Parsing log data

Parsing is the process of splitting unstructured log data into attributes (key/value pairs). You can use these attributes to facet or filter logs in useful ways. This in turn helps you build better charts and alerts.

New Relic parses log data according to rules. This document describes how logs parsing works, how to use built-in rules, and how to create custom rules.

You can also create, query, and manage your log parsing rules by using API,

Parsing example

A good example is a default NGINX access log containing unstructured text. It is useful for searching but not much else. Here's an example of a typical line:

```
127.180.71.3 - - [10/May/1997:08:05:32 +0000] "GET /downloads/product_1 HTTP/1.1" 304 0 "-"
"Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
```

In an unparsed format, you would need to do a full text search to answer most questions. After parsing, the log is organized into attributes, like response code and request URL:

```
{
 "remote_addr":"93.180.71.3",
 "time":"1586514731",
 "method":"GET",
 "path":"/downloads/product_1",
 "version":"HTTP/1.1",
 "response":"304",
 "bytesSent": 0,
 "user_agent": "Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
}
```

Parsing makes it easier to create custom queries that facet on those values. This helps you understand the distribution of response codes per request URL and quickly find problematic pages.

How log parsing works

Here's an overview of how New Relic implements parsing of logs:

Log parsing	How it works
What	• All parsing takes place against the message field; no other fields can be parsed. • Each parsing rule is created by using a NRQL WHERE clause that determines which logs the rule will attempt to parse. • To simplify the matching process, we recommend adding a logtype attribute to your logs. However, you are not limited to

Log parsing	How it works
	using logtype; one or more attributes can be used as matching criteria in the NRQL WHERE clause.
When	• Parsing will only be applied once to each log message. If multiple parsing rules match the log, only the first that succeeds will be applied. • Parsing takes place during log ingestion, before data is written to NRDB. Once data has been written to storage, it can no longer be parsed. • Parsing occurs in the pipeline **before** data enrichments take place. Be careful when defining the matching criteria for a parsing rule. If the criteria is based on an attribute that doesn't exist until after parsing or enrichment take place, that data won't be present in the logs when matching occurs. As a result, no parsing will happen.
How	• Rules can be written in Grok, regex, or a mixture of the two. Grok is a collection of patterns that abstract away complicated regular expressions. • If the content of the message field is JSON, it will be parsed automatically.

Parse attributes using Grok

Parsing patterns are specified using Grok, an industry standard for parsing log messages. Any incoming log with a logtype field will be checked against our built-in patterns, and if possible, the associated Grok pattern is applied to the log.

Grok is a superset of regular expressions that adds built-in named patterns to be used in place of literal complex regular expressions. For instance, instead of having to remember that an integer can be matched with the regular expression (?:[+-]?(?:[0-9]+)), you can just write %{INT} to use the Grok pattern INT, which represents the same regular expression.

You can always use a mix of regular expressions and Grok pattern names in your matching string. For more information, see our list of Grok syntax and supported types.

Variable names must be explicitely set and be lowercase like %{URI:uri}. Expressions such as %{URI} or %{URI:URI} would not work.

Organizing by logtype

New Relic's log ingestion pipeline can parse data by matching a log event to a rule that describes how the log should be parsed. There are two ways log events can be parsed:

- Use a built-in rule.
- Define a custom rule.

Rules are a combination of matching logic and parsing logic. Matching is done by defining a query match on an attribute of the logs. Rules are not applied retroactively. Logs collected before a rule is created are not parsed by that rule.

The simplest way to organize your logs and how they are parsed is to include the logtype field in your log event. This tells New Relic what built-in rule to apply to the logs.

IMPORTANT

Once a parsing rule is active, data parsed by the rule is permanently changed. This cannot be reverted.

Limits

Parsing is computationally expensive, which introduces risk. Parsing is done for custom rules defined in an account and for matching patterns to a log. A large number of patterns or poorly defined custom rules will consume a huge amount of memory and CPU resources while also taking a very long time to complete.

In order to prevent problems, we apply two parsing limits: per-message-per-rule and per-account.

Limit	Description
Per-message-per-rule	The per-message-per-rule limit prevents the time spent parsing any single message from being greater than 100 ms. If that limit is reached, the system will cease attempting to parse the log message with that rule. The ingestion pipeline will attempt to run any other applicable on that message, and the message will still be passed through the ingestion pipeline and stored in NRDB. The log message will be in its original, unparsed format.
Per-account	The per-account limit exists to prevent accounts from using more than their fair share of resources. The limit considers the total time spent processing **all** log messages for an account per-minute. The limit is not a fixed value; it scales up or down proportionally to the volume of data stored daily by the account and the environment size that is subsequently allocated to support that customer.

Built-in parsing rules

Common log formats have well-established parsing rules already created for them. To get the benefit of built-in parsing rules, add the logtype attribute when forwarding logs. Set the value

to something listed in the following table, and the rules for that type of log will be applied automatically.

List of built-in rules

The following logtype attribute values map to a predefined parsing rule. For example, to query the Application Load Balancer:

- From the New Relic UI, use the format logtype:"alb".
- From NerdGraph, use the format logtype = 'alb'.

To learn what fields are parsed for each rule, see our documentation about built-in parsing rules.

logtype	Log source	Example matching query
apache	Apache access logs	logtype:"apache"
apache_error	Apache error logs	logtype:"apache_error"
alb	Application load balancer logs	logtype:"alb"
cassandra	Cassandra logs	logtype:"cassandra"
cloudfront-web	CloudFront web logs	logtype:"cloudfront-web"
elb	Elastic Load Balancer logs	logtype:"elb"
haproxy_http	HAProxy logs	logtype:"haproxy_http"
ktranslate-health	KTranslate container health logs	logtype:"ktranslate-health"
linux_cron	Linux cron	logtype:"linux_cron"
linux_messages	Linux messages	logtype:"linux_messages"
iis_w3c	Microsoft IIS server logs - W3C format	logtype:"iis_w3c"
mongodb	MongoDB logs	logtype:"mongodb"
monit	Monit logs	logtype:"monit"
mysql-error	MySQL error logs	logtype:"mysql-error"
nginx	NGINX access logs	logtype:"nginx"
nginx-error	NGINX error logs	logtype:"nginx-error"
postgresql	Postgresql logs	logtype:"postgresql"
rabbitmq	Rabbitmq logs	logtype:"rabbitmq"
redis	Redis logs	logtype:"redis"
route-53	Route 53 logs	logtype:"route-53"
syslog-rfc5424	Syslogs with RFC5424 format	logtype:"syslog-rfc5424"

Add the logtype attribute

When aggregating logs, it's important to provide metadata that makes it easy to organize, search, and parse those logs. One simple way of doing this is to add the attribute logtype to the log messages when they are shipped. Built-in parsing rules are applied by default to certain logtype values.

TIP

The fields logType, logtype, and LOGTYPE are all supported for built-in rules. For ease of searching, we recommend that you align on a single syntax in your organization.

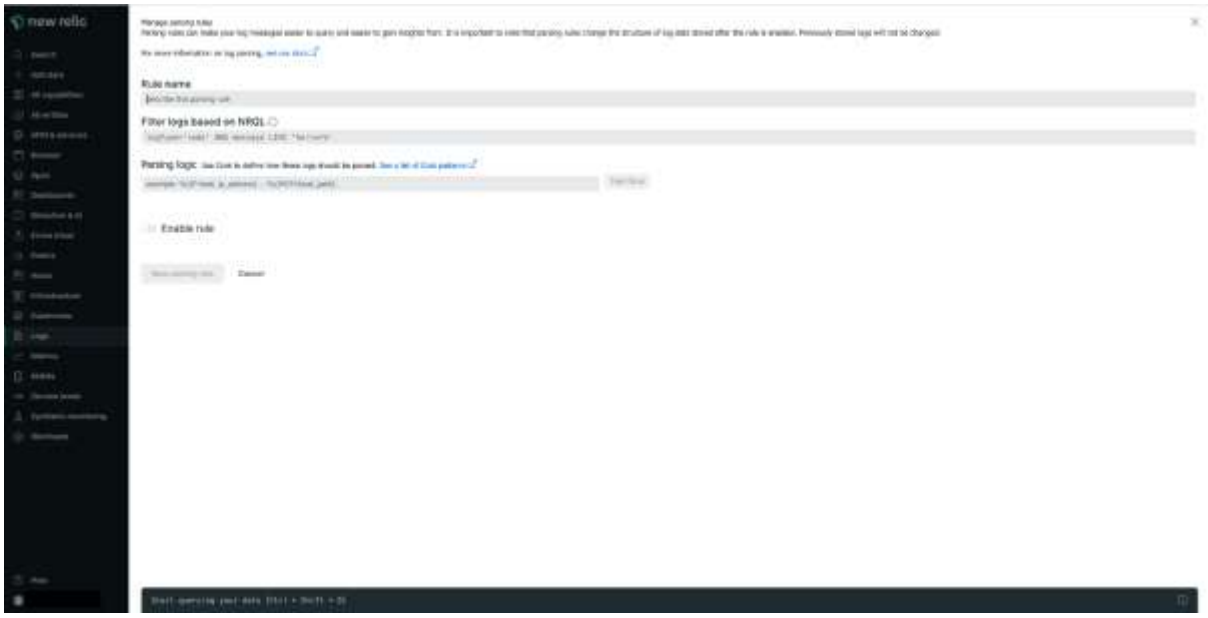
You can add attributes to the JSON request sent to New Relic. In this example we add a logtype attribute of value nginx to trigger the built-in NGINX parsing rule.

Logs API.

```
POST /log/v1 HTTP/1.1
Host: log-api.newrelic.com
Content-Type: application/json
X-License-Key: YOUR_LICENSE_KEY
Accept: */*
Content-Length: 133
{
 "timestamp": TIMESTAMP_IN_UNIX_EPOCH,
 "message": "User 'xyz' logged in",
 "logtype": "accesslogs",
 "service": "login-service",
 "hostname": "login.example.com"
}
```

Create and view custom parsing rules

Many logs are formatted or structured in a unique way. In order to parse them, custom logic must be built and applied.



Troubleshooting

If parsing is not working the way you intended, it may be due to:

- **Logic:** The parsing rule matching logic does not match the logs you want.
- **Timing:** If your parsing matching rule targets a value that doesn't exist yet, it will fail. This can occur if the value is added later in the pipeline as part of the enrichment process.
- **Limits:** There is a fixed amount of time available every minute to process logs via parsing, patterns, drop filters, etc. If the maximum amount of time has been spent, parsing will be skipped for additional log event records.

.