You are looking at documentation for an older release. Not what you want? See the current release documentation (../current/index.html).

grok edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Version: 3.4.0
- Released on: March 1, 2017
- Changelog (https://github.com/logstash-plugins/logstash-filter-grok/blob/master/CHANGELOG.md#340)

Getting Help edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

For questions about the plugin, open a topic in the Discuss (http://discuss.elastic.co) forums. For bugs or feature requests, open an issue in Github (https://github.com/elastic/logstash). For the list of Elastic supported plugins, please consult the Elastic Support Matrix (https://www.elastic.co/support/matrix#show_logstash_plugins).

Description edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

Parse arbitrary text and structure it.

Grok is currently the best way in logstash to parse crappy unstructured log data into something structured and queryable.

This tool is perfect for syslog logs, apache and other webserver logs, mysql logs, and in general, any log format that is generally written for humans and not computer consumption.

Logstash ships with about 120 patterns by default. You can find them here: https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns (https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns). You can add your own trivially. (See the `patterns_dir` setting)

If you need help building patterns to match your logs, you will find the http://grokdebug.herokuapp.com (http://grokdebug.herokuapp.com) and http://grokconstructor.appspot.com/ (http://grokconstructor.appspot.com/) applications quite useful!

Grok Basics

Grok works by combining text patterns into something that matches your logs.

The syntax for a grok pattern is `%{SYNTAX:SEMANTIC}`

The `SYNTAX` is the name of the pattern that will match your text. For example, `3.44` will be matched by the `NUMBER` pattern and `55.3.244.1` will be matched by the `IP` pattern. The syntax is how you match.

The `SEMANTIC` is the identifier you give to the piece of text being matched. For example, `3.44` could be the duration of an event, so you could call it simply `duration`. Further, a string `55.3.244.1` might identify the `client` making a request.

For the above example, your grok filter would look something like this:

```
%{NUMBER:duration} %{IP:client}
```

Optionally you can add a data type conversion to your grok pattern. By default all semantics are saved as strings. If you wish to convert a semantic's data type, for example change a string to an integer then suffix it with the target data type. For example `%{NUMBER:num:int}` which converts the `num` semantic from a string to an integer. Currently the only supported conversions are `int` and `float`.

Examples: With that idea of a syntax and semantic, we can pull out useful fields from a sample log like this fictional http request log:

```
55.3.244.1 GET /index.html 15824 0.043
```

The pattern for this could be:

```
%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duratio
n}
```

A more realistic example, let's read these logs from a file:

```
input {
  file {
    path => "/var/log/http.log"
  }
}
filter {
  grok {
    match => { "message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{N
UMBER:bytes} %{NUMBER:duration}" }
  }
}
```

After the grok filter, the event will have a few extra fields in it:

- `client: 55.3.244.1`
- `method: GET`
- `request: /index.html`
- `bytes: 15824`
- `duration: 0.043`

Regular Expressions

Grok sits on top of regular expressions, so any regular expressions are valid in grok as well. The regular expression library is Oniguruma, and you can see the full supported regexp syntax on the Oniguruma site (https://github.com/kkos/oniguruma/blob/master/doc/RE).

Custom Patterns

Sometimes logstash doesn't have a pattern you need. For this, you have a few options.

First, you can use the Oniguruma syntax for named capture which will let you match a piece of text and save it as a field:

```
(?<field_name>the pattern here)
```

For example, postfix logs have a `queue id` that is an 10 or 11-character hexadecimal value. I can capture that easily like this:

```
(?<queue_id>[0-9A-F]{10,11})
```

Alternately, you can create a custom patterns file.

- Create a directory called `patterns` with a file in it called `extra` (the file name doesn't matter, but name it meaningfully for yourself)

- In that file, write the pattern you need as the pattern name, a space, then the regexp for that pattern.

For example, doing the postfix queue id example as above:

```
# contents of ./patterns/postfix:
POSTFIX_QUEUEID [0-9A-F]{10,11}
```

Then use the `patterns_dir` setting in this plugin to tell logstash where your custom patterns directory is. Here's a full example with a sample log:

```
Jan  1 06:25:43 mailserver14 postfix/cleanup[21403]: BEF25A72965: message-id=<201301
01142543.5828399CCAF@mailserver14.example.com>
```

```
filter {
  grok {
    patterns_dir => ["./patterns"]
    match => { "message" => "%{SYSLOGBASE} %{POSTFIX_QUEUEID:queue_id}: %{GREEDYDAT
A:syslog_message}" }
  }
}
```

The above will match and result in the following fields:

- `timestamp: Jan 1 06:25:43`
- `logsource: mailserver14`
- `program: postfix/cleanup`
- `pid: 21403`
- `queue_id: BEF25A72965`
- `syslog_message: message-id=<20130101142543.5828399CCAF@mailserver14.example.com>`

The `timestamp`, `logsource`, `program`, and `pid` fields come from the `SYSLOGBASE` pattern which itself is defined by other patterns.

Another option is to define patterns *inline* in the filter using `pattern_definitions`. This is mostly for convenience and allows user to define a pattern which can be used just in that filter. This newly defined patterns in `pattern_definitions` will not be available outside of that particular `grok` filter.

Synopsis

This plugin supports the following configuration options:

Required configuration options:

```
grok {
}
```

Available configuration options:

Setting	Input type	Required
`add_field` (plugins-filters-grok.html#plugins-filters-grok-add_field)	hash (configuration-file-structure.html#hash)	No
`add_tag` (plugins-filters-grok.html#plugins-filters-grok-add_tag)	array (configuration-file-structure.html#array)	No
`break_on_match` (plugins-filters-grok.html#plugins-filters-grok-break_on_match)	boolean (configuration-file-structure.html#boolean)	No
`enable_metric` (plugins-filters-grok.html#plugins-filters-grok-enable_metric)	boolean (configuration-file-structure.html#boolean)	No
`id` (plugins-filters-grok.html#plugins-filters-grok-id)	string (configuration-file-structure.html#string)	No
`keep_empty_captures` (plugins-filters-grok.html#plugins-filters-grok-keep_empty_captures)	boolean (configuration-file-structure.html#boolean)	No
`match` (plugins-filters-grok.html#plugins-filters-grok-match)	hash (configuration-file-structure.html#hash)	No
`named_captures_only` (plugins-filters-grok.html#plugins-filters-grok-named_captures_only)	boolean (configuration-file-structure.html#boolean)	No
`overwrite` (plugins-filters-grok.html#plugins-filters-grok-overwrite)	array (configuration-file-structure.html#array)	No
`pattern_definitions` (plugins-filters-grok.html#plugins-filters-grok-pattern_definitions)	hash (configuration-file-structure.html#hash)	No

Setting	Input type	Required
`patterns_dir` (plugins-filters-grok.html#plugins-filters-grok-patterns_dir)	array (configuration-file-structure.html#array)	No
`patterns_files_glob` (plugins-filters-grok.html#plugins-filters-grok-patterns_files_glob)	string (configuration-file-structure.html#string)	No
`periodic_flush` (plugins-filters-grok.html#plugins-filters-grok-periodic_flush)	boolean (configuration-file-structure.html#boolean)	No
`remove_field` (plugins-filters-grok.html#plugins-filters-grok-remove_field)	array (configuration-file-structure.html#array)	No
`remove_tag` (plugins-filters-grok.html#plugins-filters-grok-remove_tag)	array (configuration-file-structure.html#array)	No
`tag_on_failure` (plugins-filters-grok.html#plugins-filters-grok-tag_on_failure)	array (configuration-file-structure.html#array)	No
`tag_on_timeout` (plugins-filters-grok.html#plugins-filters-grok-tag_on_timeout)	string (configuration-file-structure.html#string)	No
`timeout_millis` (plugins-filters-grok.html#plugins-filters-grok-timeout_millis)	number (configuration-file-structure.html#number)	No

Details

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

`add_field`

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is hash (configuration-file-structure.html#hash)
- Default value is `{}`

If this filter is successful, add any arbitrary fields to this event. Field names can be dynamic and include parts of the event using the `%{field}`.

Example:

```
filter {
  grok {
    add_field => { "foo_%{somefield}" => "Hello world, from %{host}" }
  }
}
```

```
# You can also add multiple fields at once:
filter {
  grok {
    add_field => {
      "foo_%{somefield}" => "Hello world, from %{host}"
      "new_field" => "new_static_value"
    }
  }
}
```

If the event has field `"somefield" == "hello"` this filter, on success, would add field `foo_hello` if it is present, with the value above and the `%{host}` piece replaced with that value from the event. The second example would also add a hardcoded field.

add_tag

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is array (configuration-file-structure.html#array)
- Default value is `[]`

If this filter is successful, add arbitrary tags to the event. Tags can be dynamic and include parts of the event using the `%{field}` syntax.

Example:

```
filter {
  grok {
    add_tag => [ "foo_%{somefield}" ]
  }
}
```

```
# You can also add multiple tags at once:
filter {
  grok {
    add_tag => [ "foo_%{somefield}", "taggedy_tag"]
  }
}
```

If the event has field `"somefield" == "hello"` this filter, on success, would add a tag `foo_hello` (and the second example would of course add a `taggedy_tag` tag).

break_on_match

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is boolean (configuration-file-structure.html#boolean)
- Default value is `true`

Break on first match. The first successful match by grok will result in the filter being finished. If you want grok to try all patterns (maybe you are parsing different things), then set this to false.

`enable_metric`

- Value type is boolean (configuration-file-structure.html#boolean)
- Default value is `true`

Disable or enable metric logging for this specific plugin instance by default we record all the metrics we can, but you can disable metrics collection for a specific plugin.

`id`

- Value type is string (configuration-file-structure.html#string)
- There is no default value for this setting.

Add a unique `ID` to the plugin configuration. If no ID is specified, Logstash will generate one. It is strongly recommended to set this ID in your configuration. This is particularly useful when you have two or more plugins of the same type, for example, if you have 2 grok filters. Adding a named ID in this case will help in monitoring Logstash when using the monitoring APIs.

```
output {
 stdout {
   id => "my_plugin_id"
 }
}
```

`keep_empty_captures`

- Value type is boolean (configuration-file-structure.html#boolean)
- Default value is `false`

If `true`, keep empty captures as event fields.

`match`

- Value type is hash (configuration-file-structure.html#hash)
- Default value is `{}`

A hash of matches of field ⇒ value

For example:

```
filter {
  grok { match => { "message" => "Duration: %{NUMBER:duration}" } }
}
```

If you need to match multiple patterns against a single field, the value can be an array of patterns

```
filter {
  grok { match => { "message" => [ "Duration: %{NUMBER:duration}", "Speed: %{NUMBER:
speed}" ] } }
}
```

named_captures_only

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is boolean (configuration-file-structure.html#boolean)
- Default value is `true`

If `true`, only store named captures from grok.

overwrite

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is array (configuration-file-structure.html#array)
- Default value is `[]`

The fields to overwrite.

This allows you to overwrite a value in a field that already exists.

For example, if you have a syslog line in the `message` field, you can overwrite the `message` field with part of the match like so:

```
filter {
  grok {
    match => { "message" => "%{SYSLOGBASE} %{DATA:message}" }
    overwrite => [ "message" ]
  }
}
```

In this case, a line like `May 29 16:37:11 sadness logger: hello world` will be parsed and `hello world` will overwrite the original message.

pattern_definitions

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is hash (configuration-file-structure.html#hash)
- Default value is `{}`

A hash of pattern-name and pattern tuples defining custom patterns to be used by the current filter. Patterns matching existing names will override the pre-existing definition. Think of this as inline patterns available just for this definition of grok

`patterns_dir`

- Value type is array (configuration-file-structure.html#array)
- Default value is `[]`

Logstash ships by default with a bunch of patterns, so you don't necessarily need to define this yourself unless you are adding additional patterns. You can point to multiple pattern directories using this setting. Note that Grok will read all files in the directory matching the patterns_files_glob and assume it's a pattern file (including any tilde backup files).

```
patterns_dir => ["/opt/logstash/patterns", "/opt/logstash/extra_patterns"]
```

Pattern files are plain text with format:

```
NAME PATTERN
```

For example:

```
NUMBER \d+
```

The patterns are loaded when the pipeline is created.

`patterns_files_glob`

- Value type is string (configuration-file-structure.html#string)
- Default value is `"*"`

Glob pattern, used to select the pattern files in the directories specified by patterns_dir

`periodic_flush`

- Value type is boolean (configuration-file-structure.html#boolean)
- Default value is `false`

Call the filter flush method at regular interval. Optional.

`remove_field`

- Value type is array (configuration-file-structure.html#array)
- Default value is `[]`

If this filter is successful, remove arbitrary fields from this event. Example:

```
filter {
  grok {
    remove_field => [ "foo_%{somefield}" ]
  }
}
```

```
# You can also remove multiple fields at once:
filter {
  grok {
    remove_field => [ "foo_%{somefield}", "my_extraneous_field" ]
  }
}
```

If the event has field `"somefield" == "hello"` this filter, on success, would remove the field with name `foo_hello` if it is present. The second example would remove an additional, non-dynamic field.

remove_tag

- Value type is array (configuration-file-structure.html#array)
- Default value is `[]`

If this filter is successful, remove arbitrary tags from the event. Tags can be dynamic and include parts of the event using the `%{field}` syntax.

Example:

```
filter {
  grok {
    remove_tag => [ "foo_%{somefield}" ]
  }
}
```

```
# You can also remove multiple tags at once:
filter {
  grok {
    remove_tag => [ "foo_%{somefield}", "sad_unwanted_tag"]
  }
}
```

If the event has field `"somefield" == "hello"` this filter, on success, would remove the tag `foo_hello` if it is present. The second example would remove a sad, unwanted tag as well.

tag_on_failure

- Value type is array (configuration-file-structure.html#array)
- Default value is `["_grokparsefailure"]`

Append values to the `tags` field when there has been no successful match

`tag_on_timeout`

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is string (configuration-file-structure.html#string)
- Default value is `"_groktimeout"`

Tag to apply if a grok regexp times out.

`timeout_millis`

edit (https://github.com/logstash-plugins/logstash-filter-grok/edit/master/lib/logstash/filters/grok.rb)

- Value type is number (configuration-file-structure.html#number)
- Default value is `30000`

Attempt to terminate regexps after this amount of time. This applies per pattern if multiple patterns are applied This will never timeout early, but may take a little longer to timeout. Actual timeout is approximate based on a 250ms quantization. Set to 0 to disable timeouts

« geoip (plugins-filters-geoip.html) i18n » (plugins-filters-i18n.html)

Top Videos

- Elasticsearch Demo (https://www.elastic.co/webinars/getting-started-elasticsearch?baymax=default&elektra=docs&storm=top-video)
- Kibana 101 (https://www.elastic.co/webinars/getting-started-kibana?baymax=default&elektra=docs&storm=top-video)
- Logstash Primer (https://www.elastic.co/webinars/getting-started-logstash?baymax=default&elektra=docs&storm=top-video)

On this page