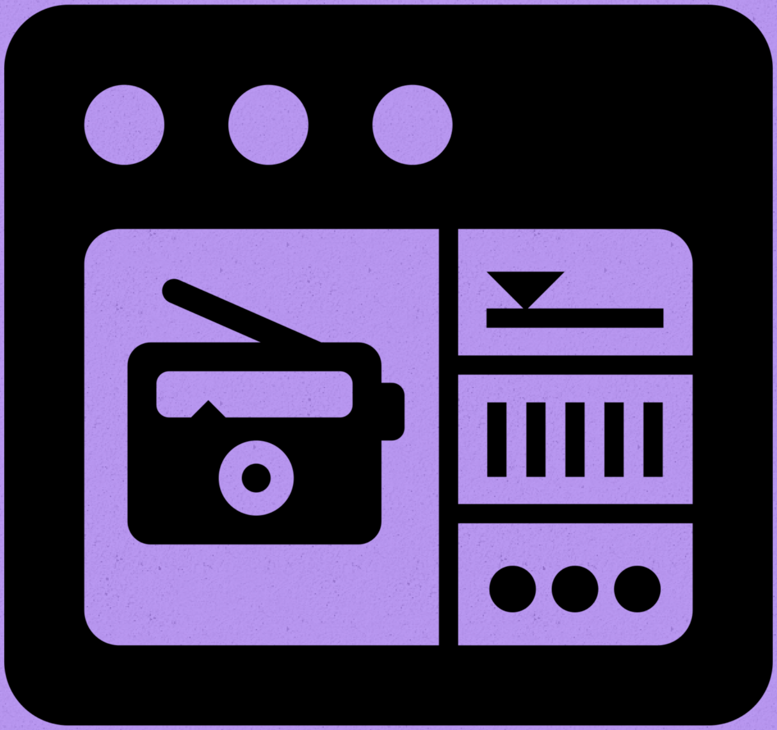
The Logstash Config Guide

Jurgens du Toit
@jrgns



Pain Free Logstash Configuration

The Logstash Config Guide

Pain Free Logstash Configuration

Jurgens du Toit

©2015 Jurgens du Toit

Tweet This Book!

Please help Jurgens du Toit by spreading the word about this book on Twitter!

The suggested hashtag for this book is #logstashconfig.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#logstashconfig

Contents

Welcome!

It always bothers me if the documentation of a project or library I'm using isn't complete. It slows down my development time and frustrates me if I have to jump into the source code to get it to work. This guide is the product of that frustration coupled with my enthusiasm for the Logstash project. It stands out for me as a system that can make connecting disconnected systems easy to do. With each new input or output plugin, the number of possibilities just grows and grows.

This guide's main aim is to provide an easy to read reference to the different Logstash plugins. If you can't configure a plugin after reading the guide on that plugin, then I have failed. Feel free to let me know what tripped you up and I'll make immediate ammends.

How to read this guide

Don't read it front to back. Well, you can, but I don't recommend it. It has a chapter per plugin. You can scan through all of the plugins to get an idea of what's available, but it's main use is as a reference: When you're working (or struggling!) with a specific plugin, go to that plugin's chapter and read through it. Get an idea of what the different options do and what the plugin requires, and take it from there.

Problems?

This is a WIP

This book is at the moment very much a Work In Progress. New chapters will be added regularly. If you've already bought the book, you'll be entitled to get these updates, free of charge! Please don't let the unfinished nature of the book prevent you from using it! Rather let me know what you think is missing, and I'll try and add it.

As I said, if you experience any issues with the guide, or if there's any plugin missing that you'd like to see covered, let me know, and I'll follow up as quickly as I can. You can contact me using the following means:

- Email: jrgns@eagerelk.com
- Twitter: @eagerelk / @jrgns
- Web: http://eagerelk.com

Happy stashing!

This is a Sample

If you looked at the table of contents, and thought that this is a very short guide, you're wrong. This is only a free sample of the larger Logstash Configuration Guide. It contains a number of chapters from each of the main sections of the guide to give you an idea of what you can expect from the guide.

How do I get the full guide?

The guide is available to buy online, and I'd greatly appreciate it if you will pay money for it. I'm sure you can get a bootleg copy somewhere, but I put a lot of hard work into the guide, and would appreciate the support. Besides, it's relatively cheap and will provide you with loads of value!

Go to [http://www.logstashconfigguide.com/][www.logstashconfigguide.com/] to buy it.

Thanks!

How to configure Logstash

Logstash has a simple configuration DSL that enables you to specify inputs, outputs and filters along with their specific options. Order matters, specifically around filters and outputs, as the configuration is basically converted into code and then executed. Keep this in mind when you're writing your configs and try to debug them.

Structure

Your configurations will generally have three sections: Inputs, outputs and filters. You can have multiple instances of each of these instances, which means that you can group related plugins together in a config file, instead of grouping them by type. My Logstash configs are generally structured as follows:

```
#/etc/logstash/conf.d/
- apache_to_elasticsearch.conf
- haproxy_to_elasticsearch.conf
- syslog_to_s3.conf
```

You'll see that I have a configuration file for each of the functions or integrations I'd like Logstash to perform. Each of those files will contain the necessary inputs, filters and outputs to perform that function. Let's look at the `apache_to_elasticsearch.conf` file, as it's typical of what you'd see in a logstash config file:

```
input {
  file {
    path => "/var/log/apache/access.log"
    type => "apache-access"
  }
}

filter {
  if [type] == "apache-access" {
    grok {
      type => "apache-access"
      pattern => "%{COMBINEDAPACHELOG}"
    }
  }
}

```

```
output {
  if [type] == "apache-access" {
    if "_grokparsefailure" in [tags] {
      null {}
    }

    elasticsearch {
    }
  }
}
```

The input section tells Logstash to pull logs from the Apache access log, and specify the `type` of those events as `apache-access`. Setting the `type` is important, as it will be used to selectively apply filters and outputs later on in the event's lifetime. It's also used to organize the events when it's eventually pushed to Elasticsearch.

In the filter section we specifically apply a `grok` filter to events that have the `apache-access` type. This conditional ensures that only the `apache-access` events get filtered. If it wasn't there, Logstash will attempt to apply the grok filter to events from other inputs as well. This filter parses the log string and populates the event with the relevant information from the Apache logs.

Lastly we see the output section. The first conditional ensures, once again, that we only operate on the `apache-access` events. The next, nested, conditional sends all the events that didn't match our grok pattern to the null output. Since they didn't conform to the specified pattern, we assume that it's log lines that contain information we're not interested in, and discard it. Since order is important in filters and outputs, this will ensure that only events that were succesfully parsed will make it to the `elasticsearch` output.

Each of the configuration files can contain these three sections. Logstash will typically combine all of our configuration files and consider it as one large config. Since you can have multiple inputs, it's recommended that you tag your events or assign types to them so that it's easy to identify them at a later stage. Also ensure that you wrap your filters and outputs that are specific to a category or type of event in a conditional, otherwise you might get some surprising results.

On Pushing and Pulling Data

Logstash requires a significant amount of resources to run properly. It's been reported that it can take up to five minutes for Lgostash to boot up on a 1st generation Raspberry Pi. This makes it less than ideal as a log shipper from various sources. My recommendation is to not configure Logstash as a publisher of data, but rather as a subscriber. This means that you'll have small, simple shippers that all publish data to a central point or buffer from where Logstash can then pull the data and transform it as necessary.

The ideal setup is having a shipper right next to the source, using a native library to send either the raw logs or a minimally parsed version of it to a queuing system or buffer. This buffer can either be Logstash itself through something like the TCP input operating as a server, or a message queue like RabbitMQ or even Redis. Use what ever is available on the source system so that you don’t add an extra burden on it. Logstash can then pull the raw logs from the buffer, transform it as necessary, and output it to the desired location.

Inputs

One of the things that make Logstash great is its ability to source logs and events from various sources. As of version 1.4.2 theres 41 different inputs on the Logstash documentation page. That's 41 different technologies, locations and services from where you can pull events and manipulate them. These include monitoring systems like `collectd`, databases like `redis`, services like `twitter` and various others like `file` and `rabbitmq`. By using these inputs you can import data from multiple sources and manipulate them however you want, and eventually send them to other systems for storage or processing.

Inputs are the starting point of any configuration. If you define no input, Logstash will automatically create a `stdin` input. Since you can create multiple inputs, it's important to type and tag them so that you can properly manipulate them in filters and outputs.

This section of the guide will look at the more interesting inputs and explain how to configure each of them. We'll consider a number of simple use cases, and explain how to get the data you want into your system.

"You can have data without information, but you cannot have information without data."

$\sim$ Daniel Keys Moran

IMAP Input

This chapter will look at the IMAP input. IMAP is an alternative protocol to POP3 to fetch emails from a server. This input will allow you to retrieve emails from an IMAP account and process them using Logstash. Check out this post on using IMAP and Elasticsearch to manage error emails for a use case of this input.

At a glance

Links: Source | Documentation Version: 0.1.4
Requirements:

IMAP account details

- Host
- Username
- Password

The short version

Setting up Logstash to connect to IMAP is really simple: Just set the IMAP account details in the config file and watch the emails roll in.

The only required options are `host`, `username` and `password`:

```
# Bare minimum
input {
  imap {
    host => "imap.mailserver.com"
    user => "you@mailserver.com"
    password => "$uper$ecret"
  }
}
```

Logstash will now check the `Inbox` folder of the `you@mailserver.com` account every 300 seconds and fetch up to 50 emails.

The longer version

Some email providers will require more settings than just the three required ones to work. You can also tweak how often the account should be checked for new emails, and how many emails should be fetched every time. Let's look at a couple of extra examples:

Fetch more, less often

You can set the fetch interval and and count using the well named `check_interval` and `fetch_-count` settings:

```
# Fetch 100 emails every hour
input {
  imap {
    host => "imap.mailserver.com"
    user => "you@mailserver.com"
    password => "$uper$ecret"
    check_interval => 3600 # Specified in seconds
    fetch_count => 100
  }
}
```

Give Logstash it's own GMail account

GMail and Google Apps accounts are widely used, not just by people but by some services as well. We'll be using the `port` setting to use the correct port, and the `secure` setting to encrypt the connection. The following config shows how you can check your Google emails using Logstash:

```
# Connecting to GMail
input {
  imap {
    host => "imap.gmail.com"
    user => "you@gmail.com"
    password => "$uper$ecret"
    port => 993
    secure => true
  }
}
```

All of the options

`check_interval`
: Numeric - How long, in seconds, Logstash waits before checking the IMAP account again. Defaults to 300.

`content_type`
: String - Emails can be sent as Multipart messages. Use this setting to specify the content type of the message part you're interested in. Defaults to `text/plain`.

`delete`
: Boolean - Specify whether or not emails should be deleted from the server after they've been fetched by Logstash. The default value is false.

`fetch_count`
: Numeric - The number of emails to retrieve every time Logstash connects to the IMAP server. Defaults to 50.

`folder`
: String - The folder that should be polled for new emails. Defaults to `INBOX`.

`host`
: String - The hostname of the server on which the IMAP account is hosted. This setting is required.

`lowercase_headers`
: Boolean - Specify whether or not the headers of the email should be lowercased. This is usefull if you want to ensure that all the headers (that are added to the event) are homogenous across all the events. The default value is true.

`password`
: String - The password for the IMAP account in plain text. This setting is required.

`port`
: Numeric - The port number on which to connect to the IMAP service. There is no default for this setting.

`secure`
: Boolean - Specify whether or not to use a secure connect. The default value is true.

`user`
: String - The full username for the IMAP account. This setting is required.

`verify_cert`
: Boolean - Specify whether or not to verify the SSL certificate (used for secure communication) when connecting to the server. This is usefull if you're connecting to an internal server with a self signed certificate. The default value is true.

Filters

If Logstash was just a dumb pipe between a number of inputs and outputs, you could easily replace it with a service like IFTTT or Zapier. Luckily it isn't. It also comes with a number of very powerful filters with which you can manipulate, measure and create events. It's the power of these filters that makes Logstash a very versatile and valueable tool.

Logstash events can come from multiple sources, so, as with outputs, it's important to do checks on whether or not an event should be processed by a particular filter.

In this section we'll look at a number of the available Logstash filters and how they can be used to increase the value of your event stream.

"If you torture the data long enough, it will confess to anything."

∼ Ronald Coase

Metrics Filter

It often happens that you're not intersted in actual events, but just the number of times an event occurs in a specific period of time. Logstash supports counting events and measuring their velocity rates through the metrics filter.

At a glance

Links: Source | Documentation Version: 0.1.8
Requirements:

- An active Logstash input

The short version

The metrics filter can operate in one of two modes: Meter or timer. In `meter` mode it will count the number of events, and emit a new event containing the total number of events and the short, mid and long term rates of events at regular intervals. In `timer` mode it will emit a new event containing the count, short, mid and long term rates and various statistics about the number of events that passed through it at set intervals.

```
# Bare minimum Meter
filter {
  metric {
    meter => [ "log_events" ]
  }
}

# Bare minimum Timer
filter {
  metric {
    timer => [ "log_events", "%{request_time}" ]
  }
}
```

The **meter** filter will create a new event every 5 seconds, giving you the following stats of events up to this point:

- log_events.count - The total number of events up to now

- log_events.rate_1m - The average number of events per minute
- log_events.rate_5m - The average number of events in 5 minutes
- log_events.rate_15m - The average number of events in 15 minutes

You'll notice that the value of the `meter` option was used to name the event properties. This is useful to differentiate between different metrics.

The **timer** filter will create a new event every 5 seconds, giving you the same stats of the event as the `meter` filter above, as well as the following stats on the `request_time` property of the event:

- request_time.min - The smallest `request_time` value the filter has seen so far.
- request_time.max - The largest `request_time` value the filter has seen so far.
- request_time.stddev - The standard deviation of the value the filter has seen so far.
- request_time.mean - The mean of the values the filter has seen so far.
- request_time.p1 - The 1th percentile for the measured values.
- request_time.p5 - The 5th percentile for the measured values.
- request_time.p10 - The 10th percentile for the measured values.
- request_time.p90 - The 90th percentile for the measured values.
- request_time.p95 - The 95th percentile for the measured values.
- request_time.p99 - The 99th percentile for the measured values.
- request_time.p100 - The 100th percentile for the measured values.

Once again the value passed as the key of the hash is used to construct the event properties.

The long version

The metrics filter has a couple of options with which you can tweak it's behaviour to suit your own needs. By default Logstash will keep on counting events and report on all the events it has received so far until it is stopped. The first example shows how you can use the `clear_interval` setting to reset the count ever 60 minutes:

```
# Only count for 60 minutes
filter {
  metric {
    meter => [ "log_events" ]
    clear_interval => 3600 # This needs to be multiples of 5
  }
}
```

By default a new event is created every 5 seconds. This can be altered using the `flush_interval` option:

```
# Create a new event every 5 minutes
filter {
  metric {
    timer => [ "log_events", "%{request_time}" ]
    flush_interval => 300 # This needs to be multiples of 5
  }
}
```

Each new event contains a lot of data. You can limit that using the `rates` and `percentiles` options:

```
# Don't report on rates, only give 50th and 95th percentiles
filter {
  metric {
    timer => [ "log_events", "%{request_time}" ]
    percentiles => [50, 95]
    rates => []
  }
}
```

The `percentiles` is only applicable to the `timer` metric. The `rates` options to both the `timer` and `meter` metrics.

All of the options

`meter`
: Array - The metered rates you want to generate. You need to specify at least one of `timer` or `meter`.

`timer`
: Hash - Specify the timed rates as well as the value you want to measure. You need to specify at least one of `timer` or `meter`.

`percentiles`
: Array - The percentile values you want to include in your `timer` metrics. Defaults to `[1, 5, 10, 90, 95, 99, 100]`.

`rates`
: Array - The velocity rates you want to include in your `timer` and `meter` metrics. The only possible values are 1, 5, and 15. Defaults to `[1, 5, 15]`.

`clear_interval`
: Number - How often, in seconds, the counters should be cleared and start over. It needs to be a multiple of 5. Defaults to -1, which means never.

`flush_interval`
: Number - How often, in seconds, a metric event should be generated. It needs to be a multiple of 5, and defaults to 5.

`ignore_older_than`
: Number - You can use this option to exclude events that are too old. This is useful to only include near real time events. Defaults to 0 (disabled).

Outputs

As with the inputs, Logstash comes with a number of outputs that enable you to push your events to various locations, services and technologies. You can store events using outputs such as `file`, `csv` and `s3`, convert them into messages with `rabbitmq` and `sqs`, or send them to various services like `hipchat`, `pagerduty` or `irc`. The number of combinations of inputs and outputs in Logstash makes it a really versatile event transformer.

Logstash events can come from multiple sources, so, as with filters, it's important to do checks on whether or not an event should be processed by a particular output. If you define no output, Logstash will automatically create a `stdout` output.

The outputs section of the guide will look at a number of outputs and what is required to set them up. We'll also discuss some of the use cases and emergent behaviour.

"Don't cry because it's over. Smile because it happened."

$\sim$ Dr. Seuss

SNS Output

Logstash comes with a number of plugins that interact with AWS, Amazon's cloud infrastructure. This chapter will look at the configuration options for the Simple Notification Service (or SNS) output that will allow you to convert Logstash events to push messages to various mobile devices, as well as to more traditional messages such as SMS texts and emails.

At a glance

Links: Source | Documentation Version: 0.1.4
Requirements:

- AWS Account
- AWS Credentials with access to the SNS service OR
- An instance of Logstash running on EC2 with access to the SNS service.

The short version

Logstash needs to know what AWS credentials it should use to access the SNS service. Configure those using the AWS config plugin options. All of these examples assume that access to the service is set up using the environment variables.

The only two required options (other than the credentials) are `region` and `arn`:

```
# Bare minimum
output {
  sns {
    region => "us-east-1" # The AWS region for the service
    arn => "arn:aws:sns:us-east-1:770975001275:logstash-testing" # The SNS topic\
 to which the event should be sent
  }
}
```

With this setup you'll be sending all of your events to the `logstash-testing` topic in plain text format. You can omit the `arn` parameter if your event contains an `sns` field. See below for more detail.

The longer version

You can use the fields present in the Logstash event to further customize the SNS output.

SNS alert content

The subject and the body of the SNS alert will be set to the contents of the `sns_subject` and `sns_message` fields respectively if they are present in the event. If the `sns_subject` field is not present, the event's source will be used instead. If the `sns_message` field isn't present, the whole event will be converted to plain text (by default) or JSON (if `format` is set to `json`).

The subject will be truncated to 100 characters and the message to 32768 bytes.

```
# SNS message content
filter {
  mutate {
    rename => [ "message", "sns_message" ] # Use the message property as the con\
tent of the SNS alert
    add_field => { "sns_subject" => "Alert: %{type}" # Use the type property as \
the subject of the SNS alert
  }
}
```

ARN

Logstash events can be sent to different SNS topics by populating the `sns` field of the event.

```
# Custom ARN
filter {
  mutate {
    add_field => { "sns" => "arn:aws:sns:us-east-1:770975001275:type-%{type}" # \
Send events to different topics dependent on the event type
  }
}
```

All of the options

`arn` String - The Amazon Resource Name (ARN) for the topic the message should be published to. You need to specify this option.

`format`
String - This parameter can be used to specify that the message should be encoded as JSON or plain text. The possible values are `json` and `plain`. This option won't be used if the `sns_message` property is present in the event. Defaults to `plain`.

`publish_boot_message_arn`

String - Specify an ARN for a SNS topic in this parameter if you'd like to be notified when this plugin boots up. This happens in the registration phase of the plugin.

General

This section will look at a number of concepts and configurations that are common accross all the plugins. All of the plugins inherit a number of options from a base plugin. If you're familiar with Object Oriented Programming, all of the plugins are child classes of the `Logstash::Plugin` class. Concepts such as Amazon authentication is also reused among some of the plugins, so I devote a chapter to explaining the concepts around that as well.

Beyond that we'll also discuss a number of use cases and implementation of Logstash in real life.

"A good programmer is someone who looks both ways before crossing a one-way street."

~ Doug Linder

AWS services plugins

A number of Logstash's plugins interact with AWS, Amazon's cloud infrastructure. This chapter will look at the common configuration options for these plugins.

At a glance:

Links: Source
Requirements:

- AWS Account
- AWS Credentials with access to the required service OR
- An instance of Logstash running on EC2 with access to the required service.

Minimum setup

- If you're running Logstash on an EC2 instance, you don't need to do anything.
- If you have multiple plugins using the same credentials, use ENV variables.
- If you have multiple plugins using different credentials, use the `aws_credentials_file` parameter.
- Otherwise go for the `access_key_id` and `secret_access_key` parameters.

All the AWS related options

All of Logstash's AWS related plugins try several methods to retrieve the AWS credentials it should use to authenticate itself to AWS. Here's the options in order of priority, along with a short discussion on the usability of each option:

1. The key and secret specified in the logstash config options

The plugin first checks for the access key id and secret access key in the `access_key_id` and `secret_access_key` parameters for the specific plugin, in this case the SNS output. You'll need to specify both:

```
output {
    sns {
        access_key_id => "Access Key ID"
        secret_access_key => "Access Key Secret"
    }
}
```

The fact that the credentials is saved in plain text in something other than your configuration management system might pose a security risk. On the other hand, if you're using a template in your config management to create the file, it shouldn't be a problem.

2. The key and secret specified in an external config file

If the key and secret isn't specified as parameters, Logstash checks for a credentials file specified with `aws_credentials file` that should contain the key and secret. The file should contain YAML:

```
output {
    sns {
        aws_credentials_file => "/etc/aws_credentials.yaml"
    }
}
```

This is a simpler and more centralized way to handle the credentials than the first option. If you have multiple plugins using the same credentials, this is the way to go. Of all the options, it's the easiest to manage, has the most flexibility, and you're less likely to commit and push your AWS credentials to your repo.

```
# /etc/aws_credentials.yaml
:access_key_id: "12345"
:secret_access_key: "54321"
```

3. Environment variables

If you prefer to have your secrets as environment variables, Logstash will check the following pairs of environment variables for the key and secret.

Firstly, it checks `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

Secondly, it checks `AMAZON_ACCESS_KEY_ID` and `AMAZON_SECRET_ACCESS_KEY`.

The drawback with using ENV variables is that all the plugins in the current environment will be using the same credentials. If you have different access profiles for the different services, you won't be able to use this.

4. IAM Instance Profile

As a last gasp effort, Logstash will check for an IAM Instance Profile. This is only available if Logstash is running on an EC2 instance.

It has the same drawback as placing the key and secret as ENV variables, in that you only have one set of credentials. It's also the easiest setup in that you have to do absolutely nothing to set it up.

Logstash plugins using AWS

SQS Input
Send SQS events to Logstash for further processing. Read more about the SQS input config.

S3 Output
Write Logstash events to an AWS S3 bucket. Read more about the S3 output config.

SQS Output
Push Logstash events to an AWS SQS queue. Read more about the SQS output config.

SNS Output
Send Logstash events as notifications through the AWS SNS service. Read more about the SNS output config.

Cloudwatch Output
Record metrics using the AWS Cloudwatch service. Read more about the Cloudwatch output config.

And we’re done!

If you’re reading this, you probably disregarded my advice to read the guide front to back. Nice. :)

Either way, well done on reaching the end. I hope you learned a lot and that the guide provided you with a lot of value.

Feel free to contact me if you have any questions or comments, or if you’d just like to share your experience of using Logstash!

Thanks

Jurgens du Toit | jrgns@eagerelk.com | @jrgns