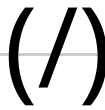


[Docs \(/guide\)](#)[Docs \(/guide\)](#)

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

[Logstash Reference \[5.3\] \(index.html\)](#) » [Configuring Logstash \(configuration.html\)](#) » [Logstash Configuration Examples](#)

« [Using Environment Variables in the Configuration](#) [Reloading the Config File](#) » ([reloading-config.html](#)) ([environment-variables.html](#))

Logstash Configuration Examples

[edit \(https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc\)](#)

The following examples illustrate how you can configure Logstash to filter events, process Apache logs and syslog messages, and use conditionals to control what events are processed by a filter or output.

Configuring Filters

[edit \(https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc\)](#)

Filters are an in-line processing mechanism that provide the flexibility to slice and dice your data to fit your needs. Let's take a look at some filters in action. The following configuration file sets up the `grok` and `date` filters.

```
input { stdin { } }

filter {
  grok {
    match => { "message" => "%{COMBINEDAPACHELOG}" }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

Run Logstash with this configuration:

```
bin/logstash -f logstash-filter.conf
```

Now, paste the following line into your terminal so it will be processed by the stdin input:

```
127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] "GET /xampp/status.php HTTP/1.1" 200 3891
"http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"
```

You should see something returned to stdout that looks like this:

```
{
  "message" => "127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] \"GET /xampp/status.php HTTP/1.1\" 200 3891 \"http://cadenza/xampp/navi.php\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\"",
  "@timestamp" => "2013-12-11T08:01:45.000Z",
  "@version" => "1",
  "host" => "cadenza",
  "clientip" => "127.0.0.1",
  "ident" => "-",
  "auth" => "-",
  "timestamp" => "11/Dec/2013:00:01:45 -0800",
  "verb" => "GET",
  "request" => "/xampp/status.php",
  "httpversion" => "1.1",
  "response" => "200",
  "bytes" => "3891",
  "referrer" => "\"http://cadenza/xampp/navi.php\"",
  "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\""
}
```

As you can see, Logstash (with help from the `grok` filter) was able to parse the log line (which happens to be in Apache "combined log" format) and break it up into many different discrete bits of information. This is extremely useful once you start querying and analyzing our log data. For example, you'll be able to easily run reports on HTTP response codes, IP addresses, referrers, and so on. There are quite a few `grok` patterns included with Logstash out-of-the-box, so it's quite likely if you need to parse a common log format, someone has already done the work for you. For more information, see the list of Logstash `grok` patterns (<https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns>) on GitHub.

The other filter used in this example is the `date` filter. This filter parses out a timestamp and uses it as the timestamp for the event (regardless of when you're ingesting the log data). You'll notice that the `@timestamp` field in this example is set to December 11, 2013, even though Logstash is ingesting the event at some point afterwards. This is handy when backfilling logs. It gives you the ability to tell Logstash "use this value as the timestamp for this event".

Processing Apache Logs

[edit \(https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc\)](https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc)

Let's do something that's actually **useful**: process apache2 access log files! We are going to read the input from a file on the localhost, and use a conditional (event-dependent-configuration.html#conditionals) to process the event according to our needs. First, create a file called something like *logstash-apache.conf* with the following contents (you can change the log's file path to suit your needs):

```
input {
  file {
    path => "/tmp/access_log"
    start_position => "beginning"
  }
}

filter {
  if [path] =~ "access" {
    mutate { replace => { "type" => "apache_access" } }
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  }
  date {
    match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
  }
  stdout { codec => rubydebug }
}
```

Then, create the input file you configured above (in this example, "/tmp/access_log") with the following log entries (or use some from your own webserver):

```
71.141.244.242 - kurt [18/May/2011:01:48:10 -0700] "GET /admin HTTP/1.1" 301 566 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/
3.6.3"
134.39.72.245 - - [18/May/2011:12:40:18 -0700] "GET /favicon.ico HTTP/1.1" 200 1189
"-" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.5
0727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2; .NET4.0C; .NET4.0E)"
98.83.179.51 - - [18/May/2011:19:35:08 -0700] "GET /css/main.css HTTP/1.1" 200 1837
"http://www.safesand.com/information.htm" "Mozilla/5.0 (Windows NT 6.0; WOW64; rv:
2.0.1) Gecko/20100101 Firefox/4.0.1"
```

Now, run Logstash with the -f flag to pass in the configuration file:

```
bin/logstash -f logstash-apache.conf
```

Now you should see your apache log data in Elasticsearch! Logstash opened and read the specified input file, processing each event it encountered. Any additional lines logged to this file will also be captured, processed by Logstash as events, and stored in Elasticsearch. As an added bonus, they are stashed with the field "type" set to "apache_access" (this is done by the `type => "apache_access"` line in the input configuration).

In this configuration, Logstash is only watching the apache access_log, but it's easy enough to watch both the access_log and the error_log (actually, any file matching `*log`), by changing one line in the above configuration:

```
input {  
  file {  
    path => "/tmp/*_log"  
    ...  
  }  
}
```

When you restart Logstash, it will process both the error and access logs. However, if you inspect your data (using `elasticsearch-kopf`, perhaps), you'll see that the access_log is broken up into discrete fields, but the error_log isn't. That's because we used a `grok` filter to match the standard combined apache log format and automatically split the data into separate fields. Wouldn't it be nice if we could control how a line was parsed, based on its format? Well, we can...

Note that Logstash did not reprocess the events that were already seen in the access_log file. When reading from a file, Logstash saves its position and only processes new lines as they are added. Neat!

Using Conditionals

[edit \(https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc\)](https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc)
You use conditionals to control what events are processed by a filter or output. For example, you could label each event according to which file it appeared in (access_log, error_log, and other random files that end with "log").

```
input {
  file {
    path => "/tmp/*_log"
  }
}

filter {
  if [path] =~ "access" {
    mutate { replace => { type => "apache_access" } }
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
      match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  } else if [path] =~ "error" {
    mutate { replace => { type => "apache_error" } }
  } else {
    mutate { replace => { type => "random_logs" } }
  }
}

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

This example labels all events using the `type` field, but doesn't actually parse the `error` or `random` files. There are so many types of error logs that how they should be labeled really depends on what logs you're working with.

Similarly, you can use conditionals to direct events to particular outputs. For example, you could:

- alert nagios of any apache events with status 5xx
- record any 4xx status to Elasticsearch
- record all status code hits via statsd

To tell nagios about any http event that has a 5xx status code, you first need to check the value of the `type` field. If it's `apache`, then you can check to see if the `status` field contains a 5xx error. If it is, send it to nagios. If it isn't a 5xx error, check to see if the `status` field contains a 4xx error. If so, send it to Elasticsearch. Finally, send all apache status codes to statsd no matter what the `status` field contains:

```

output {
  if [type] == "apache" {
    if [status] =~ /^5\d\d/ {
      nagios { ... }
    } else if [status] =~ /^4\d\d/ {
      elasticsearch { ... }
    }
    statsd { increment => "apache.#{status}" }
  }
}

```

Processing Syslog Messages

[edit \(https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc\)](https://github.com/elastic/logstash/edit/5.3/docs/static/configuration.asciidoc)

Syslog is one of the most common use cases for Logstash, and one it handles exceedingly well (as long as the log lines conform roughly to RFC3164). Syslog is the de facto UNIX networked logging standard, sending messages from client machines to a local file, or to a centralized log server via rsyslog. For this example, you won't need a functioning syslog instance; we'll fake it from the command line so you can get a feel for what happens.

First, let's make a simple configuration file for Logstash + syslog, called *logstash-syslog.conf*.

```

input {
  tcp {
    port => 5000
    type => syslog
  }
  udp {
    port => 5000
    type => syslog
  }
}

filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[ %{POSINT:syslog_pid} \])?: %{GREEDYDATA:syslog_message}" }
      add_field => [ "received_at", "%{@timestamp}" ]
      add_field => [ "received_from", "%{host}" ]
    }
    date {
      match => [ "syslog_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
  }
}

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}

```

Run Logstash with this new configuration:

```
bin/logstash -f logstash-syslog.conf
```

Normally, a client machine would connect to the Logstash instance on port 5000 and send its message. For this example, we'll just telnet to Logstash and enter a log line (similar to how we entered log lines into STDIN earlier). Open another shell window to interact with the Logstash syslog input and enter the following command:

```
telnet localhost 5000
```

Copy and paste the following lines as samples. (Feel free to try some of your own, but keep in mind they might not parse if the `grok` filter is not correct for your data).

```
Dec 23 12:11:43 louis postfix/smtpd[31499]: connect from unknown[95.75.93.154]
Dec 23 14:42:56 louis named[16000]: client 199.48.164.7#64817: query (cache) 'amster
damboothuren.com/MX/IN' denied
Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php /usr/share/cacti/site/poller.ph
p >/dev/null 2>/var/log/cacti/poller-error.log)
Dec 22 18:28:06 louis rsyslogd: [origin software="rsyslogd" swVersion="4.2.0" x-pid
="2253" x-info="http://www.rsyslog.com"] rsyslogd was HUPed, type 'lightweight'.
```

Now you should see the output of Logstash in your original shell as it processes and parses messages!

```
{
  "message" => "Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php
/usr/share/cacti/site/poller.php >/dev/null 2>/var/log/cacti/poller-error.log)",
  "@timestamp" => "2013-12-23T22:30:01.000Z",
  "@version" => "1",
  "type" => "syslog",
  "host" => "0:0:0:0:0:0:1:52617",
  "syslog_timestamp" => "Dec 23 14:30:01",
  "syslog_hostname" => "louis",
  "syslog_program" => "CRON",
  "syslog_pid" => "619",
  "syslog_message" => "(www-data) CMD (php /usr/share/cacti/site/poller.php
>/dev/null 2>/var/log/cacti/poller-error.log)",
  "received_at" => "2013-12-23 22:49:22 UTC",
  "received_from" => "0:0:0:0:0:0:1:52617",
  "syslog_severity_code" => 5,
  "syslog_facility_code" => 1,
  "syslog_facility" => "user-level",
  "syslog_severity" => "notice"
}
```

« Using Environment Variables in the Configuration Reloading the Config File » (reloading-config.html)
(environment-variables.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>)

Logstash Reference: 5.3 ▼
Logstash Introduction (introduction.html)
Getting Started with Logstash (getting-started-with-logstash.html)
How Logstash Works (pipeline.html)
Setting Up and Running Logstash (setup-logstash.html)
Breaking changes (breaking-changes.html)
Upgrading Logstash (upgrading-logstash.html)
Configuring Logstash (configuration.html) <ul style="list-style-type: none"> Structure of a Config File (configuration-file-structure.html) Accessing Event Data and Fields in the Configuration (event-dependent-configuration.html) Using Environment Variables in the Configuration (environment-variables.html) Logstash Configuration Examples (config-examples.html) Reloading the Config File (reloading-config.html) Managing Multiline Events (multiline.html) Glob Pattern Support (glob-support.html)
Working with Filebeat Modules (filebeat-modules.html)
Deploying and Scaling Logstash (deploying-and-scaling.html)
Performance Tuning (performance-tuning.html)
Monitoring APIs (monitoring.html)
Working with plugins (working-with-plugins.html)
Input plugins (input-plugins.html)
Output plugins (output-plugins.html)
Filter plugins (filter-plugins.html)