added user login and VPN session tracking tags to the SonicWall application.

- **LZ-3083 - Report Templates** - Enhanced report templates with additional data sources and output format options.
- **LZ-2700 - Search Results Formatting** - Improved formatting options for search results display.
- **LZ-3186 - AI Provider Expansion** - Added support for additional AI providers including Gemini, vLLM, Ollama, and OpenRouter. Note: v6.41 consolidated the `vllm` and `ollama` model types into a single `vllm/ollama` option.
- **LZ-3038 - API Validation Enhancements** - Added input validators and refined API response handling.

UI Updates

- **LZ-3272 - AI Session Copy Function** - Enabled copying of individual AI chat responses within a session.
- **LZ-3265 - AI Copilot Settings Save** - Corrected a disabled save button in the AI Copilot settings section.
- **LZ-3262 - App Store List Display** - Corrected truncation of the last item in the App Store list view.
- **LZ-3258 - Severity Chart Labels** - Corrected severity name mapping so text labels appear instead of integers in all chart types.
- **LZ-3251 - Error Message Improvements** - Improved error handling and messaging after legacy interface deprecation.
- **LZ-3240 - Trigger Filter Protection** - Restricted editing of regex-based trigger filters to avoid accidental overwriting.

Version v6.39

API Updates

- **LZ-3248 - Windows Agent Origin Computer** - Added origin computer information from Windows event data as a user tag.
- **LZ-3242 - AI Query Timestamp Validation** - Addressed an issue affecting some customers where an exception surfaced when timestamp range validation failed in AI queries.
- **LZ-3238 - AppStore Config Validation Testing** - Added automated testing to validate application configurations against their schemas.
- **LZ-3237 - Operations Dashboard App** - Created a new Operations app providing a centralized dashboard for high-severity events, service disruptions, application errors, and infrastructure changes.
- **LZ-3230 - User-Defined MCP Servers** - Added support for user-defined MCP servers to enhance AI context for queries such as root cause analysis and event correlation.
- **LZ-3227 - Container File Descriptor Limits** - Corrected the open-file limit in containers so services requiring many file handles operate as expected.
- **LZ-3219 - JSON Rules User Tags** - Added support for user tags in the rewrite section of JSON rules.
- **LZ-3217 - Lua Rules Event Generation** - Lua rules can now generate new events, supporting additional event processing workflows.
- **LZ-3214 - Rule Validation Fix** - Addressed an issue affecting validation of rule files with list structures at the top level.
- **LZ-3208 - Integrated AI Chatbot** - Introduced an AI chatbot integrated with the LogZilla API, removing the need for separate containers or databases.
- **LZ-3205 - WebSocket Migration** - Migrated WebSocket functionality from Tornado to Django Channels for improved performance and maintainability.

- **LZ-3204 - Help Menu Link Update** - Updated the help menu to link directly to the LogZilla documentation website.
- **LZ-3192 - Unified Query API Filters** - Simplified query API filter logic to improve usability and AI query generation accuracy.
- **LZ-3184 - Palo Alto Prisma App** - Created a new LogZilla app for Palo Alto Prisma integration.
- **LZ-3176 - HTTP Receiver Fix** - Addressed an attribute error in the HTTP receiver token management component.
- **LZ-3157 - SonicWall App User Tags** - Added user information tags to the SonicWall app for tracking VPN login activity.
- **LZ-3156 - Minimum Word Length Setting** - Updated the minimum word length setting for search indexing.
- **LZ-3146 - HTTP Receiver Documentation** - Corrected a path typo in the HTTP Event Receiver documentation.
- **LZ-3120 - Ubiquiti Rule Source Check** - Updated the Ubiquiti rule to check for source tags so filtering works regardless of port configuration.
- **LZ-3118 - Palo Alto App Documentation** - Updated Palo Alto app documentation with Traffic, System, and Config event format examples.
- **LZ-3116 - CDR Loader Startup Fix** - Addressed an issue affecting CDR Loader startup on some deployments.
- **LZ-3086 - Dashboard Export** - Added the ability to export dashboards directly from the dashboard menu.
- **LZ-3085 - Widget Export** - Added the ability to export widget data in JSON, XLS, and PNG formats.
- **LZ-3083 - Report Templates Enhancement** - Enhanced report templates with additional data sources, format selectors, and Excel integration.
- **LZ-3081 - New Reports Engine** - Introduced a new reports engine supporting scheduled reports, multiple output formats, and direct exports from the interface.
- **LZ-2858 - Ubiquiti UniFi OS Support** - Added dedicated receiving port and rules for Ubiquiti UniFi device logs including iptables events.
- **LZ-2720 - AV Rules to Apps Conversion** - Initiated conversion of antivirus vendor rules to LogZilla apps for improved management.
- **LZ-2693 - Checkpoint firewall app** - Added a new Checkpoint firewall application to the AppStore.
- **LZ-2482 - Juniper Dashboard installation** - Addressed an issue affecting some customers where the Juniper dashboard was missing after app installation.
- **LZ-2480 - Implement Ford triggers** - Added over 300 triggers for Cisco WAAS, Cisco ACI, Cisco IOS, Cisco Nexus, Cisco ACE, SDWAN, Cisco ASA, and F5 platforms.
- **LZ-3166 - AI chat API** - Created a REST API for the LogZilla AI chatbot functionality.
- **LZ-3236 - AI installation documentation** - Improved formatting and clarity in the AI installation documentation.
- **LZ-3235 - Splunk HEC forwarder documentation** - Updated documentation to clarify both HTTP and HTTPS support for Splunk HEC forwarding.
- **LZ-3232 - End of support policy documentation** - Added software lifecycle policy documentation with end-of-support and end-of-life definitions.
- **LZ-3194 - Cloud onboarding video** - Added cloud onboarding video to the documentation.
- **LZ-3189 - API documentation correction** - Corrected errors in the detailed API method documentation.
- **LZ-3148 - API documentation table formatting** - Corrected table formatting in the API method documentation.

- **LZ-3145 - HTTP ingestion endpoint documentation** - Clarified usage of `incoming` versus `incoming/raw` endpoints for HTTP ingestion.
- **LZ-3144 - HTTP event receiver documentation** - Updated documentation to explain the difference between `extra_fields` and `json` parameters.
- **LZ-3129 - Trigger script documentation** - Unified duplicated trigger execution script help sections into a single reference.
- **LZ-3044 - API documentation section correction** - Corrected section 9.2 so API documentation appears in place of forwarder information.
- **LZ-2992 - Static IP configuration guide** - Added step-by-step instructions for setting a static IP address in the LogZilla VM.
- **LZ-2932 - ServiceNow integration documentation** - Created documentation and examples for ServiceNow ticket integration.
- **LZ-2684 - Detailed installation guide** - Added an installation guide covering system requirements through post-installation verification.
- **LZ-2599 - Documentation rewrite** - Completed a major rewrite of the LogZilla documentation.

UI Updates

- **LZ-3251 - Error handling improvements** - Improved error-message handling and added graceful fallback when the legacy UI is disabled.
- **LZ-3241 - AI chat message copy functionality** - Addressed an issue affecting some customers where copying AI chat messages did not include content above embedded charts.
- **LZ-3225 - Search export functionality** - Addressed an error that could surface when exporting search results to CSV or XLS format.
- **LZ-3224 - Icon package replacement** - Replaced the icons package and applied style improvements throughout the interface.
- **LZ-3218 - AI chat integration** - Integrated AI chat functionality into the new interface.
- **LZ-3179 - AI interface connection** - Connected the AI user interface to the new AI API.
- **LZ-3113 - Account management buttons** - Added activate and delete buttons to the account management interface.

Server Licensing

View and update a LogZilla server license through the UI or the logzilla license CLI, including host key retrieval and offline activation

Server license information can be obtained via the UI or command line.

Viewing Host Key

To locate server license information in the LogZilla interface, navigate to `Settings -> System Settings -> License Information`.

The License Information section displays details about the current server license, such as the host key, expiration date, and allowed features.

Checking Host Key via Shell

To view the LogZilla server's host key, access the server's shell using the console or SSH. Once logged in, execute the following command:

```
logzilla license key
```

The logzilla license key command returns the unique server key, which can be provided to the LogZilla account manager. For example:

```
73cde9bfce9a15a0ae3a97f0c501231712813fc6
```

Updating License

After the LogZilla account manager confirms that the license has been updated on the licensing server, update the server's license by running the following command:

```
logzilla license download
```

LogZilla does not need to be restarted for the key to take effect.

Manually Installing License

If the server is offline, download the license from a different system and manually transfer it to the server. In the example below, the host key `73cde9bfce9a15a0ae3a97f0c501231712813fc6` is used, but replace it with the actual key obtained from one of the methods noted above.

Browser Download

To download the license using a web browser, visit the LogZilla license server URL with the host key appended. For example, if the host key is `73cde9bfce9a15a0ae3a97f0c501231712813fc6`, visit:

`https://license.logzilla.net/keys/73cde9bfce9a15a0ae3a97f0c501231712813fc6`

Command Line Download

To download the license using a terminal from another Linux system, use wget with the host key URL:

```
wget https://license.logzilla.net/keys/73cde9bfce9a15a0ae3a97f0c501231712813fc6 -O
lic.json
```

Replace `73cde9bfce9a15a0ae3a97f0c501231712813fc6` with the actual host key.

Installing the Downloaded License

After obtaining the JSON license file from the license server:

Save the downloaded JSON content to a file (e.g., `lic.json`)

Load the license on the offline server using the following command:

```
logzilla license load lic.json
```

This action places the license file in the appropriate directory, allowing the LogZilla server to recognize and use the updated license information.

License Enforcement

When a LogZilla server exceeds a license limit, the interface redirects signed-in users to a dedicated license error page that displays the specific reason for the restriction, such as an expired license or an exceeded events per day (EPD) allowance. Administrators can use the reported reason to determine the appropriate remediation, which typically involves contacting the LogZilla account manager to adjust the license entitlement and then downloading the updated license as described in the preceding sections.

The same error reason is also recorded in the server logs to support troubleshooting in environments where the interface is not immediately accessible.

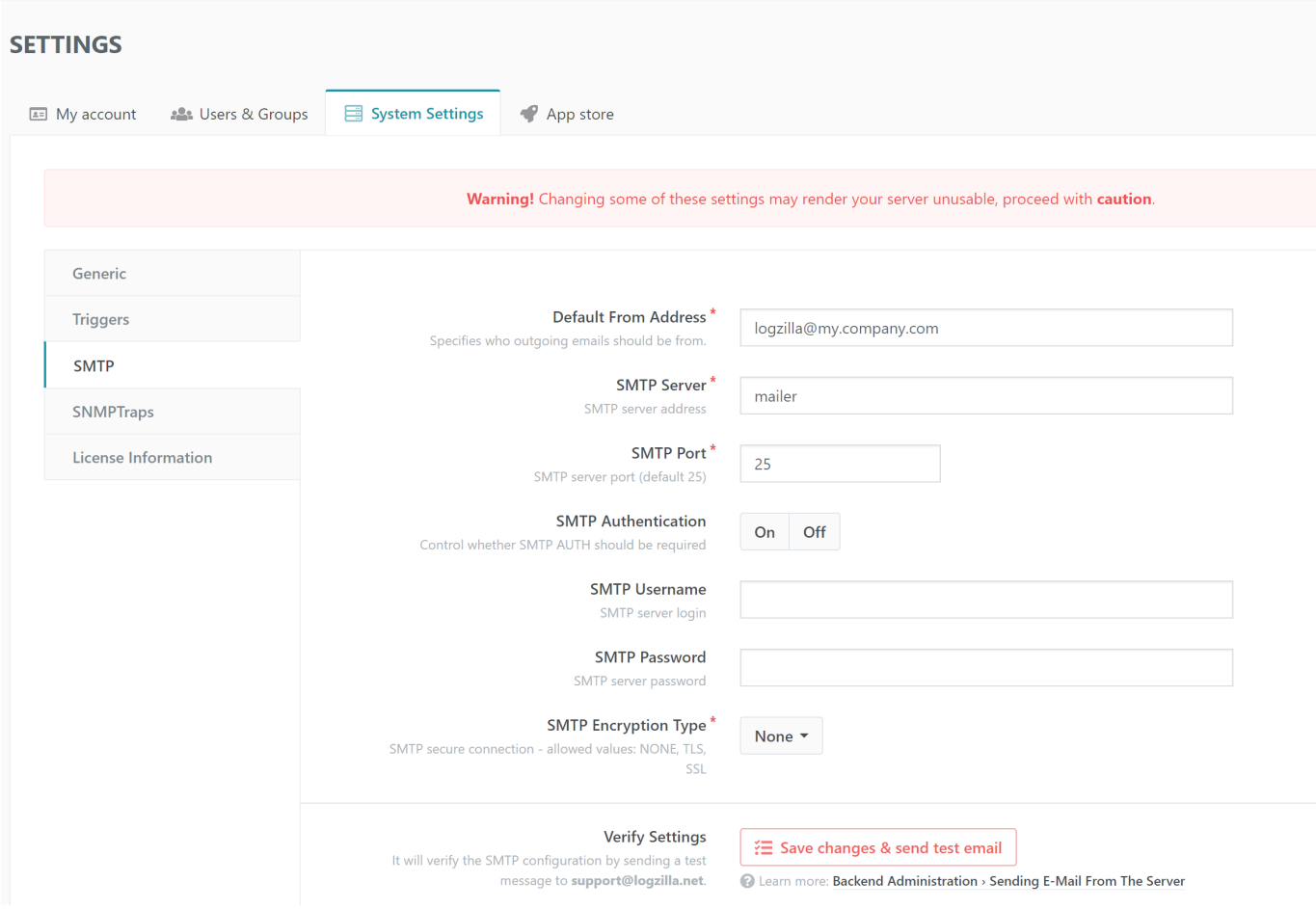
Sending Email

Configure the LogZilla SMTP mailer for trigger alerts and system notifications, covering sender address, server host, ports, and authentication

LogZilla requires SMTP server configuration to send email alerts and notifications. Administrators can configure email settings through the web interface to enable automated email delivery for triggers, alerts, and system notifications.

Accessing SMTP Settings

To configure email settings, navigate to `/settings/system/smtp` in the LogZilla web interface or access it through the Settings menu under System Settings.



SMTP Configuration Options

Basic Settings

Default From Address Specifies the sender email address for outgoing emails. This address appears in the "From" field of all emails sent by LogZilla.

SMTP Server The hostname or IP address of the SMTP mail server. This can be an internal mail server or an external service like Gmail, Office 365, or other email providers.

SMTP Port The port number used to connect to the SMTP server. Common ports include:

- Port 25: Standard SMTP (unencrypted)
- Port 587: SMTP with STARTTLS (recommended)
- Port 465: SMTP over SSL/TLS

Authentication

SMTP Authentication Toggle to enable or disable SMTP authentication. Most modern mail servers require authentication to prevent unauthorized email sending.

SMTP Username The username for authenticating with the SMTP server. This is typically an email address or account name.

SMTP Password The password associated with the SMTP username. For security, this field is masked in the interface.

Security Settings

SMTP Encryption Type Configures the encryption method for secure email transmission:

- **NONE**: No encryption (not recommended for production)
- **STARTTLS**: Upgrades the connection to encrypted after initial handshake
- **SSL/TLS**: Establishes an encrypted connection from the start

SMTP Connection Timeout Sets the timeout duration (in seconds) for establishing connections to the SMTP server. Default is typically 5 seconds.

Testing Configuration

After configuring SMTP settings, use the "Send test email" button to verify the configuration. The test email will be sent to the specified recipient to confirm that:

- SMTP server connectivity is working
- Authentication credentials are correct
- Email delivery is functioning properly

Common SMTP Providers

Gmail Configuration

- **SMTP Server**: smtp.gmail.com
- **Port**: 587
- **Encryption**: STARTTLS
- **Authentication**: Required
- **Note**: Requires app-specific password for accounts with 2FA enabled

Office 365 Configuration

- **SMTP Server**: smtp.office365.com
- **Port**: 587
- **Encryption**: STARTTLS
- **Authentication**: Required

Internal Mail Servers

Configuration varies based on the specific mail server software and security policies. Consult with the email administrator for proper settings.

Troubleshooting

Connection Issues

- Verify SMTP server hostname and port
- Check firewall rules allowing outbound connections
- Confirm network connectivity to the mail server

Authentication Failures

- Verify username and password credentials
- Check if the account requires app-specific passwords
- Ensure the account has permission to send email via SMTP

Email Delivery Problems

- Check spam/junk folders on recipient systems
- Verify the "From" address is properly configured
- Review mail server logs for delivery errors

Security Considerations

- Use encrypted connections (STARTTLS or SSL/TLS) when possible

- Store SMTP credentials securely
- Regularly rotate SMTP account passwords
- Monitor email sending for unusual activity
- Configure appropriate firewall rules for SMTP traffic

Network Communications

Network protocols, ports, and transports used by LogZilla for syslog ingest, raw data listeners, HTTP receivers, and management endpoints

LogZilla supports multiple network communication protocols and ports to accommodate diverse logging environments. The platform receives data through standard syslog protocols, raw data formats, and HTTP-based interfaces.

Syslog Communication

LogZilla receives syslog data using industry-standard protocols and formats:

RFC 3164 (BSD Syslog)

- **Default Port**: 514
- **Protocols**: TCP and UDP
- **Format**: Traditional BSD syslog format
- **Use Case**: Legacy devices and standard syslog implementations

RFC 5424 (Structured Syslog)

- **Default Port**: 601
- **Protocol**: TCP only
- **Format**: Modern structured syslog with enhanced metadata
- **Use Case**: Applications requiring structured data fields

Raw Data Communication

LogZilla accepts non-syslog data formats for specialized use cases:

Text Data

- **Default Port**: 516
- **Protocols**: TCP and UDP
- **Format**: Plain text messages
- **Use Case**: Devices sending non-standard or malformed syslog data

JSON Data

- **Default Port**: 515
- **Protocol**: TCP only
- **Format**: JSON-formatted messages

- **Use Case**: Applications sending structured JSON logs

Note: Raw data requires LogZilla apps or parsing rules to interpret and process the incoming messages effectively.

HTTP/HTTPS Communication

Web Interface

- **HTTP Port**: 80
- **HTTPS Port**: 443
- **Purpose**: User interface access and API endpoints

HTTP Log Reception

The same HTTP/HTTPS ports support log data ingestion. For detailed configuration, refer to the HTTP Event Receiver documentation.

Port Configuration

Network ports can be customized using the `logzilla config` command:

```
logzilla config SYSLOG_RAW_PORT 516
```

Available Configuration Options

Configuration Option	Default	Description
SYSLOG_BSD_TCP_PORT	514	TCP port for RFC 3164/BSD syslog messages
SYSLOG_BSD_UDP_PORT	514	UDP port for RFC 3164/BSD syslog messages
SYSLOG_RFC5424_PORT	601	TCP port for RFC 5424 syslog messages
SYSLOG_JSON_PORT	515	TCP port for raw JSON messages
SYSLOG_RAW_PORT	516	TCP port for raw text messages
SYSLOG_RAW_UDP_PORT	516	UDP port for raw text messages

Network Security Considerations

Firewall Configuration

Ensure appropriate firewall rules allow traffic on configured ports:

- Standard syslog ports (514, 601) for log reception

- HTTP/HTTPS ports (80, 443) for web interface access
- Custom ports if non-default configurations are used

Protocol Selection

- **Use TCP** for reliable delivery and flow control
- **Use UDP** for high-volume environments where occasional loss is acceptable
- **Use HTTPS** for encrypted web interface access in production

Access Control

- Restrict syslog port access to authorized log sources
- Implement network segmentation for sensitive logging infrastructure
- Monitor unusual traffic patterns on logging ports

Performance Optimization

High-Volume Environments

- Use TCP for better flow control during traffic bursts
- Monitor queue depths and connection counts
- Configure syslog receivers, batching, and buffering (see Server Settings and Syslog Settings).

Network Bandwidth

- Consider log compression for high-volume remote sources
- Implement log filtering at the source to reduce network traffic
- Use structured formats (RFC 5424, JSON) for more efficient parsing

Troubleshooting Network Issues

Connection Problems

- Verify firewall rules allow traffic on configured ports
- Check network connectivity between sources and LogZilla server
- Confirm port configurations match between senders and receivers

Performance Issues

- Monitor network utilization during peak logging periods
- Check for packet loss on UDP connections
- Verify adequate bandwidth for expected log volume

Configuration Validation

- Test port changes in staging environments first
- Verify configuration changes take effect after service restart
- Monitor logs for connection errors after configuration changes

Syslog Basics

Syslog protocol fundamentals covering RFC 3164 BSD format, RFC 5424 structured format, and RFC 6587 TCP and TLS transport for reliable log delivery

Syslog is the standard protocol for operational telemetry across servers, network infrastructure, security appliances, and applications. Originally developed in the 1980s by Eric Allman as part of the Sendmail project, syslog enables centralized logging for faster incident triage, auditability, and long-term analysis.

Modern syslog implementations support multiple transport protocols and security options to meet current operational and compliance requirements.

Syslog Standards Overview

Syslog has evolved through multiple RFC specifications:

- **RFC 3164 (2001)**: The traditional BSD syslog format, widely supported but with limitations in timestamp precision and message structure
- **RFC 5424 (2009)**: The modern syslog standard with enhanced structure, precise timestamps, and structured data support
- **RFC 6587**: Defines transmission protocols (TCP/TLS) for reliable delivery

RFC 3164 Message Structure (Legacy Format)

Traditional syslog messages consist of three main components:

PRI (Priority) Encodes both facility and severity in a single numeric value. The facility identifies the originating subsystem (e.g., auth, daemon, local0-local7), while severity indicates the importance level.

Header Contains the timestamp and hostname of the sending device. Proper time synchronization using NTP is essential for accurate log correlation.

Message The payload containing the program name and event details.

RFC 3164 Format Example

```
<34>Oct 11 22:14:15 myhost su[1234]: 'su root' failed for jdoe on /dev/pts/2
```

RFC 5424 Message Structure (Modern Format)

RFC 5424 provides enhanced structure and capabilities:

PRI (Priority) Same as RFC 3164 - encodes facility and severity.

VERSION Identifies the syslog protocol version (always "1" for RFC 5424).

TIMESTAMP High-precision timestamp in ISO 8601 format with timezone information and optional fractional seconds.

HOSTNAME Fully qualified domain name or IP address of the originating device.

APP-NAME Name of the application or process generating the message.

PROCID Process ID or thread ID of the generating process.

MSGID Message type identifier for categorizing similar events.

STRUCTURED-DATA Optional key-value pairs in standardized format for machine parsing and analysis.

MSG Free-form message text, typically human-readable.

Facilities and Severities

Severity Levels (0-7)

Code	Level	Description
0	Emergency	System unusable
1	Alert	Immediate action required
2	Critical	Critical conditions
3	Error	Error conditions
4	Warning	Warning conditions
5	Notice	Normal but significant condition
6	Informational	Informational messages
7	Debug	Debug-level messages

Common Facilities

Facilities identify the source system or application type:

- **kern**: Kernel messages
- **user**: User-level messages
- **mail**: Mail system messages
- **daemon**: System daemon messages
- **auth**: Security/authorization messages

- **local0-local7**: Custom facility codes for specific device types

Many organizations map device types (firewalls, routers, servers) to specific local facilities for easier routing and filtering.

Transport Protocols

UDP (Port 514)

- **Advantages**: Lightweight, ubiquitous support, minimal overhead
- **Disadvantages**: No delivery guarantees, potential message loss
- **Use Cases**: Legacy devices, high-volume environments where occasional loss is acceptable

TCP (Port 514)

- **Advantages**: Reliable delivery, ordered messages, flow control
- **Disadvantages**: Higher overhead, connection management complexity
- **Use Cases**: Critical systems requiring guaranteed delivery

TLS (Port 6514)

- **Advantages**: Encrypted transport, peer authentication, data integrity
- **Disadvantages**: Certificate management overhead, higher resource usage
- **Use Cases**: Sensitive data, cross-network boundaries, compliance requirements

Transport Selection Guidelines

Use TLS when:

- Transmitting sensitive or regulated data
- Crossing network boundaries or untrusted networks
- Compliance requirements mandate encryption
- Peer authentication is required

Use TCP when:

- Reliability and ordered delivery are critical
- Message loss is unacceptable
- TLS overhead is not justified

Use UDP when:

- Working with legacy or constrained devices
- High-volume environments where occasional loss is tolerable
- Minimal overhead is required

LogZilla Syslog Implementation

LogZilla utilizes the industry-standard syslog-ng daemon to receive syslog messages and forward them to LogZilla's processing architecture. This implementation supports all standard syslog transports and provides advanced parsing and filtering capabilities.

Supported Protocols

- **UDP 514**: Standard syslog reception
- **TCP 514**: Reliable syslog reception
- **TLS 6514**: Encrypted syslog reception with certificate validation
- **Custom ports**: Configurable for specific network requirements

Implementation Best Practices

Time Synchronization

Ensure all syslog sources maintain accurate time synchronization using NTP. Inconsistent timestamps complicate log correlation and incident analysis. Standardize on UTC across all systems where possible.

Facility Mapping

Establish consistent facility usage across the organization:

- Map device types to specific local facilities (local0-local7)
- Document facility assignments for operational teams
- Use facilities for automated routing and filtering rules

Message Formatting

Modern syslog implementations benefit from structured data:

- **Prefer RFC 5424**: Use RFC 5424 format when supported by devices for enhanced parsing and analysis capabilities
- **Structured Data**: Leverage RFC 5424 structured data elements for machine-readable context
- **Consistent Timestamps**: Use high-precision ISO 8601 timestamps with timezone information
- **Meaningful MSG-IDs**: Implement consistent message type identifiers for automated processing
- **Security**: Avoid embedding sensitive data in log messages

Security Considerations

- Use TLS transport for sensitive or regulated data
- Implement certificate-based authentication where required
- Configure appropriate firewall rules for syslog traffic
- Monitor for unusual syslog traffic patterns

Common Implementation Patterns

Enterprise Network Infrastructure

Large networks typically implement hierarchical syslog collection:

- Edge devices forward to regional collectors via UDP for simplicity
- Regional collectors aggregate and forward to central systems via TLS
- Central systems apply parsing, enrichment, and routing rules
- Archive systems maintain long-term retention for compliance

Security-Focused Deployments

Security-sensitive environments prioritize integrity and confidentiality:

- All syslog transport uses TLS with mutual authentication
- Certificate-based device identity validation
- Structured data formats for consistent parsing
- Real-time forwarding to SIEM with preprocessing to reduce noise

Troubleshooting Common Issues

Message Truncation

Syslog messages have practical size limits:

- UDP: typically 1024 bytes to avoid fragmentation
- TCP/TLS: larger messages supported but check receiver limits
- Use structured data to organize information within constraints

Timestamp Problems

Inconsistent timestamps complicate log correlation:

- Synchronize device clocks using NTP
- Configure consistent timestamp formats (prefer RFC 3339)
- Document systems that cannot be standardized

Certificate Management

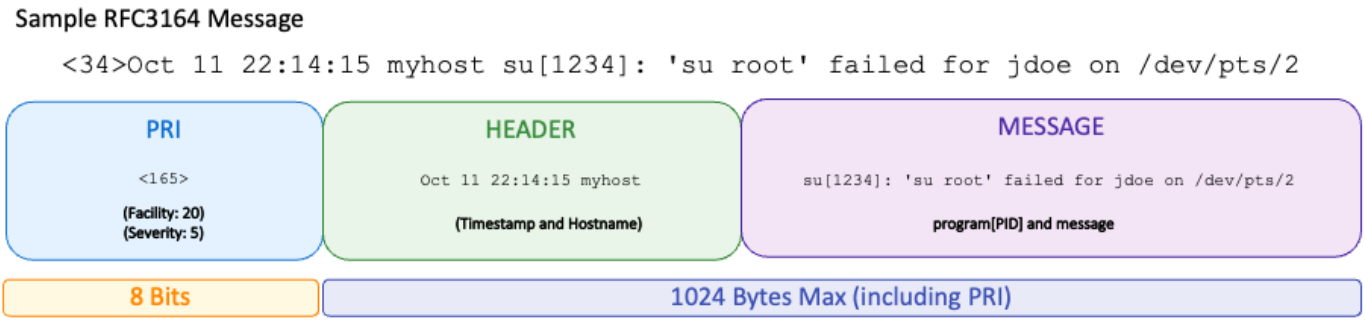
TLS deployments require ongoing certificate lifecycle management:

- Automate certificate renewal before expiration
- Monitor certificate validity across all endpoints
- Test certificate validation in staging environments.

Relays are used to forward logs from local networks to remote networks. This is the most reliable and common way to ensure message reception on the primary server when utilizing a wide-area network. For help configuring a relay, refer to the Syslog Relays section.

Syslog Message Format and Contents

RFC 3164 Format (Traditional)

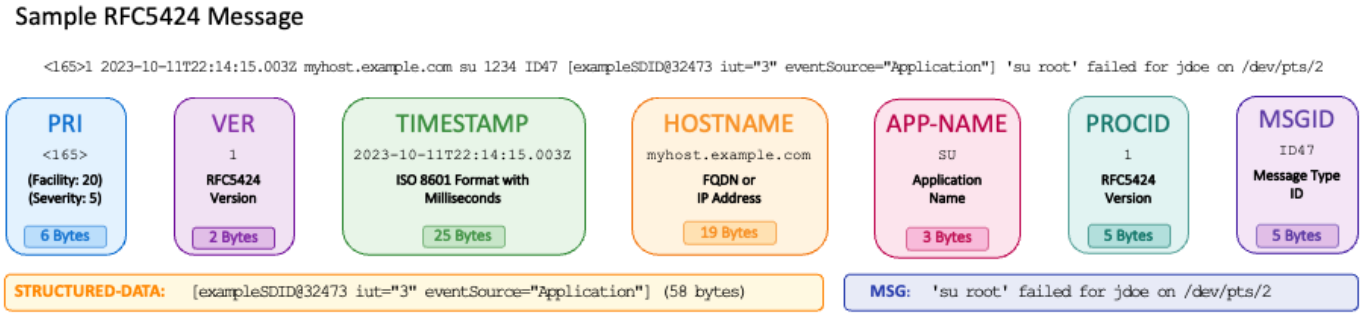


The traditional RFC 3164 syslog message has three distinct parts:

- **PRI** (priority)
- **HEADER** (timestamp and hostname)
- **MSG** (message text)

For RFC 3164 compliant events, the total length cannot exceed 1024 bytes.

RFC 5424 Format (Modern)



RFC 5424 messages provide enhanced structure with these components:

- **PRI** (priority) - Same as RFC 3164
- **VERSION** - Protocol version ("1")
- **TIMESTAMP** - ISO 8601 format with high precision
- **HOSTNAME** - FQDN or IP address

- **APP-NAME** - Application identifier
- **PROCID** - Process/thread identifier
- **MSGID** - Message type identifier
- **STRUCTURED-DATA** - Machine-readable key-value pairs
- **MSG** - Human-readable message text

RFC 5424 messages can be significantly larger than the 1024-byte RFC 3164 limit, with practical limits determined by transport protocol and implementation.

Structured Data in RFC 5424

Structured data provides machine-readable context using standardized syntax:

Structured Data Element Format

```
[SD-ID param1="value1" param2="value2"]
```

Multiple Elements Example

```
[timeQuality tzKnown="1" isSynced="1"][origin software="rsyslogd" swVersion="8.24.0"]
```

Common Structured Data Elements

timeQuality: Timestamp reliability information

- `tzKnown`: "1" if timezone is known, "0" if unknown
- `isSynced`: "1" if clock is synchronized, "0" if not

origin: Message origin information

- `software`: Name of originating software
- `swVersion`: Software version
- `ip`: IP address of originating device

meta: Additional metadata

- `sequenceId`: Message sequence number
- `sysUpTime`: System uptime when message was generated

Enterprise-Specific Elements

Organizations can define custom structured data elements using Private Enterprise Numbers (PEN):

```
[myCompany@32473 eventType="login" userId="jdoe" result="failure"]
```

Message Size Considerations

RFC 3164 Limitations

- Maximum message size: 1024 bytes (including PRI)
- No standardized timestamp format
- Limited structure for automated parsing
- Hostname length restrictions

RFC 5424 Advantages

- No fixed message size limit (transport-dependent)
- Standardized ISO 8601 timestamps with microsecond precision
- Structured data for machine parsing
- Clear field separation and encoding rules
- Support for UTF-8 encoding

Transport-Specific Limits

UDP Transport:

- Practical limit: ~1400 bytes to avoid IP fragmentation
- Consider network MTU when sizing messages
- Fragmented packets may be dropped by firewalls

TCP/TLS Transport:

- Much larger messages supported (typically 64KB+)
- Reliable delivery ensures message integrity
- Flow control prevents receiver overload

Syslog PRI Code

The Priority field is an 8-bit number that represents both the `Facility` and `Severity` of the message. The three least significant bits represent the Severity of the message (with three bits you can represent eight different Severities), and the other five bits represent the Facility of the message.

Note: Syslog Daemons (running on the syslog receiver) do not generate these Priority and Facility values. The values are created by the syslog sender (applications or hardware) from which the event is generated.

Syslog Facilities

Syslog messages are broadly categorized on the basis of the sources that generate them. These categories, referred to as `Facilities`, are represented by integers in the syslog packet. The `local` facilities are not reserved; the processes and applications that do not have pre-assigned Facility values may choose any of the eight local use facilities.

Integer	Facility
0	Kernel messages
1	User-level messages
2	Mail system
3	System daemons
4	Security/authorization messages
5	Messages generated internally by Syslogd
6	Line printer subsystem
7	Network news subsystem
8	UUCP subsystem
9	Clock daemon
10	Security/authorization messages
11	FTP daemon
12	NTP subsystem
13	Log audit
14	Log alert
15	Clock daemon
16	Local use 0 (local0)
17	Local use 1 (local1)
18	Local use 2 (local2)
19	Local use 3 (local3)
20	Local use 4 (local4)
21	Local use 5 (local5)
22	Local use 6 (local6)

Integer	Facility
23	Local use 7 (local7)

Syslog Severities

The log sender (device or software generating the message) specifies the severity of that message using single-digit integers 0-7

Note: When configuring the sending device or application, the recommended logging levels are 0-6 under normal operation, level 7 (debug) should only be used for local troubleshooting on that system.

Integer	Facility
0	Emergency: System is unusable
1	Alert: Action must be taken immediately
2	Critical: Critical conditions
3	Error: Error conditions
4	Warning: Warning conditions
5	Notice: Normal but significant condition
6	Informational: Informational messages
7	Debug: Debug-level messages

RFC 5424 Implementation Guidelines

Timestamp Best Practices

Use High-Precision Timestamps:

```
2023-10-11T22:14:15.003Z        # Millisecond precision
2023-10-11T22:14:15.003456Z     # Microsecond precision
```

Include Timezone Information:

```
2023-10-11T22:14:15.003+00:00   # UTC with offset
2023-10-11T22:14:15.003-05:00   # EST with offset
```

Application Name Guidelines

- Use consistent, descriptive application names

- Avoid spaces and special characters
- Consider using service names for system services
- Examples: `sshd`, `httpd`, `myapp`, `backup-service`

Process ID Usage

- Include actual process ID when available
- Use thread ID for multi-threaded applications
- Use "-" (NILVALUE) if not applicable
- Helps with process-specific log correlation

Message ID Strategies

Consistent Categorization:

```
ID01    # Authentication events
ID02    # Authorization events
ID03    # System errors
ID04    # Performance alerts
```

Event-Specific IDs:

```
LOGIN_SUCCESS    # Successful login
LOGIN_FAILURE    # Failed login attempt
CONFIG_CHANGE    # Configuration modification
```

Structured Data Design

Security Events:

```
[auth@32473 user="jdoe" src_ip="192.168.1.100" method="password"]
```

Performance Metrics:

```
[perf@32473 cpu_usage="85.2" memory_mb="2048" response_time_ms="150"]
```

Application Context:

```
[app@32473 module="payment" transaction_id="tx_12345" customer_id="cust_67890"]
```

Migration from RFC 3164 to RFC 5424

Assessment Phase

Device Inventory: Catalog all syslog sources and their RFC 5424 support

Parser Compatibility: Verify log management systems support RFC 5424

Network Capacity: Assess bandwidth impact of larger messages

Integration Testing: Test RFC 5424 with existing SIEM/analysis tools

Gradual Migration Strategy

Pilot Group: Start with modern devices supporting RFC 5424

Dual Format: Run both formats during transition period

Parser Updates: Update log processing rules for new format

Legacy Accommodation: Maintain RFC 3164 support for older devices

Monitoring: Track message delivery and parsing success rates

Common Migration Challenges

Increased Message Size:

- Monitor network utilization
- Adjust UDP buffer sizes if needed
- Consider TCP transport for larger messages

Parser Compatibility:

- Update regular expressions for new format
- Handle NILVALUE ("-") in empty fields
- Parse structured data elements correctly

Timestamp Parsing:

- Update timestamp parsing for ISO 8601 format
- Handle timezone information correctly
- Preserve microsecond precision where supported

Custom Syslog Configurations

For LogZilla-specific syslog configurations and advanced receiving options, refer to the Receiving Syslog Events section.

LogZilla RFC 5424 Support

LogZilla fully supports both RFC 3164 and RFC 5424 formats:

- **Automatic Detection**: Identifies message format automatically

- **Structured Data Parsing**: Extracts structured data elements into searchable fields
- **High-Precision Timestamps**: Preserves microsecond timestamp precision
- **Unicode Support**: Handles UTF-8 encoded messages correctly
- **Large Message Support**: Processes messages beyond 1024-byte limit via TCP/TLS transport

Using Tls Tunnels

Secure syslog transport over TLS on LogZilla, covering server certificate generation, client setup, and encrypted delivery across untrusted networks

TLS (Transport Layer Security) provides encrypted communication channels for syslog data transmission. LogZilla supports TLS encryption for secure log transport across untrusted networks, ensuring data confidentiality and integrity during transmission.

TLS Overview

TLS encryption protects syslog data by:

- **Encrypting data in transit** between clients and LogZilla server
- **Authenticating endpoints** using digital certificates
- **Ensuring data integrity** through cryptographic verification
- **Preventing eavesdropping** on sensitive log information

Server Configuration

Certificate Generation

Generate TLS certificates for the LogZilla server. The Common Name must match the server's hostname or IP address that clients will use to connect.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out tls.crt
```

Provide accurate certificate information when prompted:

```
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]: California
Locality Name (eg, city) []: San Francisco
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Company Name
Organizational Unit Name (eg, section) []: IT Department
Common Name (e.g. server FQDN or name) []: logzilla.company.com
Email Address []: admin@company.com
```

Important: The Common Name field must exactly match the hostname or IP address that clients use to connect to the LogZilla server.

Certificate Installation

Copy the generated certificates to the LogZilla syslog-ng directory:

```
cp tls.key tls.crt /etc/logzilla/syslog-ng
```

Set appropriate file permissions for security:

```
chmod 600 /etc/logzilla/syslog-ng/tls.key
chmod 644 /etc/logzilla/syslog-ng/tls.crt
```

Certificate File Locations

Purpose	Path	Permissions
Private Key	`/etc/logzilla/syslog-ng/tls.key`	600
Certificate	`/etc/logzilla/syslog-ng/tls.crt`	644

Configuring *syslog-ng*

By default, LogZilla syslog TLS is disabled (which is when `SYSLOG_TLS_PORT` is set to 0). The port can be changed (for example, the customary port is `6514`) with the following command:

```
logzilla config SYSLOG_TLS_PORT 6514
```

Then the LogZilla syslog module must be restarted:

```
logzilla restart -c syslog
```

To check if TLS support is working, use the `openssl` command as shown below. Replace `11.22.33.44:6514` with the LogZilla server address and TLS port.

```
openssl s_client -connect 11.22.33.44:6514 < /dev/null
```

If the output shows identification information (`C`, `ST`, `L`, `O`, etc.), certificate details from the `tls.crt` file, and TLS cipher and key specifications in use, then TLS support is operational.

If an error occurs, verify the steps from the start of this document and restart if necessary:

Adding Key Files to Client Systems

On the syslog-sending system, create a new directory:

```
mkdir -p /etc/syslog-ng/ssl
```

Transfer the key and certificate files created earlier on the **LogZilla Server** to the **Client** system, placing them in the `/etc/syslog-ng/ssl` directory. You can use `scp` or a similar method.

Configuring *syslog-ng* on the Client

There are two scenarios:

A local LogZilla instance forwards events to another LogZilla instance.

A standalone syslog-ng on the client server forwards events to a LogZilla instance.

Forwarding Events from One LogZilla Instance to Another

Replace `LZ_SERVER` below with the DNS Name or IP Address of the LogZilla Server. Change the port number accordingly if a different port number was configured at the receiving site. In the `log{}` section, update the `source` according to the sources configured in the `/etc/syslog-ng/syslog-ng.conf` file.

Create a new file named `/etc/syslog-ng/conf.d/tls_to_LogZilla.conf` and put the following content into it:

```
destination d_tls {
    syslog-ng(
        server("LZ_SERVER")
        port(6514)
        transport(tls)
        tls(ca-file("/etc/syslog-ng/ssl/tls.crt"))
    );
};

log {
  source(s_src);
  destination(d_tls);
};
```

Restart syslog-ng on the Client system:

```
service syslog-ng restart
```

Checking configuration

Check the LogZilla server to verify that events are now being received from this Client.

If issues occur, refer to the Syslog Troubleshooting section.

Advanced server configuration

For more than a single source port with TLS transport, TLS can be added to any syslog source by directly editing the `/etc/logzilla/syslog-ng/config.yaml` file. Find the `sources` array element and for any source, add `transport: tls` and then `tls_key_file` and `tls_cert_file` options. For example, to enable TLS transport for JSON input, add this:

```
- name: json-tls
  enabled: True
  type: network
  transport: tls
  port: 6515
  tls_cert_file: "/etc/logzilla/syslog-ng/tls.crt"
  tls_key_file: "/etc/logzilla/syslog-ng/key.crt"
  flags:
    - no-parse
  program_override: _JSON
```

After any change to this configuration file, the LogZilla syslog module must be restarted:

```
logzilla restart -c syslog
```

Using Https

Enable HTTPS on the LogZilla web interface with self-signed or CA-signed certificates, including key generation, installation, and UI configuration

HTTPS provides encrypted communication for the LogZilla web interface, ensuring secure access to dashboards, configuration, and administrative functions. This guide covers certificate generation, installation, configuration, and the related web-UI settings.

Certificate Requirements

LogZilla supports both self-signed certificates and certificates from trusted Certificate Authorities (CAs). For production environments, CA-signed certificates are recommended for better security and user trust.

All examples in this guide use the following directory on the LogZilla host to store TLS key and certificate files:

```
/tmp/logzilla-certs
```

You can use a different directory if preferred; just adjust the paths in the commands accordingly.

Self-Signed Certificate Generation

Generate a self-signed certificate in a temporary directory:

```
mkdir -p /tmp/logzilla-certs
cd /tmp/logzilla-certs

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout logzilla.key \
  -out logzilla.crt
```

A prompt for certificate details will appear:

```
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]: California
Locality Name (eg, city) []: San Francisco
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Company Name
Organizational Unit Name (eg, section) []: IT Department
Common Name (e.g. server FQDN or name) []: logzilla.company.com
Email Address []: admin@company.com
```

`.` and Enter can be pressed to leave any field blank if not used or not needed.

Important: The Common Name must match the hostname or IP address used to access the LogZilla web interface (for example, `logzilla.company.com` or the server's IP).

Modern browsers also validate the Subject Alternative Name (SAN) extension; ensure the hostname you use to access LogZilla is present there.

Certificate Installation

Using Self-Signed Certificates

After generating the certificate in `/tmp/logzilla-certs`, enable HTTPS:

```
logzilla https --set /tmp/logzilla-certs/logzilla.key /tmp/logzilla-certs/logzilla.crt
```

Using CA-Signed Certificates

To use a certificate obtained from a trusted Certificate Authority:

Copy the key and certificate to a directory (for example, `/tmp/logzilla-certs`):

```
mkdir -p /tmp/logzilla-certs
cp /path/to/logzilla-ca.key /tmp/logzilla-certs/
cp /path/to/logzilla-ca.crt /tmp/logzilla-certs/
```

Enable HTTPS using those files:

```
logzilla https --set /tmp/logzilla-certs/logzilla-ca.key /tmp/logzilla-certs/logzilla-
ca.crt
```

The filenames are not important as long as the correct paths are used in the `logzilla https --set` command.

HTTPS Configuration Options

Enable HTTPS

Activate HTTPS for the primary web interface:

```
logzilla https --set /tmp/logzilla-certs/<keyfile>.key /tmp/logzilla-
certs/<certificate>.crt
logzilla config HTTPS_PORT 443
logzilla restart -c front
```

Disable HTTPS

Return to HTTP-only access:

```
logzilla config HTTPS_PORT 0
logzilla restart -c front
```

Force HTTPS Redirects

Redirect all HTTP requests to HTTPS automatically:

```
logzilla config FORCE_HTTPS true
```

Disable forced HTTPS redirects:

```
logzilla config FORCE_HTTPS false
```

Verification

After enabling HTTPS, verify the configuration by accessing the LogZilla web interface using `https://` instead of `http://`. The browser should show a secure connection indicator.

For self-signed certificates, browsers will display a security warning that can be safely bypassed for internal use.

Certificate Renewal

Self-signed certificates expire after the specified validity period (365 days in the example above). For production environments, implement a certificate renewal process or use automated certificate management tools like Let's Encrypt.

Troubleshooting

Common Issues

- **Certificate path errors**: Ensure certificate files exist at the paths configured for `logzilla https --set` (for example, under `/tmp/logzilla-certs`) and are readable by LogZilla.
- **Common Name mismatch**: Verify the certificate Common Name matches the hostname or IP address used to access LogZilla.
- **Port conflicts**: Ensure port 443 (or the chosen HTTPS port) is available for HTTPS traffic.

Checking Certificate Details

View certificate information:

```
openssl x509 -in /tmp/logzilla-certs/<certificate>.crt -text -noout
```

Testing HTTPS Connection

Test the HTTPS configuration:

```
openssl s_client -connect hostname:443 -servername hostname
```

Replace `hostname` with the actual host or IP used to reach the LogZilla web interface.

Web Interface Port Settings

(Settings → System Settings → Front)

The **Front** section of System Settings controls which ports the LogZilla web UI and API listen on, and how HTTP/HTTPS are used.

Http Port

UI and API HTTP port

The TCP port used for **unencrypted** HTTP access to the primary LogZilla UI and API (for example, `80`).

- Set to `0` to disable HTTP on the primary UI.
- Common values: `80` (default) or another port (for example, to use LogZilla with an external reverse proxy).

Https Port

UI and API HTTPS port

The TCP port used for **encrypted** HTTPS access to the primary LogZilla UI and API.

- Set to `0` to disable HTTPS on the primary UI.
- Typical production value: `443`.
- Must correspond to the port set with `logzilla config HTTPS_PORT`.

Http Port Ui2

UI and API HTTP port for UI2 (experimental)

HTTP port for the **UI2** interface (new UI, version 2) and its API.

- Leave at the default (for example, `8080`) unless explicitly using the UI2 frontend.
- Set to `0` to disable HTTP access for UI2.

Https Port Ui2

UI and API HTTPS port for UI2 (experimental)

HTTPS port for the **UI2** interface and its API.

- Default example: `8889`.
- Set to `0` to disable HTTPS for UI2.
- If UI2 is exposed over HTTPS, ensure the certificate configured via `logzilla https --set` is valid for the host/port combination you will use.

Force Https

Use HTTPS instead of HTTP

Controls whether HTTP requests are automatically redirected to HTTPS.

- **On** —- All HTTP requests to the UI/API are redirected to the configured HTTPS port. This corresponds to `logzilla config FORCE_HTTPS true` and is recommended for production deployments.
- **Off** —- Both HTTP and HTTPS are allowed. Users can access the UI via either scheme.

Note that forcing HTTPS only has an effect if an HTTPS port is configured and reachable.

Http Client Body Buffer Size Mb

If client body request is above this size, it will be buffered on the disk, which can make processing slower. This affects mostly Kinesis receiver requests.

Maximum size (in megabytes) that LogZilla will buffer **in memory** for the HTTP request body. When a request body exceeds this size:

- The body is temporarily buffered to disk instead of memory.
- This reduces memory usage but can slow down very large requests.

Guidelines:

- Default: `50` MB (suitable for most installations).
- Increase this value if large payloads are frequently sent and there is sufficient memory.
- Decrease this value if memory pressure is a concern and slower large-request handling is acceptable (for example, some Kinesis or bulk ingestion endpoints).

Changes to these settings usually require a front-end restart:

```
logzilla restart -c front
```

to fully apply.

Server Settings

LogZilla YAML-based server configuration in /etc/logzilla/settings/ managed through the logzilla settings CLI for advanced backend parameters

LogZilla uses a modern YAML-based configuration system that provides advanced settings not available through the web interface. These backend settings control core system behavior, performance parameters, and security features.

Warning: Changing these settings may cause system instability or data loss. Exercise extreme caution and test changes in a non-production environment first.

Modern Configuration System

LogZilla stores configuration in YAML files located in `/etc/logzilla/settings/`. Each file corresponds to a specific configuration group (e.g., `smtp.yaml`, `ldap.yaml`, `generic.yaml`).

Configure with the command line

Administrators use the `logzilla settings` command to manage configuration:

```
# List all available settings
logzilla settings list

# List settings for a specific group
logzilla settings list ldap

# Update a setting value
logzilla settings update SETTING_NAME=new_value

# Update multiple settings at once
logzilla settings update SMTP_SERVER=mail.company.com SMTP_PORT=587

# Manage multiple instances (e.g., secondary LDAP server)
logzilla settings update --id 1 LDAP_ENABLED=true
```

Edit the configuration file directly

Advanced users with root access can edit YAML files directly:

```
# Navigate to settings directory
cd /etc/logzilla/settings/

# Edit a configuration file
vim smtp.yaml
```

After direct edits, see Applying Changes to reload the affected configuration group.

Applying Changes

After making configuration changes, apply them using one of these methods:

Method 1: Automatic Application (Recommended)

The `logzilla settings update` command automatically applies changes to compatible modules.

Method 2: Manual Reload (After Direct Edits)

If editing YAML files directly, notify the system of changes:

```
# Reload a specific configuration group
logzilla settings reload smtp
logzilla settings reload ldap
```

Method 3: Container Restart (When Required)

Some settings require container restarts to take effect:

```
# Restart specific containers
logzilla restart -c gunicorn celeryworker celerybeat

# Or restart all LogZilla services
logzilla restart
```

Note: API and Celery containers typically require restart for changes to take effect, while core processing modules support live reloading for most settings.

These are the modules used by LogZilla, which can be restarted with `logzilla restart -c <module>`:

Module	Description
celerybeat	Schedules periodic background tasks for the platform.
gunicorn	API server for the LogZilla backend.
celeryworker	Executes background jobs and module orchestration tasks.

Module	Description
parsermodule	Parses, normalizes, and enriches incoming events using rules/apps.
querymodule	Runs searches/analytics, evaluates triggers, and publishes live results.
storagemodule-1	Event storage and indexing service (dedup, archiving, aggregates).
ai	AI Copilot backend; powers AI features when enabled.
watcher	Orchestrates, monitors, and restarts LogZilla containers.
snmptrapd	Receives SNMP traps (UDP 162) and injects them as events.
httpreceiver	HTTP/HTTPS event ingest API (token-protected) into LogZilla.
sec	Simple Event Correlator for stateful, multi-event correlation.
scriptserver	Runs user-defined scripts for trigger actions/correlation.
front	Web frontend serving UI and proxying API/ingest endpoints.
syslog	Syslog-ng receiver for BSD/RFC5424, JSON, RAW, and TLS inputs.
logcollector	Collects and consolidates internal container logs for diagnostics.
mailer	Outbound SMTP service for alerts and test emails.
telegraf	Collects internal metrics and ships them to time-series storage.
qdrant	Vector database backing AI features (embeddings/search).
influxdb	Time-series database for metrics and aggregates.
postgres	Primary relational store for users, dashboards, triggers, settings.
redis	In-memory caching and queues for ephemeral data.

Configuration Files and Settings

The following sections organize settings by their configuration files in `/etc/logzilla/settings/`:

Generic Settings (`generic.yaml`)

Setting	Description	Default	Range
`SEARCH_DEFAULT_LIMIT`	Default max results limit for search queries	`1000`	Integer
`FREE_DISK_SPACE_SOFT_LIMIT_GB`	Warning threshold for free disk space	`10`	Integer (GB)
`FREE_DISK_SPACE_HARD_LIMIT_GB`	Critical threshold for free disk space	`2`	Integer (GB)

Setting	Description	Default	Range
EXTERNAL_BASE_URL	External URL of the LogZilla instance	null	Valid URL
TIME_ZONE	Server timezone	UTC	Timezone string
RBAC_ENABLED	Enable Role Based Access Control	true	true, false
TASKS_ENABLED	Enable Tasks feature	true	true, false
AIR_GAPPED	Disable external service communication	false	true, false
PRUNE_DOCKER_IMAGES	Remove old Docker images during upgrades	true	true, false
SHOW_ADVANCED_SETTINGS	Show advanced settings in UI	false	true, false

Search Settings (`sphinx.yaml`)

Setting	Description	Default	Range
SPHINX_MIN_WORD_LENGTH	Minimum word length to index	-	Integer
SPHINX_MIN_PREFIX_LENGTH	Minimum prefix length to index	-	Integer
SPHINX_MIN_INFIX_LENGTH	Minimum infix length to index	-	Integer
SPHINX_MAX_DOCUMENTS_PER_INDEX	Max documents per index	-	Integer
SPHINX_MAX_INDEXING_TIME	Max indexing time per chunk (seconds)	-	Integer
SPHINX_MIN_INDEX_LEN	Min events per indexing batch	-	Integer
SPHINX_REINDEX_PROC_MAX	Max concurrent indexing processes	-	Integer
SPHINX_MERGING_PROC_MAX	Max concurrent merging processes	-	Integer
SPHINX_REINDEX_DELAY	Delay between reindexing (seconds)	-	Integer
SPHINX_MAX_MATCHES	Maximum matches per query	-	Integer
SPHINX_MYSQL_PORT	Sphinx MySQL port (0 to disable)	-	Integer
SPHINX_HTTP_PORT	Sphinx HTTP port (0 to disable)	-	Integer
SPHINX_REALTIME_MODE	Enable realtime mode (experimental)	-	true, false

Trigger Settings (`triggers.yaml`)

Setting	Description	Default	Range
`TRIGGERS_ENABLED`	Enable or disable triggers	`true`	`true`, `false`
`SEND_MAIL_PERIOD`	Email resend interval (seconds)	`60`	Integer
`SEND_WEBHOOK_PERIOD`	Webhook resend interval (seconds, 0=always)	`0`	Integer
`EXEC_SCRIPT_PERIOD`	Script rerun interval (seconds)	`60`	Integer

SMTP Settings (`smtp.yaml`)

Setting	Description	Default	Range
`MAIL_SENDER`	Email address used as sender	`logzilla@localhost`	Email address
`SMTP_SERVER`	SMTP server address	`localhost`	Hostname/IP
`SMTP_PORT`	SMTP server port	`25`	Integer
`SMTP_AUTH_REQUIRED`	Enable SMTP authentication	`false`	`true`, `false`
`SMTP_USER`	SMTP username for authentication	(empty)	String
`SMTP_PASS`	SMTP password for authentication	(empty)	String
`SMTP_CRYPT`	Encryption method	`NONE`	`TLS`, `SSL`, `NONE`
`SMTP_TIMEOUT`	Connection timeout (seconds)	`30`	Integer

Syslog Daemon Settings (`syslogng.yaml`)

Setting	Description	Default	Range
`SYSLOG_BSD_TCP_PORT`	TCP port for BSD syslog (0 to disable)	`514`	Integer
`SYSLOG_BSD_UDP_PORT`	UDP port for BSD syslog (0 to disable)	`514`	Integer
`SYSLOG_RFC5424_PORT`	Port for RFC5424 syslog (0 to disable)	`601`	Integer

Setting	Description	Default	Range
SYSLOG_JSON_PORT	Port for JSON syslog (0 to disable)	515	Integer
SYSLOG_RAW_PORT	Port for RAW TCP events (0 to disable)	516	Integer
SYSLOG_RAW_UDP_PORT	Port for RAW UDP events (0 to disable)	516	Integer
SYSLOG_TLS_PORT	Port for TLS syslog (0 to disable)	6514	Integer
SYSLOG_TLS_CERT_FILE	Path to TLS certificate file	/etc/logzilla/nginx/server.crt	File path
SYSLOG_TLS_KEY_FILE	Path to TLS key file	/etc/logzilla/nginx/server.key	File path
SYSLOG_MAX_CONNECTIONS	Maximum concurrent TCP connections	500	Integer
SYSLOG_FLOW_CONTROL	Enable flow control	true	true, false
SYSLOG_DISK_BUFFER	Enable disk buffer	false	true, false
SYSLOG_BUFFER_RELIABLE	Enable reliable disk buffer	false	true, false
SYSLOG_MEMORY_BUFFER_SIZE_MB	Memory buffer size (MB)	4	Integer
SYSLOG_MEMORY_BUFFER_LENGTH	Memory buffer length (messages)	50000	Integer
SYSLOG_DISK_BUFFER_SIZE_MB	Disk buffer size (MB)	1	Integer

Setting	Description	Default	Range
SYSLOG_DESTINATION_WORKERS	Number of destination workers	2	Integer
SYSLOG_DESTINATION_BATCH_LINES	Lines per batch	10000	Integer
SYSLOG_DESTINATION_BATCH_TIMEOUT	Batch timeout (milliseconds)	500	Integer
SYSLOG_DEBUG	Enable debug logging (TSV format)	false	true, false
SYSLOG_DEBUG_JSON	Enable debug logging (JSON format)	false	true, false
PCI_COMPLIANT_LOGS	Enable PCI compliant logging	false	true, false

Application Ports (`application_ports.yaml`)

Dedicated syslog ports for vendors whose log format requires separate handling. Events received on these ports are automatically tagged for processing by the corresponding application rules. Each port listens on both TCP and UDP.

Setting	Description	Default	Range
SYSLOG_ARISTA_EOS_PORT	Arista EOS syslog port (0 to disable)	0	Integer
SYSLOG_CHECKPOINT_PORT	Check Point syslog port (0 to disable)	0	Integer
SYSLOG_DATAPOWER_PORT	IBM DataPower syslog port (0 to disable)	0	Integer
SYSLOG_DELL_N_SERIES_PORT	Dell N-Series switch syslog port (0 to disable)	0	Integer
SYSLOG_FIREEYE_PORT	FireEye syslog port (0 to disable)	0	Integer
SYSLOG_INFOBLOX_PORT	Infoblox NIOS syslog port (0 to disable)	0	Integer
SYSLOG_MERAKI_PORT	Cisco Meraki syslog port (0 to disable)	0	Integer
SYSLOG_PALOALTO_PORT	Palo Alto syslog port (0 to disable)	0	Integer
SYSLOG_PALOALTO_SDWAN_ION_PORT	Palo Alto Prisma SD-WAN ION syslog port (0 to disable)	0	Integer
SYSLOG_SYMANTEC_PORT	Symantec Endpoint Protection syslog port (0 to disable)	0	Integer

Setting	Description	Default	Range
SYSLOG_UNIFI_PORT	Ubiquiti UniFi syslog port (0 to disable)	0	Integer
SYSLOG_VMWARE_PORT	VMware syslog port (0 to disable)	0	Integer

SNMP Trap Settings (`snmptrapd.yaml`)

Setting	Description	Default	Range
SNMPTRAPD_ENABLED	Enable SNMP trap daemon module	False	True, False
SNMPTRAPD_FORMAT	Message field format (see snmptrapd(8))	See below	-
SNMPTRAPD_PROGRAM	Program field value for SNMP trap events	SNMPTrap	-
SNMPTRAPD_FACILITY	Facility field value for SNMP trap events	LOCAL0	-
SNMPTRAPD_SEVERITY	Severity field value for SNMP trap events	INFO	-
SNMPTRAPD_PORT	SNMP trap daemon port	162	-

Additional Configuration Files

Other important configuration files include:

- **`storage.yaml`** : Event storage, deduplication, and archiving settings
- **`parser.yaml`** : Message parsing engine configuration
- **`logger.yaml`** : Logging levels for different components
- **`forwarder.yaml`** : Event forwarding to external systems
- **`sec.yaml`** : Simple Event Correlator (SEC) integration
- **`front.yaml`** : Nginx frontend proxy configuration (HTTP/HTTPS ports)
- **`ldap.yaml`** : LDAP/Active Directory integration (supports multiple instances)
- **`ai.yaml`** : AI/ML feature settings
- **`httpreceiver.yaml`** : HTTP event receiver configuration
- **`influxdb.yaml`** : InfluxDB integration settings
- **`django_login.yaml`** : Django authentication settings
- **`license.yaml`** : License configuration
- **`secrets.yaml`** : Sensitive configuration data

Configuration Management Best Practices

Multi-Instance Configuration

LogZilla supports multiple instances of certain configurations. For example, to configure a secondary LDAP server:

```
# Configure primary LDAP server
logzilla settings update LDAP_ENABLED=true LDAP_SERVER=ldap1.company.com

# Configure secondary LDAP server
logzilla settings update --id 1 LDAP_ENABLED=true LDAP_SERVER=ldap2.company.com
```

This creates `ldap.yaml` and `ldap__1.yaml` files respectively.

Configuration Validation

The `logzilla settings` command validates all changes against predefined schemas before saving, preventing invalid configurations:

```
# This will fail with validation error if port is invalid
logzilla settings update SMTP_PORT=invalid_port
```

Backup and Recovery

Before making significant configuration changes, backup the settings directory:

```
# Create backup
cp -r /etc/logzilla/settings /etc/logzilla/settings.backup.$(date +%Y%m%d)

# Restore from backup if needed
cp -r /etc/logzilla/settings.backup.20241201 /etc/logzilla/settings
logzilla settings reload generic  # Reload as needed
```

Important Notes

Live Reloading vs Container Restart

Live Reloading: Core processing modules (parser, storage, query) support live reloading for most settings through the pub/sub mechanism.

Container Restart Required: API and Celery containers require restart for changes to take effect:

- SMTP settings
- LDAP configuration

- Database connections
- Security settings

SMTP Configuration

SMTP settings control outgoing email functionality for alerts and notifications. After changing SMTP settings, restart the API containers:

```
docker restart lz_gunicorn_1 lz_celeryworker_1
```

Configuration File Access

Inspect settings with the command line

The `logzilla settings list` command prints the current values for a specific configuration group:

```
logzilla settings list generic
logzilla settings list smtp
```

Inspect the configuration file directly

Administrators with root access can read the raw YAML file on disk:

```
cat /etc/logzilla/settings/generic.yaml
```

SNMP Trap Format Default

The default `SNMPTRAPD_FORMAT` setting is:

```
Enterprise OID: %N, Trap Type: %W, Trap Sub-Type: %q,
Uptime: %T, Description: %W,
PDU Attribute/Value Pair Array: %v
```

Legacy Configuration Command

Deprecated: The `logzilla config` command is deprecated and should not be used. It lacks support for multi-instance configurations and validation. Use `logzilla settings` instead.

Search Engine Settings

Tune the LogZilla Sphinx search engine via sphinx.yaml for full-text indexing, word length, infix matching, and performance parameters

LogZilla uses an advanced search engine for full-text indexing and searching of log events. The search engine configuration is managed through the `sphinx.yaml` file and provides extensive customization options for indexing behavior, performance tuning, and search capabilities.

Critical Warning: Modifying search engine settings can cause system instability, performance degradation, or data loss. Always test changes in a non-production environment first and create backups before making modifications. Contact LogZilla support if unsure about any setting.

Search Engine Configuration

Search settings are stored in `/etc/logzilla/settings/sphinx.yaml` and can be managed using the modern `logzilla settings` command.

Basic Configuration

```
# View current search settings
logzilla settings list sphinx

# Update search settings
logzilla settings update SPHINX_MIN_WORD_LENGTH=3
logzilla settings update SPHINX_MIN_INFIX_LENGTH=3

# Apply changes (automatic with settings update)
logzilla settings reload sphinx
```

Core Indexing Settings

Word and Text Processing

Setting	Description	Impact	Recommended
`SPHINX_MIN_WORD_LENGTH`	Minimum word length to index	Shorter = more matches, larger index	`3–4`
`SPHINX_MIN_PREFIX_LENGTH`	Minimum prefix length for prefix searches	Affects `word*` searches	`3–4`

Setting	Description	Impact	Recommended
SPHINX_MIN_INFIX_LENGTH	Minimum infix length for wildcard searches	Enables *word* and *word searches	3–4 or 0

Wildcard Search Capabilities

Default Behavior: LogZilla supports suffix wildcards (`start*`) by default.

Infix Wildcards: To enable prefix (`*end`) and middle (`*middle*`) wildcard searches, configure infix indexing:

```
# Enable infix searching (matches SPHINX_MIN_WORD_LENGTH)
logzilla settings update SPHINX_MIN_INFIX_LENGTH=4
```

Example: With `SPHINX_MIN_INFIX_LENGTH=3` , the word "testing" generates indexes for: `tes` , `est` , `sti` , `tin` , `ing` , `test` , `esti` , `stin` , `ting` , `testi` , `estin` , `sting` , `testin` , `esting` , and `testing` .

Performance Warning: Infix indexing increases index size and can significantly impact performance. Use values below 4 only when necessary and with adequate system resources.

Data Safety: Changes to indexing settings only affect new data. Existing data retains its original indexing. Incorrect settings may cause search failures or system instability.

Performance and Scaling Settings

Index Management

Setting	Description	Commonly Changed
SPHINX_MIN_WORD_LENGTH	Minimum word length to index	Yes - tune for data
SPHINX_MIN_PREFIX_LENGTH	Minimum prefix length to index	Yes - tune for data
SPHINX_MIN_INFIX_LENGTH	Minimum infix length to index	Yes - tune for data
SPHINX_MAX_DOCUMENTS_PER_INDEX	Max documents per index	Rarely
SPHINX_MAX_INDEXING_TIME	Max indexing time (seconds)	Rarely
SPHINX_MIN_INDEX_LEN	Min events per indexing batch	Rarely
SPHINX_REINDEX_PROC_MAX	Max concurrent indexing processes	Yes - tune for CPU
SPHINX_MERGING_PROC_MAX	Max concurrent merging processes	Yes - tune for CPU
SPHINX_REINDEX_DELAY	Delay between reindexing (seconds)	Sometimes

Setting	Description	Commonly Changed
SPHINX_MAX_MATCHES	Maximum matches per query	Yes - tune for usage
SPHINX_DISABLE_AUTO_SPLITTING	Disable automatic index splitting	Advanced use only
SPHINX_DISABLE_MERGING	Disable index merging	Troubleshooting only
SPHINX_DISABLE_TEMPORARY_INDEXES	Disable temporary indexes	Troubleshooting only

Concurrent Processing

Setting	Description	Resource Impact
SPHINX_REINDEX_PROC_MAX	Max concurrent indexing processes	CPU and I/O intensive
SPHINX_MERGING_PROC_MAX	Max concurrent merging processes	I/O intensive
SPHINX_REINDEX_DELAY	Delay between reindexing (seconds)	Reduces system load

Query Limits

Setting	Description	Memory Impact
SPHINX_MAX_MATCHES	Maximum matches per query	Higher = more memory usage

Advanced Configuration

Debug and Troubleshooting

```
# Enable searchd debug mode
logzilla settings update SPHINX_SEARCHD_DEBUG=true

# Enable searchd logging
logzilla settings update SEARCHD_LOGS_ENABLED=true
```

Optimization Guidelines

For High-Volume Environments

```
# Increase batch processing
logzilla settings update SPHINX_MIN_INDEX_LEN=50000
logzilla settings update SPHINX_MAX_INDEXING_TIME=300

# Optimize concurrent processing
logzilla settings update SPHINX_REINDEX_PROC_MAX=4
logzilla settings update SPHINX_MERGING_PROC_MAX=2

# Increase reindex delay to reduce load
logzilla settings update SPHINX_REINDEX_DELAY=30
```

For Search-Heavy Workloads

```
# Increase query result limits
logzilla settings update SPHINX_MAX_MATCHES=2000000

# Increase concurrent processing
logzilla settings update SPHINX_REINDEX_PROC_MAX=4
logzilla settings update SPHINX_MERGING_PROC_MAX=4
```

For Storage-Constrained Systems

```
# Disable infix indexing to save space
logzilla settings update SPHINX_MIN_INFIX_LENGTH=0

# Increase minimum word length
logzilla settings update SPHINX_MIN_WORD_LENGTH=5

# Disable temporary index splitting
logzilla settings update SPHINX_DISABLE_AUTO_SPLITTING=true
```

Important Considerations

Critical Safety Information

Warning: Search engine settings directly affect system performance and stability. Incorrect configurations can cause:

- Complete search functionality failure
- Excessive memory usage leading to system crashes

- Data processing delays or failures
- Inability to access historical data

Always consult LogZilla support before modifying these settings in production.

Index Rebuilding

Important: Changes to word length and infix settings only apply to new events. Existing events retain their original indexing. Modifying these settings may require system maintenance windows and can affect search performance during the transition period.

Performance Impact

Memory Usage: Infix indexing can increase memory usage by 3-10x depending on text content and minimum lengths.

Disk Space: Index size grows significantly with shorter minimum lengths and infix indexing enabled.

Search Speed: More indexed terms can slow search performance, especially for common terms.

Recommended Settings by Use Case

Standard Deployment:

```
SPHINX_MIN_WORD_LENGTH=4
SPHINX_MIN_PREFIX_LENGTH=4
SPHINX_MIN_INFIX_LENGTH=0  # Disabled for performance
```

Advanced Search Requirements:

```
SPHINX_MIN_WORD_LENGTH=3
SPHINX_MIN_PREFIX_LENGTH=3
SPHINX_MIN_INFIX_LENGTH=3  # Enable with caution
```

High-Performance Environment:

```
SPHINX_MIN_WORD_LENGTH=5
SPHINX_MIN_PREFIX_LENGTH=5
SPHINX_MIN_INFIX_LENGTH=0  # Disabled
SPHINX_MIN_INDEX_LEN=100000
SPHINX_REINDEX_DELAY=60
```

Troubleshooting

Common Issues

Slow Searches: Reduce infix length or disable infix indexing.

High Memory Usage: Increase minimum word lengths and disable infix indexing.

Missing Search Results: Check minimum word length settings and ensure terms meet the configured thresholds.

Indexing Delays: Adjust concurrent processing limits and batch sizes.

Monitoring

```
# Check indexing status and statistics
logzilla events stats
```

Troubleshooting Search Issues

For search-related problems, check the main LogZilla log:

```
# View recent log entries
tail -f /var/log/logzilla/logzilla.log

# Search for indexing errors
grep -i "sphinx\|search\|index" /var/log/logzilla/logzilla.log
```

Note: Advanced search engine diagnostics require container access and should only be performed by experienced administrators or under LogZilla support guidance.

Data Archiving And Retention

Configure LogZilla online and archived storage retention, hourly chunk transitions, and compressed archive policies to control disk footprint

LogZilla provides automated data archiving to manage storage space while maintaining access to historical log data. The archiving system moves older data from active storage to compressed archive storage, where it remains searchable but with slower performance.

Important: Archive settings directly affect data retention and storage usage. Incorrect configuration can result in premature data loss. Always verify settings before making changes in production environments.

How Archiving Works

LogZilla manages data in two states:

- **Online Data**: Recent data stored in active indexes for fast searching
- **Archived Data**: Older data moved to compressed storage, still searchable but with reduced performance

Data is organized into hourly chunks and automatically transitioned based on configured retention policies.

Archive Configuration

Archive settings are managed through the storage configuration:

```
# View current archive settings
logzilla settings list storage

# Configure archive settings
logzilla settings update AUTO_ARCHIVE_ENABLED=true
logzilla settings update ARCHIVE_FLUSH_DAYS=365
logzilla settings update ARCHIVE_EXPIRE_DAYS=30
```

Archive Settings

Setting	Description	Default	Impact
`AUTO_ARCHIVE_ENABLED`	Enable automatic archiving	`true`	Disabling stops all automatic archiving
`ARCHIVE_FLUSH_DAYS`	Days before data is archived	`365`	Shorter = less online storage used
`ARCHIVE_EXPIRE_DAYS`	Days before archived data is deleted	`5`	**Critical**: Data permanently lost after this period

Setting	Description	Default	Impact
AUTOARCHIVE_CRON_HOUR	Daily archive job time (24-hour format)	5	Schedule during low-usage periods

Important: `ARCHIVE_EXPIRE_DAYS` determines when archived data is permanently deleted. Total data retention is `ARCHIVE_FLUSH_DAYS`

- `ARCHIVE_EXPIRE_DAYS`. Verify this meets the site's retention requirements.

Automatic Archiving

LogZilla automatically manages data lifecycle:

Daily Processing: Archive job runs at configured hour

Data Archiving: Online data older than `ARCHIVE_FLUSH_DAYS` is archived

Data Expiration: Archived data older than `ARCHIVE_EXPIRE_DAYS` is deleted

No Downtime: Process runs in background without service interruption

Manual Archive Management

Use the `logzilla archives` command for manual operations:

Archive Specific Date Ranges

```
# Archive data for specific period
logzilla archives archive --ts-from 2024-01-01 --ts-to 2024-02-01

# Archive data older than specified days
logzilla archives archive --expire-days 30

# Round timestamps to nearest hour (aligns with LogZilla's hourly chunks)
logzilla archives --round archive --ts-from 2024-01-01 --ts-to 2024-02-01
```

Remove Archived Data

```
# Permanently remove archived data for specific period
logzilla archives remove --ts-from 2023-01-01 --ts-to 2023-02-01
```

Warning: The `remove` command permanently deletes data. This action cannot be undone. Ensure a backup exists if the data might be needed later.

Automatic Archive Searching

LogZilla automatically determines whether to search archived data based on the date range specified:

Recent Data: Searches within the online data retention period use only active indexes for fast results

Historical Data: Searches with date ranges extending beyond online data automatically include archived data

Performance Impact: Archive search performance depends on the storage hardware used for archived data

Transparent Access: Users don't need to manually "restore" data - LogZilla handles archive access transparently

No Rehydration Required

Unlike traditional log management systems that require users to manually **rehydrate** archived data back to active storage, LogZilla provides direct access to archived logs without any manual intervention.

Traditional Rehydration Process (Not Required with LogZilla)

Most log management platforms require administrators to:

Identify archived data needed for analysis

Request rehydration of specific time ranges or data sets

Wait for rehydration to complete (often hours or days)

Pay additional costs for rehydration processing and temporary storage

Manage rehydrated data lifecycle and cleanup

LogZilla's Transparent Archive Access

LogZilla eliminates the rehydration bottleneck entirely:

- **No rehydration delays**: Archived data is immediately accessible
- **No rehydration costs**: No additional charges for accessing historical data
- **No rehydration management**: No need to track or cleanup rehydrated datasets
- **No rehydration planning**: No advance planning required for historical analysis
- **No rehydration limits**: Access any archived timeframe without restrictions

Benefits Over Rehydration-Based Systems

Immediate Access: Start analyzing historical data instantly without waiting for rehydration processes to complete.

Cost Efficiency: Eliminate rehydration fees and temporary storage costs associated with bringing archived data back online.

Operational Simplicity: Remove the complexity of managing rehydration workflows, scheduling, and cleanup processes.

Forensic Readiness: Respond to security incidents immediately without rehydration delays that could impact investigation timelines.

Compliance Reporting: Generate compliance reports spanning any timeframe without advance rehydration planning.

Competitive Advantage: LogZilla's architecture provides direct archive access capabilities that eliminate the operational overhead and costs associated with traditional rehydration workflows found in other log management platforms.

API Queries

Authentication Required: All LogZilla API requests require authentication. See Getting Started for API token creation and usage instructions.

LogZilla automatically determines whether to search archived data based on the date range specified in the query. When the time range extends beyond the online data retention period, archived data is automatically included.

Example: Recent Data Search (Online Only)

```
# List existing API tokens (requires admin access)
sudo logzilla authtoken list

# Use an existing USER token (replace with your actual token)
TOKEN="6160ce50a098067f39d1acc72396b31c5518a5ca7b178538"

# Or create a new USER token if needed (creates user-prefixed token)
# TOKEN=$(sudo logzilla authtoken create | tail -1)

# Search last hour (online data only - fast)
curl -X POST \
  -H "Authorization: token $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "type": "Search",
    "params": {
      "time_range": {"preset": "last_1_hours"},
      "filter": [{"field": "message", "value": "error"}],
      "page_size": 100
    }
  }' \
  "http://your-logzilla-server/api/query"
```

Example: Historical Data Search (Includes Archives)

```
# Search last 30 days (includes archived data - slower)
curl -X POST \
  -H "Authorization: token $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "type": "Search",
    "params": {
      "time_range": {"preset": "last_30_days"},
      "filter": [{"field": "message", "value": "error"}],
      "page_size": 100
    }
  }' \
  "http://your-logzilla-server/api/query"
```

Example: Specific Date Range Search

```
# Search specific date range (may include archives)
curl -X POST \
  -H "Authorization: token $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "type": "Search",
    "params": {
      "time_range": {
        "ts_from": 1609459200,
        "ts_to": 1609545600
      },
      "filter": [
        {"field": "host", "value": "web-server-01"},
        {"field": "severity", "op": "le", "value": 3}
      ],
      "sort": ["first_occurrence"],
      "page_size": 50
    }
  }' \
  "http://your-logzilla-server/api/query"
```

Performance Note: Queries spanning archived data will take longer to complete and may return status `202 ACCEPTED` for asynchronous processing. Use the returned `query_id` to check results or implement websocket subscriptions for real-time updates.

Monitoring and Logs

Archive Activity Logs

Monitor archive operations through LogZilla's system logs:

```
# View archive-related log entries
sudo grep -i archive /var/log/logzilla/logzilla.log

# Monitor real-time archive activity
sudo tail -f /var/log/logzilla/logzilla.log | grep -i archive
```

API-Based Archive Monitoring

```
# Use existing API token (replace with your actual token)
TOKEN="6160ce50a098067f39d1acc72396b31c5518a5ca7b178538"

# Or create new USER token if needed (creates user-prefixed token)
# TOKEN=$(sudo logzilla authtoken create | tail -1)

# Query archive-related system events
curl -X POST \
  -H "Authorization: token $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "type": "Search",
    "params": {
      "time_range": {"preset": "last_24_hours"},
      "filter": [
        {"field": "program", "value": "logzilla"},
        {"field": "message", "value": "archive"}
      ],
      "sort": ["-first_occurrence"],
      "page_size": 50
    }
  }' \
  "http://your-logzilla-server/api/query"
```

Archive Status

Monitor current archive status:

```
# View archived data chunks
 curl -H 'Authorization: token user-914012b8e78f8b305db63a6c04019d77655a9a3daa7badda'
http://your-logzilla-server/api/archives
```

System Logs

Archive operations are logged to the main LogZilla log:

```
# Monitor archive operations
grep -i "archive" /var/log/logzilla/logzilla.log
```

Configuration Examples

Common Retention Scenarios

90 Days Online, 1 Year Total Retention:

```
logzilla settings update ARCHIVE_FLUSH_DAYS=90
logzilla settings update ARCHIVE_EXPIRE_DAYS=275
# Total retention: 90 + 275 = 365 days
```

30 Days Online, 3 Years Total Retention:

```
logzilla settings update ARCHIVE_FLUSH_DAYS=30
logzilla settings update ARCHIVE_EXPIRE_DAYS=1065
# Total retention: 30 + 1065 = 1095 days (3 years)
```

7 Days Online, 6 Months Total Retention:

```
logzilla settings update ARCHIVE_FLUSH_DAYS=7
logzilla settings update ARCHIVE_EXPIRE_DAYS=173
# Total retention: 7 + 173 = 180 days (6 months)
```

Best Practices

Storage Planning

- **Calculate Total Retention**: `ARCHIVE_FLUSH_DAYS` + `ARCHIVE_EXPIRE_DAYS`
- **Monitor Disk Usage**: Archive storage is located at `/var/lib/logzilla-archive/`
- **Plan for Growth**: Consider data volume increases over time
- **Use `--round` Flag**: When manually archiving, use `--round` to align with hourly chunks

Performance Optimization

- **Schedule Wisely**: Run archive jobs during low-usage periods
- **Limit Archive Searches**: Use archived data searches sparingly

- **Monitor Impact**: Archive operations can affect system performance

Data Safety

- **Verify Settings**: Confirm retention periods meet compliance requirements
- **Test Procedures**: Validate archive and search functionality in test environments
- **Document Policies**: Maintain clear data retention documentation

Relocating Archive Storage

Use this procedure to move the archive volume to a new host directory (bind mount) when reallocating disk space.

Check Current Archive Size

```
# Inspect size of the archive volume (no host path assumptions)
docker run --rm -v lz_archive:/archive \
  logzilla/runtime sh -lc 'du -csh /archive'
```

Move the Archive to a New Host Directory

```
logzilla stop

# Ensure /new_archive_dir exists and has sufficient space
docker run --rm \
  -v /new_archive_dir:/new_archive_dir \
  -v lz_archive:/temp_archive \
  logzilla/runtime sh -lc 'mv /temp_archive/* /new_archive_dir/'

# Recreate the volume as a bind mount to the new directory
docker rm lz_watcher
docker volume rm lz_archive
docker volume create --opt type=none --opt o=bind \
  --opt device=/new_archive_dir lz_archive

logzilla start
```

Notes:

- Replace `/new_archive_dir` with the destination directory on the host. Create it before running the commands.
- The archive path inside the container remains `/var/lib/logzilla-archive/`. The `lz_archive` volume binds that path to the specified host directory.
- Ensure no containers are using the archive during the move; `logzilla stop` handles this.

Troubleshooting

Common Issues

Archive Job Failures: Check system logs and available disk space

Slow Archive Searches: Verify system resources and consider limiting search scope

Missing Archived Data: Confirm data hasn't exceeded `ARCHIVE_EXPIRE_DAYS`

Storage Space Issues: Monitor both online and archive storage locations

Recovery Procedures

For archive-related issues:

Check system logs for error messages

Verify storage configuration and available space

Contact LogZilla support for data recovery assistance

Maintain external backups for critical data protection

Ldap Authentication

Integrate LogZilla with LDAP and Active Directory for single sign-on, including multiple server configurations, group-based access, and TLS binding

LogZilla supports LDAP/Active Directory authentication with multiple server configurations. The LDAP system integrates with the modern settings framework and supports advanced features like group-based access control and TLS encryption.

Important: Before enabling LDAP authentication, ensure any existing local accounts with the same usernames or email addresses as LDAP accounts are renamed or removed to prevent conflicts.

LDAP Configuration

Web Interface Configuration (Recommended)

The **recommended approach** for configuring LDAP is through the LogZilla web interface, which provides a user-friendly experience with built-in validation and testing capabilities:

Access LDAP Settings:

- Log into the LogZilla web interface as an administrator
- Navigate to **Settings** → **Authentication** → **LDAP**

Configure Connection Settings:

- **Server URL**: Enter the LDAP server URL (e.g., `ldap://ldap.company.com:389`)
- **Bind DN**: Service account for LDAP searches
- **Bind Password**: Password for the service account
- **Configuration Name**: Friendly name for this LDAP server

Set User and Group Search:

- **User Search DN**: Base DN for user searches (e.g., `ou=users,dc=company,dc=com`)
- **Group Search DN**: Base DN for group searches (e.g., `ou=groups,dc=company,dc=com`)
- **Group Object Class**: Select appropriate class (`posixGroup`, `groupOfNames`, etc.)

Configure Field Mappings:

- **Username Field**: LDAP attribute for username (`uid`, `sAMAccountName`)
- **First Name Field**: LDAP attribute for first name (`givenName`)
- **Last Name Field**: LDAP attribute for last name (`sn`)
- **Email Field**: LDAP attribute for email (`mail`)

Set Access Control (Optional):

- **Required Groups**: Specify groups users must belong to
- **Group Whitelist**: Limit which groups are imported
- **Group Blacklist**: Exclude specific groups

Configure Security (Recommended):

- **Enable TLS/SSL**: Use encrypted connections
- **Certificate Validation**: Set appropriate validation level
- **Upload Certificates**: Add CA certificates if needed

Test and Enable:

- Use the **Test Connection** button to verify settings
- Test with actual user credentials
- **Enable** the LDAP configuration once testing succeeds

Advantages of Web Interface:

- **User-friendly forms** with validation and help text
- **Built-in testing tools** for immediate feedback
- **Certificate upload functionality** for TLS/SSL
- **Multiple server management** through the interface
- **No command-line knowledge required**

Advanced: Command Line Configuration

Note: Command-line configuration is provided for advanced users who specifically require shell access. Most users should use the web interface above, which provides the same functionality with a better user experience.

For users who prefer or require command-line administration:

```
# Initialize LDAP configuration interactively
logzilla ldap init
```

This interactive wizard will prompt for the same settings available in the web interface. After initial setup, individual settings can be modified:

```
# View current settings
logzilla settings list ldap

# Modify specific settings
logzilla settings update LDAP_TLS_START_TLS=true
logzilla settings update LDAP_OPT_DISABLE_REFERRALS=true
```

LDAP Configuration Overview

Regardless of configuration method, LDAP settings are stored in the modern LogZilla settings system:

- **Primary Configuration**: `/etc/logzilla/settings/ldap.yaml`
- **Additional Servers**: `/etc/logzilla/settings/ldap__1.yaml`, `ldap__2.yaml`, etc.
- **Certificate Storage**: `/etc/logzilla/settings/` (alongside configuration files)

The system supports multiple LDAP servers for redundancy and different organizational units.

LDAP Configuration Settings

All LDAP settings can be configured using the `logzilla settings update` command. Settings are stored in `/etc/logzilla/settings/ldap.yaml`.

Core Settings

Setting	Description	Example
LDAP_ENABLED	Enable/disable this LDAP configuration	true
LDAP_SERVER_URL	LDAP server URL	ldap://ldap.company.com:389
LDAP_BIND_DN	Service account DN for searches	cn=service,ou=users,dc=company,dc=com
LDAP_BIND_PASSWORD	Service account password	password
LDAP_CONFIG_NAME	Friendly name for this configuration	Primary Active Directory

User and Group Search

Setting	Description	Example
LDAP_USER_SEARCH_DN	Base DNs for user searches (list)	["ou=users,dc=company,dc=com"]
LDAP_GROUP_SEARCH_DN	Base DNs for group searches (list)	["ou=groups,dc=company,dc=com"]
LDAP_GROUP_OBJECT_CLASS	LDAP object class for groups	posixGroup or groupOfNames

Setting	Description	Example
LDAP_GROUP_SEARCH_DN_FILTER	LDAP filter for group searches	(objectClass=posixGroup)

Access Control

Setting	Description	Example
LDAP_REQUIRE_GROUP_DN	Required group membership (list)	["cn=logzilla-users,ou=groups,dc=company,dc=com"]
LDAP_GROUP_NAMES	Whitelist of group names to import (list)	["logzilla-admins", "logzilla-users"]
LDAP_GROUP_NAMES_EXCLUDE	Blacklist of group names to exclude (list)	["disabled-users"]

Field Mapping

Setting	Description	Common Values
LDAP_FIELDS_USERNAME	LDAP attribute for username	uid, sAMAccountName
LDAP_FIELDS_FIRST_NAME	LDAP attribute for first name	givenName
LDAP_FIELDS_LAST_NAME	LDAP attribute for last name	sn
LDAP_FIELDS_EMAIL	LDAP attribute for email	mail

TLS/SSL Settings

Setting	Description	Values
LDAP_TLS_START_TLS	Enable StartTLS	true, false
LDAP_TLS_REQUIRE_CERT	Certificate validation policy	NEVER, ALLOW, DEMAND
LDAP_TLS_CA_CERTFILE	CA certificate file path	/etc/logzilla/settings/ca.pem
LDAP_TLS_CERTFILE	Client certificate file path	/etc/logzilla/settings/client.pem
LDAP_TLS_KEYFILE	Client key file path	/etc/logzilla/settings/client.key

Advanced Options

Setting	Description	Default
LDAP_OPT_DISABLE_REFERRALS	Disable LDAP referrals (helps with AD)	false
LDAP_OPT_NETWORK_TIMEOUT	Network timeout in seconds	30

Multiple LDAP Servers

LogZilla supports multiple LDAP servers for redundancy or different organizational units.

Adding Additional Servers (Web Interface)

Navigate to LDAP Settings: Go to **Settings** → **Authentication** → **LDAP**

Add New Server: Click **Add LDAP Server** or similar option

Configure Second Server: Follow the same configuration steps as the primary server

Test and Enable: Test the additional server before enabling

Adding Additional Servers (Command Line)

For command-line users:

```
# Create additional LDAP configuration
logzilla ldap create

# Initialize the second LDAP server
logzilla ldap init --id 1

# Configure settings for second server
logzilla settings update --id 1 LDAP_SERVER_URL=ldap://ldap2.company.com:389
logzilla settings update --id 1 LDAP_CONFIG_NAME="Secondary LDAP"

# List all LDAP configurations
logzilla ldap list
```

Note: LDAP configuration IDs start at 0. Use `--id 1` when creating an additional LDAP configuration (the primary configuration is `--id 0`).

Testing and Activation

Testing LDAP Configuration

Web Interface Testing (Recommended):

Use the **Test Connection** button in the LDAP settings page

Enter test user credentials when prompted

Verify successful authentication and group retrieval

Review any error messages and adjust settings as needed

Command Line Testing: For users managing LDAP via command line:

```
# Test primary LDAP server
logzilla ldap test -u testuser -p testpassword

# Test specific LDAP server
logzilla ldap test --id 1 -u testuser -p testpassword
```

Enabling LDAP Authentication

Web Interface Activation:

After successful testing, use the **Enable** toggle in the web interface

The change takes effect immediately

Users can now log in using their LDAP credentials

Command Line Activation:

```
# Enable primary LDAP server
logzilla ldap enable

# Enable specific LDAP server
logzilla ldap enable --id 1

# Disable LDAP server if needed
logzilla ldap disable --id 1
```

Configuration Examples

These examples show common LDAP configurations. **Use the web interface** to enter these values through the user-friendly forms, or use the command-line examples if you prefer shell access.

Active Directory Configuration

Web Interface Settings:

- **Server URL**: `ldap://ad.company.com:389`
- **Bind DN**: `cn=ldapservice,ou=Service Accounts,dc=company,dc=com`
- **User Search DN**: `ou=Users,dc=company,dc=com`
- **Group Search DN**: `ou=Groups,dc=company,dc=com`
- **Username Field**: `sAMAccountName`

- **Group Object Class**: `group`
- **Advanced Options**: Enable "Disable Referrals"

Command Line Equivalent:

```
logzilla settings update LDAP_SERVER_URL=ldap://ad.company.com:389
logzilla settings update LDAP_BIND_DN="cn=ldapservice,ou=Service
Accounts,dc=company,dc=com"
logzilla settings update LDAP_USER_SEARCH_DN='["ou=Users,dc=company,dc=com"]'
logzilla settings update LDAP_GROUP_SEARCH_DN='["ou=Groups,dc=company,dc=com"]'
logzilla settings update LDAP_FIELDS_USERNAME=sAMAccountName
logzilla settings update LDAP_GROUP_OBJECT_CLASS=group
logzilla settings update LDAP_OPT_DISABLE_REFERRALS=true
```

OpenLDAP Configuration

Web Interface Settings:

- **Server URL**: `ldap://openldap.company.com:389`
- **Bind DN**: `cn=admin,dc=company,dc=com`
- **User Search DN**: `ou=people,dc=company,dc=com`
- **Group Search DN**: `ou=groups,dc=company,dc=com`
- **Username Field**: `uid`
- **Group Object Class**: `posixGroup`

Command Line Equivalent:

```
logzilla settings update LDAP_SERVER_URL=ldap://openldap.company.com:389
logzilla settings update LDAP_BIND_DN="cn=admin,dc=company,dc=com"
logzilla settings update LDAP_USER_SEARCH_DN='["ou=people,dc=company,dc=com"]'
logzilla settings update LDAP_GROUP_SEARCH_DN='["ou=groups,dc=company,dc=com"]'
logzilla settings update LDAP_FIELDS_USERNAME=uid
logzilla settings update LDAP_GROUP_OBJECT_CLASS=posixGroup
```

TLS/SSL Configuration Example

Web Interface Settings:

- **Enable TLS**: Check the TLS/SSL option
- **Certificate Validation**: Set to "Required" for production
- **Upload CA Certificate**: Use the certificate upload feature
- **For LDAPS**: Use `ldaps://ldap.company.com:636` as Server URL

Command Line Equivalent:

```
logzilla settings update LDAP_TLS_START_TLS=true
logzilla settings update LDAP_TLS_REQUIRE_CERT=DEMAND
logzilla settings update LDAP_TLS_CA_CERTFILE=/etc/logzilla/settings/ca.pem

# For LDAPS (SSL)
logzilla settings update LDAP_SERVER_URL=ldaps://ldap.company.com:636
```

Group-Based Access Control

Web Interface Settings:

- **Required Groups**: Add groups that users must belong to
- **Group Whitelist**: Specify which groups to import
- **Group Blacklist**: Specify which groups to exclude

Command Line Equivalent:

```
logzilla settings update LDAP_REQUIRE_GROUP_DN='["cn=logzilla-
users,ou=groups,dc=company,dc=com"]'
logzilla settings update LDAP_GROUP_NAMES='["logzilla-admins", "logzilla-users",
"logzilla-viewers"]'
logzilla settings update LDAP_GROUP_NAMES_EXCLUDE='["disabled-accounts", "temp-users"]'
```

User Authentication

Login Requirements

Users should authenticate using their LDAP username only:

Correct Login Format:

- `jdoe`
- `john.doe`

Incorrect Login Formats:

- `jdoe@company.com` ❌
- `COMPANY\jdoe` ❌
- `cn=jdoe,ou=users,dc=company,dc=com` ❌

User Account Creation

When users successfully authenticate via LDAP:

LogZilla automatically creates local user accounts

User information is populated from LDAP attributes

Group memberships are synchronized

Users inherit permissions based on their LDAP groups

Troubleshooting

Common Issues

Authentication Failures:

```
# Check LDAP connectivity
logzilla ldap test -u testuser -p testpassword

# Verify settings
logzilla settings list ldap
```

Group Synchronization Issues:

```
# Check group search configuration
logzilla settings list ldap | grep GROUP

# Test with specific group filters
logzilla settings update LDAP_GROUP_SEARCH_DN_FILTER="(objectClass=posixGroup)"
```

TLS/SSL Problems:

```
# Disable certificate validation for testing
logzilla settings update LDAP_TLS_REQUIRE_CERT=NEVER

# Check certificate paths
ls -la /etc/logzilla/settings/*.pem
```

Log Analysis

LDAP authentication events are logged to the main LogZilla log:

```
# Monitor LDAP authentication attempts
grep -i "ldap\|auth" /var/log/logzilla/logzilla.log

# Check for specific errors
grep -i "ldap.*error" /var/log/logzilla/logzilla.log
```

Best Practices

Security

- **Use TLS/SSL**: Always encrypt LDAP communications in production
- **Service Accounts**: Use dedicated service accounts with minimal privileges
- **Group-Based Access**: Implement group-based access control
- **Regular Testing**: Periodically test LDAP connectivity

Performance

- **Multiple Servers**: Configure multiple LDAP servers for redundancy
- **Network Timeouts**: Adjust timeouts based on network conditions
- **Group Filtering**: Use specific group filters to reduce search scope

Maintenance

- **Monitor Logs**: Regularly check authentication logs
- **Update Certificates**: Keep TLS certificates current
- **Test Changes**: Always test configuration changes in non-production first
- **Document Settings**: Maintain documentation of LDAP configuration

Pci Compliance

Enable PCI DSS Requirement 10.3 raw message preservation in LogZilla, storing unaltered logs in a segregated directory for audit trail integrity

LogZilla provides PCI DSS compliance features to meet audit trail requirements for secure log management. The system can maintain unaltered copies of raw log messages in a segregated directory structure, supporting compliance with PCI DSS Requirement 10.3 for audit trail integrity.

Critical Warning: PCI compliance logging will more than **double the log storage requirements** as it creates complete copies of all incoming log messages without deduplication. Monitor disk usage closely and implement log rotation and retention policies immediately after enabling this feature to prevent disk space exhaustion.

PCI Compliance Overview

LogZilla's PCI compliance feature provides:

- **Raw Message Preservation**: Stores original, unmodified log messages
- **Segregated Storage**: Maintains compliance logs separate from operational logs
- **Structured Organization**: Organizes logs by date for easy management
- **Audit Trail Integrity**: Preserves complete message content for compliance audits

Enabling PCI Compliance Logging

Web Interface Configuration

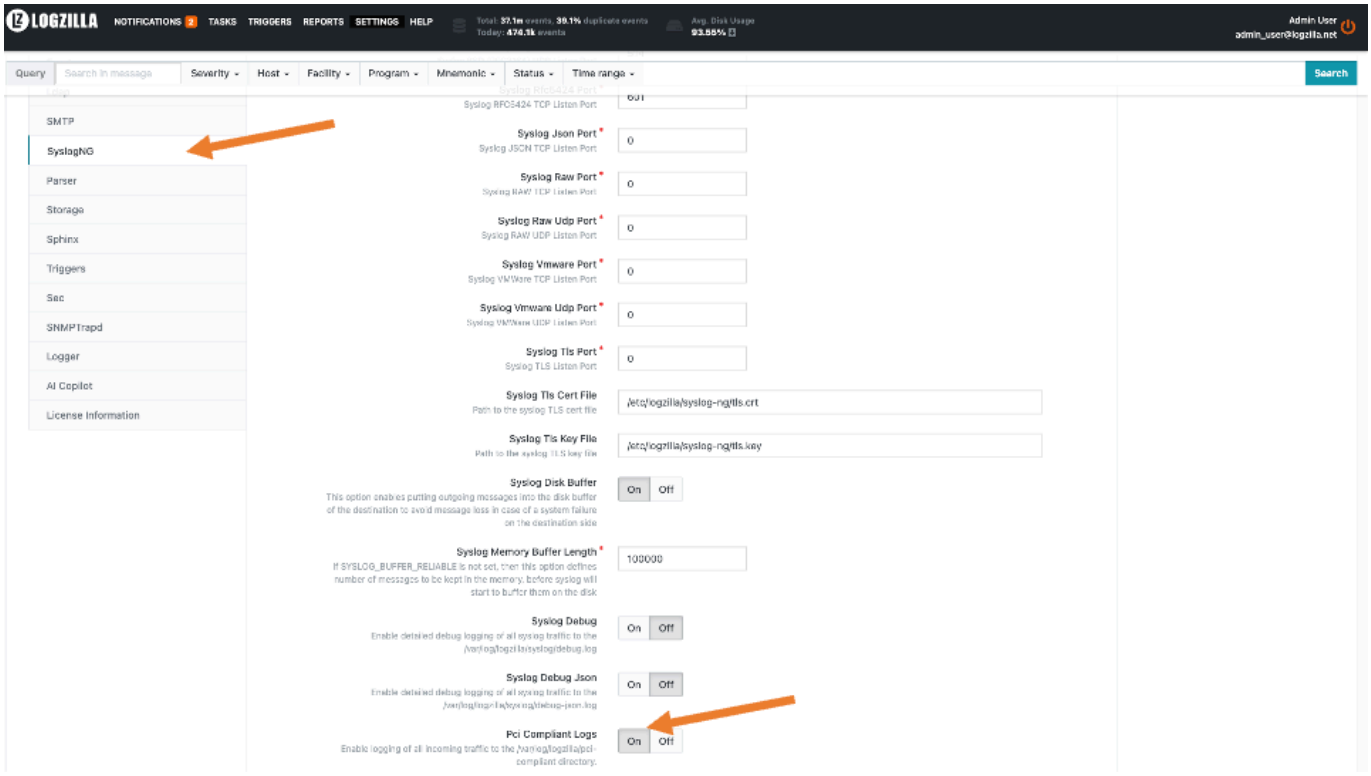
The **recommended approach** for enabling PCI compliance logging is through the LogZilla web interface, which provides a user-friendly experience with immediate validation and status feedback:

Access PCI Settings:

- Log into the LogZilla web interface as an administrator
- Navigate to **Settings** → **Syslog Daemon** → **PCI Compliance**

Enable PCI Logging:

- Locate the **"PCI Compliant Logs"** setting
- Toggle the setting to **"On"**
- Enabling takes effect immediately



Verify Log Generation:

- Check that PCI log files begin appearing in the designated directory under `/var/log/logzilla/pci-compliant/`

Command Line Configuration

Note: Command-line configuration is provided for advanced users who specifically require shell access. Most users should use the web interface above, which provides the same functionality with better user experience.

For users who prefer command-line administration:

```
# Enable PCI compliance logging
logzilla settings update PCI_COMPLIANT_LOGS=true

# Verify the setting
logzilla settings list syslogng | grep PCI
```

PCI Logging Deactivation

Note: Disabling PCI logging stops new log collection but does not automatically remove existing PCI compliance log files. Administrators must manually manage existing files according to their retention policies.

For UI users:

Access PCI Settings:

- Log into the LogZilla web interface as an administrator
- Navigate to **Settings** → **Syslog Daemon** → **PCI Compliance**

Disable PCI Logging:

- Locate the **"PCI Compliant Logs"** setting
- Toggle the setting to **"Off"**
- Disabling takes effect immediately

For command-line users:

```
# Disable PCI compliance logging
logzilla settings update PCI_COMPLIANT_LOGS=false

# Verify the setting is disabled
logzilla settings list syslogng | grep PCI
```

Log Storage and Organization

Directory Structure

PCI compliance logs are stored in a structured format:

```
/var/log/logzilla/pci-compliant/
├── 2025-09/
│   ├── 2025-09-25.log
│   ├── 2025-09-26.log
│   └── ...
├── 2025-08/
│   ├── 2025-08-25.log
│   ├── 2025-08-26.log
│   └── ...
└── checksums (if using external integrity monitoring)
```

Log Content

PCI compliance logs contain:

- **Raw Messages**: Complete, unaltered syslog messages (`$RAWMSG`)
- **Original Format**: Messages exactly as received by LogZilla (standard BSD syslog format, RFC 3164)
- **No Processing**: No parsing, normalization, or modification applied
- **Complete Audit Trail**: All incoming log data for compliance verification

File Integrity and Checksums

Built-in Security

LogZilla provides basic file security:

- **File Permissions**: Logs written with `0644` permissions
- **Directory Permissions**: Directories created with `0755` permissions
- **Segregated Storage**: Compliance logs isolated from operational data

External Integrity Monitoring

Important: The following scripts are **administrator-created examples** and are not built-in LogZilla utilities. These scripts must be created, tested, and maintained by system administrators according to their specific compliance requirements.

For enhanced compliance, implement external file integrity monitoring:

```
# Create daily checksum and compression script
cat << 'EOF' > /usr/local/bin/logzilla-pci-checksum.sh
#!/bin/bash
# LogZilla PCI Compliance Checksum Script
# This is an EXAMPLE script - customize for your environment

PCI_DIR="/var/log/logzilla/pci-compliant"
CHECKSUM_FILE="$PCI_DIR/checksums"
YESTERDAY=$(date -d "yesterday" +%Y-%m-%d)
YESTERDAY_MONTH=$(date -d "yesterday" +%Y-%m)
LOG_FILE="$PCI_DIR/$YESTERDAY_MONTH/$YESTERDAY.log"

# Process yesterday's log file if it exists
if [ -f "$LOG_FILE" ]; then
    echo "$(date): Processing $YESTERDAY.log" >> "$CHECKSUM_FILE"

    # Create SHA256 checksum of original file (before compression)
    sha256sum "$LOG_FILE" >> "$CHECKSUM_FILE"

    # Compress the log file
    gzip "$LOG_FILE"

    # Create SHA256 checksum of compressed file
    sha256sum "$LOG_FILE.gz" >> "$CHECKSUM_FILE"

    echo "$(date): Completed processing $YESTERDAY.log.gz" >> "$CHECKSUM_FILE"

    # Optional: Set immutable attribute to prevent tampering (requires ext2/3/4)
    # chattr +i "$LOG_FILE.gz"
else
    echo "$(date): Warning - Log file $LOG_FILE not found" >> "$CHECKSUM_FILE"
fi
EOF

# Make script executable
chmod +x /usr/local/bin/logzilla-pci-checksum.sh
```

Automated Daily Processing

Set up automated daily processing:

```
# Create cron job for daily checksum processing
cat << 'EOF' > /etc/cron.d/logzilla-pci
# LogZilla PCI Compliance - Daily log processing
# Runs at 12:01 AM daily to process previous day's logs
# IMPORTANT: This is an example - test thoroughly before production use
1 0 * * * root /usr/local/bin/logzilla-pci-checksum.sh >> /var/log/logzilla/pci-
processing.log 2>&1

# Optional: Weekly verification of checksums
# 0 2 * * 0 root /usr/local/bin/logzilla-pci-verify.sh >> /var/log/logzilla/pci-
verification.log 2>&1
EOF
```

Compliance Management

Log Retention

Note: The following retention script is an **example template** that administrators should customize for their specific environment and compliance requirements.

Implement appropriate retention policies:

```
# Example: Retain PCI logs for 1 year, then archive
cat << 'EOF' > /usr/local/bin/logzilla-pci-retention.sh
#!/bin/bash
# LogZilla PCI Log Retention Script

PCI_DIR="/var/log/logzilla/pci-compliant"
ARCHIVE_DIR="/backup/logzilla-pci-archive"
RETENTION_DAYS=365

# Create archive directory if it doesn't exist
mkdir -p "$ARCHIVE_DIR"

# Find and archive logs older than retention period
find "$PCI_DIR" -name "*.log.gz" -mtime +$RETENTION_DAYS -type f \
    -exec mv {} "$ARCHIVE_DIR/" \;

# Log retention activity
echo "$(date): PCI log retention completed" >> "$PCI_DIR/retention.log"
EOF

chmod +x /usr/local/bin/logzilla-pci-retention.sh
```

Backup Procedures

Note: The following backup script is an **example template** that administrators should adapt for their specific backup infrastructure and security requirements.

Implement secure backup procedures:

```
# Example backup script
cat << 'EOF' > /usr/local/bin/logzilla-pci-backup.sh
#!/bin/bash
# LogZilla PCI Compliance Backup Script

PCI_DIR="/var/log/logzilla/pci-compliant"
BACKUP_DEST="/secure/backup/location"
DATE=$(date +%Y%m%d)

# Create compressed backup
tar -czf "$BACKUP_DEST/logzilla-pci-$DATE.tar.gz" -C "$PCI_DIR" .

# Verify backup integrity
if [ $? -eq 0 ]; then
    echo "$(date): PCI backup completed successfully" >> "$PCI_DIR/backup.log"
else
    echo "$(date): PCI backup FAILED" >> "$PCI_DIR/backup.log"
fi
EOF

chmod +x /usr/local/bin/logzilla-pci-backup.sh
```

Monitoring and Verification

Check PCI Logging Status

```
# Verify PCI logging is enabled
logzilla settings list syslogng | grep PCI_COMPLIANT_LOGS

# Check recent PCI log files
ls -la /var/log/logzilla/pci-compliant/$(date +%Y-%m)/

# Monitor log file growth
watch "ls -lh /var/log/logzilla/pci-compliant/$(date +%Y-%m)/$(date +%Y-%m-%d).log"
```

Verify Log Content

```
# Sample recent PCI log entries
tail -n 10 /var/log/logzilla/pci-compliant/$(date +%Y-%m)/$(date +%Y-%m-%d).log

# Check for specific time period
grep "$(date +%H:%M)" /var/log/logzilla/pci-compliant/$(date +%Y-%m)/$(date +%Y-%m-
%d).log
```

Integrity Verification

```
# Verify checksums
md5sum -c /var/log/logzilla/pci-compliant/checksums

# Check for file modifications
find /var/log/logzilla/pci-compliant -name "*.log.gz" -newer /var/log/logzilla/pci-
compliant/checksums
```

Best Practices

Security

- **Access Control**: Restrict access to PCI log directories
- **File Permissions**: Maintain appropriate file and directory permissions
- **Secure Backup**: Store backups in secure, encrypted locations
- **Regular Audits**: Periodically verify log integrity and completeness

Performance

- **Disk Space**: Monitor disk usage in PCI log directories
- **I/O Impact**: Consider I/O impact of dual logging on system performance
- **Log Rotation**: Implement timely compression and archival
- **Network Storage**: Consider dedicated storage for compliance logs

Compliance

- **Retention Policies**: Implement policies meeting regulatory requirements
- **Documentation**: Maintain documentation of PCI logging procedures
- **Regular Testing**: Test backup and recovery procedures
- **Audit Preparation**: Ensure logs are readily available for compliance audits

Troubleshooting

Common Issues

PCI Logging Not Working:

```
# Check if setting is enabled
logzilla settings list syslogng | grep PCI_COMPLIANT_LOGS
```

Missing Log Files:

```
# Check directory permissions
ls -ld /var/log/logzilla/pci-compliant/
```

Disk Space Issues:

```
# Check available space
df -h /var/log/logzilla/pci-compliant/

# Find largest log files
find /var/log/logzilla/pci-compliant -type f -exec ls -lh {} \; | sort -k5 -hr | head -10
```

Role Based Access Control

Configure LogZilla RBAC with group-based permissions, UI component restrictions, host and subnet filtering, and fine-grained feature controls

LogZilla's Role-Based Access Control (RBAC) system provides granular control over user access to system resources, data, and functionality. RBAC enables administrators to create groups with specific permissions and restrict user access based on their organizational roles and responsibilities.

Important: RBAC must be enabled in the system settings before groups and permissions can be configured. Navigate to **Settings** → **System Settings** → **Generic** and ensure **RBAC_ENABLED** is set to **true**.

RBAC Overview

LogZilla's RBAC system provides access control through:

- **Group-Based Permissions**: Organize users into groups with specific access rights
- **UI Component Control**: Restrict access to dashboards, search, triggers, and other interface elements
- **Host-Based Filtering**: Limit data visibility to specific hosts, IP ranges, or subnets
- **User Management**: Assign users to multiple groups with inherited permissions
- **Granular Controls**: Fine-tune access to individual system features

Managing Groups and Permissions

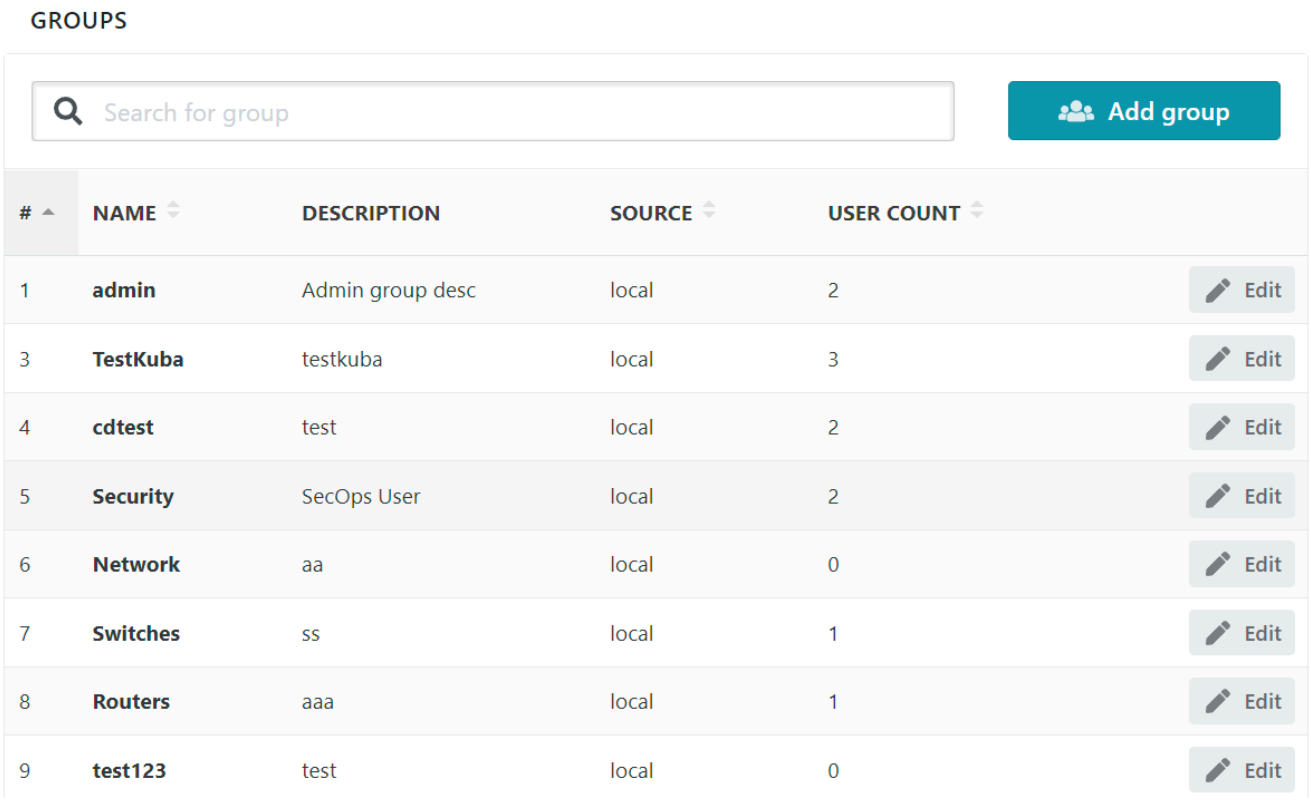
Web Interface Management (Recommended)

The **recommended approach** for managing RBAC is through the LogZilla web interface:

Accessing RBAC Settings

Navigate to User Management:

- Log into LogZilla as an administrator
- Go to **Settings** → **Users & Groups**
- Select the **Groups** tab to view existing groups



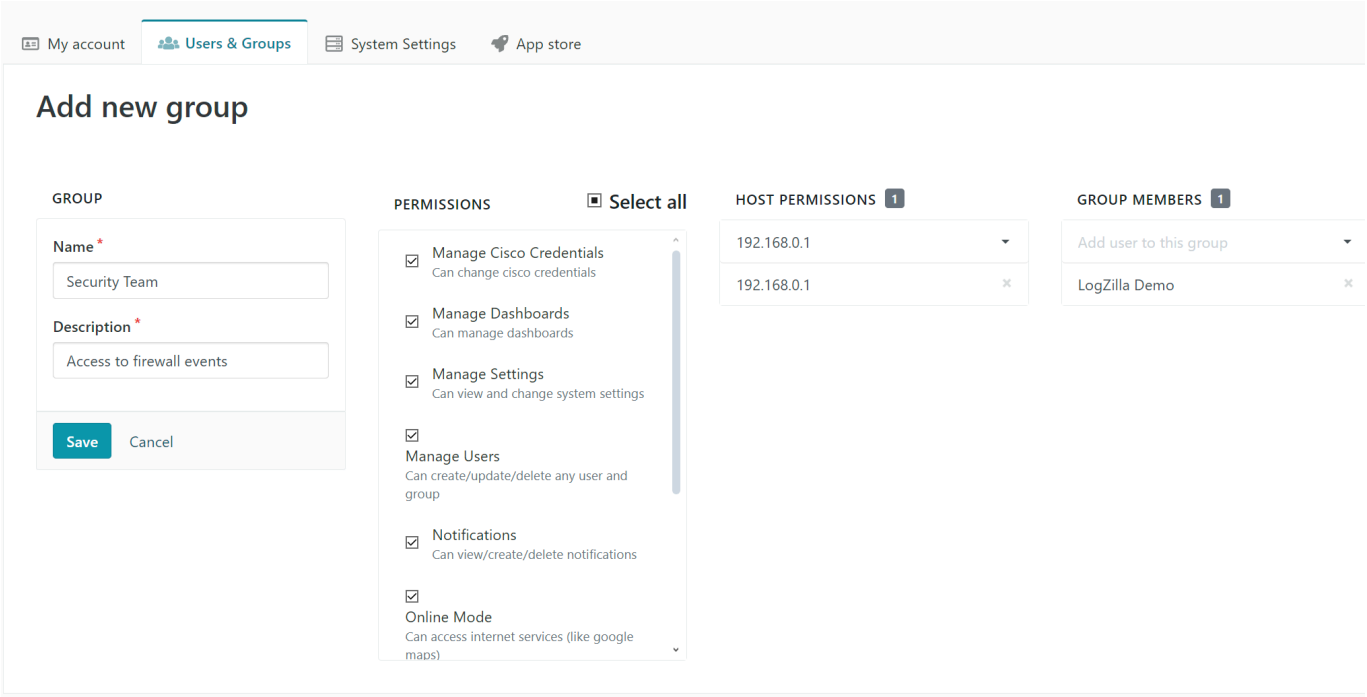
Creating New Groups

Start Group Creation:

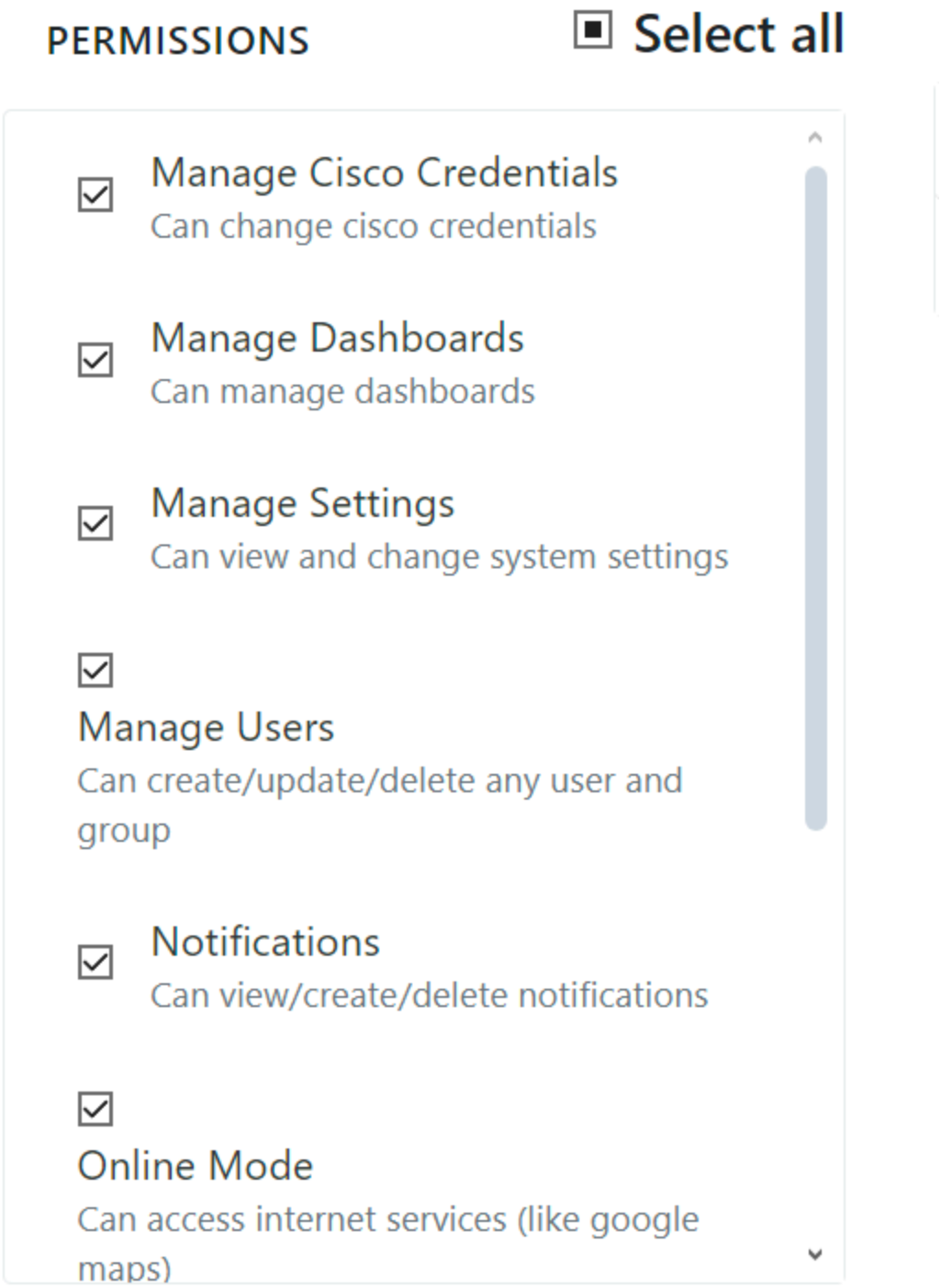
- Click the **"Add group"** button in the Groups interface
- This opens the group configuration form

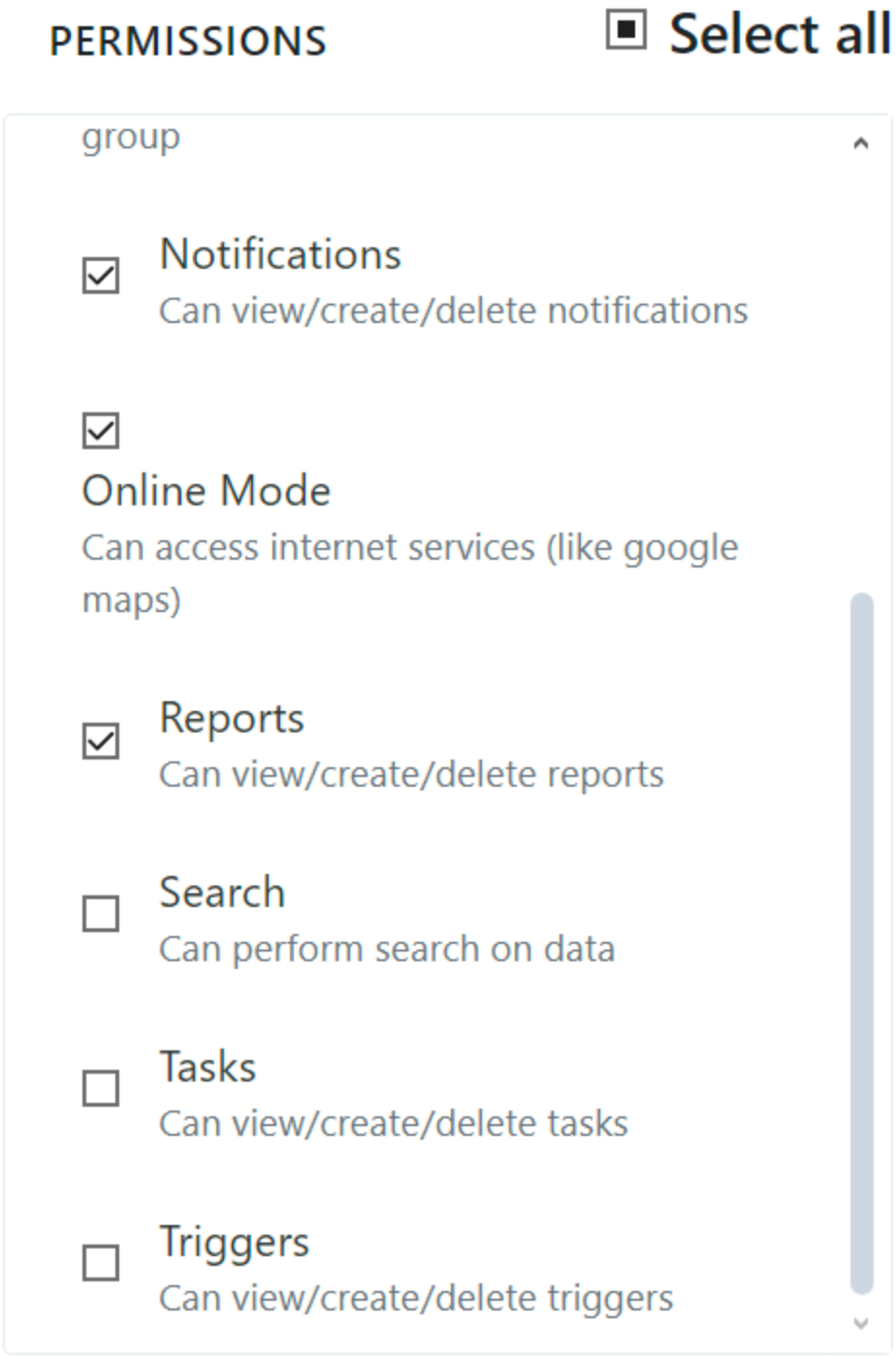
Configure Basic Group Information:

- **Name**: Enter a descriptive group name (e.g., "Security Team", "Network Operators")
- **Description**: Provide a detailed description of the group's purpose



Set UI Permissions:


PERMISSIONS
Select all
Manage Cisco Credentials
Can change cisco credentials
Manage Dashboards
Can manage dashboards
Manage Settings
Can view and change system settings
Manage Users
Can create/update/delete any user and group
Notifications
Can view/create/delete notifications
Online Mode
Can access internet services (like google maps)



- **Individual Permissions**: Check specific permissions for granular control:
  - **Manage Cisco Credentials**: Change Cisco credentials
  - **Manage Dashboards**: Create, edit, and delete dashboards
  - **Manage Settings**: Change system settings
  - **Manage Users**: Create, edit, and delete users

  - **Notifications**: View and manage alert notifications
  - **Online Mode**: Access external internet services
  - **Reports**: Can view/create/delete reports
  - **Search**: Perform log searches and queries
  - **Tasks**: Can view/create/delete tasks
  - **Triggers**: Create and manage automated triggers
- **Select All**: Enable all available UI permissions for full access
- **Permission Descriptions**: Each permission includes helpful descriptions

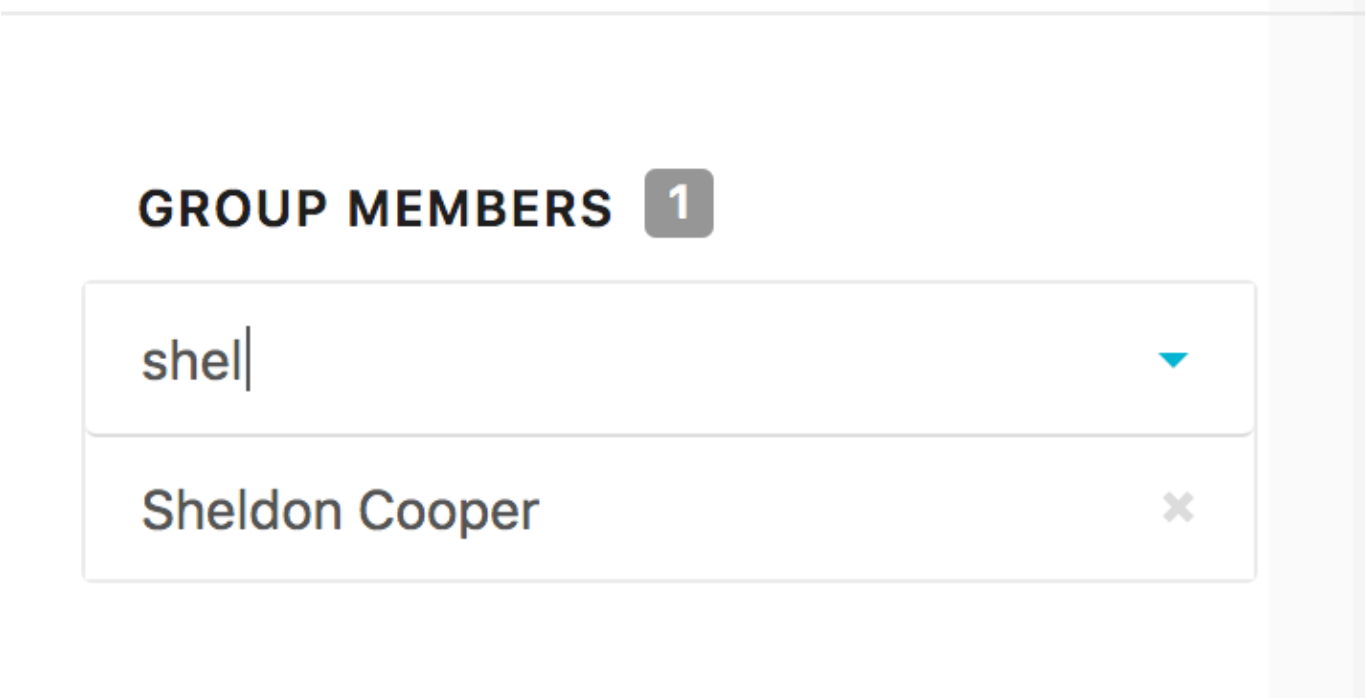
Configure Host Permissions:

- **Add Specific Hosts**: Enter individual hostnames or IP addresses
- **Use Wildcards**: Implement pattern-based filtering:
  - `192.168.28.*` - Entire subnet access
  - `web-server-*` - All hosts matching pattern
  - `*.company.com` - All hosts in domain
- **Multiple Entries**: Add multiple host patterns for complex filtering

Assign Group Members:

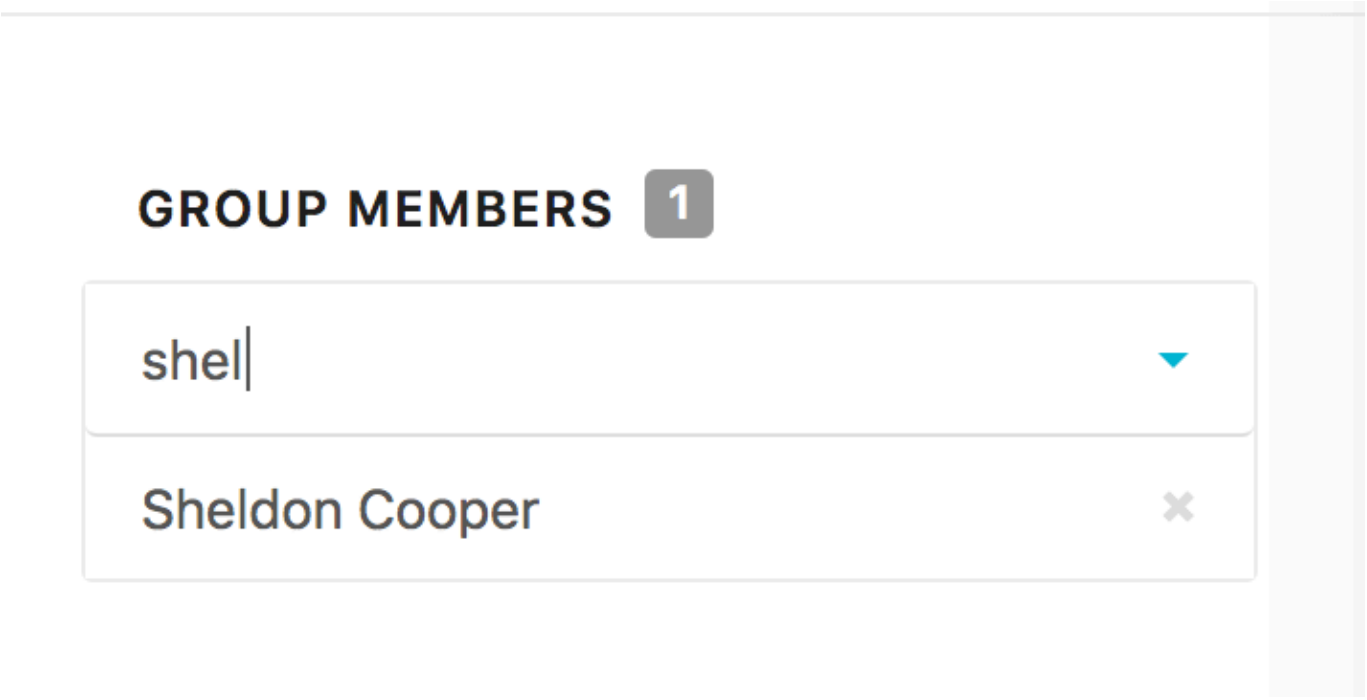
- **Search Users**: Type to search for existing users
- **Select Members**: Choose users to add to the group
- **Multiple Selection**: Add multiple users simultaneously

- **User Display**: Shows full names for easy identification



Assign Group Members:

- **Search Users**: Type to search for existing users
- **Select Members**: Choose users to add to the group
- **Multiple Selection**: Add multiple users simultaneously
- **User Display**: Shows full names for easy identification



Save and Apply:

- Click **"Save"** to create the group
- Changes take effect immediately
- Users receive new permissions upon next login

Advantages of Web Interface:

- **Visual permission management** with clear descriptions
- **Real-time validation** of host patterns and user selections
- **Intuitive group creation** with guided workflows
- **Immediate feedback** on configuration changes
- **No command-line knowledge required**

Advanced: Command Line Management

Note: Command-line RBAC management is provided for advanced users who specifically require shell access or automated group management. Most users should use the web interface above for better user experience and validation.

For advanced users who prefer command-line administration:

```
# Enable RBAC system-wide
logzilla settings update RBAC_ENABLED=true

# View current RBAC settings
logzilla settings list generic | grep RBAC
```

- Users receive new permissions upon next login

Advantages of Web Interface:

- **Visual permission management** with clear descriptions
- **Real-time validation** of host patterns and user selections
- **Intuitive group creation** with guided workflows
- **Immediate feedback** on configuration changes
- **No command-line knowledge required**
- **Built-in help text** for each permission type

Permission Types and Capabilities

UI Permissions

Permission	Description (paraphrased)	Impact
Manage Cisco Credentials	Modify stored Cisco device credentials for integrations	Integration access and device connectivity
Manage Dashboards	Create, update, and remove dashboards	Full dashboard control
Manage Settings	Change global system configuration	System-wide configuration changes
Manage Users	Create, edit, and delete user accounts	User lifecycle and access management
Notifications	View and manage alert notifications	Alert system access
Online Mode	Allow access to external internet services	External connectivity
Reports	View, create, and delete reports	Reporting data creation and removal
Search	Perform log searches and queries	Core search functionality
Tasks	View, create, and delete tasks	Task workflow management
Triggers	Create and manage automated triggers	Automation control

Host Permissions

Exact Matching:

- `192.168.1.100` - Specific IP address
- `web-server-01` - Specific hostname
- `mail.company.com` - Specific FQDN

Wildcard Patterns:

- `192.168.1.*` - Entire subnet (192.168.1.0/24)
- `192.168.*.*` - Larger subnet range
- `web-server-*` - All hosts starting with "web-server-"
- `*.company.com` - All hosts in company.com domain
- `*database*` - Any host containing "database"

Multiple Patterns:

- Combine multiple patterns for complex filtering
- Each pattern is evaluated independently
- Users see logs from ANY matching pattern

User Management

Assigning Users to Groups

During Group Creation: Add users when creating new groups

Edit Existing Groups: Modify group membership anytime

Multiple Group Membership: Users can belong to multiple groups

Permission Inheritance: Users inherit ALL permissions from ALL groups

User Permission Resolution

Additive Permissions: Users receive the union of all group permissions **Host Access**: Users can access hosts from ALL assigned groups **UI Access**: Users get the broadest UI permissions from any group

Best Practices

Security

- **Principle of Least Privilege**: Grant minimum necessary permissions
- **Regular Audits**: Periodically review group memberships and permissions
- **Role-Based Design**: Create groups based on job functions, not individuals
- **Host Segmentation**: Use specific host patterns rather than broad wildcards

Organization

- **Descriptive Names**: Use clear, meaningful group names
- **Detailed Descriptions**: Document group purposes and intended users
- **Logical Grouping**: Organize permissions by department or function
- **Documentation**: Maintain records of RBAC decisions and changes

Performance

- **Efficient Patterns**: Use specific host patterns to reduce processing overhead
- **Group Optimization**: Avoid excessive group proliferation
- **Regular Cleanup**: Remove unused groups and inactive users

Common Use Cases

Network Operations Team

Permissions: Search, Dashboards, Notifications **Host Access**: `router-*`, `switch-*`, `firewall-*` **Purpose**: Monitor network infrastructure

Security Team

Permissions: All UI permissions **Host Access**: `*` (all hosts) **Purpose**: Full system access for security monitoring

Application Support

Permissions: Search, Dashboards **Host Access**: `app-server-*`, `web-server-*` **Purpose**: Monitor specific application infrastructure

Database Administrators

Permissions: Search, Dashboards, Triggers **Host Access**: `db-*`, `*database*` **Purpose**: Monitor and manage database systems

Troubleshooting

Common Issues

Users Cannot Access Expected Data:

Verify RBAC is enabled in system settings
Check user group memberships
Verify host permission patterns
Confirm UI permissions are granted

Host Patterns Not Working:

Test patterns with specific examples
Verify wildcard syntax (`*` for any characters)
Check for typos in hostnames or IP addresses
Ensure patterns match actual log data

Permission Changes Not Applied:

Users may need to log out and back in
Verify group configuration was saved
Check for conflicting group memberships
Confirm RBAC system is enabled

Verification

```
# Check if RBAC is enabled
logzilla settings list generic | grep RBAC_ENABLED

# Verify user can see expected hosts in search results
# (This verification is done through the web interface)
```

Example: Security Team Configuration

This example demonstrates creating a security team group:

Group Configuration:

- **Name**: "Security Team"
- **Description**: "Full access for security monitoring and incident response"
- **UI Permissions**: All permissions enabled
- **Host Permissions**: `*` (all hosts)
- **Members**: Security analysts and incident responders

Result: Security team members can:

- Access all system logs from any host
- Create and manage dashboards
- Set up automated triggers for security events
- Receive and manage security notifications
- Perform searches across all data

In this configuration, a user like "Sheldon Cooper" assigned to a group with `192.168.28.*` host permissions would only see log events from devices in the 192.168.28.0/24 subnet, providing effective data segmentation for role-based access control.

Offline Installs And Upgrades

Install or upgrade LogZilla in air-gapped environments using the offline kickstart package, which bundles all required Docker images and scripts

LogZilla supports offline installations and upgrades for air-gapped environments where direct internet access is not available. The offline installation package contains all necessary Docker images and installation scripts, enabling complete LogZilla deployment without external network connectivity.

Important: Offline installations require Docker to be pre-installed on the target system. The offline package does not include Docker itself.

Offline Installation Overview

LogZilla's offline installation system provides:

- **Self-Contained Package**: Complete installation bundle with all required Docker images
- **Air-Gap Support**: No internet connectivity required on target systems
- **Automated Installation**: Streamlined installation process via kickstart script
- **Upgrade Capability**: In-place upgrades using offline packages
- **Version Verification**: Built-in version checking and validation

Prerequisites

System Requirements

- **Target System**: LogZilla server (air-gapped or offline)
- **Download System**: Any system with internet access for package download
- **Root Access**: Administrative privileges on the LogZilla server
- **Docker**: Docker Engine installed and running on target system
- **System Resources**: Minimum CPU and RAM requirements as per LogZilla specifications

File Transfer Method

Choose an appropriate method for transferring the offline package:

- **SCP/RSYNC**: Secure copy over existing network connections
- **Network Share**: Shared storage accessible to both systems
- **Physical Media**: USB drives, CD/DVD, or other removable media

Downloading the Offline Package

Web Interface Download (Recommended)

The **recommended approach** for obtaining the offline package is through the LogZilla license portal:

Access Download Portal:

- Navigate to https://license.logzilla.net/download/
- The system automatically provides the latest stable version
- Download begins immediately upon accessing the URL

Package Information:

- **File Format**: `logzilla-v6.x.y.tar.gz`
- **Package Contents**: All required Docker images and installation scripts
- **File Size**: Typically 2-4 GB depending on version
- **Naming Convention**: Version number included in filename

Verify Download:

- Confirm complete download by checking file size
- Note the exact filename for use in subsequent commands
- Verify file integrity if checksums are provided

Package Transfer

Transfer the downloaded package to the offline LogZilla server using one of these methods:

Network Transfer:

```
# SCP transfer
scp logzilla-v6.x.y.tar.gz user@offline-server:/tmp/

# RSYNC transfer
rsync -avz logzilla-v6.x.y.tar.gz user@offline-server:/tmp/
```

Physical Media Transfer:

```
# Copy to removable media
cp logzilla-v6.x.y.tar.gz /media/usb-drive/
sync && umount /media/usb-drive
```

Note: Replace `logzilla-v6.x.y.tar.gz` with the actual filename of the downloaded package throughout this document.

Determining Installation vs Upgrade

Critical Warning: Running a new installation on an existing LogZilla system will **destroy all data and configuration**. Always verify whether LogZilla is already installed before proceeding.

Check for Existing Installation

Before proceeding with any installation, verify if LogZilla is already installed:

```
# Check if logzilla command exists
which logzilla

# If logzilla exists, check current version
logzilla version 2>/dev/null

# Check for LogZilla Docker containers
docker ps -a | grep lz_

# Check for LogZilla settings directory
ls -la /etc/logzilla/ 2>/dev/null
```

Decision Matrix:

- **If `logzilla version` returns a version number**: Use the **Upgrade Procedure** below
- **If `logzilla` command does not exist**: Use the **New Installation Procedure** below
- **If unsure**: Contact LogZilla support before proceeding

Upgrade Procedure (Existing Systems)

Important: This section is for systems that **already have LogZilla installed**. If LogZilla is not currently installed, skip to the New Installation section below.

Pre-Upgrade Preparation

System Verification:

```
# Check current LogZilla version
logzilla version

# Check available disk space for Docker
docker info --format '{{.DockerRootDir}}' | xargs df -h
```

Backup Recommendations:

- Create system backup before upgrading

- Document current configuration settings
- Note any custom modifications or integrations

Command Line Upgrade Process

Extract Upgrade Package:

```
# Navigate to transfer location
cd /tmp

# Extract the new version package
tar xzvf logzilla-v6.x.y.tar.gz
```

Execute Upgrade:

```
# Run upgrade command pointing to extracted directory
logzilla upgrade --offline-dir /tmp/logzilla-v6.x.y
```

Upgrade Process: The upgrade command automatically:

- Loads new Docker images from the offline directory
- Stops existing LogZilla containers in proper sequence
- Starts new containers with upgraded images
- Updates system configurations

Verify Upgrade:

```
# Check new version
logzilla version

# Test web interface accessibility
curl -X GET http://localhost/api/ping
```

Upgrade Output Example:

```
Loading /tmp/logzilla-v6.x.y/logzilla-runtime:v6.x.y.tar.gz ...
Loading /tmp/logzilla-v6.x.y/postgres:15.2-alpine.tar.gz ...
Starting LogZilla upgrade to 'v6.x.y'
Decommissioning old containers...
Starting new containers...
LogZilla successfully upgraded to 'v6.x.y'
```

Post-Upgrade Verification

System Health Check:

```
# Verify version upgrade
logzilla version

# Test log ingestion
logger "Test message after upgrade"
```

Web Interface Verification:

Access LogZilla web interface

Confirm user authentication works

Verify dashboard functionality

Test search capabilities

Check system settings and configurations

New Installation Procedure (Fresh Systems Only)

Critical: This procedure is **only for systems that do not have LogZilla installed**. Running this on an existing LogZilla system will destroy all data. Use the Upgrade Procedure above for existing installations.

Post-Installation Setup

After completing the offline installation, the first administrator login triggers a mandatory setup sequence:

Initial Login: Log in with the initial administrator credentials

EULA Acceptance: Accept the End-User License Agreement

Instance Registration: Complete the license registration form with:

- User Name
- Company Name
- Email Address
- Phone Number

License Retrieval: The system automatically obtains a license based on registration information

Dashboard Access: Access the main LogZilla interface and dashboards

Installation Process

Prerequisites Check:

```
# Verify Docker is installed and running
docker --version
systemctl status docker

# Ensure sufficient system resources
free -h  # Check available RAM
nproc    # Check CPU cores

# CRITICAL: Verify LogZilla is NOT already installed
logzilla version 2>/dev/null && echo "ERROR: LogZilla already installed! Use upgrade
procedure instead."
```

Installation Steps:

Extract Package:

```
# Navigate to transfer location
cd /tmp

# Extract the offline package
tar xzvf logzilla-v6.x.y.tar.gz
```

This creates a directory named `logzilla-v6.x.y` containing:

- Docker image archives (`.tar.gz` files)
- Installation script (`kickstart.sh`)
- Version information (`version.txt`)
- Package manifest

Run Installation Script:

```
# Navigate to extracted directory
cd logzilla-v6.x.y

# Execute installation with root privileges
sudo bash kickstart.sh
```

Installation Process: The `kickstart.sh` script automatically:

- Verifies system requirements (CPU, RAM, Docker)
- Loads all Docker images from the package
- Executes `logzilla install --version <version> --offline`
- Configures initial system settings

Post-Installation Setup:

```
# Verify installation
logzilla version
```

Installation Output Example:

```
Checking system requirements...
✓ CPU cores: 4 (minimum 2 required)
✓ RAM: 8GB (minimum 4GB required)
✓ Docker: 20.10.x (supported version)

Loading Docker images...
✓ logzilla-runtime:v6.x.y
✓ logzilla-front:v6.x.y
✓ postgres:15.2-alpine
✓ redis:6.2.6-alpine

Starting LogZilla installation...
LogZilla successfully installed!
```

Complete Upgrade Example

This example demonstrates a complete offline upgrade from version 6.28.0 to 6.31.8 in an air-gapped environment.

Pre-Upgrade Status

Check Current Version:

```
root@logzilla-server:/tmp$ logzilla version
v6.28.0
```

Verify Air-Gapped Environment:

```
root@logzilla-server:~$ ping 8.8.8.8
ping: connect: Network is unreachable
```

Package Extraction

Extract Offline Package:

```
root@logzilla-server:~$ cd /tmp
root@logzilla-server:/tmp$ tar xzvf logzilla-v6.31.8.tar.gz
logzilla-v6.31.8/
logzilla-v6.31.8/kickstart.sh
logzilla-v6.31.8/library-influxdb:1.8.10-alpine.tar.gz
logzilla-v6.31.8/library-postgres:15.2-alpine.tar.gz
logzilla-v6.31.8/library-redis:6.2.6-alpine.tar.gz
logzilla-v6.31.8/library-telegraf:1.20.4-alpine.tar.gz
logzilla-v6.31.8/logzilla-etcd:v3.5.7.tar.gz
logzilla-v6.31.8/logzilla-front:v6.31.8.tar.gz
logzilla-v6.31.8/logzilla-mailer:v6.31.8.tar.gz
logzilla-v6.31.8/logzilla-runtime:v6.31.8.tar.gz
logzilla-v6.31.8/logzilla-sec:v6.31.8.tar.gz
logzilla-v6.31.8/logzilla-syslogng:v6.31.8.tar.gz
```

Upgrade Execution

Run Upgrade Command:

```
root@logzilla-server:/tmp$ logzilla upgrade --offline-dir /tmp/logzilla-v6.31.8
```

Upgrade Process Output:

```
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/library-influxdb:1.8.10-alpine.tar.gz
...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/library-postgres:15.2-alpine.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/library-redis:6.2.6-alpine.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/library-telegraf:1.20.4-alpine.tar.gz
...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-etcd:v3.5.7.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-front:v6.31.8.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-mailer:v6.31.8.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-runtime:v6.31.8.tar.gz ...
lz.manager INFO     Assuming version v6.31.8
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-sec:v6.31.8.tar.gz ...
lz.manager INFO     Loading /tmp/logzilla-v6.31.8/logzilla-syslogng:v6.31.8.tar.gz ...

Starting LogZilla upgrade to 'v6.31.8'
lz.setup   INFO     Setup init
lz.docker  INFO     Decommission: queryupdatemodule, front
lz.docker  INFO     Decommission: httpreceiver, celerybeat, queryeventsmodule-1
lz.docker  INFO     Decommission: triggersactionmodule, gunicorn, aggregatesmodule-1
lz.docker  INFO     Decommission: storagemodule-1
lz.docker  INFO     Decommission: logcollector, telegraf, tornado, mailer
lz.docker  INFO     Decommission: syslog
lz.docker  INFO     Decommission: postgres
lz.docker  INFO     Decommission: redis, influxdb
lz.docker  INFO     Decommission: etcd
lz.docker  INFO     Start: etcd
lz.docker  INFO     Start: influxdb, redis
lz.docker  INFO     Start: postgres
lz.containers.postgres INFO     Running postgres v15 migration ...
lz.containers.postgres INFO     Postgres v15 migration finished successfully

Operations to perform:
  Apply all migrations: admin, api, auth, contenttypes, django_celery_beat, sessions
Running migrations:
  No migrations to apply.

lz.setup   INFO     Update group permissions
lz.setup   INFO     Update internal triggers
lz.docker  INFO     Start: syslog
lz.docker  INFO     Start: logcollector, tornado, telegraf, mailer
lz.docker  INFO     Start: storagemodule-1
lz.docker  INFO     Start: triggersactionmodule, celeryworker, dictionarymodule
lz.docker  INFO     Start: celerybeat, httpreceiver, queryeventsmodule-1
lz.docker  INFO     Start: queryupdatemodule, front
lz.docker  INFO     Start: watcher

LogZilla successfully upgraded to 'v6.31.8'
```

Post-Upgrade

Verify New Version:

```
root@logzilla-server:/tmp$ logzilla version
v6.31.8
```

Test Log Ingestion:

```
root@logzilla-server:/tmp$ logger "Test message after upgrade"
```

Then log into the LogZilla web interface and search for `Test message after upgrade` to verify log ingestion is working correctly.

Troubleshooting

Common Issues

Package Extraction Fails:

```
# Check file integrity
ls -la logzilla-v6.x.y.tar.gz

# Verify sufficient disk space
df -h /tmp

# Re-extract with verbose output
tar -xzvf logzilla-v6.x.y.tar.gz
```

Docker Image Loading Issues:

```
# Check Docker service status
systemctl status docker

# Verify Docker disk space
docker system df
```

Upgrade Command Fails:

```
# Verify offline directory path
ls -la /tmp/logzilla-v6.x.y/

# Check system resources
free -h && df -h
```

Post-Upgrade Issues:

```
# Check service logs
logzilla logs --tail 50

# Restart services if needed
logzilla restart
```

Custom Dns

Override Docker DNS resolvers and host mappings for LogZilla containers via /etc/docker/daemon.json, enabling custom internal name resolution

Specifying custom DNS servers

Configure Docker's DNS so all LogZilla containers use the desired resolvers.

Create or edit `/etc/docker/daemon.json` on the host.

Add DNS settings (JSON):

```
{
"dns": ["1.2.3.4", "5.6.7.8"],
"dns-search": ["example.com"]
}
```

Notes:

- Replace the IPs and domain with values for the environment.
- Keep the file valid JSON; merge with existing keys if present.
- Restarting Docker will interrupt running containers. Plan a short maintenance window.

Restart Docker:

```
sudo systemctl restart docker
```

Custom host mappings (without changing DNS)

When DNS reverse lookups are unavailable or specific overrides are needed, configure static host mappings for LogZilla containers using `/etc/logzilla/hosts.in` (loaded into container `/etc/hosts`).

Create or edit `/etc/logzilla/hosts.in` on the host. Format matches `/etc/hosts` (multiple names per line allowed):

```
1.2.3.4 foo.bar.baz
2.3.4.5 baz.lab.com
10.11.12.13 somedevice somedevice.foo.com
```

Restart affected container(s) to apply the mapping. For syslog:

```
logzilla restart -c syslog
```

Verify inside the container that the entry exists:

```
docker exec -ti lz_syslog cat /etc/hosts
```

The `hosts.in` file is read by LogZilla during container (re)creation and populates container `extra_hosts`. Update the file and restart relevant containers whenever mappings change.

Docker Containers

Reference for the Docker containers that make up a LogZilla deployment, including syslog, parser, storage, query, API, AI, and auxiliary services

Docker Containers Used by LogZilla

LogZilla operates by means of multiple docker containers handling various facets of its operation. The following are the containers used:

Container Name	Description
lz_celerybeat	Schedules periodic background tasks for the platform.
lz_gunicorn	API server for the LogZilla backend.
lz_celeryworker	Executes background jobs and module orchestration tasks.
lz_parsermodule	Parses, normalizes, and enriches incoming events using rules/apps.
lz_querymodule	Runs searches/analytics, evaluates triggers, and publishes live results.
lz_storagemodule-1	Event storage and indexing service (dedup, archiving, aggregates).
lz_ai	AI Copilot backend; powers AI features when enabled.
lz_watcher	Orchestrates, monitors, and restarts LogZilla containers.
lz_snmptrapd	Receives SNMP traps (UDP 162) and injects them as events.
lz_httpreceiver	HTTP/HTTPS event ingest API (token-protected) into LogZilla.
lz_sec	Simple Event Correlator for stateful, multi-event correlation.
lz_scriptserver	Runs user-defined scripts for trigger actions/correlation.
lz_front	Web frontend serving UI and proxying API/ingest endpoints.
lz_syslog	Syslog-ng receiver for BSD/RFC5424, JSON, RAW, and TLS inputs.
lz_logcollector	Collects and consolidates internal container logs for diagnostics.
lz_mailer	Outbound SMTP service for alerts and test emails.
lz_telegraf	Collects internal metrics and ships them to time-series storage.
lz_qdrant	Vector database backing AI features (embeddings/search).
lz_influxdb	Time-series database for metrics and aggregates.

Container Name	Description
lz_postgres	Primary relational store for users, dashboards, triggers, settings.
lz_redis	In-memory caching and queues for ephemeral data.

Note: AI containers appear only when the AI feature is enabled and configured in the web interface under Settings → System → AI or via the CLI with the `logzilla settings update` command. (`AI_ENABLED` true with `API_KEY` and `MODEL_NAME`). When enabled, the web interface proxies AI under the `/ai` path.

Overview and Architecture Placement

LogZilla uses a set of Docker containers that cooperate to receive, normalize, store, query, and present log data. These containers are grouped by function so the platform can scale pieces independently and restart modules without impacting the entire system.

At a high level:

- Ingestion enters through `lz_syslog` and optional SNMP and HTTP receivers.
- Processing and parsing occur in `lz_parsermodule` , with supporting modules such as `lz_dictionarymodule` and `lz_forwardermodule` for enrichment and forwarding.
- Durable application state is stored in `lz_postgres` ; time-series metrics in `lz_influxdb` .
- Queries and correlation are executed by `lz_querymodule` and `lz_sec` , with storage managed by `lz_storagemodule-1` .
- Results are returned through the API (`lz_gunicorn`) and UI proxy (`lz_front`).
- Background workers and the watcher manage orchestration and periodic tasks.

Container Groups and Roles

Frontend and API

- `lz_front`
  - Web frontend serving UI and proxying API/ingest endpoints.
  - Terminates HTTP/HTTPS and proxies traffic to internal services.
- `lz_gunicorn`
  - API server for the LogZilla backend.
  - REST API service used by the UI and CLI.

Workers and Orchestration

- `lz_celeryworker`
  - Executes background jobs and module orchestration tasks.

- `lz_celerybeat`
  - Schedules periodic background tasks for the platform.
- `lz_watcher`
  - Orchestrates, monitors, and restarts LogZilla containers.

Ingestion and Parsing

- `lz_syslog`
  - Syslog-ng receiver for BSD/RFC5424, JSON, RAW, and TLS inputs.
  - Port defaults are documented in Network Communications. Settings live under `syslogng` in Server Settings.
- `lz_snmptrapd`
  - Receives SNMP traps (UDP 162) and injects them as events.
- `lz_httpreceiver`
  - HTTP/HTTPS event ingest API (token-protected) into LogZilla.
- `lz_parsermodule`
  - Parses, normalizes, and enriches incoming events using rules/apps.

Query, Storage, and Correlation

- `lz_storagemodule-1`
  - Event storage and indexing service (dedup, archiving, aggregates).
- `lz_querymodule`
  - Runs searches/analytics, evaluates triggers, and publishes live results.
- `lz_sec`
  - Simple Event Correlator for stateful, multi-event correlation.
- `lz_scriptserver`
  - Runs user-defined scripts for trigger actions/correlation.

Datastores and Metrics

- `lz_postgres`
  - Primary relational store for users, dashboards, triggers, settings.
- `lz_influxdb`
  - Time-series database for metrics and aggregates.
- `lz_redis`

    - Ephemeral cache and queues (e.g., query results and task coordination).
- `lz_telegraf`
    - Collects internal metrics and ships them to time-series storage.
- `lz_mailer`
    - Outbound SMTP service for alerts and test emails.
- `lz_logcollector`
    - Collects and consolidates internal container logs for diagnostics.

AI (Conditional)

- `lz_ai`
    - AI Copilot backend; powers AI features when enabled.
    - Available only when AI is enabled and configured.
- `lz_qdrant`
    - Vector database backing AI features (embeddings/search).
    - Runs only when AI is enabled. Persistent volume name: `qdrant`.

Enablement: set `AI_ENABLED=true` and configure `API_KEY` and `MODEL_NAME` in the `ai` settings group. When enabled, `lz_front` proxies the AI backend under `/ai`.

Lifecycle and Control

Container lifecycle is managed through the LogZilla CLI (see System Commands for full details). For Docker-specific troubleshooting, check container status:

```
docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Image}}'
```

Logs and Troubleshooting

Primary platform log:

```
sudo tail -f /var/log/logzilla/logzilla.log
```

Per-container logs (examples):

```
docker logs -n 100 lz_front
docker logs -n 100 lz_syslog
docker logs -n 100 lz_gunicorn
```

Quick checks:

- Frontend is reachable on configured HTTP/HTTPS ports.
- Syslog receivers are listening on the configured ports (see Network Communications for defaults and options).
- Postgres and InfluxDB are running and healthy.

If a module configuration has been updated and does not appear to take effect, use a targeted restart for that component.

Data Persistence and Volumes

The platform persists data across container restarts via Docker volumes.

- Postgres stores durable application data.
- Archive data is managed by the storage pipeline (see Data Archiving and Retention for lifecycle and relocation guidance).
- InfluxDB stores time-series metrics.
- Qdrant stores AI vectors when AI is enabled (volume `qdrant`).

Before upgrades or large configuration changes, snapshots and backups are recommended. See Offline Installs and Upgrades.

Networking Model

`lz_front` terminates HTTP/HTTPS and proxies to internal services by container name. Syslog receiver ports are configurable; defaults are documented in Network Communications. Name resolution for containers uses the Docker network and service names. When custom host mappings are needed, use `/etc/logzilla/hosts.in` (see Custom DNS).

Security and Best Practices

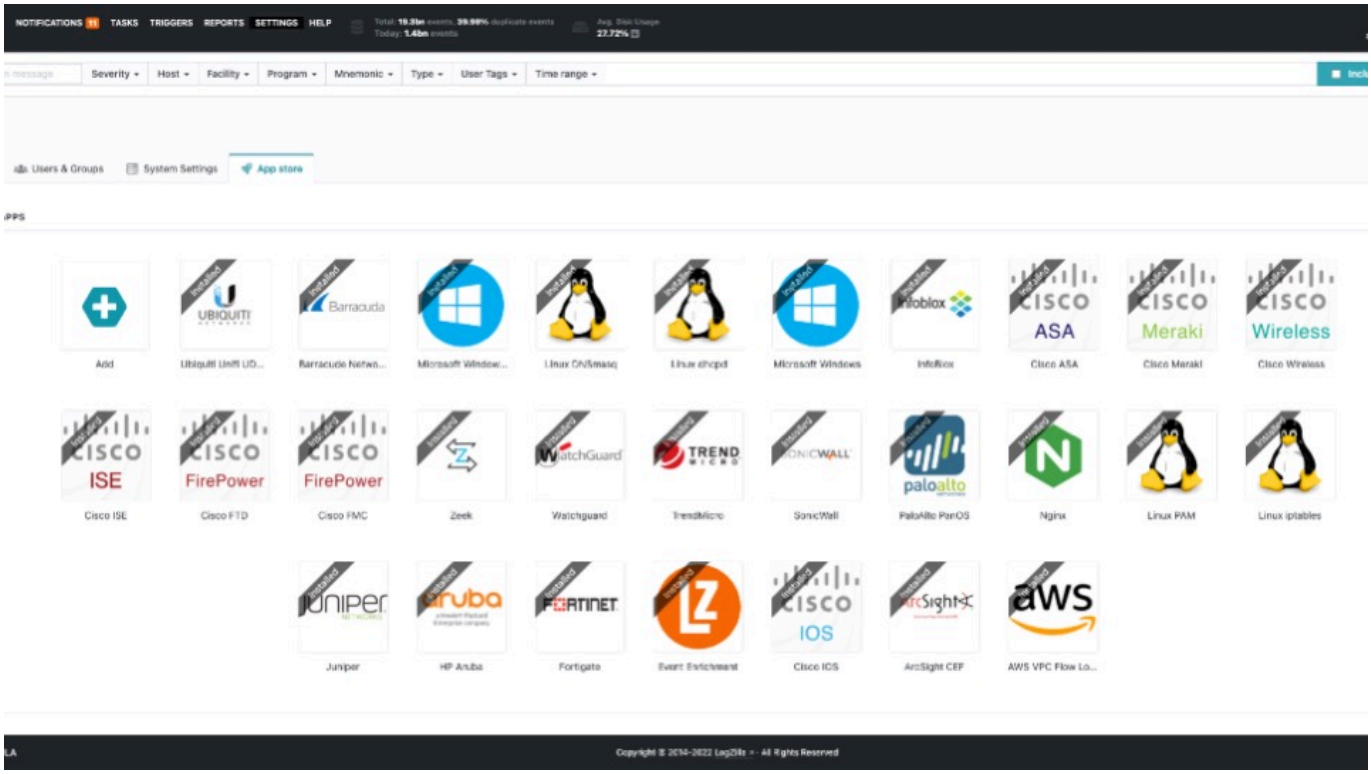
- Use HTTPS in production for the web UI and API. See Using HTTPS.
- Restrict inbound syslog ports to authorized sources using firewalls and network policy.
- Prefer the `logzilla` CLI for lifecycle and configuration changes.
- Take snapshots/backups before upgrades or significant configuration changes.
- Monitor disk utilization for Postgres, archives, and InfluxDB volumes.

Logzilla Apps

LogZilla App Store packages bundle vendor-specific parsing rules, dashboards, and triggers for sources such as Cisco, FortiGate, and Palo Alto

LogZilla Apps

LogZilla Apps are vendor, or technology-specific, packages that extend the platform. Each app typically includes parsing rules, dashboards, and triggers tailored to a data source. After installation, incoming events are correctly parsed and enriched, dashboards render meaningful views, and alerts are ready to use with minimal configuration.



Watch: LogZilla Apps Videos

- Video: LogZilla App Store
  - The App Store video Demonstrates navigating to Settings → Applications, reviewing app details (including the in-app README), and installing or uninstalling apps from the UI.
- Video: LogZilla Apps and Rules
  - The Apps and Rules video Explains how apps package vendor-specific parsing rules, dashboards, and triggers so events are normalized and immediately useful for search, dashboards, and alerts.

LogZilla App Store

LogZilla Apps and Rules

Manage Apps in the UI

The Applications area is available to users with permission to manage settings. Access it under Settings → Applications.

- Installed: Shows apps currently installed on the system.
- Available: Shows apps that can be installed.
- Details: Shows full information for a selected app.

Installed Tab

- Filter by name using the search box.
- Open an app's details to view metadata and, if installed, the option to uninstall.
- Use the "Add" tile to jump to the Available tab.

Available Tab

- Filter by name using the search box.
- Badges indicate whether an app is installed or requires a license.
- Open an app's details to review its README and install.

App Details

- Shows name, version, description, and icon.
- Renders the app's README for guidance and requirements prior to install.
- Provides Install or Uninstall. When a paid app requires a license, a Purchase action is shown instead.
- If an app requires a parent, the UI indicates the missing dependency and disables the install until the parent app is installed.
- If the app deprecates older items (for example, dashboards), the UI shows a confirmation message before proceeding.

How Apps Work (High Level)

Apps are self-contained packages that the system discovers and presents in the UI. An app typically contains:

- Parsing rules to recognize and enrich vendor logs.
- Pre-built dashboards and widgets for fast visibility.
- Triggers for proactive alerting on common conditions.
- Optional configuration files that adjust behavior without code changes.
- An in-app README rendered in the App Details view.

Benefits

- Immediate parsing and enrichment of popular device logs.
- Ready-made dashboards and widgets for common use cases.
- Pre-defined triggers to alert on important events.
- In-app documentation to guide installation and usage.
- Safer installs via parent-dependency checks and deprecation warnings.

Troubleshooting

- If an app does not behave as expected, uninstall it from the Details view and then re-install it.
- Review the primary platform log for errors: `/var/log/logzilla/logzilla.log`. Installation and runtime issues are typically recorded there.

Logzilla On Vmware

Deploy LogZilla on VMWare using the downloadable virtual appliance, and expand LVM-backed storage online by attaching a second virtual disk

LogZilla on VMWare

Users may download a Virtual LogZilla instance for testing or smaller deployments. For larger environments, the same VM can be used with additional resources (CPU, RAM, and storage). The VM is configured to use Linux Logical Volume Manager (LVM) so storage can be expanded online.

Default disk and expansion

- Default VM disk size: 50 GB.
- Expansion method: add a second virtual disk and extend LVM.
- The VM does not need to be powered off to add a new disk.

Add more disk space (without downtime)

Add a new virtual disk in VMWare. Do not grow the original disk; add a second disk instead.

Connect to the server console or SSH as root.

Identify the new disk:

```
fdisk -l | grep -E '/dev/[sv]d'
```

Look for a disk without partitions (likely the new one). Replace `/dev/vdb` below with the correct device name.

Prepare the new disk and create an LVM PV/partition:

```
# Replace with your disk (e.g., /dev/vdb)
disk="/dev/vdb"

# Create a new partition of type 8e (Linux LVM)
printf 'n\n\n\n\n\nt\n8e\np\nw\n' | fdisk -c -u "$disk"

# Inform the OS of partition table changes
partprobe "$disk"

# Initialize the LVM physical volume
pvcreate "${disk}1"
```

Extend the volume group to include the new PV:

```
vg=$(vgdisplay -c | cut -d ':' -f 2 | head -1)
vgextend "$vg" "${disk}1"
```

Identify the logical volume used by root (`/`):

```
lvpath=$(df --output=source / | tail -1)
```

Extend the logical volume and resize the filesystem:

```
lvextend -l+100%FREE "$lvpath"
resize2fs "$lvpath"
```

Note: If the disk is 100% full, `vgextend` may fail. Free disk space and retry.

Verify changes:

```
partprobe
```

After these steps, the volume should report the increased size. No reboot is required.

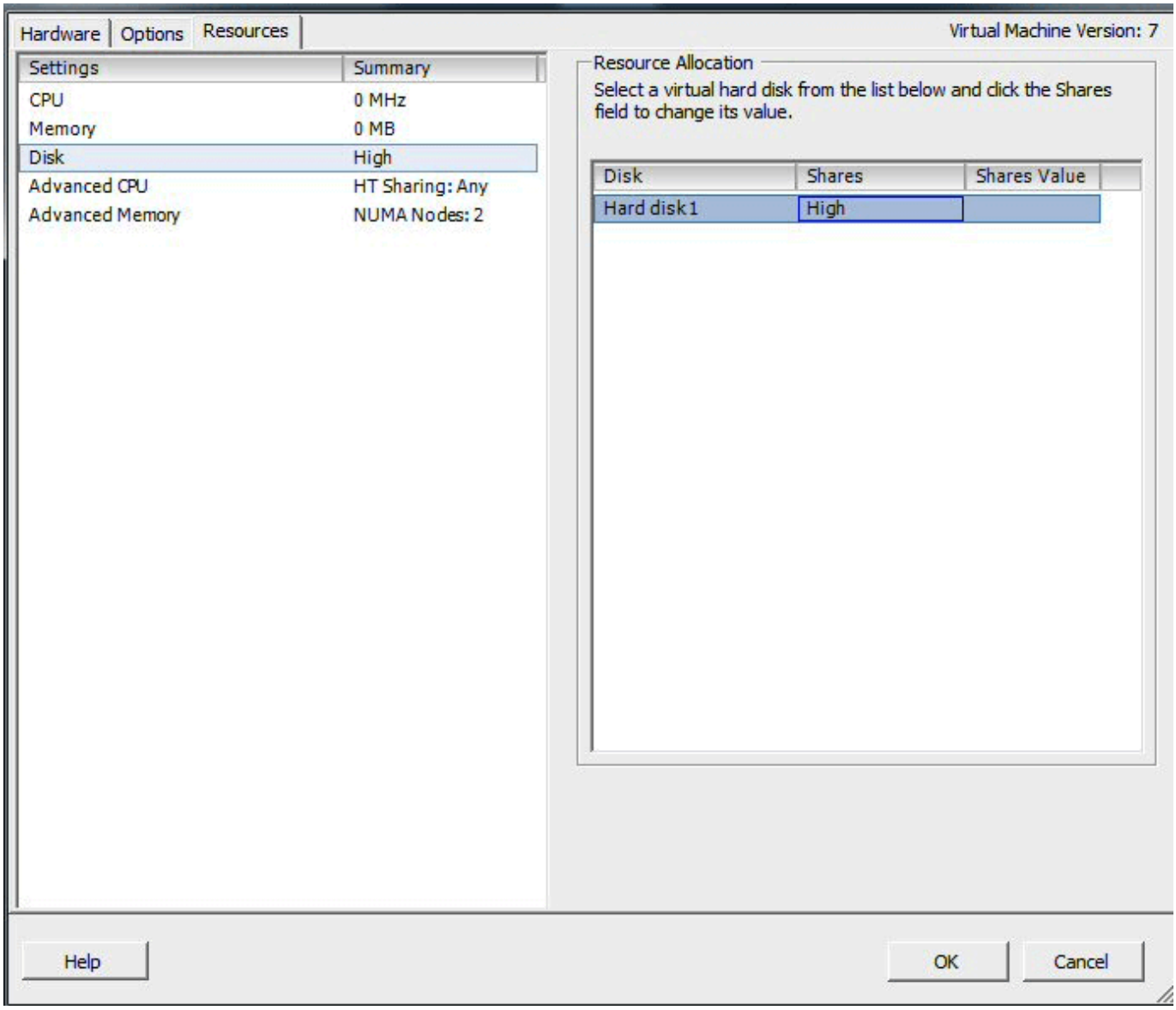
Performance considerations on VMWare

For VMware-based deployments, set the virtual disk resource allocation to High for improved I/O responsiveness.

Right-click the VM and select Edit settings.

Open the Resources tab.

Select Disk, then change the allocation from Normal to High.



VMWare Resource Allocation: Set Disk to High

Note: For large-scale, high-throughput deployments, ensure the team operating VMware is experienced with optimizing disk I/O performance.

Network Tuning

Tune UDP syslog ingest performance on the LogZilla host, including netcat and loggen benchmarking and kernel socket buffer adjustments

Networking performance (UDP)

Steps to test UDP ingest performance and adjust kernel parameters. Commands are examples; ports and values must be verified for the environment.

Testing UDP performance

- Use a non-conflicting high port for testing (for example, 5514) to avoid disrupting the running syslog receiver and to avoid privileged-port requirements.
- Start a UDP listener using netcat:

  ```
  netcat -u -p 5514 -l > /tmp/logs
  ```

  Alternatively (on systems where `nc` is used):

  ```
  nc -u -l 5514 > /tmp/logs
  ```

- In another terminal, generate messages using loggen (part of syslog-ng):

  ```
  ./loggen -r 10000 -D -I 10 127.0.0.1 5514
  ```

- Example output from loggen:

  ```
  average rate = 10877.62 msg/sec, count=108783, time=10.006, msg size=256,
  bandwidth=2719.40 kB/sec
  ```

- Verify line count matches (or closely matches) the loggen reported count:

  ```
  wc -l /tmp/logs
  ```

  Example:

```
108783 /tmp/logs
```

- Check UDP statistics:

```
netstat -su
```

Modern equivalent using `ss`:

```
ss -u -s    # UDP summary
ss -u -a    # all UDP sockets
```

Key fields:

- `packets received`: total packets since last boot.
- `packets to unknown port`: packets sent to a closed port (for example, service stopped while senders keep sending).
- `packet receive errors`: errors while receiving/processing; a single packet can generate multiple errors.

Tuning UDP buffers

If `netstat -su` shows errors, increase UDP buffer sizes incrementally and re-test after each change.

- Increase receive buffer max (example: 32 MB):

```
sysctl -w net.core.rmem_max=33554432
```

The default on many Linux distributions is about 122 KB (`net.core.rmem_default = 124928`).

- Additional tuning options (examples):

```
# UDP memory (in pages), min rmem
# NOTE: values are environment-dependent; increase gradually and test
echo 'net.ipv4.udp_mem = 192576 256768 385152' >> /etc/sysctl.conf
echo 'net.ipv4.udp_rmem_min = 4096' >> /etc/sysctl.conf

# Alternative runtime example for udp_mem
sysctl -w net.ipv4.udp_mem='262144 327680 393216'
```

Optional transmit buffer tuning (device-dependent):

```
sysctl -w net.core.rmem_default=212992
sysctl -w net.core.wmem_default=212992
sysctl -w net.core.wmem_max=33554432
```

net.ipv4.udp_mem values are page-based. With `PAGE_SIZE=4096`, the maximum above equals `385152 * 4096 = 1,577,582,592` bytes.

- Increase the incoming packet queue size:

```
sysctl -w net.core.netdev_max_backlog=2000
```

Persisting settings

Settings changed with `sysctl -w` are not persistent. To make the changes permanent, append to `/etc/sysctl.conf` and reload:

```
# Example: persist rmem_max at 32 MB
echo 'net.core.rmem_max=33554432' >> /etc/sysctl.conf

# Apply changes
sysctl -p
```

Re-run tests after each change to validate impact and adjust values as needed.

Alternatively, create a drop-in file and reload all settings:

```
cat >/etc/sysctl.d/99-logzilla-tuning.conf <<'EOF'
net.core.rmem_max=33554432
EOF

sysctl --system
```

Advanced NIC tuning (optional)

These options can improve performance on multi-queue NICs in high-throughput environments. Hardware and kernel support vary; test changes incrementally in staging before applying to production.

Multi-queue visibility

```
ls -d /sys/class/net/eth0/queues/rx-* 2>/dev/null || true
ls -d /sys/class/net/eth0/queues/tx-* 2>/dev/null || true
```

Receive Packet Steering (RPS) and Receive Flow Steering (RFS)

View and set global flow entries (example values):

```
cat /proc/sys/net/core/rps_sock_flow_entries
echo 32768 > /proc/sys/net/core/rps_sock_flow_entries
```

Distribute per-queue flow counts and CPUs (example; adjust mask to available CPUs, for example `f` = CPUs 0-3):

```
for q in /sys/class/net/eth0/queues/rx-*/rps_flow_cnt; do echo 4096 > "$q"; done
for q in /sys/class/net/eth0/queues/rx-*/rps_cpus; do echo f > "$q"; done
```

Transmit Packet Steering (XPS)

Pin transmit queues to CPUs (example; adjust CPU mask):

```
for q in /sys/class/net/eth0/queues/tx-*/xps_cpus; do echo f > "$q"; done
```

Note: RPS/RFS/XPS settings reset on reboot. Persist via udev rules or a systemd unit executed at boot. Not all systems support `xps_cpus`, in which case it will indicate `write error: No such file or directory`.

NIC ring buffers

Inspect and (if supported) increase ring sizes cautiously:

```
ethtool -g eth0          # query
ethtool -G eth0 rx 4096 tx 4096   # set (hardware/driver dependent)
```

NIC offload features

View current offload settings:

```
ethtool -k eth0
```

Only disable specific offloads during targeted troubleshooting. Defaults are generally optimal.

Interrupt mapping visibility

```
grep -E 'eth0' /proc/interrupts
```

Most distributions run `irqbalance` to spread IRQ load automatically. Manual pinning is advanced and environment-specific.

System Tuning

Tune CPU frequency governors, SpeedStep, and TurboBoost behavior on LogZilla hosts for consistent throughput on high-volume log processing workloads

CPU frequency governors

Recent Intel CPUs provide energy-saving and boost capabilities (SpeedStep and TurboBoost). These features adjust core frequencies based on load, which may not be desirable for servers focused on consistent performance.

Checking current processor speed

```
cat /proc/cpuinfo | grep MHz
```

Example output:

```
cpu MHz   : 1400.000
cpu MHz   : 1400.000
cpu MHz   : 1400.000
cpu MHz   : 1400.000
cpu MHz   : 1400.000
cpu MHz   : 1400.000
cpu MHz   : 3500.000
cpu MHz   : 3500.000
```

Only some cores may run at top speed with default settings, which can be undesirable for throughput-focused workloads.

Set governor temporarily (root)

Add this helper function in a root shell or in `.bash_aliases`:

```
function setgov ()
{
    # usage:
    # setgov ondemand
    # setgov performance
    echo "Current setting: $(cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor |
sort -u)"
    echo "Current CPU Speeds:"
    cat /proc/cpuinfo | grep 'cpu MHz'
    [[ -z $1 ]] && { echo "Missing argument (ondemand|performance)"; return 1; }
    echo "$1" | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
    echo "New CPU Speeds:"
    cat /proc/cpuinfo | grep 'cpu MHz'
}
```

Load it with `source ~/.bash_aliases`, then run `setgov performance`.

Permanent change (root)

```
apt-get install -y --no-install-recommends cpufrequtils
echo 'GOVERNOR="performance"' >/etc/default/cpufrequtils
systemctl enable --now cpufrequtils
```

Note that this will not work in all environments.

Alternative methods (if available)

If the following tools are installed on the distribution, they can be used to set performance profiles:

```
# Using cpupower (part of kernel tools)
cpupower frequency-set -g performance

# Using tuned (RHEL/Rocky/Alma families)
tuned-adm profile throughput-performance
```

Notes:

- TurboBoost only runs when other cores are throttled due to TDP. With the `performance` governor, cores run at nominal frequency.
- TurboBoost depends on SpeedStep; disabling SpeedStep in BIOS disables boosting and throttling.

Filesystem performance

These recommendations affect disk layout and virtual memory. Verify target disk names and understand that some commands are destructive.

Warning: Commands like `parted`, `mkfs`, and `lvcreate` can destroy data if run against the wrong device. Ensure device names are correct before proceeding.

Disk format (LVM recommended)

OS disks should use Logical Volume Manager (LVM). Align sectors properly.

Identify the target disk and existing layout before partitioning:

```
lsblk -o NAME,SIZE,TYPE,FSTYPE,MOUNTPOINT
```

Format disk using `parted` (example uses `/dev/sda`). The following commands are run inside the parted prompt:

```
disk=/dev/sda
parted -a optimal ${disk}
mklabel gpt
unit s
mkpart primary 2048s 100%
align-check opt 1
set 1 lvm on
print
```

Create the physical volume:

```
pvcreate -M 2 --dataalignment 4k ${disk}${partition}
```

Check alignment (first PE should be `1.00m`):

```
pvs -o +pe_start
```

Create the volume group (if not already present):

```
volumeName="vg0"
partition=1 # partition created above with parted
vgcreate ${volumeName} ${disk}${partition}
```

Create a logical volume using remaining free space (named `data`):

```
lvcreate -n data -l 100%FREE ${volumeName}
```

Create a filesystem on the new logical volume:

```
# CAUTION: The next command will destroy data on the target device.
mkfs.ext4 /dev/${volumeName}/data
```

Create an fstab entry (replace the mount point as appropriate):

```
/dev/${volumeName}/data    /mnt/data    ext4    defaults    0    2
```

During a new OS install, installers typically configure LVM automatically. LVM allows adding disks and resizing volumes online.

Swap

For server deployments, avoid swap unless required for a temporary mitigation. If swap must be enabled, tune the settings as follows.

View current swappiness:

```
cat /proc/sys/vm/swappiness
```

Set swappiness temporarily (example: 10):

```
sysctl vm.swappiness=10
```

Persist swappiness across reboots by adding to `/etc/sysctl.conf` and reloading:

```
echo 'vm.swappiness=10' >> /etc/sysctl.conf
sysctl -p
```

Alternatively, create a drop-in file and reload all settings:

```
cat >/etc/sysctl.d/99-logzilla-vm.conf <<'EOF'
vm.swappiness=10
EOF

sysctl --system
```

vfs_cache_pressure

This controls how aggressively the kernel reclaims cached inode and dentry information.

View the current value:

```
cat /proc/sys/vm/vfs_cache_pressure
```

Set a more conservative value (example: 50):

```
sysctl vm.vfs_cache_pressure=50
```

Make it persistent:

```
echo 'vm.vfs_cache_pressure=50' >> /etc/sysctl.conf
sysctl -p
```

Alternatively, persist with a drop-in file (can reuse the same file as above):

```
echo 'vm.vfs_cache_pressure=50' >> /etc/sysctl.d/99-logzilla-vm.conf
sysctl --system
```

Server Migration

Migrate a LogZilla installation to a new server using the migrate-dump CLI, including version compatibility rules and post-restore license updates

Migrating LogZilla to a new server involves transferring all data and configuration using the built-in `migrate-dump` command. This modern approach provides a safe, reliable method for moving LogZilla installations between servers while maintaining data integrity.

Migration Overview

The migration process uses LogZilla's dedicated `migrate-dump` command:

Create dump: Generate a complete backup on the source server

Transfer dump: Move the backup files to the destination server

Restore dump: Import the backup on the destination server

Update license: Configure licensing for the new server

Version Compatibility

The `migrate-dump` command has specific version requirements:

- **Same versions**: Works without restrictions
- **Newer dump version**: Will not work (dump version > destination version)
- **Older dump version**: Requires `--force` option (dump version < destination version)

Important: Ensure version compatibility before starting the migration process. The safest approach is using identical LogZilla versions on both servers.

Source Server Migration

Step 1: Create Migration Dump

Create a complete backup of the LogZilla installation using the migrate-dump command. Specify a directory to store the dump files:

```
logzilla migrate-dump create --dump-dir /tmp/logzilla-migration
```

This command creates a full backup including:

- All log data and archives

- Database contents (PostgreSQL, InfluxDB, Redis, etcd)
- Configuration settings
- User accounts and permissions

Note: Ensure sufficient disk space is available in the dump directory. The backup size depends on the amount of stored log data.

Step 2: Transfer Migration Files

Copy the migration dump to the destination server using scp, rsync, or another secure transfer method:

```
scp -r /tmp/logzilla-migration user@destination-server:/tmp/
```

Alternatively, use rsync for large transfers:

```
rsync -avz /tmp/logzilla-migration/ user@destination-server:/tmp/logzilla-migration/
```

Destination Server Setup

Step 1: Install LogZilla

Install LogZilla on the destination server. Ensure the version matches or is newer than the source server version:

```
logzilla install
```

Step 2: Restore Migration Dump

Restore the complete LogZilla installation from the migration dump:

```
logzilla migrate-dump restore --dump-dir /tmp/logzilla-migration
```

For version compatibility issues, use the force option if the dump version is older than the destination version:

```
logzilla migrate-dump restore --dump-dir /tmp/logzilla-migration --force
```

The restore process automatically:

- Stops LogZilla services if running
- Restores all data and configuration
- Updates internal references and paths

- Starts LogZilla services after completion

Step 3: Update License

After successful restoration, update the license for the new server:

```
logzilla license download
```

This command replaces the migrated license with a demo license valid for the new server. Contact LogZilla support to extend the license as needed.

Post-Migration Verification

Monitor Startup Progress

LogZilla initialization time depends on data volume. Monitor startup progress using the API endpoint:

```
http://your-server.com/api/monitor
```

Check System Logs

Monitor LogZilla logs for any errors during startup:

```
tail -f /var/log/logzilla/logzilla.log -n 40
```

Verify Data Integrity

Once LogZilla starts:

- Confirm historical data appears in the interface
- Verify user accounts and permissions
- Test log ingestion from sources
- Validate dashboard and trigger configurations

Troubleshooting

Common Issues

Migration Dump Creation Fails

- Verify sufficient disk space in the dump directory
- Check LogZilla service status before creating dump
- Review system logs for specific error messages

- Ensure proper file permissions on dump directory

Version Compatibility Errors

- Confirm LogZilla versions on source and destination servers
- Use `--force` option for older dump versions (with caution)
- Consider upgrading destination server if dump version is newer

Restore Process Fails

- Verify dump directory exists and contains valid migration files
- Check destination server has sufficient disk space
- Ensure network connectivity during restore process
- Review restore logs for specific error details

License Issues

- Ensure `logzilla license download` completed successfully
- Contact support if demo license expires before production license
- Verify network connectivity for license server communication

Legacy Migration Method

For older LogZilla versions without `migrate-dump` support, the manual Docker volume method may be required. Contact LogZilla support for guidance on legacy migration procedures.

Best Practices

- **Test migrations** in a staging environment first
- **Verify version compatibility** before starting migration
- **Schedule downtime** during low-usage periods
- **Monitor disk space** on both source and destination servers
- **Validate migration** thoroughly before decommissioning source server
- **Coordinate with teams** that depend on LogZilla services

Syslog Settings

Configure the LogZilla syslog-ng listener ports, buffering, debug logging, TLS, and PCI-compliant capture from the Syslog Daemon settings pane

Syslog Settings

The Syslog Daemon settings page configures the built-in syslog-ng listeners: ports, buffering, debug logging, and PCI-compliant capture.

For device or application setup, see Receiving Data. For advanced pipeline customization using `config.yaml` and `conf.d/`, see Syslog pipeline customization.

Location in UI

- Settings → System Settings → Syslog Daemon

Listener ports

The following fields appear in the Syslog Daemon settings pane. Enter a port number to enable the listener, or leave `0` to disable it.

Field (UI label)	Protocol
Syslog Bsd Tcp Port	BSD RFC3164 TCP
Syslog Bsd Udp Port	BSD RFC3164 UDP
Syslog Rfc5424 Port	RFC5424 TCP
Syslog Json Port	JSON over TCP
Syslog Raw Tcp Port	Raw TCP
Syslog Raw Udp Port	Raw UDP
Syslog Tls Port	TLS

Application Ports

LogZilla provides dedicated syslog ports for vendors whose log format requires separate handling. These are configured under **Settings → System Settings → Application Ports**. Enter a port number in the matching field to enable reception, or leave it set to `0` to disable it.

Setting	Covers
Syslog Arista Eos Port	Arista EOS switches
Syslog Checkpoint Port	Check Point firewalls
Syslog Datapower Port	IBM DataPower gateways
Syslog Dell N Series Port	Dell N-Series switches
Syslog Fireeye Port	FireEye appliances
Syslog Infoblox Port	Infoblox NIOS (DNS, DHCP, audit, ADP)
Syslog Meraki Port	Cisco Meraki devices
Syslog Paloalto Port	Palo Alto firewalls
Syslog Paloalto Sdwan Ion Port	Palo Alto Prisma SD-WAN ION appliances
Syslog Symantec Port	Symantec Endpoint Protection
Syslog Unifi Port	Ubiquiti UniFi devices
Syslog Vmware Port	VMware ESXi and vCenter

Notes:

- TLS requires valid cert and key files. The paths are set in the `Syslog Tls Cert File` and `Syslog Tls Key File` fields on the same settings page.

Flow control and buffering

The following fields control queue behavior and buffer sizing:

- Syslog Max Connections
- Syslog Flow Control
- Syslog Disk Buffer
- Syslog Buffer Reliable
- Syslog Memory Buffer Size Mb
- Syslog Memory Buffer Length
- Syslog Disk Buffer Size Mb

Destination batching

- Syslog Destination Workers
- Syslog Destination Batch Lines
- Syslog Destination Batch Timeout

Debug logging

- Syslog Debug - writes to `/var/log/logzilla/syslog/debug.log`
- Syslog Debug Json - writes to `/var/log/logzilla/syslog/debug-json.log`

Important:

- Disable debug toggles after troubleshooting to reduce load and disk usage.

PCI compliant logs

- Pci Compliant Logs - writes to `/var/log/logzilla/pci-compliant/`

Extra log rules (advanced)

- Syslog Extra Log Rules - injects directives into the main `log {}` section. Use carefully. For full customization, see Syslog pipeline customization.

Managing these settings from the command line

The same fields are available via the CLI. Use `logzilla config list` and `logzilla config update` rather than `logzilla settings list` / `logzilla settings update`. The CLI uses the underlying field names in `UPPER_SNAKE_CASE`:

UI field	CLI key
Syslog Bsd Udp Port	SYSLOG_BSD_UDP_PORT
Syslog Bsd Tcp Port	SYSLOG_BSD_TCP_PORT
Syslog Rfc5424 Port	SYSLOG_RFC5424_PORT
Syslog Json Port	SYSLOG_JSON_PORT
Syslog Raw Tcp Port	SYSLOG_RAW_PORT
Syslog Raw Udp Port	SYSLOG_RAW_UDP_PORT
Syslog Tls Port	SYSLOG_TLS_PORT
Syslog Tls Cert File	SYSLOG_TLS_CERT_FILE
Syslog Tls Key File	SYSLOG_TLS_KEY_FILE
Syslog Max Connections	SYSLOG_MAX_CONNECTIONS
Syslog Flow Control	SYSLOG_FLOW_CONTROL
Syslog Disk Buffer	SYSLOG_DISK_BUFFER
Syslog Buffer Reliable	SYSLOG_BUFFER_RELIABLE

UI field	CLI key
Syslog Memory Buffer Size Mb	SYSLOG_MEMORY_BUFFER_SIZE_MB
Syslog Memory Buffer Length	SYSLOG_MEMORY_BUFFER_LENGTH
Syslog Disk Buffer Size Mb	SYSLOG_DISK_BUFFER_SIZE_MB
Syslog Destination Workers	SYSLOG_DESTINATION_WORKERS
Syslog Destination Batch Lines	SYSLOG_DESTINATION_BATCH_LINES
Syslog Destination Batch Timeout	SYSLOG_DESTINATION_BATCH_TIMEOUT
Syslog Debug	SYSLOG_DEBUG
Syslog Debug Json	SYSLOG_DEBUG_JSON
Pci Compliant Logs	PCI_COMPLIANT_LOGS
Syslog Extra Log Rules	SYSLOG_EXTRA_LOG_RULES

Related topics

- Syslog troubleshooting
- Syslog basics

Syslog Pipeline Customization

Extend the LogZilla syslog-ng pipeline via config.yaml, conf.d includes, SYSLOG_EXTRA_LOG_RULES, and custom sources for specialized transports

Syslog pipeline customization (advanced)

Administrators who need to customize the syslog-ng pipeline beyond what is available in the UI can use `config.yaml`, the `conf.d/` include directory, dedicated sources, and safe ways to inject extra rules into the main pipeline.

For common changes such as ports, batching, buffering, and debug toggles, use Syslog Settings.

Configuration locations

- `/etc/logzilla/syslog-ng/config.yaml`
  - Primary YAML used to render the syslog-ng configuration.
- `/etc/logzilla/syslog-ng/conf.d/`
  - Additional `*.conf` files included by the main template.

Important:

- Avoid creating custom top-level `log` statements. Use `SYSLOG_EXTRA_LOG_RULES` or the patterns shown below to keep the main pipeline intact.

When to customize

- Add a dedicated listener or transport not covered by UI.
- Tag a source via `source_type` for specialized parsing.
- Insert targeted filters/rewrites into the main pipeline.
- Create raw TCP/UDP inputs for non-syslog data.

Sources (standard)

Standard sources are generated from settings and usually require no changes:

- `bsd` –- TCP on `SYSLOG_BSD_TCP_PORT` (BSD syslog)
- `bsd_udp` –- UDP on `SYSLOG_BSD_UDP_PORT` (BSD syslog)
- `rfc5424` –- TCP on `SYSLOG_RFC5424_PORT` (RFC 5424 syslog)
- `json` –- TCP on `SYSLOG_JSON_PORT` (newline JSON)
- `tls` –- TCP on `SYSLOG_TLS_PORT` (TLS RFC 5424)
- `raw` –- TCP on `SYSLOG_RAW_PORT` (no parsing)

- `raw_udp` —- UDP on `SYSLOG_RAW_UDP_PORT` (no parsing)

Dedicated sources with `source_type`

To target a subset of events for specific parsing rules:

Set `source_type` on the syslog-ng source in `config.yaml`.

In the relevant Lua rule, set `SOURCE_FILTER = "<tag>"`.

Ensure the tag is included in Parser settings as required.

Only events with the matching `source_type` are processed by rules that declare that tag.

Example: add a TLS source with a dedicated tag

```
sources:
  - name: tls_west
    enabled: true
    type: network
    port: 6514
    transport: tls
    tls_cert_file: /etc/ssl/logzilla/server.crt
    tls_key_file: /etc/ssl/logzilla/server.key
    flags: ["syslog-protocol"]
    program_override: "tls-wf"
    extra_fields:
      site: "west-dc"
    source_type: "west"
```

If rules declare `SOURCE_FILTER = "west"`, ensure the tag is permitted in Parser settings.

Example: add a raw UDP source for unparsed logs

```
sources:
  - name: raw_udp_1516
    enabled: true
    type: network
    port: 1516
    transport: udp
    flags: ["no-parse"]
    program_override: "raw-udp"
    extra_fields:
      log_type: "raw"
    source_type: "devices"
```

Injecting extra log rules (advanced)

Use `SYSLOG_EXTRA_LOG_RULES` in settings or set `extra_log_rules` in `config.yaml` to inject statements into the main `log {}`. Keep injected logic minimal and well-scoped.

```
extra_log_rules: "filter(f_only_host);"
```

When more complex filters are needed, place reusable building blocks in `conf.d/` and reference them from `extra_log_rules`.

Restarting after changes

```
logzilla restart -c syslog
```

Related topics

- Syslog Settings
- Syslog troubleshooting
- Syslog basics

Http Receiver Settings

LogZilla HTTP Receiver module settings for the /incoming endpoint, covering access tokens, authentication headers, and interactive API docs

HTTP Receiver Settings

HTTP Receiver module settings are available in the web interface. UI fields map to underlying settings and behaviors.

For how to send events over HTTP and verify ingestion, see HTTP Event Receiver. For relay-side (sender) configuration, see Syslog Relays.

Where HTTP events are received

- External clients send to the configured HTTP/HTTPS port on the LogZilla host at path `/incoming`.
- Internally, the HTTP Receiver serves endpoints under that path. The reverse proxy forwards `/incoming` to the receiver.
- Interactive API docs are provided at `/incoming/docs`.
- For endpoint reference and the OpenAPI schema, see HTTP Event Receiver.

Authentication (access tokens)

The HTTP Receiver accepts any of the following:

- `Authorization: token <YOUR_TOKEN>` header
- `X-LZ-Access-Key: <YOUR_TOKEN>` header
- `AUTHTOKEN=<YOUR_TOKEN>` query parameter

Tokens can be created and managed in the Authtokens UI or CLI. The receiver accepts both permanent tokens and dynamically fetched tokens from LogZilla's internal API.

Settings

- HTTP Receiver Workers Number → `HTTP_RECEIVER_WORKERS_NUM`
  - Number of worker processes the receiver runs.
- HTTP Receiver Show Access Logs → `HTTP_RECEIVER_SHOW_ACCESS_LOGS`
  - Enables HTTP access logging for the receiver.
- HTTP Receiver Log Invalid Request → `HTTP_RECEIVER_LOG_INVALID_REQUEST`
  - When an ingest request is invalid (for example, a 422 response), logs a short diagnostic dump to aid troubleshooting. Disable in high-volume environments if logging is too chatty.

Notes

- Listener ports are set in `Front` settings (HTTP/HTTPS). The HTTP Receiver does not have its own public port; it is exposed at `/incoming` on the UI/API port.
- If configuration changes do not take effect automatically, restart the module:

```
logzilla restart -c httpreceiver
```

Related topics

- HTTP Event Receiver
- Syslog Relays

Syslog Troubleshooting

Diagnose syslog reception issues on a LogZilla host using logzilla.log inspection, tcpdump packet capture, and syslog-ng debug output

Syslog Troubleshooting

Practical checks and tools for diagnosing syslog reception issues in LogZilla. The focus is on verifying that events arrive at the host, that syslog-ng is processing them, and that debug outputs are available.

Check the LogZilla log

Use the primary LogZilla log to look for errors and pipeline state:

```
sudo tail -f /var/log/logzilla/logzilla.log
```

Verify that the source is sending

Confirm that traffic from the sender reaches the LogZilla server. Adjust the port and interface as needed for the environment.

```
# Listen on the default gateway interface for UDP 514 (RFC 3164)
sudo tcpdump -vvv -i $(awk '$2 == 00000000 { print $1 }' /proc/net/route) \
  udp port 514
```

If a different interface is required:

```
sudo tcpdump -vvv -i p1p1 udp port 514
```

Enable syslog-ng debug mode (advanced)

After confirming traffic arrives, enable syslog-ng debugging in the `lz_syslog` container to inspect parsing behavior.

```
sudo docker exec -it lz_syslog bash -c 'syslog-ng-ctl debug --set=on'
sudo docker logs lz_syslog --tail 100 -f
```

Disable debug mode after checking the output:

```
sudo docker exec -it lz_syslog bash -c 'syslog-ng-ctl debug --set=off'
```

Syslog debugging controls

Use the UI (recommended)

Troubleshooting toggles are available in the web interface:

Open `Settings` → `System Settings` → `Syslog Daemon`.

Toggle `Syslog Debug` to `On` to write to `/var/log/logzilla/syslog/debug.log`.

Toggle `Syslog Debug Json` to `On` to write to `/var/log/logzilla/syslog/debug-json.log`.

Disable these toggles after troubleshooting to avoid extra load and disk usage.

View logs using:

```
sudo tail -F /var/log/logzilla/syslog/debug.log
sudo tail -F /var/log/logzilla/syslog/debug-json.log
```

Use the console (fallback)

When the UI is unavailable, the same settings can be changed via the CLI:

```
sudo logzilla settings update SYSLOG_DEBUG=true
sudo logzilla settings update SYSLOG_DEBUG_JSON=true
```

Disable after troubleshooting:

```
sudo logzilla settings update SYSLOG_DEBUG=false
sudo logzilla settings update SYSLOG_DEBUG_JSON=false
```

Generate local test messages

Use `logger` to generate local test events and verify they appear in LogZilla:

```
sudo logger -T -P 514 --rfc3164 -n localhost -p local0.emerg \
  -t "test" "rfc3164 event test on TCP Port 514 from $(hostname)"

sudo logger -d -P 514 --rfc3164 -n localhost -p local0.emerg \
  -t "test" "rfc3164 event test on UDP Port 514 from $(hostname)"
```

Raw capture for support

If events still do not appear, capture a sample for support analysis. Adjust duration (`-G`) based on volume and reproduction window.

```
# Example: capture 3 hours
# 86400 would be 24 hours
# Include fragmented packets and compress the capture

sudo tcpdump -i $(awk '$2 == 00000000 { print $1 }' /proc/net/route) \
  "udp port 514 or (ip[6:2] & 0x1fff) != 0" \
  -nnvvXSs 0 -G 10800 -W 1 -z gzip -w /tmp/$(hostname).pcap
```

Include LogZilla version information in support tickets:

```
sudo logzilla version
```

Syslog Relays

Configure syslog-ng and rsyslog relays to forward events to a LogZilla server over TCP, UDP, TLS, or HTTP, including token-authenticated HTTPS

Syslog Relays

Syslog relays (sender-side systems) forward logs to LogZilla using syslog-ng or rsyslog over TCP/UDP, TLS, or HTTP/HTTPS. Receiver-side configuration appears in HTTP Receiver Settings.

Architecture overview

```
remote site/devices  ->  relay (syslog-ng/rsyslog)  ->  LogZilla server
                                  |                     (syslog or HTTP)
                                  +-- TLS/JSON/filters
```

Prerequisites

- LogZilla server host and port information.
- For HTTP/HTTPS: an access token and the `/incoming` path on the LogZilla server.
- Optional TLS materials for syslog TLS (certificates/keys) when using port 6514.

syslog-ng (TCP/UDP)

The following example receives on TCP/UDP 514 and forwards to LogZilla over TCP 514. Adjust ports and transports as needed.

```
# filename: /etc/syslog-ng/conf.d/logzilla-relay.conf

options {
  flush_lines(100);
  threaded(yes);
  use_dns(yes);
  use_fqdn(no);
  keep_hostname(yes);
  dns-cache-size(2000);
  dns-cache-expire(87600);
};

source s_network {
  network(transport("tcp") port(514));
  network(transport("udp") so_rcvbuf(1048576) port(514));
};

destination d_logzilla {
  network("<LOGZILLA_HOST>" port(514) transport(tcp));
};

log {
  # Disable s_src if local events are not needed
  source(s_src);
  source(s_network);
  destination(d_logzilla);
  flags(flow-control);
};
```

syslog-ng with TLS (6514)

Use TLS (RFC 5425) for encrypted forwarding. Ensure certificate paths are valid on the relay host.

```
destination d_logzilla_tls {
  network(
    "<LOGZILLA_HOST>"
    port(6514)
    transport(tls)
    tls(
      ca_dir("/etc/syslog-ng/ca.d")
      key_file("/etc/syslog-ng/key.d/relay-key.pem")
      cert_file("/etc/syslog-ng/cert.d/relay-cert.pem")
    )
  );
};
```

Update the `log {}` path to use `d_logzilla_tls` when forwarding securely.

Notes:

- **Relay trust store (`ca_dir` / `ca-file`)**: Use `ca_dir("/etc/syslog-ng/ca.d")` or `ca-file("/path/to/ca.pem")` to trust the LogZilla server certificate. The server's public cert can be copied to the relay and referenced with `ca-file("/path/to/tls.crt")`, or use a proper CA bundle.
- **Relay client key/cert are optional**: `key_file` and `cert_file` on the relay are only needed for mutual TLS (client authentication). For typical server-auth-only TLS, omit them.
- **Server-side TLS materials**: The LogZilla server's TLS listener uses `SYSLOG_TLS_CERT_FILE` and `SYSLOG_TLS_KEY_FILE` (defaults: `/etc/logzilla/syslog-ng/tls.crt` and `/etc/logzilla/syslog-ng/tls.key`). These files live on the server and are not copied to relays.
- **Client certs not required by default**: The server's syslog TLS listener does not require client certificates by default (peer verification is optional). Enable mutual TLS only if users have configured the server to require and trust client certs.

Minimal server-auth-only example (no client key/cert on the relay):

```
destination d_logzilla_tls {
  network(
    "<LOGZILLA_HOST>"
    port(6514)
    transport(tls)
    tls(
      # Either trust the issuing CA or the server's public cert
      ca-file("/path/to/logzilla-ca.crt")
      # or: ca_dir("/etc/syslog-ng/ca.d")
    )
  );
};
```

Background on certificates and TLS ports can be found in Using TLS Tunnels.

syslog-ng over HTTP/HTTPS (to /incoming)

Forward events using HTTP/HTTPS to LogZilla's HTTP Receiver at `/incoming`. Include an access token in headers.

```
destination d_logzilla_http {
  http(
    url("https://<LOGZILLA_HOST>:<PORT>/incoming")
    method("POST")
    user-agent("syslog-ng Relay")
    headers(
      "Content-Type: application/json",
      "Authorization: token <YOUR_TOKEN>"
    )
    body-prefix("{\"events\": [\n")
    delimiter(",\n")
    body('$(format-json
      --pair priority=int($PRI)
      --pair host="$HOST"
      --pair program="$PROGRAM"
      --pair message="$MESSAGE"
    )')
    body-suffix("\n]}")
    batch-lines(10000)
    batch-bytes(10485760)
    batch-timeout(500)
  );
};

log {
  source(s_src);
  destination(d_logzilla_http);
  flags(flow-control);
};
```

- Endpoint reference and minimal tests appear in HTTP Event Receiver.
- Interactive API docs are available at `/incoming/docs` on the LogZilla server. Accepted token header forms are documented in the HTTP Event Receiver documentation.

rsyslog with TLS

Use port 6514 for TLS syslog; 443 may be used only in constrained environments.

```
$DefaultNetstreamDriver gtls
$DefaultNetstreamDriverCAFile /etc/rsyslog.d/keys/ca.pem
$DefaultNetstreamDriverCertFile /etc/rsyslog.d/keys/client-cert.pem
$DefaultNetstreamDriverKeyFile /etc/rsyslog.d/keys/client-key.pem

$ActionSendStreamDriverAuthMode x509/name
$ActionSendStreamDriverPermittedPeer <LOGZILLA_HOST>
$ActionSendStreamDriverMode 1

*.* action(type="omfwd" Target="<LOGZILLA_HOST>" Port="6514" Protocol="tcp")
```

Best practices

Load balance high-volume sources across multiple relays.

Enable disk buffering to prevent loss during outages:

```
destination d_logzilla {
  network(
    "<LOGZILLA_HOST>"
    port(514)
    transport(tcp)
    disk-buffer(
      mem-buf-size(10000)
      disk-buf-size(2000000)
      reliable(yes)
    )
  );
};
```

Tag forwarded messages with relay identification (e.g., `relay_id`).

Apply coarse filtering at the relay to reduce traffic.

For WAN links, deploy both local and central relays for resilience.

Verification

- End-to-end checks and packet captures: Syslog Troubleshooting.
- HTTP relays: endpoint details and sample requests: HTTP Event Receiver.

CHAPTER 11

Command Line Tools

LogZilla command line reference for system administration, data management, query automation, and scripted workflows outside the web interface

Command Line Tools

LogZilla provides comprehensive command line tools for system administration, data management, and automation. These tools enable administrators to configure systems, manage data lifecycles, and build automated workflows without requiring web interface access.

Documentation Overview

CLI Overview and Setup

Learn the fundamentals of LogZilla command line tools including authentication, common patterns, and basic usage. Essential reading for all CLI users.

Key Topics:

- Command structure and help system
- Authentication methods (API tokens, config files)
- Environment variables and best practices
- Common patterns and error handling

System Administration Commands

Master system configuration, service management, and administrative tasks. These commands handle core system operations and maintenance.

Key Topics:

- Settings and license management
- Application installation and updates
- User management and LDAP configuration
- Service control and system maintenance

Data Management Commands

Control the complete data lifecycle from ingestion to archival. These tools manage data flow, storage optimization, and processing rules.

Key Topics:

- Archive management and retention policies
- Event statistics and performance monitoring
- Data forwarding and network analysis
- Parser rules and trigger management

Query Command Basics

Introduction to LogZilla's powerful query system for data analysis and reporting. Learn core query types and parameter structures.

Key Topics:

- Authentication and configuration setup
- Basic query types (Search, TopN, EventRate)
- Time ranges, filters, and operators
- Output formats and result handling

Advanced Query Types

Explore specialized queries for system monitoring, performance analysis, and advanced data aggregation.

Key Topics:

- System performance monitoring queries
- LogZilla internal statistics
- Advanced aggregation features
- Administrative and workflow queries

Query Examples and Automation

Practical examples, automation scripts, and integration patterns for building comprehensive LogZilla workflows.

Key Topics:

- Report generation examples
- System monitoring automation
- SIEM and ticketing system integration
- Scheduled automation and best practices

Quick Start Guide

Essential First Steps

Authentication Setup

```
# Create API token
logzilla authtoken create --user admin
export LOGZILLA_API_KEY="your-token"
```

System Status Check

```
# Verify LogZilla is running
logzilla version
logzilla license info
```

Basic Query

```
# Simple data query
echo '{"time_range": {"preset": "last_1_hours"}}' > test.json
logzilla query --type Search --params test.json
```

Common Use Cases

Task	Primary Commands	Documentation
System setup	`settings`, `license`, `apps`	System Commands
User management	`password`, `authtoken`, `ldap`	System Commands
Data lifecycle	`archives`, `events`, `drop`	Data Commands
Event processing	`rules`, `triggers`, `forwarder`	Data Commands
Data analysis	`query` with various types	Query Basics
System monitoring	System performance queries	Advanced Queries
Automation	Scripts and scheduled tasks	Query Examples

Command Categories

Administrative Commands

- **System Configuration**: `settings`, `license`, `https`
- **Service Management**: `start`, `stop`, `restart`, `version`
- **Application Management**: `apps install/uninstall`
- **User Management**: `password`, `authtoken`, `ldap`

Data Management Commands

- **Storage Lifecycle**: `archives`, `events`, `drop`
- **Data Flow**: `forwarder`, `sniffer`
- **Processing Rules**: `rules`, `triggers`
- **Performance**: Event statistics and optimization

Query and Analysis Commands

- **Basic Queries**: `Search`, `TopN`, `EventRate`, `LastN`
- **System Monitoring**: `System_CPU`, `System_Memory`, `System_DF`
- **Internal Statistics**: `StorageStats`, `ProcessingStats`
- **Administrative**: `Notifications`, `Tasks`

Best Practices Summary

Security

- Use API tokens instead of passwords for automation
- Protect configuration files with appropriate permissions
- Rotate credentials regularly
- Monitor command execution and access

Performance

- Use specific time ranges to limit query scope
- Apply filters to reduce data processing
- Schedule resource-intensive operations during off-peak hours
- Monitor system resources during large operations

Automation

- Validate parameters and handle errors gracefully
- Implement logging and monitoring for automated tasks
- Use configuration files for complex parameters
- Test automation in development environments first

Documentation and Maintenance

- Document automation purposes and dependencies
- Keep scripts and configurations under version control
- Monitor automation success rates and performance
- Maintain up-to-date parameter templates

Getting Help

Each command provides detailed help information:

```
# General help
logzilla --help

# Command-specific help
logzilla <command> --help
logzilla query --help
logzilla settings --help
```

For complex scenarios and troubleshooting, refer to the specific documentation sections above or contact LogZilla support with detailed command output and system information.

The LogZilla CLI provides powerful tools for every aspect of log management. Start with the basics and progressively build more sophisticated workflows as your requirements evolve.

Cli Overview

Introduction to the logzilla command line tool, covering command structure, subcommands, options, arguments, and the built-in help system

LogZilla Command Line Tools Overview

LogZilla provides a command line interface for system administration, data management, and automation. The CLI tools enable administrators to configure systems, manage data, and automate workflows without requiring web interface access.

Command Structure

All LogZilla commands follow a consistent pattern:

```
logzilla <command> [subcommand] [options] [arguments]
```

Examples:

```
logzilla version                    # Simple command
logzilla settings update TIME_ZONE=UTC  # Command with argument
logzilla query --type Search        # Command with options
logzilla apps install cisco-asa     # Command with subcommand and argument
```

Getting Help

Every command supports detailed help information:

```
# General help
logzilla --help

# Command-specific help
logzilla <command> --help
logzilla query --help
logzilla settings --help
```

Authentication and Access

Root User Requirement

Most LogZilla CLI commands require root privileges:

```
# Switch to root user
sudo su -

# Or run individual commands with sudo
sudo logzilla version
```

API Authentication

Commands that interact with the LogZilla API support multiple authentication methods:

Method 1: API Token (Recommended)

```
# Create an API token
logzilla authtoken create --user admin

# Use token with commands
logzilla query --authtoken <your-token> --type Search

# Set token in environment variable
export LOGZILLA_API_KEY="your-token-here"
logzilla query --type Search  # Uses environment variable
```

Method 2: Username and Password

```
logzilla query --user admin --password mypassword
```

Method 3: Configuration File

Create a configuration file at `$HOME/.lz5query`:

```
[lz5query]
authtoken = your-token-here
base_url = http://front/api
```

Or with username/password:

```
[lz5query]
user = admin
password = mypassword
base_url = http://front/api
```

Command Categories

LogZilla CLI tools are organized into functional categories:

System Administration

Configure LogZilla system settings and manage the service lifecycle.

Command	Purpose
`settings`	Configure system parameters
`license`	Manage licensing
`apps`	Install/manage LogZilla applications
`https`	Configure SSL certificates
`start/stop/restart`	Control LogZilla service
`version`	Display version information
`upgrade`	Upgrade LogZilla

User and Access Management

Manage user accounts, passwords, and authentication systems.

Command	Purpose
`password`	Change user passwords
`authtoken`	Manage API tokens
`ldap`	Configure LDAP/Active Directory

Data Management

Control data ingestion, storage, and lifecycle management.

Command	Purpose
`archives`	Manage data archiving
`events`	View event statistics
`drop`	Delete event data
`forwarder`	Configure event forwarding
`sniffer`	Capture network traffic

Event Processing

Configure how LogZilla processes and responds to events.

Command	Purpose
`rules`	Manage parser rules
`triggers`	Configure automated actions

Querying and Analysis

Search and analyze event data programmatically.

Command	Purpose
`query`	Perform structured queries

Troubleshooting

Diagnose system issues and gather diagnostic information.

Command	Purpose
`logs`	View LogZilla system logs
`inspect-dump`	Create diagnostic packages

Common Patterns

Working with Time Ranges

Many commands accept time range parameters:

```
# Using presets
--ts-from "3 days ago"
--ts-from "24 hours ago"
--ts-from "today"
--ts-from "yesterday"

# Using specific dates
--ts-from "2024-01-01" --ts-to "2024-01-31"
--ts-from "2024-01-01 00:00:00" --ts-to "2024-01-01 23:59:59"

# Using Unix timestamps
--ts-from 1704067200 --ts-to 1706745600
```

Output Formats

Commands that return data often support multiple output formats:

```
# JSON output (default for most commands)
logzilla query --type TopN --output $HOME/results.json

# Excel format
logzilla query --type TopN --output /root/report.xlsx --format xlsx
```

Batch Operations

Many commands support batch operations through configuration files:

```
# Query parameters in JSON file
logzilla query --type Search --params search-config.json

# Import configurations from files
logzilla triggers import --input triggers.yaml
logzilla forwarder import --input-file forwarder-config.yaml
```

Error Handling

CLI commands use standard exit codes:

- **0**: Success
- **1**: General error
- **2**: Invalid arguments
- **3**: Authentication failure

Check command success in scripts:

```
if logzilla version; then
    echo "LogZilla is accessible"
else
    echo "Error accessing LogZilla"
    exit 1
fi
```

Environment Variables

LogZilla CLI tools recognize several environment variables:

Variable	Purpose	Example
LOGZILLA_API_KEY	Default API token	export LOGZILLA_API_KEY="token123"
LOGZILLA_BASE_URL	API base URL	export LOGZILLA_BASE_URL="http://logzilla.local/api"
LOGZILLA_USER	Default username	export LOGZILLA_USER="admin"

Best Practices

Security

- **Use API tokens** instead of passwords when possible
- **Protect configuration files** with appropriate file permissions
- **Use environment variables** for automation scripts
- **Rotate API tokens** regularly

Automation

- **Check exit codes** in scripts for error handling
- **Use configuration files** for complex parameters
- **Log command output** for audit trails
- **Test commands** in development environments first

Performance

- **Use specific time ranges** to limit data processing
- **Leverage caching** where available
- **Run resource-intensive operations** during off-peak hours
- **Monitor system resources** during large operations

Quick Start Examples

Check System Status

```
# Verify LogZilla is running
logzilla version

# Check license status
logzilla license info

# View recent system logs
tail -f /var/log/logzilla/logzilla.log
```

Basic Data Query

```
# Create API token
TOKEN="$(logzilla authtoken create --user admin | tail -n1 | tr -d '\r')"

# Simple search query
echo '{"time_range": {"preset": "last_1_hours"}, "limit": 10}' > search.json
logzilla query --type Search --params search.json --authtoken $TOKEN
```

System Maintenance

```
# Archive old data
logzilla archives archive --expire-days 90

# Check system performance
logzilla events stats --ts-from "24 hours ago"

# Create diagnostic package
logzilla inspect-dump --output /tmp/diagnostics-$(date +%Y%m%d)
```

Next Steps

- **System Commands**: Learn about system administration commands
- **Data Commands**: Explore data management and lifecycle tools
- **Query Tools**: Master the query interface
- **Automation**: Build scripts and automated workflows

The LogZilla CLI provides tools for every aspect of log management. Each command category builds upon these foundational concepts to provide specialized functionality for different administrative tasks.

System Commands

LogZilla CLI system commands for settings management, licensing, application installation, service control, and administrative operations

System Commands

LogZilla provides command line tools for system configuration, service management, and administrative tasks. These commands handle core system operations including settings management, licensing, application installation, and service control.

System Configuration

Settings Management

Configure LogZilla system parameters including time zones, SMTP settings, retention policies, and feature toggles.

View Settings

```
# List all current settings (all groups)
logzilla settings list

# List settings for a specific group (e.g., generic)
logzilla settings list generic
```

Modify Settings

```
# Change settings
logzilla settings update TIME_ZONE=UTC
logzilla settings update SMTP_SERVER=mail.company.com

# Set multiple settings at once
logzilla settings update TIME_ZONE=EST SMTP_SERVER=mail.local.com
```

Tip: Use `logzilla settings list` to discover exact setting names and groups available in your installation before updating values.

License Management

Manage LogZilla licensing, view usage limits, and handle license updates.

View License Information

```
# Display license details
logzilla license info
```

License Installation

```
# Get server's unique host key (needed for licensing)
logzilla license key

# Download license from LogZilla servers (requires internet)
logzilla license download

# Load license from file
logzilla license load /path/to/license.txt

# Verify license validity
logzilla license verify
```

Application Management

Install and manage pre-built parsing rules, dashboards, and integrations for specific products such as Cisco ASA, Palo Alto Networks, and other systems.

List Applications

```
# List available applications
logzilla apps list
```

Install Applications

```
# Install an application by code
logzilla apps install cisco-asa

# Reinstall if already installed
logzilla apps install cisco-asa --force
```

Manage Applications

```
# Remove an application
logzilla apps uninstall cisco-asa

# Test application definition without installing
logzilla apps test cisco-asa
```

Create Custom Applications

```
# Create a custom application from template
logzilla apps create my-custom-app
```

HTTPS Configuration

Configure SSL/TLS certificates to secure the LogZilla web interface.

Enable HTTPS

```
# Enable HTTPS with certificate and key files
logzilla https --set /path/to/private.key /path/to/certificate.crt
```

Note: The HTTPS tool currently supports setting the key and certificate. Other management actions are handled automatically by the platform.

User and Access Management

Password Management

Change passwords for local LogZilla user accounts.

```
# Change password for a user (interactive prompt)
logzilla password username

# Provide password non-interactively (use with caution)
logzilla password username --password 'newpass'
```

API Token Management

Create and manage API tokens for programmatic access to LogZilla.

Create Tokens

```
# Create token for a user (defaults to admin if omitted)
logzilla authtoken create --user admin

# Create ingest-only token (for sending data)
logzilla authtoken create --ingest-only
```

Manage Tokens

```
# List all active tokens (table)
logzilla authtoken list

# List tokens for specific user (JSON)
logzilla authtoken list --user admin --json-dump

# View token details
logzilla authtoken info <token>

# Revoke a token
logzilla authtoken revoke <token>
```

LDAP Authentication

Configure Active Directory or LDAP authentication for user management.

Initial Configuration

```
# Initialize LDAP configuration (interactive wizard)
logzilla ldap init

# Create an additional LDAP configuration instance
logzilla ldap create
```

Manage LDAP Settings

```
# List configured LDAP instances
logzilla ldap list

# Enable LDAP configuration
logzilla ldap enable --id 1

# Disable LDAP configuration
logzilla ldap disable --id 1
```

Test LDAP

```
# Test LDAP connection and user authentication
logzilla ldap test --id 1 --username testuser --password testpass

# Show detailed debug logs
logzilla ldap test --id 1 --show-debug-logs
```

Control LDAP

```
# Enable LDAP configuration
logzilla ldap enable --id 1

# Disable LDAP configuration
logzilla ldap disable --id 1

# Sync users from LDAP
logzilla ldap sync --id 1

# Import LDAP groups
logzilla ldap import-groups --id 1
```

Service Control

Basic Service Operations

Control the LogZilla service lifecycle and check system status.

```
# Start LogZilla
logzilla start

# Stop LogZilla
logzilla stop

# Restart LogZilla
logzilla restart
```

Version and Updates

```
# Display current version
logzilla version

# Upgrade to latest version
sudo logzilla upgrade

# Upgrade to a specific version
logzilla upgrade --version v6.2.0

# Upgrade using offline images
logzilla upgrade --offline-dir /path/to/images
```

System Maintenance

System Operations

```
# Uninstall LogZilla (use with caution)
logzilla uninstall --force-removal

# Factory reset (removes data)
logzilla reset --all --force-removal

# Reset only events
logzilla reset --events --force-removal

# Reset admin password to 'admin'
logzilla reset --adminpass
```

Configuration Management

Snapshots

```
# Create snapshot
logzilla snapshot create

# List available snapshots
logzilla snapshot list

# Restore from snapshot (latest if --id omitted)
logzilla snapshot restore --id <snapshot-id>

# Remove old snapshots automatically
logzilla snapshot autoremove
```

Migration

```
# Create migration package
logzilla migrate-dump create --dump-dir /tmp/migration

# Restore from migration package
logzilla migrate-dump restore --dump-dir /tmp/migration --force
```

Troubleshooting Commands

System Diagnostics

```
# Create comprehensive diagnostic package
logzilla inspect-dump --output /tmp/diagnostics-$(date +%Y%m%d)

# Test system performance
logzilla speedtest
```

Log Management

```
# Show the path to follow logs (tail command)
logzilla logs

# Example follow
sudo tail -f /var/log/logzilla/logzilla.log
```

Container Access

```
# Access a container shell (default: watcher)
logzilla shell --container watcher

# Run a command in a specific container
logzilla shell --container syslog -- bash -lc 'ls -la /'
```

Best Practices

Security

- **Regularly rotate API tokens** and remove unused ones
- **Use HTTPS** in production environments
- **Implement strong password policies** for local accounts

- **Monitor LDAP connectivity** and authentication logs

Maintenance

- **Create snapshots** before major configuration changes
- **Monitor license expiration** and usage limits
- **Keep applications updated** to latest versions
- **Regular system backups** for disaster recovery

Performance

- **Monitor system resources** during maintenance operations
- **Schedule upgrades** during maintenance windows
- **Test configuration changes** in development first
- **Use appropriate retention policies** to manage disk usage

These system administration commands provide the foundation for managing LogZilla infrastructure. Combined with proper monitoring and maintenance procedures, they ensure reliable operation and optimal performance.

Data Commands

LogZilla CLI data commands for archive creation, retention policies, rule management, trigger execution, dashboard import and export, and event lifecycle

Data Management Commands

LogZilla provides command line tools for managing the complete data lifecycle, from ingestion and processing to archival and deletion. These commands handle data flow, storage optimization, event processing rules, and automated responses to events.

Data Storage and Lifecycle

Archive Management

Control data archiving to manage disk space and implement data retention policies. Archives move older data to compressed storage while maintaining searchability.

Create Archives

```
# Archive events older than N days
logzilla archives archive --expire-days 30

# Archive specific date range (rounded to full hours)
logzilla archives --round archive --ts-from "2024-01-01 00:00" --ts-to "2024-01-31 23:00"

# Round provided timestamps to full hours automatically
logzilla archives --round archive --expire-days 60
```

Migrate Old Archives

```
# Migrate archives to the latest format for direct querying
logzilla archives migrate --ts-from "2023-01-01 00:00" --ts-to "2023-12-31 23:00"
```

Remove Archives

```
# Remove archived data permanently (use with caution)
logzilla archives remove --ts-from "2022-01-01 00:00" --ts-to "2022-12-31 23:00"
```

Event Statistics and Monitoring

Monitor data volume, system performance, and storage utilization.

View Event Statistics

```
# View event statistics for a date range (print totals)
logzilla events stats --ts-from "2024-01-01 00:00" --ts-to "2024-01-31 23:59"

# Include full per-chunk table
logzilla events stats --ts-from "last 24 hours" --full-table

# Exclude archive or live data from the result
logzilla events stats --ts-from "3 days ago" --exclude-archive
```

Parser Performance

```
# View parser performance metrics
logzilla events parser-stats
```

Field Analysis

```
# Show field and tag cardinality (unique values)
logzilla events cardinality --scope all

# Filter fields by regex and compute exact values
logzilla events cardinality --scope fields --filter '^host|program$' --exact

# Show actual values seen (fields or tags)
logzilla events values --scope fields --limit 50
logzilla events values --scope tags --limit 100
```

Data Deletion

Permanently remove event data from LogZilla. Use these commands with extreme caution as data deletion cannot be undone.

```
# Permanently delete events for date range (timestamps must align to full hours)
logzilla drop --ts-from "2023-01-01 00:00" --ts-to "2023-01-31 00:00"

# Automatically round to the nearest hour
logzilla drop --ts-from "2023-01-01 00:05" --ts-to "2023-01-31 00:02" --round

# Skip confirmation (use with extreme caution)
logzilla drop --ts-from "2023-01-01 00:00" --ts-to "2023-01-31 00:00" --force-removal
```

Warning: The `drop` command permanently deletes data and cannot be undone. Always verify date ranges carefully. Use `--round` to correct non-hour-aligned timestamps.

Data Ingestion and Forwarding

Event Forwarding

Configure LogZilla to forward events to external systems such as SIEMs, notification platforms, or other log management systems.

For configuration structure, options, and examples, see the Forwarding Module.

View Forwarder Configuration

```
# Display current forwarder configuration
logzilla forwarder print

# Display configuration split per files
logzilla forwarder print-files
```

Manage Forwarder Rules

```
# Import forwarder rules from file
logzilla forwarder import --input-file /path/to/forwarder.yaml

# Cleanup all forwarder configuration and counters (prompts for consent)
logzilla forwarder cleanup
```

Control Forwarder

```
# Reload forwarder configuration after changes
logzilla forwarder reload
```

Forwarder Statistics

```
# View forwarding statistics
logzilla forwarder stats

# View statistics for specific time range
logzilla forwarder stats --ts-from "2024-01-01 00:00" --ts-to "2024-01-31 23:59"
```

Network Traffic Analysis

Capture and analyze syslog traffic for troubleshooting ingestion issues and testing configurations.

Traffic Capture

```
# Start capturing syslog traffic
logzilla sniffer start --filename sniff-{t}.out --time 60 --live

# Store raw (unparsed) events
logzilla sniffer start --raw --time 120
```

Traffic Analysis

```
# View capture status
logzilla sniffer status

# Stop traffic capture
logzilla sniffer stop
```

Traffic Replay

```
# Replay captured traffic for testing
logzilla sniffer replay sniff-2025-01-01.out

# Keep original timestamps if present
logzilla sniffer replay sniff-2025-01-01.out --keep-ts
```

Event Processing Rules

Parser Rules Management

Manage rules that extract fields, normalize log messages, and enhance event data with additional context.

List and View Rules

```
# List all parser rules
logzilla rules list

# Filter by name using wildcards (quote the pattern)
logzilla rules list "*cisco*"

# Show recent runtime errors for rules
logzilla rules errors
```

Manage Rules

```
# Enable specific rule
logzilla rules enable "Custom Rule"

# Disable specific rule
logzilla rules disable "Old Rule"

# Add new rule from file
logzilla rules add /path/to/rule.yaml --name "Custom Rule"

# Remove rule
logzilla rules remove "Unused Rule"
```

Rule Testing and Validation

```
# Test rule for syntax errors
logzilla rules test "Custom Rule"

# Test rule against sample data
logzilla rules test --path /path/to/custom-rule.lua

# Validate rule file structure before adding
logzilla rules validate /path/to/new-rule.lua

# Dry run rule changes
logzilla rules test --path /path/to/custom-rule.lua --exitfirst
```

Rule Operations

```
# Reload all rules after changes
logzilla rules reload

# Export rules to file
logzilla rules export --output /backup/parser-rules.yaml

# Import rules from file
logzilla rules import --input /backup/parser-rules.yaml

# Run rule performance tests
logzilla rules performance --detailed
```

Trigger Management

Configure automated actions that execute when specific events occur, such as sending notifications, executing scripts, or creating tickets.

List and View Triggers

```
# List all triggers
logzilla triggers list

# Filter by name using wildcards (quote the pattern)
logzilla triggers list "*security*"
```

Manage Triggers

```
# Import triggers from file
logzilla triggers import --input /path/to/triggers.yaml --owner admin

# Export trigger to file
logzilla triggers export --trigger-id 123 --output /backup/security-trigger.yaml

# Export all triggers
logzilla triggers export --output /backup/all-triggers.yaml

# Update trigger configuration fields
logzilla triggers update --trigger-id 123 --set add_note=true severity=4
```

Delete Triggers

```
# Delete by id or by name filter (prompts for consent)
logzilla triggers delete --trigger-id 123
logzilla triggers delete "*old*"
```

Trigger Performance

```
# Run trigger performance tests
logzilla triggers performance
```

Data Quality and Maintenance

Repair and Cleanup

```
# Run automated repair for storage and aggregates
logzilla events fix --ts-from "2024-01-01 00:00" --ts-to "2024-01-02 00:00" --scope
storage
logzilla events fix --ts-from "2024-01-01 00:00" --ts-to "2024-01-02 00:00" --scope
aggregates
```

Notes

- Some operations require LogZilla to be running and properly licensed.
- Timestamps that represent date ranges should align to full hours for storage operations. Use `--round` where available.

Best Practices

Data Lifecycle Management

- **Implement retention policies** based on compliance requirements
- **Archive old data regularly** to manage disk space
- **Monitor storage usage** and plan for growth
- **Test archive and restore procedures** regularly

Optimizing Performance

- **Monitor parser performance** and optimize rules
- **Use appropriate archival schedules** for your data volume
- **Regular maintenance tasks** to keep system optimized
- **Monitor forwarder performance** and adjust as needed

Data Quality

- **Validate parser rules** before deployment
- **Monitor trigger execution** for errors
- **Regular data integrity checks** to ensure consistency
- **Test configurations** in development environments

Security and Compliance

- **Secure forwarder connections** with encryption
- **Audit data access** and modification activities
- **Implement proper data retention** for compliance

- **Monitor and log administrative actions**

These data management commands provide control over the LogZilla data lifecycle. Proper use of these tools ensures optimal performance, compliance with retention policies, and reliable data processing workflows.

Query Basics

Run structured searches against LogZilla from the command line using logzilla query with parameter files, authtoken setup, and query type selection

LogZilla Query Command Basics

The `logzilla query` command provides programmatic access to LogZilla data through structured queries. This tool enables administrators to generate reports, extract specific data sets, and automate data analysis workflows.

Getting Started

Prerequisites

The query command requires authentication and proper permissions:

```
# Ensure you have root access or appropriate permissions
sudo su -

# Create API token for authentication
logzilla authtoken create --user admin
```

Basic Command Structure

```
logzilla query --type <QueryType> --params <parameters-file> [options]
```

Required Parameters:

- `--type` : Specifies the query type (Search, TopN, EventRate, etc.)
- `--params` : Path to JSON file containing query parameters

Common Options:

- `--authtoken` : API authentication token
- `--user` / `--password` : Username/password authentication
- `--output-file` : Save results to file
- `--format` : Output format (json, xlsx)

Authentication Setup

Method 1: API Token (Recommended)

```
# Create token and save to environment
TOKEN="$(logzilla authtoken create | tail -n1 | tr -d '[:space:]')"
export LOGZILLA_API_KEY="$TOKEN"

# Use in queries
logzilla query --type Search --params search.json --authtoken $LOGZILLA_API_KEY
```

Method 2: Configuration File

Create `~/.lz5query` configuration file:

```
[lz5query]
authtoken = your-api-token-here
base_url = http://localhost/api
```

Method 3: Username/Password

```
# Interactive password prompt
logzilla query --type Search --params search.json --user admin --password

# Direct password (less secure)
logzilla query --type Search --params search.json --user admin --password mypass
```

Core Query Types

Search Query

Retrieve specific events matching filter criteria. Most flexible query type for detailed event analysis.

Parameters File (search-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours"
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [3]
    }
  ],
  "limit": 100,
  "sort": ["-first_occurrence"]
}
```

Execute Query:

```
logzilla query --type Search --params search-params.json --authtoken $TOKEN
```

TopN Query

Find the most frequent values for a specific field. Ideal for identifying top talkers, most active hosts, or common error patterns.

Parameters File (topn-params.json):

```
{
  "field": "host",
  "limit": 10,
  "time_range": {
    "preset": "last_7_days"
  },
  "filter": [
    {
      "field": "program",
      "op": "eq",
      "value": ["sshd"]
    }
  ]
}
```

Execute Query:

```
logzilla query --type TopN --params topn-params.json --authtoken $TOKEN
```

EventRate Query

Calculate event rates over time periods. Useful for trend analysis and capacity planning.

Parameters File (eventrate-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours",
    "step": 3600
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ]
}
```

Execute Query:

```
logzilla query --type EventRate --params eventrate-params.json --authtoken $TOKEN
```

Common Parameters

Time Range Configuration

Time ranges define the period for data retrieval. Multiple formats supported:

Preset Time Ranges

```
{
  "time_range": {
    "preset": "today"
  }
}
```

Available Presets:

- `today` - Current day from midnight
- `yesterday` - Previous day
- `last_1_hours` - Last 1 hour
- `last_24_hours` - Last 24 hours

- `last_7_days` - Last 7 days
- `last_30_days` - Last 30 days
- `last_365_days` - Last 365 days

Note: Dynamic presets are supported with the pattern `last_N_unit`, where `unit` is one of `seconds`, `minutes`, `hours`, `days`, or `weeks`. Examples:

- `last_6_hours`
- `last_12_hours`
- `last_90_days`
- `last_2_weeks`

Absolute Time Ranges

```
{
  "time_range": {
    "ts_from": "2024-01-01 00:00:00",
    "ts_to": "2024-01-31 23:59:59"
  }
}
```

Unix Timestamps

```
{
  "time_range": {
    "ts_from": 1704067200,
    "ts_to": 1706745600
  }
}
```

Time Steps for Aggregation

```
{
  "time_range": {
    "preset": "last_24_hours",
    "step": 3600
  }
}
```

Common Step Values:

- `60` - 1 minute intervals

- `900` - 15 minute intervals
- `3600` - 1 hour intervals
- `86400` - 1 day intervals

Filter Configuration

Filters limit query results to specific criteria. Multiple filters are combined with AND logic.

Basic Filter Structure

```
{
  "filter": [
    {
      "field": "field_name",
      "op": "operator",
      "value": ["value1", "value2"]
    }
  ]
}
```

Available Operators

Operator	Description	Example
`eq`	Equals (default)	`"value": ["error"]`
`ne`	Not equals	`"value": ["debug"]`
`lt`	Less than	`"value": [5]`
`le`	Less than or equal	`"value": [3]`
`gt`	Greater than	`"value": [100]`
`ge`	Greater than or equal	`"value": [1000]`
`qp`	Query phrase (message search)	`"value": ["connection failed"]`
`=*`	Wildcard match	`"value": ["error*"]`
`!*`	Not wildcard match	`"value": ["debug*"]`
`=~`	Regex match	`"value": ["^ERROR.*"]`
`!~`	Not regex match	`"value": ["^DEBUG.*"]`

Filter Examples

Host Filter:

```
{
  "field": "host",
  "op": "eq",
  "value": ["web-server-01", "web-server-02"]
}
```

Severity Filter:

```
{
  "field": "severity",
  "op": "le",
  "value": [3]
}
```

Message Search:

```
{
  "field": "message",
  "op": "qp",
  "value": ["authentication failed"]
}
```

Wildcard Program Filter:

```
{
  "field": "program",
  "op": "=*",
  "value": ["ssh*"]
}
```

Archive Data Access

Include archived data in queries using the `with_archive` parameter:

```
{
  "time_range": {
    "preset": "last_365_days"
  },
  "with_archive": true,
  "filter": [
    {
      "field": "host",
      "op": "eq",
      "value": ["critical-server"]
    }
  ]
}
```

Note: Queries with `with_archive: true` may take longer to execute as they search both active and archived data.

Result Limiting and Sorting

Limit Results

```
{
  "limit": 50,
  "page_size": 25
}
```

Sort Results

```
{
  "sort": ["-first_occurrence", "host"]
}
```

Sort Fields:

- `first_occurrence` / `-first_occurrence` (descending)
- `last_occurrence` / `-last_occurrence` (descending)
- `count` / `-count` (descending)
- `host`, `program`, `severity`

Output Formats

JSON Output (Default)

```
logzilla query --type Search --params search.json --output results.json
```

Results saved as structured JSON for programmatic processing.

Excel Output

```
logzilla query --type TopN --params topn.json --output report.xlsx --format xlsx
```

Results formatted as Excel spreadsheet for business reporting.

Console Output

```
logzilla query --type Search --params search.json
```

Results displayed directly in terminal for immediate review.

Common Field Names

Standard Event Fields

Field	Description	Example Values
`host`	Source hostname	`web-server-01`, `192.168.1.100`
`program`	Program/service name	`sshd`, `kernel`, `nginx`
`message`	Log message text	`Connection established`
`severity`	Syslog severity (0-7)	`0` (Emergency) to `7` (Debug)
`facility`	Syslog facility (0-23)	`16` (Local0), `23` (Local7)
`first_occurrence`	First seen timestamp	`1704067200.123`
`last_occurrence`	Last seen timestamp	`1704067260.456`
`count`	Event occurrence count	`1`, `50`, `1000`

Enhanced Fields

Field	Description	Example Values
`cisco_mnemonic`	Cisco device message ID	`ASA-6-302013`
`status`	Event processing status	`0` (Unknown), `1` (Known)
`user_tags`	Custom user tags	`{"environment": "production"}`

Error Handling

Common Issues

Authentication Errors:

```
# Error: Invalid token
# Solution: Create new token
logzilla authtoken create --user admin
```

Parameter Errors:

```
# Error: Invalid JSON in parameters file
# Solution: Validate JSON syntax
cat params.json | jq .
```

Time Range Errors:

```
# Error: Invalid time range
# Solution: Check date format and range validity
```

Debugging

```
# Enable debug output
logzilla query --type Search --params search.json --debug

# Validate parameters file
cat search.json | jq .

# Test with minimal parameters
echo '{"time_range": {"preset": "last_1_hours"}}' > test.json
logzilla query --type Search --params test.json
```

Troubleshooting

Verify that the ~/.lz5query file is correct and contains the correct credentials.

```
[lz5query]
authtoken = your-api-token-here
base_url = http://localhost/api
```

Best Practices

Performance

- **Use specific time ranges** to limit data processing
- **Apply filters** to reduce result set size
- **Use appropriate limits** for large datasets
- **Consider archived data impact** on query performance

Security

- **Use API tokens** instead of passwords
- **Protect configuration files** with proper permissions
- **Rotate tokens regularly** for security
- **Limit token permissions** to required operations

Automation

- **Validate JSON parameters** before execution
- **Handle errors gracefully** in scripts
- **Log query execution** for audit trails
- **Use consistent naming** for parameter files

The query command provides the foundation for data analysis and reporting. Master these basics before moving to advanced query types and complex automation scenarios.

Advanced Query Types

Advanced logzilla query types for host system metrics, CPU, memory, EPS throughput, capacity planning, and historical performance analysis

Advanced Query Types and System Monitoring

LogZilla provides specialized query types for system monitoring, performance analysis, and advanced data aggregation. These queries enable deep insights into both log data patterns and LogZilla system health.

System Performance Queries

System queries provide real-time and historical data about LogZilla host performance. These queries are essential for monitoring system health and capacity planning.

System CPU Usage

Monitor CPU utilization across different categories and time periods.

Parameters (cpu-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours"
  },
  "cpu": "totals"
}
```

CPU Options:

- `"totals"` - Aggregate across all CPU cores
- `0`, `1`, `2`, etc. - Specific CPU core number

Execute Query:

```
logzilla query --type System_CPU --params cpu-params.json --authtoken $TOKEN
```

Result Categories:

- `user` - CPU used by user applications
- `system` - CPU used by operating system

- `idle` - CPU not doing work
- `wait` - CPU waiting for disk I/O
- `interrupt` - CPU handling hardware interrupts
- `softirq` - CPU servicing soft interrupts
- `nice` - CPU for process priority management
- `steal` - CPU allocated by hypervisor (virtualized systems)

System Memory Usage

Track memory utilization and identify memory pressure.

Parameters (memory-params.json):

```
{
  "time_range": {
    "preset": "last_12_hours"
  }
}
```

Execute Query:

```
logzilla query --type System_Memory --params memory-params.json --authtoken $TOKEN
```

Memory Categories:

- `used` - Memory used by processes
- `free` - Available memory
- `buffered` - Memory used for I/O buffers
- `cached` - Memory used for disk cache

System Disk Usage

Monitor disk space utilization across filesystems.

Parameters (disk-params.json):

```
{
  "time_range": {
    "preset": "last_7_days"
  },
  "fs": "root"
}
```

Filesystem Options:

- `"root"` - Root filesystem (always available)
- System-specific mount points (varies by configuration)

Execute Query:

```
logzilla query --type System_DF --params disk-params.json --authtoken $TOKEN
```

System I/O Operations

Track disk I/O performance and identify bottlenecks.

Parameters (iops-params.json):

```
{
  "time_range": {
    "preset": "last_6_hours"
  }
}
```

Execute Query:

```
logzilla query --type System_IOPSQuery --params iops-params.json --authtoken $TOKEN
```

I/O Metrics:

- `reads` - Read operations per second
- `writes` - Write operations per second

System Network Usage

Monitor network interface utilization and performance.

Parameters (network-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours"
  },
  "interface": "eth0"
}
```

Execute Query:

```
logzilla query --type System_Network --params network-params.json --authtoken $TOKEN
```

Network Metrics:

- `if_packets.tx` - Packets transmitted
- `if_packets.rx` - Packets received
- `if_octets.tx` - Bytes transmitted
- `if_octets.rx` - Bytes received
- `if_errors.tx` - Transmission errors
- `if_errors.rx` - Reception errors

System Network Errors

Focus specifically on network error conditions.

Parameters (net-errors-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours"
  },
  "interface": "eth0"
}
```

Execute Query:

```
logzilla query --type System_NetworkErrors --params net-errors-params.json --authtoken
$TOKEN
```

Error Metrics:

- `drop_in` - Incoming packets dropped
- `drop_out` - Outgoing packets dropped
- `err_in` - Incoming packet errors
- `err_out` - Outgoing packet errors

LogZilla Internal Queries

Storage Statistics

Monitor LogZilla's internal storage performance and utilization.

Parameters (storage-params.json):

```
{
  "time_range": {
    "preset": "last_24_hours"
  }
}
```

Execute Query:

```
logzilla query --type StorageStats --params storage-params.json --authtoken $TOKEN
```

Storage Metrics:

- `new` - New events processed (not duplicates)
- `duplicates` - Duplicate events identified
- `total` - Total events processed

Processing Statistics

Track LogZilla's event processing performance.

Note: Requires `INTERNAL_COUNTERS_MAX_LEVEL` set to `DEBUG`

Enable Processing Stats:

```
logzilla settings IC_LOG_LEVEL=DEBUG
```

Parameters (processing-params.json):

```
{
  "time_range": {
    "preset": "last_6_hours"
  }
}
```

Execute Query:

```
logzilla query --type ProcessingStats --params processing-params.json --authtoken $TOKEN
```

Processing Metrics:

- `new` - New events processed
- `duplicates` - Duplicate events found
- `oot` - Out-of-time events (outside TIME_TOLERANCE)

Advanced Data Queries

LastN Query

Retrieve the most recent values for a specific field, useful for finding latest activity or recent changes.

Parameters (lastn-params.json):

```
{
  "field": "host",
  "limit": 20,
  "time_range": {
    "preset": "last_7_days"
  },
  "filter": [
    {
      "field": "program",
      "op": "eq",
      "value": ["kernel"]
    }
  ]
}
```

Execute Query:

```
logzilla query --type LastN --params lastn-params.json --authtoken $TOKEN
```

Result Fields:

- `name` - Field value
- `count` - Occurrence count in time range
- `last_seen` - Timestamp of most recent occurrence

Advanced TopN Features

TopN queries support advanced aggregation and subfield analysis.

Parameters with Subfields (topn-advanced.json):

```
{
  "field": "host",
  "limit": 10,
  "time_range": {
    "preset": "last_24_hours"
  },
  "with_subperiods": true,
  "subfields": ["program", "severity"],
  "subfields_limit": 5,
  "show_other": true
}
```

Advanced Options:

- `with_subperiods` - Include data for each time sub-period
- `subfields` - Show breakdown by additional fields
- `subfields_limit` - Limit subfield results
- `show_other` - Include "other" category for remaining values
- `top_periods` - Show top sub-periods by activity

Administrative Queries

Notifications Query

Retrieve notification group information and associated events.

Parameters (notifications-params.json):

```
{
  "sort": ["-updated_at"],
  "time_range": {
    "preset": "last_7_days"
  },
  "time_range_field": "created_at",
  "is_private": false,
  "read": false,
  "with_events": true
}
```

Sort Options:

- `"Oldest first"`
- `"Newest first"`
- `"Oldest unread first"`

- `"Newest unread first"`

Time Range Fields:

- `"created_at"` - When notification was created
- `"updated_at"` - When notification was last updated
- `"unread_since"` - When notification became unread
- `"read_at"` - When notification was read

Execute Query:

```
logzilla query --type Notifications --params notifications-params.json --authtoken $TOKEN
```

Tasks Query

Retrieve task management information for workflow tracking.

Parameters (tasks-params.json):

```
{
  "target": "all",
  "is_overdue": false,
  "is_open": true,
  "assigned_to": [],
  "sort": ["-created_at"]
}
```

Target Options:

- `"assigned_to_me"` - Tasks assigned to current user
- `"all"` - All tasks

Execute Query:

```
logzilla query --type Tasks --params tasks-params.json --authtoken $TOKEN
```

Query Result Formats

Understanding Result Structure

All queries return results in a consistent structure:

```
{
  "query_id": "unique-query-identifier",
  "results": {
    "totals": {
      "ts_from": 1704067200,
      "ts_to": 1704153600,
      "count": 12345
    },
    "details": [
      {
        "ts_from": 1704067200,
        "ts_to": 1704070800,
        "count": 1234
      }
    ]
  }
}
```

System Query Results

System queries include aggregated statistics:

```
{
  "totals": {
    "usage_cpu": {
      "sum": 1234.56,
      "count": 240,
      "min": 0.1,
      "max": 98.7,
      "avg": 5.14,
      "last": 12.3,
      "last_ts": 1704153600
    }
  }
}
```

Aggregate Fields:

- `sum` - Total of all values
- `count` - Number of data points
- `min` - Minimum value
- `max` - Maximum value
- `avg` - Average value (sum/count)
- `last` - Most recent value

- `last_ts` - Timestamp of most recent value

Performance Considerations

Query Optimization

Time Range Optimization:

- Use specific time ranges rather than very broad ranges
- Consider system performance impact of long historical queries
- Use appropriate step sizes for time-series data

Filter Optimization:

- Apply filters to reduce data set size
- Use indexed fields (host, program, severity) for better performance
- Combine multiple filters efficiently

Archive Considerations:

- Queries with `with_archive: true` require additional processing time
- Archive queries may have different performance characteristics
- Consider splitting large archive queries into smaller time ranges

Resource Management

Memory Usage:

- Large result sets consume significant memory
- Use appropriate limits to control memory usage
- Consider pagination for very large datasets

CPU Impact:

- Complex aggregations require CPU resources
- System queries add monitoring overhead
- Schedule resource-intensive queries during off-peak hours

Best Practices

System Monitoring

- **Regular health checks** using system queries
- **Set up automated monitoring** for key metrics
- **Establish baselines** for normal system behavior

- **Alert on anomalies** in system performance

Query Design

- **Start with simple queries** and add complexity gradually
- **Test query performance** before automation
- **Use appropriate time ranges** for your use case
- **Document query purposes** and parameters

Data Analysis

- **Combine multiple query types** for deeper analysis
- **Use subfields** for detailed breakdowns
- **Archive historical results** for trend analysis
- **Validate results** against known system behavior

Automation

- **Schedule regular reports** using system queries
- **Implement alerting** based on query results
- **Create reusable parameter templates** for common queries
- **Monitor query execution times** and optimize as needed

Advanced queries provide deep insights into both log data and system performance. Master these tools to build monitoring and analysis workflows.

Query Examples

Worked examples of the logzilla query command for security event reports, authentication failure summaries, scheduled automation, and external integration

Query Examples and Automation

Practical examples, automation scripts, and real-world use cases for the LogZilla query command. These examples demonstrate how to build reports, automate analysis, and integrate LogZilla data with external systems.

Report Generation Examples

Security Event Summary Report

Generate a security report showing authentication failures, privilege escalations, and suspicious activities.

Parameters (security-report.json):

```
{
  "field": "host",
  "limit": 20,
  "time_range": {
    "preset": "last_7_days"
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["\"authentication failed\" OR \"privilege escalation\" OR suspicious"]
    },
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ],
  "subfields": ["program", "severity"],
  "subfields_limit": 10
}
```

Automation Script (security-report.sh):

```
#!/bin/bash

# Security Report Generation Script
REPORT_DATE=$(date +%Y%m%d)
OUTPUT_FILE="/reports/security-report-${REPORT_DATE}.xlsx"
API_TOKEN="${LOGZILLA_API_KEY}"

# Generate security report
logzilla query --type TopN \
    --params security-report.json \
    --output-file "$OUTPUT_FILE" \
    --format xlsx \
    --authtoken "$API_TOKEN"

# Email report to security team
if [ -f "$OUTPUT_FILE" ]; then
    echo "Security report generated: $OUTPUT_FILE" | \
    mail -s "Daily Security Report - $REPORT_DATE" \
         -a "$OUTPUT_FILE" \
         security-team@company.com
fi
```

Network Traffic Analysis

Analyze network device logs to identify top talkers and traffic patterns.

Parameters (network-analysis.json):

```
{
  "field": "host",
  "limit": 15,
  "time_range": {
    "preset": "last_24_hours"
  },
  "filter": [
    {
      "field": "program",
      "op": "=*",
      "value": ["*firewall*", "*router*", "*switch*"]
    }
  ],
  "with_subperiods": true,
  "subfields": ["program"],
  "subfields_limit": 5
}
```

Application Performance Report

Monitor application performance by analyzing error rates and response times.

Parameters (app-performance.json):

```
{
  "time_range": {
    "preset": "last_24_hours",
    "step": 3600
  },
  "filter": [
    {
      "field": "program",
      "op": "eq",
      "value": ["nginx", "apache2", "tomcat"]
    },
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ]
}
```

Complete Script (app-performance.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DIR="/reports/$(date +%Y/%m)"
mkdir -p "$REPORT_DIR"

# Generate event rate report
logzilla query --type EventRate \
    --params app-performance.json \
    --output-file "${REPORT_DIR}/app-errors-$(date +%Y%m%d-%H%M).json" \
    --authtoken "$API_TOKEN"

# Generate top applications report
echo '{
  "field": "program",
  "limit": 10,
  "time_range": {"preset": "last_24_hours"},
  "filter": [{"field": "severity", "op": "le", "value": [4]}]
}' > /tmp/top-apps.json

logzilla query --type TopN \
    --params /tmp/top-apps.json \
    --output-file "${REPORT_DIR}/top-error-apps-$(date +%Y%m%d-%H%M).xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/top-apps.json
```

Data Analysis Workflows

Trend Analysis Over Time

Analyze event patterns over extended periods to identify trends.

Weekly Trend Analysis (weekly-trends.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
WEEKS=12  # Analyze last 12 weeks

# Ensure output dir exists
mkdir -p /reports/trends

for i in $(seq 0 $((WEEKS-1))); do
    START_DATE=$(date -d "-$((i+1)) weeks" +%Y-%m-%d)
    END_DATE=$(date -d "-${i} weeks" +%Y-%m-%d)

    START_TS=$(date -d "${START_DATE} 00:00:00" +%s)
    END_TS=$(date -d "${END_DATE} 23:59:59" +%s)

    # Create parameters for this week (note: integers, not strings)
    cat > "/tmp/week-${i}.json" << EOF
{
  "time_range": {
    "ts_from": ${START_TS},
    "ts_to": ${END_TS},
    "step": 86400
  },
  "with_archive": true
}
EOF

    # Generate weekly report
    logzilla query --type EventRate \
        --params "/tmp/week-${i}.json" \
        --output-file "/reports/trends/week-${START_DATE}.json" \
        --authtoken "$API_TOKEN"

    rm "/tmp/week-${i}.json"
done

# Combine results for analysis
python3 /scripts/analyze-trends.py /reports/trends/
```

Compliance Reporting

Generate compliance reports for audit requirements.

PCI Compliance Report (pci-compliance.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DATE=$(date +%Y%m%d)
MONTH_START=$(date -d "$(date +%Y-%m-01)" +%Y-%m-%d)
MONTH_END=$(date -d "$(date +%Y-%m-01) +1 month -1 day" +%Y-%m-%d)

# Ensure output dir exists
mkdir -p /compliance

# Precompute epoch seconds
MONTH_START_TS=$(date -d "${MONTH_START} 00:00:00" +%s)
MONTH_END_TS=$(date -d "${MONTH_END} 23:59:59" +%s)

# Authentication events
cat > /tmp/auth-events.json << EOF
{
  "time_range": {
    "ts_from": ${MONTH_START_TS},
    "ts_to": ${MONTH_END_TS}
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["authentication OR login OR logon OR access"]
    }
  ],
  "limit": 10000,
  "sort": ["-first_occurrence"],
  "with_archive": true
}
EOF

# Generate authentication report
logzilla query --type Search \
    --params "/tmp/auth-events.json" \
    --output-file "/compliance/auth-events-${REPORT_DATE}.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

# Administrative access events
cat > /tmp/admin-events.json << EOF
{
  "field": "host",
  "limit": 50,
  "time_range": {
    "ts_from": ${MONTH_START_TS},
    "ts_to": ${MONTH_END_TS}
```

```
  },
  "filter": [
    {
      "field": "message",
      "op": "qp",
      "value": ["sudo OR su OR admin OR privilege"]
    }
  ],
  "subfields": ["program", "message"],
  "with_archive": true
}
EOF

logzilla query --type TopN \
    --params "/tmp/admin-events.json" \
    --output-file "/compliance/admin-access-${REPORT_DATE}.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/auth-events.json /tmp/admin-events.json
```

System Monitoring Automation

Performance Monitoring Dashboard

Automated system performance data collection for dashboards.

System Health Check (health-monitor.sh):

```
#!/bin/bash
set -euo pipefail

API_TOKEN="${LOGZILLA_API_KEY}"
TIMESTAMP=$(date +%Y%m%d-%H%M%S)
METRICS_DIR="/monitoring/metrics"
mkdir -p "$METRICS_DIR"

# CPU utilization (no exporter → redirect stdout)
echo '{"time_range": {"preset": "last_1_hours"}, "cpu": "totals"}' > /tmp/cpu.json
logzilla query --type System_CPU \
    --params /tmp/cpu.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/cpu-${TIMESTAMP}.json"

# Memory utilization
echo '{"time_range": {"preset": "last_1_hours"}}' > /tmp/memory.json
logzilla query --type System_Memory \
    --params /tmp/memory.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/memory-${TIMESTAMP}.json"

# Disk utilization
echo '{"time_range": {"preset": "last_1_hours"}, "fs": "root"}' > /tmp/disk.json
logzilla query --type System_DF \
    --params /tmp/disk.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/disk-${TIMESTAMP}.json"

# Event processing rate
echo '{"time_range": {"preset": "last_1_hours", "step": 300}}' > /tmp/events.json
logzilla query --type EventRate \
    --params /tmp/events.json \
    --authtoken "$API_TOKEN" \
    > "${METRICS_DIR}/events-${TIMESTAMP}.json"

rm -f /tmp/cpu.json /tmp/memory.json /tmp/disk.json /tmp/events.json

# Process metrics for alerting
python3 /scripts/process-metrics.py "${METRICS_DIR}" "$TIMESTAMP"
```

Capacity Planning Data

Collect data for capacity planning and growth analysis.

Capacity Analysis (capacity-planning.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
OUTPUT_DIR="/capacity-planning/$(date +%Y-%m)"
mkdir -p "$OUTPUT_DIR"

# Storage statistics for last 30 days (no exporter → redirect stdout)
echo '{"time_range": {"preset": "last_30_days"}}' > /tmp/storage.json
logzilla query --type StorageStats \
    --params /tmp/storage.json \
    --authtoken "$API_TOKEN" \
    > "${OUTPUT_DIR}/storage-stats.json"

# Event rate trends
echo '{
  "time_range": {
    "preset": "last_30_days",
    "step": 86400
  }
}' > /tmp/event-trends.json

logzilla query --type EventRate \
    --params /tmp/event-trends.json \
    --output-file "${OUTPUT_DIR}/event-trends.json" \
    --authtoken "$API_TOKEN"

# Top hosts by volume
echo '{
  "field": "host",
  "limit": 100,
  "time_range": {"preset": "last_30_days"},
  "with_archive": true
}' > /tmp/top-hosts.json

logzilla query --type TopN \
    --params /tmp/top-hosts.json \
    --output-file "${OUTPUT_DIR}/top-hosts.xlsx" \
    --format xlsx \
    --authtoken "$API_TOKEN"

rm /tmp/{storage,event-trends,top-hosts}.json
```

Integration Examples

SIEM Integration

Export LogZilla data for SIEM correlation and analysis.

SIEM Export Script (siem-export.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
SIEM_DIR="/siem-exports"
BATCH_SIZE=10000

# Export security events for SIEM
cat > /tmp/security-export.json << EOF
{
  "time_range": {
    "preset": "last_1_hours"
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [4]
    }
  ],
  "limit": ${BATCH_SIZE},
  "sort": ["-first_occurrence"]
}
EOF

# Export to JSON for SIEM ingestion
EXPORT_FILE="${SIEM_DIR}/security-events-$(date +%Y%m%d-%H%M).json"
logzilla query --type Search \
    --params /tmp/security-export.json \
    --output-file "$EXPORT_FILE" \
    --authtoken "$API_TOKEN"

# Convert to SIEM format if needed
if [ -f "$EXPORT_FILE" ]; then
    python3 /scripts/convert-to-siem-format.py "$EXPORT_FILE"

    # Send to SIEM via API or file transfer
    curl -X POST \
        -H "Content-Type: application/json" \
        -H "Authorization: token ${SIEM_API_TOKEN}" \
        --data-binary "@${EXPORT_FILE}" \
        "${SIEM_ENDPOINT}/api/events"
fi

rm /tmp/security-export.json
```

Ticketing System Integration

Automatically create tickets for critical events.

Ticket Creation (create-tickets.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
TICKET_THRESHOLD=10  # Create ticket if more than 10 critical events

# Check for critical events in last hour
cat > /tmp/critical-check.json << EOF
{
  "time_range": {
    "preset": "last_1_hours"
  },
  "filter": [
    {
      "field": "severity",
      "op": "le",
      "value": [2]
    }
  ],
  "limit": 1000
}
EOF

# Get critical events
RESULT_FILE="/tmp/critical-events-$(date +%Y%m%d-%H%M).json"
logzilla query --type Search \
    --params /tmp/critical-check.json \
    --output-file "$RESULT_FILE" \
    --authtoken "$API_TOKEN"

# Check event count and create ticket if threshold exceeded
EVENT_COUNT=$(jq '.results.totals.count // 0' "$RESULT_FILE")

if [ "$EVENT_COUNT" -gt "$TICKET_THRESHOLD" ]; then
    # Create ticket via API
    TICKET_DATA=$(cat << EOF
{
  "title": "Critical Events Alert - $EVENT_COUNT events detected",
  "description": "LogZilla detected $EVENT_COUNT critical events in the last hour. Please
investigate.",
  "priority": "high",
  "category": "system-alert",
  "attachment": "$RESULT_FILE"
}
EOF
)

    curl -X POST \
        -H "Content-Type: application/json" \
        -H "Authorization: token ${TICKETING_API_TOKEN}" \
```

```
        --data "$TICKET_DATA" \
        "${TICKETING_ENDPOINT}/api/tickets"
fi

rm /tmp/critical-check.json
```

Advanced Automation Patterns

Multi-Query Reports

Combine multiple queries for deeper analysis.

Multi-Query Report (comprehensive-report.sh):

```
#!/bin/bash

API_TOKEN="${LOGZILLA_API_KEY}"
REPORT_DIR="/reports/comprehensive/$(date +%Y%m%d)"
mkdir -p "$REPORT_DIR"

# Function to run query with error handling
run_query() {
    local query_type="$1"
    local params_file="$2"
    local output_file="$3"
    local format="${4:-json}"

    if logzilla query --type "$query_type" \
        --params "$params_file" \
        --output-file "$output_file" \
        --format "$format" \
        --authtoken "$API_TOKEN"; then
        echo "✓ Generated: $output_file"
    else
        echo "✗ Failed: $output_file"
        return 1
    fi
}

# Top hosts by volume
echo '{
  "field": "host",
  "limit": 20,
  "time_range": {"preset": "last_24_hours"}
}' > /tmp/top-hosts.json

run_query "TopN" "/tmp/top-hosts.json" "${REPORT_DIR}/top-hosts.xlsx" "xlsx"

# Error rate trends
echo '{
  "time_range": {
    "preset": "last_24_hours",
    "step": 3600
  },
  "filter": [{"field": "severity", "op": "le", "value": [4]}]
}' > /tmp/error-trends.json

run_query "EventRate" "/tmp/error-trends.json" "${REPORT_DIR}/error-trends.json"

# System performance
echo '{"time_range": {"preset": "last_24_hours"}, "cpu": "totals"}' > /tmp/cpu-perf.json
run_query "System_CPU" "/tmp/cpu-perf.json" "${REPORT_DIR}/cpu-performance.json"
```

```
# Generate summary report
python3 /scripts/generate-summary.py "$REPORT_DIR"

# Clean up
rm /tmp/{top-hosts,error-trends,cpu-perf}.json
```

Scheduled Automation

Set up cron jobs for regular automated reporting.

Crontab Configuration:

```
# Daily security report at 6 AM
0 6 * * * /scripts/security-report.sh

# Hourly system health check
0 * * * * /scripts/health-monitor.sh

# Weekly capacity planning report (Sundays at 2 AM)
0 2 * * 0 /scripts/capacity-planning.sh

# Monthly compliance report (1st of month at 3 AM)
0 3 1 * * /scripts/pci-compliance.sh
```

Best Practices for Automation

Error Handling

- **Check command exit codes** and handle failures gracefully
- **Validate JSON parameters** before executing queries
- **Log automation activities** for troubleshooting
- **Implement retry logic** for transient failures

Performance Optimization

- **Use specific time ranges** to limit query scope
- **Schedule resource-intensive queries** during off-peak hours
- **Implement query result caching** where appropriate
- **Monitor automation performance** and optimize as needed

Security and Maintenance

- **Protect API tokens** and configuration files
- **Rotate credentials regularly** for security
- **Monitor automation logs** for suspicious activity

- **Keep scripts and parameters updated** with system changes

Documentation and Monitoring

- **Document automation purposes** and dependencies
- **Monitor automation success rates** and performance
- **Implement alerting** for automation failures
- **Maintain version control** for automation scripts

These examples provide the foundation for building sophisticated LogZilla automation workflows. Adapt these patterns to the specific requirements and integrate them with the existing monitoring and reporting systems.

CHAPTER 12

Logzilla Ai

LogZilla Copilot AI assistant for natural language log querying, vendor-aware search, anomaly detection, and in-product configuration guidance

LogZilla Copilot is an AI-powered log analysis assistant integrated with LogZilla. It enables natural language querying of log data and returns results informed by the schema and tag taxonomy of every installed LogZilla application.

Key Capabilities

LogZilla Copilot provides the following log intelligence functionality:

- **Natural Language Log Querying**: Translate search requirements into conversational prompts
- **Vendor-Aware Query Context**: Apply the schema and tag taxonomy of every installed LogZilla application to produce accurate results on vendor-specific searches across FortiGate, SonicWall, Cisco, Palo Alto, and the full App Store catalog
- **Log Visualization**: Generate reports and data visualizations on request
- **System Management Assistance**: Provide guidance for LogZilla configuration
- **Documentation Integration**: Surface relevant documentation directly within conversations
- **Anomaly Detection Assistance**: Identify and explain unusual patterns in log data

Quick Access

Users can access LogZilla Copilot by clicking the `Copilot` link in the top navigation. The assistant interface opens in a new browser tab at the `/ui2/chat-ai` path of the LogZilla URL.

Use Cases

Log Analysis Acceleration

LogZilla Copilot significantly reduces time spent analyzing log data by allowing natural language queries and providing contextually relevant insights, enabling faster incident resolution.

Operational Efficiency

The system automates routine log analysis tasks, freeing technical staff to focus on higher-value activities while maintaining monitoring capabilities.

Knowledge Enhancement

LogZilla Copilot serves as both assistant and educator, helping teams understand log patterns and system behaviors through conversational interactions and explanations.

Implementation Considerations

Note: As with all AI systems, information provided by LogZilla Copilot should be verified before implementation. Users should exercise professional judgment when applying AI-generated suggestions.

LogZilla Copilot continually improves through regular updates to enhance accuracy, performance, and capabilities.

Installation

Enable the LogZilla AI Copilot from System Settings, configuring model provider, model name, and API key for OpenAI, Anthropic, Gemini, or self-hosted endpoints

Installation of LogZilla AI Copilot Module

The LogZilla Copilot module is now included in the standard LogZilla installation package, making the setup process much simpler than before.

Prerequisites

- LogZilla instance (minimum version v6.36)
- Access to a suitable LLM provider (OpenAI, Anthropic, Google Gemini, OpenRouter, or a self-hosted OpenAI-compatible endpoint such as vLLM or Ollama)

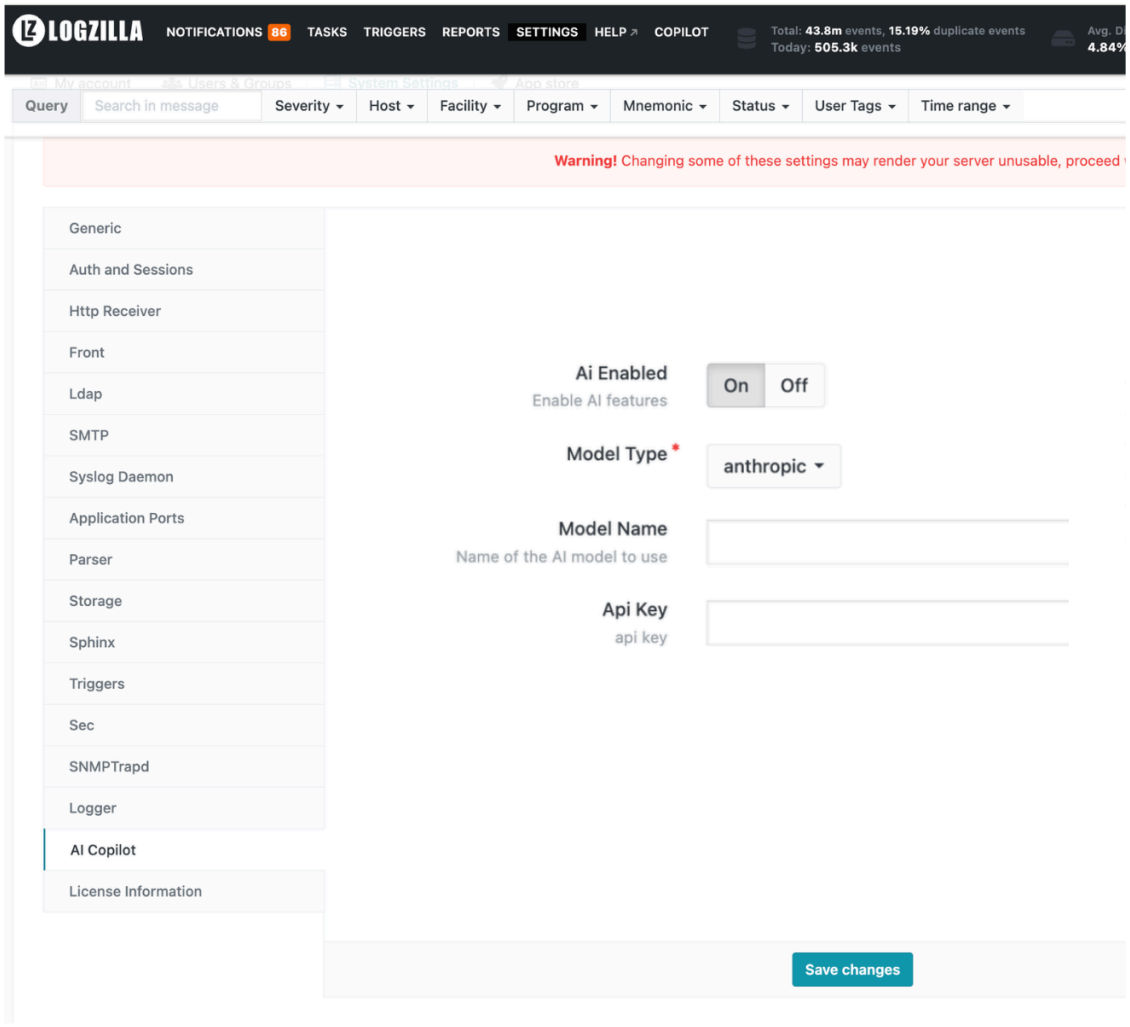
Enabling the AI Copilot

1. Open the AI Copilot settings

From the top navigation, open `Settings` and select `System Settings`. In the sidebar, click `AI Copilot`.

2. Configure the required fields

Four fields are required to enable the Copilot: **Ai Enabled**, **Model Type**, **Model Name**, and **Api Key**. The remaining fields are optional and primarily used for tuning or self-hosted deployments.



Required fields:

- **Ai Enabled**: Set to `On` to activate the AI Copilot feature.
- **Model Type**: Select the AI provider. Supported values are `openai`, `anthropic`, `gemini`, `vllm/ollama`, and `openrouter`. The `vllm/ollama` option covers any OpenAI-compatible endpoint, including vLLM and Ollama deployments.
- **Model Name**: The specific model to use. Examples: `gpt-5.2`, `claude-opus-4-6`, `claude-sonnet-4-6`, `llama3`.
- **Api Key**: The API key for the selected provider.
- **Query Loglevel**: Controls logging verbosity for AI query operations. Defaults to `DISABLED`; set to `INFO` or `DEBUG` when investigating an issue.

Optional fields:

- **Api Base Url**: Base URL for the AI API provider. Required for self-hosted deployments such as vLLM or Ollama (for example, `http://host:11434/v1` for an Ollama server).
- **Max Tokens**: Maximum number of tokens the model produces per generation. (default: 8192, minimum: 8192)

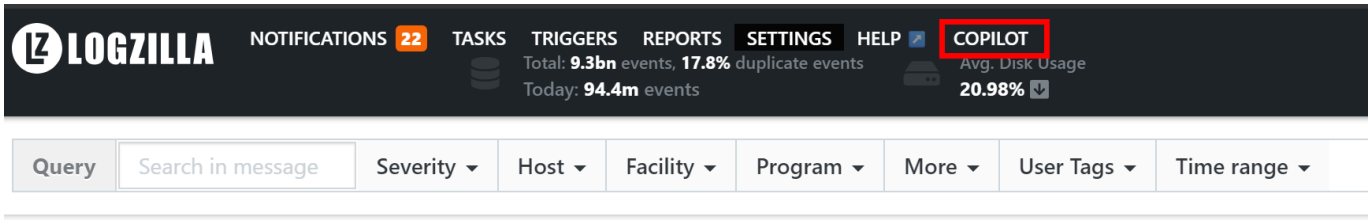
- **Context Window**: Maximum number of context tokens the model accepts. Some models require this parameter; for others it is ignored. (default: unset, minimum when set: 100000)
- **Request Timeout**: Maximum number of seconds to wait for the LLM to respond before the request is considered timed out. (default: 300, range: 30-3600)

3. Save Settings

Click `Save changes` to apply. The LogZilla UI restarts automatically.

4. Access the AI Copilot

After a few seconds, the `Copilot` link appears in the top navigation. Clicking it opens the AI Copilot interface in a new browser tab at the `/ui2/chat-ai` path of the LogZilla URL.



Advanced settings

The AI Copilot pane also exposes several advanced fields. These are hidden by default and surface when the "Show advanced settings" toggle is enabled on the Settings page. Most deployments do not need to change them.

- **Logzilla Mcp Docs Url**: Base URL for the LogZilla MCP documentation server. Defaults to `https://mcp-docs.logzilla.ai/mcp`.
- **Extra Mcp Urls**: Additional MCP servers to make available to the Copilot. Defaults to empty.
- **Tracing Enabled**: When on, AI request traces are exported to the configured tracing server. Defaults to off.
- **Tracing Project Name**: Project label applied to exported traces. Defaults to `lz-ai-tracking`.
- **Tracing Server Url**: Endpoint of the external tracing collector.
- **Tracing Api Key**: API key used when the tracing server requires authentication.

For additional information, consult the LogZilla documentation or contact LogZilla support.

Using Copilot

Learn how to leverage LogZilla Copilot's capabilities for efficient log analysis, natural language querying, and advanced log management

Overview

LogZilla Copilot is an integrated AI assistant for log analysis in LogZilla. It combines natural language processing with access to LogZilla's query engine, allowing operators to run log searches and analysis through conversational prompts.

Features

1. Natural Language Log Query Processing

- Query LogZilla data using everyday language without specialized syntax
- Extract information from events, statistics, and system status
- Run searches and filters through conversational requests
- Identify patterns across distributed log sources
- Produce vendor-aware results by applying the schema and tag taxonomy of every installed LogZilla application, covering FortiGate, SonicWall, Cisco, Palo Alto, and other supported platforms

2. System Information and Assistance

- Understand incoming log data through natural language queries
- Obtain explanations about system configurations and settings
- Receive guidance on LogZilla features and functionality
- Access troubleshooting information for common system issues

3. Contextual Documentation Integration

- Instant access to relevant LogZilla documentation sections on demand
- Context-aware assistance based on current system state and user activity
- Configuration recommendations drawn from LogZilla documentation
- Step-by-step troubleshooting guidance for analytics scenarios

4. Log Data Presentation

- Present query results in organized, readable formats
- Display log data in structured tables when requested
- Format results to highlight relevant information
- Assist with interpreting log patterns and events

Intuitive User Interface

Efficient Conversation Management

- **Left Sidebar**:
  - `New Conversation` button to start fresh interactions
  - List of existing conversations
  - Edit conversation titles (click pen icon)
  - Delete conversations (click trash icon)
  - Switch between conversations by clicking

Main Interface

- **Chat Area**:
  - Displays conversation history
  - Shows user questions and AI responses
  - Supports markdown formatting for better readability
  - Includes interactive charts and tables when requested

Control Elements

- **Input Field**: Located at bottom for entering questions
- **Top Right Corner**:
  - User information display
  - Current version information
  - LLM model selection options

Practical Usage Guide

1. Effective Log Query Examples

```
"Show error events from the last hour"
"How many events occurred in the last 24 hours?"
"What does this error pattern mean?"
"Help me understand these SSH failure logs"
"Where can I find information about configuring alerts?"
"Show all critical error events from the last hour regarding authentication"
"Display total event count and trend analysis from the last 24 hours"
"Create an automated rule to forward security-related events to Splunk"
"Analyze and explain these unusual SSH failure patterns across servers"
"Identify potential security incidents in the last 24 hours"
```

2. Core Capabilities

LogZilla Copilot assists with log management through:

- Event search and pattern analysis using natural language
- Access to LogZilla documentation and knowledge resources
- System information and configuration guidance
- Log data visualization and interpretation
- Basic troubleshooting assistance

3. Operational Best Practices

- Specific, targeted queries produce more actionable log analysis results
- Including system context and timeframes significantly improves accuracy
- Strategic follow-up questions help refine understanding of complex events
- Leveraging conversation history provides continuity for extended troubleshooting sessions
- Starting with broader queries before narrowing focus often yields better results

LogZilla Integration

- Access to LogZilla's query engine for log data retrieval
- Shared authentication with the main LogZilla system
- Integration with LogZilla documentation resources
- Available through the LogZilla navigation menu

Implementation Considerations

- AI-generated log analysis requires verification before critical implementation
- Response times vary based on query complexity and data volume
- Review of generated suggestions remains essential before production deployment
- Professional judgment should accompany all AI-assisted operations

Usage Tips

- Including specific time ranges in queries helps produce more relevant results
- Managing conversations by creating new ones for different topics improves organization
- Clear, concise questions typically yield better responses
- Following up with clarifying questions helps refine results

© LogZilla Corporation · logzilla.ai

CHAPTER 13

Logzilla Cloud

LogZilla Cloud managed log management with real-time search, dashboards, alerting, and API access, provisioned on the logzilla.cloud domain

LogZilla Cloud is a fully managed log management platform that provides real-time search, dashboards, alerting, and API access without the complexity of self-hosted infrastructure. Organizations can start collecting, analyzing, and acting on log data within minutes of signup.

Key Benefits

- **Zero Infrastructure Management**: No servers to maintain, patch, or scale
- **Instant Setup**: Begin ingesting logs immediately after provisioning
- **Enterprise Security**: SOC 2 compliant with encrypted data transmission and storage
- **Scalable Ingestion**: Handle everything from small deployments to enterprise-scale log volumes
- **Advanced Analytics**: Real-time search, custom dashboards, and intelligent alerting
- **API-First Design**: Full REST API for integration with existing tools and workflows

Getting Started

The LogZilla Cloud journey involves two main phases:

1. Purchase and Provisioning

Learn how to select the right service tier, choose a hostname, complete the purchase process, and understand what to expect during instance provisioning.

→ **Ordering and Checkout**

2. Onboarding and Configuration

Complete the initial login, secure the account, configure log ingestion from Linux hosts, set up relays for network devices, and verify that events are flowing correctly.

→ **Onboarding LogZilla Cloud**

What's Included

LogZilla Cloud instances come pre-configured with:

- **Real-Time Search**: Lightning-fast search across millions of events with advanced filtering and query capabilities
- **Interactive Dashboards**: Customizable dashboards with multiple widget types including charts, graphs, tables, and metrics
- **Intelligent Parsing**: Automatic parsing and normalization for 20+ enterprise vendors (Cisco, Juniper, Palo Alto, etc.)
- **Advanced Alerting**: Trigger-based alerting with email, Slack, and webhook notifications plus automated script execution
- **Data Transformation**: User tags, rewrite rules, and custom field extraction for enhanced log enrichment
- **AI Assistant (Copilot)**: Natural language interactions with log data for analysis and troubleshooting
- **Comprehensive API**: Full REST API access for integrations, automation, and custom applications

- **Compliance Ready**: Configurable retention policies, audit trails, and data export capabilities
- **App Store Integration**: Pre-built applications and dashboards for specific use cases and vendors

Note: Some features may only be available on certain service tiers. See the LogZilla Cloud website for current tier details and feature availability.

Support and Resources

- **Documentation**: Comprehensive guides for configuration, troubleshooting, and advanced features
- **Support Channels**: Email and priority support based on service tier
- **Slack**: Access to LogZilla Slack community for support based on service tier

Ready to get started? Begin with Getting LogZilla Cloud to learn about service tiers and the purchase process.

Ordering And Checkout

Order a LogZilla Cloud instance on the logzilla.cloud domain, choose a hostname, select a service tier, and complete checkout for managed log management

LogZilla Cloud provides hosted log management with real-time search, dashboards, alerting, and an API. Instances are provisioned under the `logzilla.cloud` domain and include an ingest-only token for secure event submission over HTTPS.

Getting Started

To purchase LogZilla Cloud:

Visit https://www.logzilla.net/cloud

Review the available service tiers and current pricing

Select the tier that fits operational requirements

Choose a hostname and proceed to subscription

Service tiers vary by ingestion volume, storage capacity, features, and support level. Current tier details and pricing are available on the LogZilla Cloud website.

Choose a Hostname

- Hostname will be of the form `<chosen>.logzilla.cloud`.
- Constraints:
  - 3-63 characters.
  - Letters (a-z, A-Z), digits (0-9), and hyphens (-) only.
  - Cannot start or end with a hyphen.

Purchase and Payment

- Payments are processed by a secure third-party provider (Stripe).
- A redirect to the provider may be required for authentication.
- LogZilla does not store payment details; they are handled by the payment provider.

Provisioning Emails

After subscribing:

- Email 1: "Provisioning Started for The LogZilla Instance at `<hostname>`.logzilla.cloud."
- Email 2 (approximately 10 minutes later): "The LogZilla Instance is Ready at `<hostname>`.logzilla.cloud."

The readiness email typically includes:

- Instance URL (note that if the one chosen was unavailable, this may be slightly different than the chosen hostname)
- Admin password
- API ingest token (ingest-only token)
- Full API token (for appropriate service tiers)

Next Steps

Proceed to onboarding to perform the first login, secure the account, and begin sending events:

- See Cloud Onboarding.

Cloud Onboarding

Onboard a new LogZilla Cloud tenant by signing in, rotating the temporary password, and configuring syslog-ng HTTPS ingest from a Linux host

After a LogZilla Cloud instance is provisioned, the first steps involve logging in, updating the password, configuring ingestion from a Linux host using syslog-ng over HTTPS, and verification. Additional resources are provided below.

Prerequisites

- Instance URL, username, and temporary password (from the provisioning email)
- API ingest token (ingest-only token) for HTTPS event submission

First Login

Navigate to the instance URL provided in the onboarding email (note that if the one chosen was unavailable, this may be slightly different than the chosen hostname).

Sign in using the provided username and temporary password.

Update Password

Open `Settings` and change the password:

- Enter the current password.
- Enter and confirm a new password.

Store the new password securely.

Ingest-Only Token

- The provisioning email includes an API ingest token (prefixed with `ingest-`).
- This token is used to authenticate HTTPS ingestion to `/incoming`.
- Security: Treat the token like a password. Rotate it if compromised and update any senders.
- More information about tokens can be found in the API documentation.

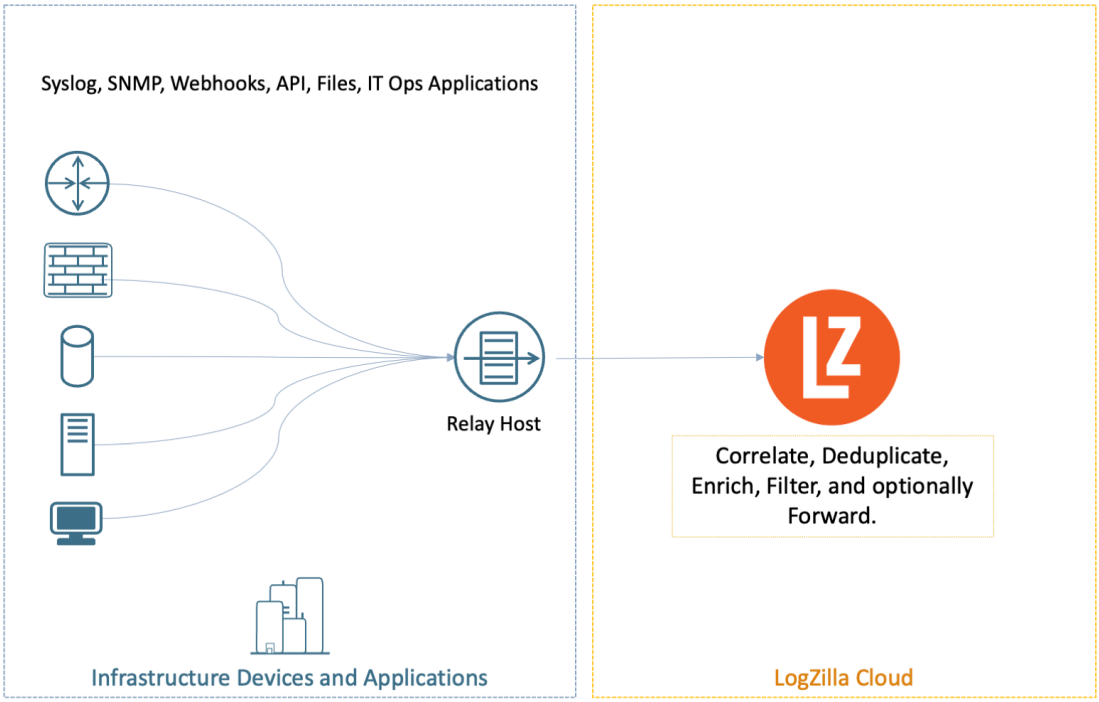
Set Up a Relay

A relay receives events from devices and applications (typically via syslog on UDP/TCP 514) and forwards them securely to LogZilla Cloud over HTTPS using the ingest-only token.

- Devices and apps send syslog to the relay.

- The relay transforms/encapsulates events as JSON and posts to `https://YOUR-HOST.logzilla.cloud/incoming` with header `Authorization: token YOUR_GENERATED_TOKEN`.
- Common choices: syslog-ng or rsyslog. Either can run on an existing Linux host or a small VM/container.

Relay flow overview:



Network and security checklist:

- Allow device-to-relay syslog (UDP/TCP 514) as appropriate.
- Allow relay outbound HTTPS (TCP 443) to `YOUR-HOST.logzilla.cloud`.
- Protect the ingest-only token; rotate it if exposed and update the relay configuration.

The system logger must be configured to relay events to LogZilla Cloud. The installation process varies depending on the system logger being used. To determine which syslog daemon is currently in use, run:

```
curl -s https://logzilla.sh/detect-syslog-env.sh | bash
```

Note: to receive logs from other non-linux devices, such as networking equipment, those devices will need to be configured to send logs via the RFC 3164 or RFC 5424 syslog specification to this linux server (or whichever is using the system logger configuration indicated below).

Data transformation and user tags are documented in Data Transforms / Rewrite Rules. This step is optional and can be skipped when user tag metadata is not needed at this stage.

Option A: syslog-ng relay (HTTPS to /incoming)

Add an HTTP destination to the syslog-ng configuration to send structured JSON to LogZilla Cloud.

Create Configuration File

`sudo vi /etc/syslog-ng/conf.d/60-logzilla-cloud.conf` (or `sudo nano ...`)

Destination and Log Path

Add a destination and a log path similar to the following. Replace placeholders before saving.

```
destination d_logzilla {
    http(
        url("https://YOUR-HOST.logzilla.cloud/incoming")
        method("POST")
        user-agent("syslog-ng User Agent")
        headers(
            "Content-Type: application/json",
            "Authorization: token YOUR_GENERATED_TOKEN"
        )
        body-prefix("{\"events\": [\n")
        delimiter(",\n")
        body('$(format-json
            --pair priority=int($PRI)
            --pair host="$HOST"
            --pair program="$PROGRAM"
            --pair message="$MESSAGE"
        )')
        body-suffix("\n]}")
        batch-lines(10000)
        batch-bytes(10485760)
        batch-timeout(500)
     );
};
```

Add User Tags (Optional)

Arbitrary metadata can be attached as user tags directly from syslog-ng using additional `--pair` entries under a `user_tags.` prefix. Example lines to include inside `$(format-json ...)`:

```
            --pair user_tags.Location="SJC-1"
            --pair user_tags.Contact="noc@example.com"
            --pair user_tags.Asset_ID="WEB-0227"
```

Replace Placeholders

- `YOUR-HOST` → chosen instance hostname.
- `YOUR_GENERATED_TOKEN` → ingest-only token from the provisioning email.

Validate and Restart syslog-ng

```
syslog-ng --syntax-only
sudo systemctl restart syslog-ng
```

Notes:

- For very high throughput, prefer a syslog-ng relay with batching.
- When `omhttp` is not available, forward rsyslog → syslog-ng locally and use Option A for the HTTPS JSON destination to `/incoming`.

Option B: rsyslog relay (HTTPS to /incoming)

If rsyslog is preferred or already standard, it can post events to LogZilla Cloud using the `omhttp` output module. Availability of `omhttp` depends on the distribution; if it is unavailable, forward rsyslog locally to a syslog-ng relay and use Option A for HTTPS.

Example Configuration

```
# /etc/rsyslog.d/60-logzilla-cloud.conf
module(load="imuxsock")
module(load="imklog")
module(load="omhttp")  # ensure this module/package is installed on the OS

# Minimal JSON envelope per event
template(name="lz_event_json" type="list") {
  constant(value="{\"events\":[{")
  constant(value="\"priority\":") property(name="pri")
  constant(value=",\"host\":\"") property(name="hostname") constant(value="\"")
  constant(value=",\"program\":\"") property(name="programname") constant(value="\"")
  constant(value=",\"message\":\"") property(name="msg") constant(value="\"")
  constant(value=",\"user_tags\":{\"Location\":\"SJC-
1\",\"Contact\":\"noc@example.com\",\"Asset_ID\":\"WEB-0227\"}}")
  constant(value="}]} ")
}

# Post each event to LogZilla Cloud
action(
  type="omhttp"
  server="YOUR-HOST.logzilla.cloud"
  serverport="443"
  usehttps="on"
  restpath="/incoming"
  header="Content-Type: application/json"
  header="Authorization: token YOUR_GENERATED_TOKEN"
  template="lz_event_json"
  # reliability queue (tune as needed)
  action.resumeRetryCount="-1"
  queue.type="LinkedList"
  queue.size="10000"
)
```

Validate and Restart rsyslog

```
sudo rsyslogd -N1
sudo systemctl restart rsyslog
```

Notes

- For very high throughput, prefer a syslog-ng relay with batching.
- When `omhttp` is not available, forward rsyslog → syslog-ng locally and use Option A for the HTTPS JSON destination to `/incoming`.

Add User Tags with rsyslog (Optional)

Rsyslog can embed a `user_tags` object in the JSON payload by extending the template. For example:

```
template(name="lz_event_json_with_tags" type="list") {
  constant(value="{\"events\":[{")
  constant(value="\"priority\":") property(name="pri")
  constant(value=",\"host\":\"") property(name="hostname") constant(value="\"")
  constant(value=",\"program\":\"") property(name="programname") constant(value="\"")
  constant(value=",\"message\":\"") property(name="msg") constant(value="\"")
  constant(value=",\"user_tags\":{\"Location\":\"SJC-
1\",\"Contact\":\"noc@example.com\",\"Asset_ID\":\"WEB-0227\"}}")
  constant(value="}]} ")
}

# Then reference this template in the omhttp action
action(
  type="omhttp"
  server="YOUR-HOST.logzilla.cloud"
  serverport="443"
  usehttps="on"
  restpath="/incoming"
  header="Content-Type: application/json"
  header="Authorization: token YOUR_GENERATED_TOKEN"
  template="lz_event_json_with_tags"
)
```

Verify Ingestion

Send Test Event

From the Linux host, send a test event:

```
logger "Test LogZilla log message"
```

Verify with Curl

Optionally, verify end-to-end with curl (replace placeholders):

```
curl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: token YOUR_GENERATED_TOKEN' \
  -X POST -d '{
    "events": [ {
      "message": "Test Message",
      "host": "curl.test",
      "program": "myapp"
    } ] }' \
  'https://YOUR-HOST.logzilla.cloud/incoming'
```

Confirm in UI

In the LogZilla UI, confirm reception:

- Events/Sec widget shows bars for received events.
- Most Recent Event Sources lists the Linux host.
- Search for the message (for example, `message:"Test LogZilla log message"`).

SaaS Administration

- Location: `https://YOUR-HOST.logzilla.cloud/settings/system/saas`.
- Manage subscription: opens the Stripe customer portal to update or cancel the subscription and manage payment methods.
- Current tier panel: shows plan name, disk space, and daily event limits.
- Change domain/hostname:
  - Request a new domain on the SaaS settings page and submit.
  - The instance will stop and start under the new domain; this typically takes a couple of minutes.
  - Update all senders and relays to the new hostname (`YOUR-HOST.logzilla.cloud`) after the change.

Other Sources

- Forwarding from non-Linux devices via relays: Relays.
- Windows event ingestion using the LogZilla Syslog Agent: LogZilla Syslog Agent

CHAPTER 14

Kubernetes Deployment

LogZilla Kubernetes reference architecture for Postgres, Redis, InfluxDB, Storage, Query, API, Ingest, and Front modules with ingress exposure guidance

When to use Kubernetes

- Kubernetes is recommended for sustained ingest above 10B events per day (EPD).
- Kubernetes is required for deployments exceeding 20B EPD.

Supported platforms

- Any conformant Kubernetes distribution is supported.
- Examples in this section include Google Kubernetes Engine (GKE), but the manifests are provider agnostic unless noted.

Image tags

- Use the `stable` tag by default for production deployments.
- If a release pinning policy exists, use the specific release tag (for example, `vX.Y.Z`).
- Avoid `unstable` in production. The `latest` tag is generally used for development workflows only.

Component map

- Backing services: PostgreSQL, Redis, InfluxDB (optional: Grafana).
- LogZilla components: Storage, Query, API (Gunicorn, Tornado, Celery worker), Ingest (syslog-ng, ParserModule, HTTP Receiver), and Front (NGINX UI).
- Ingress/Exposure: LoadBalancer Services for syslog/json/rfc5424/http ingest and an HTTP Ingress for UI/API/WebSocket.

What to do next

- GKE Quickstart:
  - GKE Quickstart
- Review prerequisites and cluster requirements:
  - Prerequisites
- Deploy backing services:
  - PostgreSQL
  - Redis
  - InfluxDB
- Configure shared ConfigMaps/Secrets:
  - Common Config and Secrets
- Deploy LogZilla components:

    - Storage Module
    - Query Module
    - API Module
    - Ingest Module
    - Front End
- Expose services:
    - Ingress (GKE example)
- Operate and tune:
    - Scaling and Tuning
    - Troubleshooting

Gke Quickstart

Bring up LogZilla on Google Kubernetes Engine using the repository helper scripts and manifests, including gcloud authentication and cluster creation

Purpose

This quickstart helps a technical operator new to Kubernetes bring up LogZilla on Google Kubernetes Engine (GKE) using the repository's existing helper scripts and manifests.

Prerequisites

- Google Cloud project with billing enabled.
- gcloud CLI and kubectl installed and on PATH.
- Authentication and project selection performed:

```
# Authenticate gcloud and select the project
gcloud auth login

# The scripts default to the project "logzilla-v7" and region "us-east1".
# If using a different project or region, edit the script noted below.
gcloud config set project logzilla-v7
```

Create a GKE cluster (scripted)

The repository provides `k8s/gcloud_scripts/create_cluster`. The quickstart uses the script as-is (project "logzilla-v7", region "us-east1").

```
# Optional: review or edit the script before running
sed -n '1,200p' k8s/gcloud_scripts/create_cluster

# Create a cluster named lz-cluster with 3 nodes (defaults in the script)
bash k8s/gcloud_scripts/create_cluster lz-cluster 3
```

Script defaults to be aware of (file: `k8s/gcloud_scripts/create_cluster`):

- Line 6: `--project "logzilla-v7"`
- Line 8: `--region "us-east1"`
- Lines 23-24: Default VPC and subnetwork values referencing the same project

If those do not match the target environment, edit the script accordingly before running.

Configure kubectl context

```
# Fetch cluster credentials for kubectl
gcloud container clusters get-credentials lz-cluster \
  --region us-east1 \
  --project logzilla-v7

# Verify connectivity
kubectl get nodes
```

Optional: create a dedicated namespace

```
kubectl create namespace logzilla
# Remember to add: -n logzilla to subsequent kubectl commands
```

Deploy backing services

Manifests are provided in `k8s/services/`.

```
kubectl apply -f k8s/services/postgres.yaml
kubectl apply -f k8s/services/redis.yaml
kubectl apply -f k8s/services/influxdb.yaml
# Optional dashboard
kubectl apply -f k8s/services/grafana.yaml
```

Deploy common config and LogZilla modules

Manifests are provided in `k8s/modules/`.

```
kubectl apply -f k8s/modules/common.yaml
kubectl apply -f k8s/modules/storage.yaml
kubectl apply -f k8s/modules/query.yaml
kubectl apply -f k8s/modules/api.yaml
kubectl apply -f k8s/modules/ingest.yaml
kubectl apply -f k8s/modules/front.yaml
```

Expose services (GKE example)

```
kubectl apply -f k8s/ingress/ingress-gke.yaml
```

This creates LoadBalancer Services for syslog (TCP/UDP), JSON, RFC5424, and HTTP ingest, and an Ingress routing the UI (`/`), static assets, API (`/api/`), WebSocket (`/ws/`), and `/incoming`.

Verify

```
kubectl get pods
kubectl get svc
kubectl describe ingress ingress-gke
```

Notes

- LogZilla images should use the `stable` tag for production. If a release pinning policy exists, use `vX.Y.Z`.
- Secrets in the docs use `stringData:` for readability. If using `data:`, base64-encode values first.

Cleanup

Scripts are provided and should be reviewed before running, as they are potentially destructive.

```
# Delete the cluster (uses region us-east1 and project logzilla-v7)
sed -n '1,200p' k8s/gcloud_scripts/delete_cluster
bash k8s/gcloud_scripts/delete_cluster lz-cluster

# Delete leftover disks in the project (uses region us-east1)
sed -n '1,200p' k8s/gcloud_scripts/delete_disks
bash k8s/gcloud_scripts/delete_disks
```

Prerequisites

Kubernetes cluster requirements for a LogZilla deployment, covering LoadBalancer services, Ingress controllers, StorageClasses, image registries, and namespaces

Cluster requirements

- LoadBalancer Service support for external syslog and HTTP ingest.
- An Ingress controller compatible with `networking.k8s.io/v1`.
- A default `StorageClass` or a named `StorageClass` for PersistentVolumeClaims.
- Sufficient node capacity for the requested CPU and memory in manifests.

Registry and images

- Pull access to LogZilla images.
- Use the `stable` tag by default for production. If a release pinning policy exists, use a specific tag (for example, `vX.Y.Z`).
- Third-party images (PostgreSQL, Redis, InfluxDB, Grafana) use stable tags in the examples.

Namespace

If using a dedicated namespace:

```
kubectl create namespace logzilla
# add -n logzilla to subsequent kubectl commands
```

Secrets: stringData vs data

- Examples in this section use `stringData` for Secrets to keep manifests readable; Kubernetes encodes them to base64 on apply.
- If using `data:` instead, base64-encode values first. Example:

```
printf '%s' 'your_token_here' | base64
```

StorageClass notes

- In some manifests, a `storageClassName` is referenced (for example, `premium-rwo`). Replace it with the cluster's StorageClass or remove the field if the default StorageClass is desired.

Order of deployment

Backing services (PostgreSQL, Redis, InfluxDB, optional Grafana)

Common ConfigMaps and Secrets

LogZilla modules (Storage, Query, API, Ingest, Front)

Ingress and external exposure

Refer to each page in this section for copy-and-paste manifests.

Postgres

PostgreSQL Kubernetes manifest for LogZilla, defining the Secret, ConfigMap, PersistentVolumeClaim, StatefulSet, and Service required by the API module

PostgreSQL manifest

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
stringData:
  POSTGRES_USER: postgres
  POSTGRES_PASSWORD: <set-strong-password>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-multimap
data:
  POSTGRES_DB: logzilla
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: postgres
  selector:
    matchLabels:
      name: postgres
  template:
    metadata:
      labels:
        name: postgres
    spec:
      containers:
      - name: postgres
        image: library/postgres:17.2-alpine
        args:
          - postgres
          - -c
```

```
        - max_connections=500
      resources:
        requests:
          memory: 500Mi
          cpu: 500m
        limits:
          memory: 1Gi
          cpu: 1000m
      ports:
      - containerPort: 5432
      livenessProbe:
        exec:
          command: [
            "pg_isready",
            "-U", "$(POSTGRES_USER)",
            "-d", "$(POSTGRES_DB)"
          ]
        failureThreshold: 5
        periodSeconds: 1
        timeoutSeconds: 15
      readinessProbe:
        exec:
          command: [
            "pg_isready",
            "-U", "$(POSTGRES_USER)",
            "-d","$(POSTGRES_DB)"
          ]
        initialDelaySeconds: 20
        timeoutSeconds: 5
        periodSeconds: 3
        failureThreshold: 2
        successThreshold: 1
      volumeMounts:
      - name: postgres
        mountPath: /var/lib/postgresql/
        readOnly: false
      env:
      - name: POSTGRES_PASSWORD
        valueFrom:
          secretKeyRef:
            name: postgres-secret
            key: POSTGRES_PASSWORD
      - name: POSTGRES_USER
        valueFrom:
          secretKeyRef:
            name: postgres-secret
            key: POSTGRES_USER
      - name: POSTGRES_DB
        valueFrom:
          configMapKeyRef:
```

```
              name: postgres-multimap
              key: POSTGRES_DB
      volumes:
        - name: postgres
          persistentVolumeClaim:
            claimName: postgres-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  ports:
    - port: 5432
      targetPort: 5432
      protocol: TCP
  type: ClusterIP
  selector:
    name: postgres
```

Notes

- Replace placeholder values in `postgres-secret` with environment-specific strong credentials. Examples use `stringData` for readability.
- If you prefer `data:` instead of `stringData`, base64-encode values.
- The readiness/liveness probe uses variables for database name and user. Ensure the variable names match the environment (for example, `POSTGRES_DB`).

Redis

Redis Kubernetes manifest for LogZilla, providing PersistentVolumeClaim, StatefulSet, and Service definitions for caching and queue workloads

Redis manifest

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
spec:
  serviceName: redis
  selector:
    matchLabels:
      name: redis
  template:
    metadata:
      labels:
        name: redis
    spec:
      containers:
      - name: redis
        image: library/redis:7.4.0-alpine
        args: ["redis-server", "--loglevel", "warning"]
        resources:
          requests:
            memory: 500Mi
            cpu: 500m
          limits:
            memory: 2Gi
            cpu: 1000m
        ports:
        - containerPort: 6379
        livenessProbe:
          exec:
            command:
              - redis-cli
              - --raw
              - incr
              - ping
          failureThreshold: 5
          periodSeconds: 1
```

```
          timeoutSeconds: 15
        volumeMounts:
        - name: redis
          mountPath: /data
          readOnly: false
      volumes:
        - name: redis
          persistentVolumeClaim:
            claimName: redis-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
    - port: 6379
      targetPort: 6379
      protocol: TCP
  type: ClusterIP
  selector:
    name: redis
```

Notes

- Adjust the PVC size and StorageClass as required by the environment.

Influxdb

InfluxDB Kubernetes manifest for LogZilla metrics, including Secret, ConfigMap, PersistentVolumeClaim, StatefulSet, and Service definitions

InfluxDB manifest

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: influxdb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: v1
kind: Secret
metadata:
  name: influxdb-secret
type: Opaque
stringData:
  INFLUXDB_ADMIN_USER: logzilla
  INFLUXDB_ADMIN_PASSWORD: <set-strong-password>
  INFLUXDB_USER: user
  INFLUXDB_USER_PASSWORD: <set-strong-password>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: influxdb-multimap
data:
  INFLUXDB_DATA_MAX_VALUES_PER_TAG: "10000"
  INFLUXDB_DATA_MAX_SERIES_PER_DATABASE: "0"
  INFLUXDB_DATA_INDEX_VERSION: inmem
  INFLUXDB_LOGGING_FORMAT: json
  INFLUXDB_LOGGING_LEVEL: warn
  INFLUXDB_COORDINATOR_LOG_QUERIES_AFTER: 10s
  INFLUXDB_HTTP_LOG_ENABLED: "false"
  INFLUXDB_HTTP_ACCESS_LOG_PATH: ""
  INFLUXDB_UDP_ENABLED: "true"
  INFLUXDB_UDP_BIND_ADDRESS: :8086
  INFLUXDB_UDP_DATABASE: lzcounters
  INFLUXDB_UDP_BATCH_TIMEOUT: 1s
  INFLUXDB_UDP_0_BATCH_SIZE: "5000"
  INFLUXDB_UDP_BATCH_SIZE: "0"
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: influxdb
spec:
```

```
  serviceName: influxdb
  selector:
    matchLabels:
      name: influxdb
  template:
    metadata:
      labels:
        name: influxdb
    spec:
      containers:
      - name: influxdb
        image: library/influxdb:1.8.10-alpine
        resources:
          requests:
            memory: 1Gi
            cpu: 500m
          limits:
            memory: 2Gi
            cpu: 1000m
        ports:
        - containerPort: 8086
          name: api
          protocol: TCP
        - containerPort: 8086
          name: udp-counters
          protocol: UDP
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /health
            port: api
            scheme: HTTP
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /health
            port: api
            scheme: HTTP
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        envFrom:
          - configMapRef:
              name: influxdb-multimap
          - secretRef:
```

```
              name: influxdb-secret
        volumeMounts:
        - name: influxdb
          mountPath: /var/lib/influxdb/
          readOnly: false
      volumes:
        - name: influxdb
          persistentVolumeClaim:
            claimName: influxdb-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: influxdb
spec:
  ports:
    - port: 8086
      name: api
      targetPort: 8086
      protocol: TCP
    - port: 8086
      name: udp-counters
      targetPort: 8086
      protocol: UDP
  type: ClusterIP
  selector:
    name: influxdb
```

Notes

- Replace placeholder values with environment-specific credentials. Examples use `stringData` for readability. If you prefer `data:`, base64-encode values.
- Adjust PVC size and StorageClass as needed.

Common Config And Secrets

Shared LogZilla Kubernetes ConfigMaps and Secrets, including Django license server settings, log levels, and credentials consumed by multiple modules

ConfigMaps and Secrets

The following manifests define shared ConfigMaps and Secrets required by multiple LogZilla components. Replace placeholder values before applying.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: django-multimap
data:
  LICENSE_SERVER_BASE_URL: https://license.logzilla.net/
  CUSTOMER_INFO: k8s_customer
  INTERNAL_CUSTOMER: "1"
  EULA_ACCEPTED: '1'
  SYSTEM_METRICS_ENABLED: "0"
  QUERYMODULE_API_URL: http://querymodule:81/
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: log-multimap
data:
  LOG_LEVEL: INFO
  IE_LOG_LEVEL: WARNING
  IC_LOG_LEVEL: DEBUG
  LOG_THREAD_INFO: "1"
  IE_TARGET_URL: http://storage-0.storage:81/internal-events
  PYTHONASYNCIODEBUG_LOG_LEVEL: DISABLED
  PYTHONASYNCIODEBUG_SLOW_CALLBACK_DURATION: "0.5"
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: module-multimap
data:
  MODULE_WAIT_FOR_API_TIMEOUT: '60'
  MODULE_DOWNLOAD_LICENSE: '1'
  IA_BASE_URL: http://api/api/
  LOG_API_TIMERS: "0"
  PYTHONOPTIMIZE: "1"
---
apiVersion: v1
kind: Secret
metadata:
  name: django-secret
type: Opaque
stringData:
  DJANGO_SECRET_KEY: <set-strong-random-value>
---
apiVersion: v1
kind: Secret
metadata:
  name: internal-api-secret
```

```
type: Opaque
stringData:
  INTERNAL_API_TOKEN: <set-strong-random-value>
---
apiVersion: v1
kind: Secret
metadata:
  name: http-ingest-token-secret
type: Opaque
stringData:
  HTTP_INGEST_TOKEN: <set-strong-random-token>
  LZ_TOKENS: <comma-separated-or-json-token-list>
```

Notes

- Examples use `stringData` for readability. Kubernetes will encode to base64 on apply. If you prefer `data:`, base64-encode values first. Example:

```
printf '%s' 'your_token_here' | base64
```

- Review URLs in `django-multimap` and `module-multimap` for environment specifics.

Security note

- Treat `DJANGO_SECRET_KEY`, `INTERNAL_API_TOKEN`, `HTTP_INGEST_TOKEN`, and `LZ_TOKENS` as sensitive credentials. Generate strong, random values and do not reuse across environments. Rotate regularly per security policy.

Storage Module

LogZilla Storage module Kubernetes manifest with the StatefulSet, InfluxDB sidecar, and headless Service for event storage, dedup, and archiving

Storage module manifest

The manifest below defines the Storage module StatefulSet with an InfluxDB sidecar and the `storage` headless Service. Replace the `storageClassName` if needed and review resource requests/limits.

```
---
apiVersion: v1
kind: Secret
metadata:
  name: sm-influxdb-secret
type: Opaque
stringData:
  INFLUXDB_ADMIN_USER: logzilla
  INFLUXDB_ADMIN_PASSWORD: <set-strong-password>
  INFLUXDB_USER: user
  INFLUXDB_USER_PASSWORD: <set-strong-password>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: sm-influxdb-multimap
data:
  INFLUXDB_DATA_MAX_VALUES_PER_TAG: "10000"
  INFLUXDB_DATA_MAX_SERIES_PER_DATABASE: "0"
  INFLUXDB_DATA_INDEX_VERSION: inmem
  INFLUXDB_LOGGING_FORMAT: json
  INFLUXDB_LOGGING_LEVEL: warn
  INFLUXDB_COORDINATOR_LOG_QUERIES_AFTER: 5s
  INFLUXDB_HTTP_LOG_ENABLED: "false"
  INFLUXDB_HTTP_ACCESS_LOG_PATH: ""
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: storage
spec:
  serviceName: storage
  podManagementPolicy: Parallel
  replicas: 5
  selector:
    matchLabels:
      name: storage
  template:
    metadata:
      labels:
        name: storage
    spec:
      enableServiceLinks: false
      containers:
      - name: storagemodule
        terminationMessagePolicy: FallbackToLogsOnError
        image: logzilla/runtime:stable
        imagePullPolicy: Always
        resources:
```

```
      requests:
        memory: 1Gi
        cpu: 2000m
      limits:
        memory: 4Gi
        cpu: 4000m
    command:
      - /usr/lib/logzilla/bin/module_run
      - StorageModule
    ports:
    - containerPort: 11413
      name: ingest-port
      protocol: TCP
    - containerPort: 81
      name: sm-api-port
      protocol: TCP
    livenessProbe:
      failureThreshold: 3
      httpGet:
        path: /liveness
        port: sm-api-port
        scheme: HTTP
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 5
    startupProbe:
      failureThreshold: 3
      httpGet:
        path: /readiness
        port: sm-api-port
        scheme: HTTP
      timeoutSeconds: 120
      initialDelaySeconds: 30
      periodSeconds: 10
      successThreshold: 1
    readinessProbe:
      failureThreshold: 3
      httpGet:
        path: /readiness
        port: sm-api-port
        scheme: HTTP
      initialDelaySeconds: 5
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 1
    envFrom:
      - configMapRef:
          name: log-multimap
      - configMapRef:
          name: module-multimap
```

```
        - secretRef:
            name: internal-api-secret
      env:
        - name: DEDUP_WINDOW
          value: "5"
        - name: INFLUXDB_HOST
          value: "localhost"
        - name: INFLUXDB_COUNTERS_HOST
          value: "influxdb"
      volumeMounts:
      - name: sm-data
        mountPath: /var/lib/logzilla/
        readOnly: false
      - name: sm-archives
        mountPath: /var/lib/logzilla-archive/
        readOnly: false
    - name: influxdb
      image: library/influxdb:1.8.10-alpine
      resources:
        requests:
          memory: 1Gi
          cpu: 500m
        limits:
          memory: 4Gi
          cpu: 1000m
      ports:
      - containerPort: 8086
        name: api
        protocol: TCP
      livenessProbe:
        failureThreshold: 3
        httpGet:
          path: /health
          port: api
          scheme: HTTP
        initialDelaySeconds: 30
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 5
      readinessProbe:
        failureThreshold: 3
        httpGet:
          path: /health
          port: api
          scheme: HTTP
        initialDelaySeconds: 5
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 1
      envFrom:
```

```
          - configMapRef:
              name: sm-influxdb-multimap
          - secretRef:
              name: sm-influxdb-secret
        volumeMounts:
        - name: influxdb-data
          mountPath: /var/lib/influxdb/
          readOnly: false
  volumeClaimTemplates:
  - metadata:
      name: sm-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: premium-rwo # replace as needed
      resources:
        requests:
          storage: 10Gi
  - metadata:
      name: sm-archives
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 10Mi
  - metadata:
      name: influxdb-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
---
apiVersion: v1
kind: Service
metadata:
  name: storage
spec:
  clusterIP: None
  ports:
    - port: 11413
      targetPort: ingest-port
      protocol: TCP
      name: sm-ingest-port
    - port: 82
      protocol: TCP
      targetPort: qe-api-port
      name: query-events-api-port
  type: ClusterIP
  selector:
    name: storage
```

Notes

- Update `storageClassName` fields per the cluster configuration, or remove to use the default StorageClass.
- The Storage module exposes an internal API on port 81 and a ZeroMQ ingest on port 11413. The headless service allows stable DNS names for StatefulSet members (e.g., `storage-0.storage`).

Query Module

LogZilla Query module Kubernetes manifest covering the StatefulSet, Service, resource requests, and storage endpoint references for search workloads

Query module manifest

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: querymodule
spec:
  serviceName: querymodule
  selector:
    matchLabels:
      name: querymodule
  template:
    metadata:
      labels:
        name: querymodule
    spec:
      enableServiceLinks: false
      containers:
      - name: querymodule
        image: logzilla/runtime:stable
        imagePullPolicy: Always
        resources:
          requests:
            memory: "1Gi"
            cpu: "1000m"
          limits:
            memory: "1Gi"
            cpu: "2000m"
        command:
          - /usr/lib/logzilla/bin/module_run
          - QueryModule
        ports:
        - containerPort: 81
          name: query-api-port
          protocol: TCP
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /ping
            port: query-api-port
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
          failureThreshold: 5
        startupProbe:
          failureThreshold: 60
          httpGet:
            path: /ping
```

```
            port: query-api-port
            scheme: HTTP
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        envFrom:
          - configMapRef:
              name: postgres-multimap
          - configMapRef:
              name: django-multimap
          - configMapRef:
              name: log-multimap
          - configMapRef:
              name: module-multimap
          - secretRef:
              name: django-secret
          - secretRef:
              name: postgres-secret
          - secretRef:
              name: internal-api-secret
        env:
          - name: SM_API_ADDRESSES
            value: http://storage-{{0-4}}.storage:81
---
apiVersion: v1
kind: Service
metadata:
  name: querymodule
spec:
  clusterIP: None
  ports:
    - port: 81
      protocol: TCP
      targetPort: query-api-port
      name: query-api-port
  type: ClusterIP
  selector:
    name: querymodule
```

Notes

- Ensure the ordinal range in `SM_API_ADDRESSES` matches the `storage` replicas (for example, `storage-0` through `storage-4` for 5 replicas).

Api Module

LogZilla API module Kubernetes manifest defining PersistentVolumeClaims and StatefulSet for Gunicorn, Tornado, and Celery worker processes

API module manifest

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: config-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: config-logs-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: api
spec:
  serviceName: api
  selector:
    matchLabels:
      name: api
  template:
    metadata:
      labels:
        name: api
    spec:
      enableServiceLinks: false
      containers:
      - name: gunicorn
        image: logzilla/runtime:stable
        imagePullPolicy: Always
        resources:
          requests:
            memory: "500Mi"
            cpu: "500m"
          limits:
            memory: "2Gi"
```

```
        cpu: "4000m"
    command:
      - /bin/sh
      - -c
      - >
        /usr/lib/logzilla/bin/lz-simple-setup &&
        gunicorn -c /usr/lib/logzilla/lib/logzilla/api/gunicorn.conf.py
        --log-level=info --workers=10 logzilla.api.wsgi:application
    ports:
    - containerPort: 80
      name: api
      protocol: TCP
    startupProbe:
      httpGet:
        path: /ping
        port: api
        scheme: HTTP
      initialDelaySeconds: 20
      periodSeconds: 5
      successThreshold: 1
      failureThreshold: 5
      timeoutSeconds: 30
    readinessProbe:
      httpGet:
        path: /ping
        port: api
        scheme: HTTP
      initialDelaySeconds: 5
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
      timeoutSeconds: 30
    livenessProbe:
      httpGet:
        path: /ping
        port: api
        scheme: HTTP
      periodSeconds: 10
      successThreshold: 1
      failureThreshold: 3
      timeoutSeconds: 30
    envFrom:
      - configMapRef:
          name: postgres-multimap
      - configMapRef:
          name: log-multimap
      - configMapRef:
          name: django-multimap
      - secretRef:
          name: django-secret
```

```
        - secretRef:
            name: postgres-secret
        - secretRef:
            name: internal-api-secret
      volumeMounts:
      - name: config
        mountPath: /etc/logzilla
        readOnly: false
      - name: logs
        mountPath: /var/log/logzilla
        readOnly: false
    - name: tornado
      image: logzilla/runtime:stable
      imagePullPolicy: Always
      resources:
        requests:
          memory: "500Mi"
          cpu: "200m"
        limits:
          memory: "1Gi"
          cpu: "1000m"
      command: ["/usr/lib/logzilla/bin/tornado_server"]
      ports:
      - containerPort: 8001
        name: ws
        protocol: TCP
      livenessProbe:
        httpGet:
          path: /ping
          port: ws
          scheme: HTTP
        periodSeconds: 10
        successThreshold: 1
        failureThreshold: 3
        timeoutSeconds: 5
      startupProbe:
        httpGet:
          path: /ping
          port: ws
          scheme: HTTP
        initialDelaySeconds: 10
        periodSeconds: 5
        successThreshold: 1
        failureThreshold: 3
        timeoutSeconds: 1
      readinessProbe:
        httpGet:
          path: /ping
          port: ws
          scheme: HTTP
```

```
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          failureThreshold: 3
          timeoutSeconds: 1
        envFrom:
          - configMapRef:
              name: postgres-multimap
          - configMapRef:
              name: log-multimap
          - configMapRef:
              name: django-multimap
          - secretRef:
              name: django-secret
          - secretRef:
              name: postgres-secret
          - secretRef:
              name: internal-api-secret
      - name: celeryworker
        image: logzilla/runtime:stable
        imagePullPolicy: Always
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
        command:
          - celery
          - -A
          - logzilla.api.celery_app
          - worker
          - --autoscale
          - 4,2
        livenessProbe:
          exec:
            command: ["/usr/lib/logzilla/bin/healthcheck", "celery"]
          periodSeconds: 10
          successThreshold: 1
          failureThreshold: 5
          timeoutSeconds: 5
        startupProbe:
          exec:
            command: ["/usr/lib/logzilla/bin/healthcheck", "celery"]
          initialDelaySeconds: 20
          periodSeconds: 10
          successThreshold: 1
          failureThreshold: 3
          timeoutSeconds: 5
```

```
        envFrom:
          - configMapRef:
              name: postgres-multimap
          - configMapRef:
              name: log-multimap
          - configMapRef:
              name: django-multimap
          - secretRef:
              name: django-secret
          - secretRef:
              name: postgres-secret
          - secretRef:
              name: internal-api-secret
        volumeMounts:
        - name: config
          mountPath: /etc/logzilla
          readOnly: false
        - name: logs
          mountPath: /var/log/logzilla
          readOnly: false
      volumes:
      - name: config
        persistentVolumeClaim:
          claimName: config-pvc
      - name: logs
        persistentVolumeClaim:
          claimName: config-logs-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: api
  annotations:
    cloud.google.com/neg: '{"ingress": true}'
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: api
    - port: 8001
      targetPort: 8001
      protocol: TCP
      name: ws
  clusterIP: None
  selector:
    name: api
```

Notes

- The `api` Service exposes HTTP (80) and WebSocket (8001). On GKE, NEG annotations are used for Ingress backends.
- Ensure secrets and config maps referenced here are created first (see Common Config and Secrets page).

Ingest Module

LogZilla Ingest module Kubernetes manifest with syslog-ng, ParserModule, and HTTP receiver containers for BSD, RFC5424, JSON, and /incoming event intake

Ingest module manifest

The Ingest module provides syslog receivers (TCP/UDP 514), JSON (TCP 515), RFC5424 (TCP 601), and an HTTP receiver for `/incoming`. It runs three containers: `syslogng`, `parsermodule`, and `httpreceiver`.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: syslogng-conf
data:
  config.yaml: |
    config_version: '6.32'
    custom_conf_dir: /etc/logzilla/syslog-ng/conf.d/
    default_max_connections: 50
    default_log_iw_size: 1000000
    default_flush_lines: 100
    default_log_fetch_limit: 100

    extra_log_rules: ''
    flow_control: true

    destinations:
    - enabled: true
      type: logzilla-http
      ingest_token: replace_me
      url: http://localhost/
      mem_buf_length: 50000
      batch_timeout: 500
      batch_lines: 10000
      retries: 999
      send_timeout: 1000
      reply_timeout: 5000
      connect_timeout: 5000
      time_reopen: 1
      name: logzilla
      port: 11412
      reliable: false
      workers: 2
      buffer_dir: /var/lib/syslog-ng/
      disk_buf_size: 1048576
      disk_buff_enabled: false
      mem_buf_size: 4194304
      qout_size: 10000

    sources:
    - enabled: true
      flags: [syslog-protocol]
      name: bsd
      port: 514
      type: network
    - enabled: true
      flags: [syslog-protocol]
      name: bsd_udp
      port: 514
```

```
      transport: udp
      type: network
    - enabled: true
      name: rfc5424
      port: 601
      type: syslog
    - enabled: true
      flags: [no-parse]
      name: json
      port: 515
      program_override: _JSON
      type: network
    - enabled: false
      flags: [syslog-protocol]
      name: tls
      port: 6514
      tls_cert_file: /etc/logzilla/syslog-ng/tls.crt
      tls_key_file: /etc/logzilla/syslog-ng/tls.key
      transport: tls
      type: network
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: ingest
spec:
  serviceName: ingest
  podManagementPolicy: Parallel
  replicas: 4
  selector:
    matchLabels:
      name: ingest
  template:
    metadata:
      labels:
        name: ingest
    spec:
      enableServiceLinks: false
      containers:
      - name: syslogng
        image: logzilla/syslogng:stable
        terminationMessagePolicy: FallbackToLogsOnError
        imagePullPolicy: Always
        command: ["/usr/local/bin/docker-entrypoint.sh"]
        args: ["--worker-threads=8", "--no-caps"]
        resources:
          requests:
            memory: "200Mi"
            cpu: "500m"
          limits:
```

```
            memory: "1Gi"
            cpu: "2000m"
        ports:
        - containerPort: 514
          protocol: TCP
          name: bsd-tcp
        - containerPort: 514
          protocol: UDP
          name: bsd-udp
        - containerPort: 515
          protocol: TCP
          name: json
        - containerPort: 601
          protocol: TCP
          name: rfc5424
        - containerPort: 8080
          protocol: TCP
          name: stats
        livenessProbe:
          exec:
            command: ["syslog-ng-ctl", "stats"]
          failureThreshold: 3
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        readinessProbe:
          exec:
            command: ["syslog-ng-ctl", "stats"]
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        volumeMounts:
        - name: syslogng-conf
          mountPath: /etc/logzilla/syslog-ng/config.yaml
          subPath: config.yaml
          readOnly: true
      - name: parsermodule
        image: logzilla/runtime:stable
        imagePullPolicy: Always
        terminationMessagePolicy: FallbackToLogsOnError
        resources:
          requests:
            memory: "200Mi"
            cpu: "200m"
          limits:
            cpu: "1000m"
            memory: "500Mi"
```

```
        command: ["/usr/lib/logzilla/bin/module_run", "ParserModule"]
        ports:
        - containerPort: 11412
          name: zmq-ingest
          protocol: TCP
        - containerPort: 81
          name: module-api
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /liveness
            port: module-api
            scheme: HTTP
          failureThreshold: 3
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        readinessProbe:
          httpGet:
            path: /readiness
            port: module-api
            scheme: HTTP
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        envFrom:
          - configMapRef:
              name: log-multimap
          - configMapRef:
              name: module-multimap
          - secretRef:
              name: internal-api-secret
        env:
        - name: PARSER_WORKERS
          value: "8"
        - name: LOG_MAX_LEVEL
          value: INFO
        - name: DOWNLOAD_API_RULES
          value: "1"
        - name: LOG_LEVEL
          value: "INFO"
        - name: SINGLE_SM_MODE
          value: "1"
        - name: SM_INGEST_URLS
          value: http://storage-{{1-4}}.storage:81/ingest
      - name: httpreceiver
        image: logzilla/runtime:stable
```

```
        terminationMessagePolicy: FallbackToLogsOnError
        imagePullPolicy: Always
        command: ["/usr/lib/logzilla/bin/http_receiver"]
        resources:
          requests:
            memory: "250Mi"
            cpu: "250m"
          limits:
            memory: "1000Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
          protocol: TCP
          name: http-recv-port
        livenessProbe:
          httpGet:
            path: /ping
            port: http-recv-port
            scheme: HTTP
          failureThreshold: 3
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        readinessProbe:
          httpGet:
            path: /ping
            port: http-recv-port
            scheme: HTTP
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        env:
        - name: LZ_TARGET_BASE_URL
          value: http://localhost:81/
        - name: IA_BASE_URL
          value: http://api/api/
        - name: WORKERS_NUM
          value: "8"
        envFrom:
          - secretRef:
              name: internal-api-secret
          - secretRef:
              name: http-ingest-token-secret
      volumes:
      - name: syslogng-conf
        configMap:
          name: syslogng-conf
```

```
          items:
          - key: config.yaml
            path: config.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: syslog
spec:
  clusterIP: None
  ports:
    - port: 514
      protocol: TCP
      targetPort: bsd-tcp
      name: bsd-tcp-svc
    - port: 514
      protocol: UDP
      targetPort: bsd-udp
      name: bsd-udp-svc
    - port: 515
      protocol: TCP
      targetPort: json
      name: json-svc
    - port: 601
      protocol: TCP
      targetPort: rfc5424
      name: rfc5424-svc
  type: ClusterIP
  selector:
    name: ingest
---
apiVersion: v1
kind: Service
metadata:
  name: httpreceiver
spec:
  clusterIP: None
  ports:
    - port: 80
      protocol: TCP
      targetPort: http-recv-port
      name: http-recv-port
  type: ClusterIP
  selector:
    name: ingest
```

Notes

- Replace `ingest_token` in `syslogng-conf` with a valid token.

- The syslog Service exposes TCP/UDP 514, JSON 515, RFC5424 601.
- `httpreceiver` exposes port 80 for `/incoming` HTTP ingest.

Front End

LogZilla NGINX front-end Kubernetes manifest with ConfigMap, Deployment, and Service definitions for serving the UI and proxying API and WebSocket traffic

Front end (NGINX) manifest

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: front-conf
data:
  nginx.conf: |
    worker_processes 2;

    user nginx nginx;
    pid /tmp/nginx.pid;
    error_log stderr info;

    events {
        worker_connections 1024;
        accept_mutex on;
    }

    http {
        include /etc/nginx/mime.types;
        default_type application/octet-stream;
        access_log /dev/stdout combined;
        sendfile on;
        client_body_buffer_size 50m;
        client_max_body_size 50m;
        proxy_buffers 16 1m;
        proxy_buffer_size 1m;

        proxy_redirect off;

        server {
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $http_host;
            proxy_set_header X-HTTPS-Protocol $ssl_protocol;
            proxy_set_header HTTP_INTERNAL_REQUEST_USERNAME "";

            root /usr/share/nginx/html;

            location /version {
                add_header Content-Type text/plain;
                add_header Cache-Control no-cache;
                try_files $uri /version.txt;
            }

            location /api/static/ {
                try_files $uri =404;
            }
```

```
            set $ui "ui1";
            if ($server_port = 80 ) { set $ui "ui1"; }
            if ($server_port = 8080 ) { set $ui "ui2"; }

            location / {
                root "/usr/share/nginx/html/$ui";
                try_files $uri /index.html;
            }

            listen 80 default_server;
            listen 8080;
        }
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front
spec:
  selector:
    matchLabels:
      name: front
  template:
    metadata:
      labels:
        name: front
    spec:
      containers:
      - name: front
        image: logzilla/front:stable
        imagePullPolicy: Always
        resources:
          requests:
            memory: "200Mi"
            cpu: "200m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
        ports:
        - containerPort: 80
          protocol: TCP
          name: front-port
        - containerPort: 8080
          protocol: TCP
          name: new-front-port
        livenessProbe:
          httpGet:
            path: /version
            port: 80
            scheme: HTTP
```

```
          failureThreshold: 3
          initialDelaySeconds: 30
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        startupProbe:
          httpGet:
            path: /version
            port: 80
            scheme: HTTP
          failureThreshold: 3
          initialDelaySeconds: 5
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        volumeMounts:
        - name: nginx-conf
          mountPath: /etc/nginx/nginx.conf
          subPath: nginx.conf
          readOnly: true
      volumes:
      - name: nginx-conf
        configMap:
          name: front-conf
---
apiVersion: v1
kind: Service
metadata:
  name: front
spec:
  ports:
    - port: 80
      targetPort: front-port
      protocol: TCP
      name: ui-port
    - port: 8080
      targetPort: new-front-port
      protocol: TCP
      name: new-ui-port
  type: ClusterIP
  selector:
    name: front
```

Notes

- The front end serves the UI and static assets and proxies to UI variants.
- Ensure the front Deployment is applied after the LogZilla API is running.

Ingress Gke

GKE LoadBalancer Services and HTTP Ingress manifest for LogZilla syslog, JSON, RFC5424, and HTTP ingest, plus UI, API, and WebSocket endpoints

Purpose

This example exposes syslog (TCP/UDP), JSON, RFC5424, and HTTP ingest via LoadBalancer Services, and publishes the UI, API, and WebSocket endpoints via an HTTP Ingress. Adapt annotations and class to other providers as needed.

GKE Ingress and Services

```
apiVersion: v1
kind: Service
metadata:
  name: syslogng-lb-tcp
spec:
  ports:
    - port: 514
      protocol: TCP
      targetPort: bsd-tcp
      name: bsd-tcp-svc-lb
    - port: 515
      protocol: TCP
      targetPort: json
      name: json-svc-lb
    - port: 601
      protocol: TCP
      targetPort: rfc5424
      name: rfc5424-svc-lb
  type: LoadBalancer
  selector:
    name: ingest
---
apiVersion: v1
kind: Service
metadata:
  name: syslogng-lb-udp
spec:
  ports:
    - port: 514
      protocol: UDP
      targetPort: bsd-udp
      name: bsd-udp-svc-lb
  type: LoadBalancer
  selector:
    name: ingest
---
apiVersion: v1
kind: Service
metadata:
  name: httpreceiver-lb
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: http-recv-port
      name: http-recv-port-ld
  type: LoadBalancer
```

```
  selector:
    name: ingest
---
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: tornado-backendconfig
spec:
  healthCheck:
    timeoutSec: 5
    checkIntervalSec: 10
  timeoutSec: 3600
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-tornado
  annotations:
    cloud.google.com/backend-config: '{"default": "tornado-backendconfig"}'
    cloud.google.com/neg: '{"ingress": true}'
spec:
  ports:
    - port: 8001
      targetPort: 8001
      protocol: TCP
  type: ClusterIP
  selector:
    name: api
---
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: api-backendconfig
spec:
  healthCheck:
    timeoutSec: 5
    checkIntervalSec: 10
---
apiVersion: v1
kind: Service
metadata:
  name: ingress-api
  annotations:
    cloud.google.com/backend-config: '{"default": "api-backendconfig"}'
    cloud.google.com/neg: '{"ingress": true}'
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
```

```
  type: ClusterIP
  selector:
    name: api
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-gke
  annotations:
    spec.ingressClassName: "gce"
    kubernetes.io/ingress.allow-http: "true"
spec:
  defaultBackend:
    service:
      name: front
      port:
        number: 80
  rules:
  - http:
      paths:
        - path: /api/static/
          pathType: Prefix
          backend:
            service:
              name: front
              port:
                number: 80
        - path: /incoming/
          pathType: Prefix
          backend:
            service:
              name: httpreceiver-lb
              port:
                number: 80
        - path: /api/
          pathType: Prefix
          backend:
            service:
              name: ingress-api
              port:
                number: 80
        - path: /ws/
          pathType: Prefix
          backend:
            service:
              name: ingress-tornado
              port:
                number: 8001
```

Notes

- The NEG annotations are specific to GKE HTTP(S) Load Balancers.
- For other providers, use the platform's Ingress class and health checks.
- LoadBalancer Services expose syslog and HTTP ingest externally.

Scaling And Tuning

Scale LogZilla Kubernetes StatefulSets for ingest, storage, and query workloads, tune CPU and memory requests, and align SM_API_ADDRESSES ordinals

Replica scaling

Increase or decrease replicas for StatefulSets to match ingest and query load:

```
# Ingest (syslog receivers, parser, http receiver)
kubectl scale statefulset ingest --replicas=8

# Storage (ensure ordinal references align, e.g., storage-0..N)
kubectl scale statefulset storage --replicas=7

# Query module (usually scale with storage)
kubectl scale statefulset querymodule --replicas=3
```

When scaling `storage`, review ordinal references used by other modules:

- `SM_API_ADDRESSES` in Query Module (e.g., `http://storage-{{0-4}}.storage:81`).
- `SM_INGEST_URLS` in Ingest ParserModule (e.g., `http://storage-{{1-4}}.storage:81/ingest`).

Adjust these ranges to reflect the desired replica counts.

Requests and limits

Manifests define CPU and memory requests/limits for each container. Increase requests to provide scheduling guarantees; adjust limits to prevent eviction. Observe actual usage via the platform metrics (e.g., `kubectl top pods`).

Guidance by component:

- Ingest `syslogng`: raise `--worker-threads` and CPU/memory for higher EPS.
- Ingest `parsermodule`: increase `PARSER_WORKERS` and resources as needed.
- API `gunicorn`: tune `--workers` (default 10 in the manifest command) and allocate CPU accordingly.
- Celery worker: adjust autoscale args and CPU/memory.
- Storage/InfluxDB: ensure disk performance and memory are adequate; consider faster StorageClass for production.

Rolling updates

```
# Check rollout status
kubectl rollout status statefulset/api

# Restart pods to pick up new config or image tag
kubectl rollout restart statefulset/api
```

Readiness and liveness probes

Probes are defined in all manifests. If a pod flaps between Ready/NotReady:

- Describe the pod and review the probe configuration and last failure:

```
kubectl describe pod <pod-name>
```

- Examine container logs:

```
kubectl logs <pod-name> -c <container-name>
```

StorageClass and PVC sizing

- Replace `storageClassName` with the provider's class as needed.
- Right-size PVC requests (e.g., `sm-data`, `sm-archives`, `influxdb-data`).

Ingress and external exposure

- For GKE, use the provided Ingress example and NEG annotations.
- For other providers, adapt annotations and class to the installed controller.

Image tags

- Use the `stable` tag by default for production deployments.
- For controlled rollouts, pin to a specific release tag (for example, `vX.Y.Z`) and update via `kubectl set image` or `kubectl rollout restart`.

Troubleshooting

Diagnose LogZilla Kubernetes deployments using kubectl pod, service, and log inspection, and resolve probe, Secret, storage, and Ingress failures

Quick checks

```
# Pods and services
kubectl get pods
kubectl get svc

# Inspect a pod
kubectl describe pod <pod-name>

# Container logs
kubectl logs <pod-name> -c <container-name>
```

Common issues

- Probes failing (NotReady / CrashLoopBackOff):
  - Inspect readiness/liveness/startup probe configuration in manifests.
  - Review container logs for stack traces or healthcheck errors.
- Secrets or ConfigMaps missing:
  - Ensure `Common Config and Secrets` are applied before modules.
  - Verify base64 values and keys match manifest references.
- Storage pending:
  - Replace `storageClassName` with the cluster's class or remove the field to use the default StorageClass.
  - Confirm PV/PVC provisioning status.
- Ingress errors:
  - Confirm the correct IngressClass and annotations for the provider.
  - On GKE, verify NEG backends and health checks.
- External ports not reachable:
  - Confirm LoadBalancer Services were assigned external IPs and firewall rules permit inbound TCP/UDP as required.

Component-specific tips

- Ingest
  - `syslogng` exposes TCP/UDP 514, JSON 515, RFC5424 601; ensure the `syslog` Service exists and pods are Ready.
  - `httpreceiver` serves `/incoming` on port 80; verify the `httpreceiver` Service and Ingress/LB route.
- Storage
  - Check `storagemodule` and InfluxDB logs for disk or memory pressure.
  - Validate PVCs (`sm-data` , `sm-archives` , `influxdb-data`).
- API
  - `gunicorn` health endpoint: `/ping` on port 80.
  - `tornado` health endpoint: `/ping` on port 8001.
- Query
  - Ensure `SM_API_ADDRESSES` points to the actual `storage-<ordinal>` range.

Reapply and rollouts

```
# Reapply a manifest after editing
kubectl apply -f <file>.yaml

# Restart a statefulset to pick up changes
kubectl rollout restart statefulset/<name>

# Monitor rollout
kubectl rollout status statefulset/<name>
```