Purpose

This document describes how to customize Nagios Log Server Filters, it is intended for use by Nagios Log Server Administrators.

Target Audience

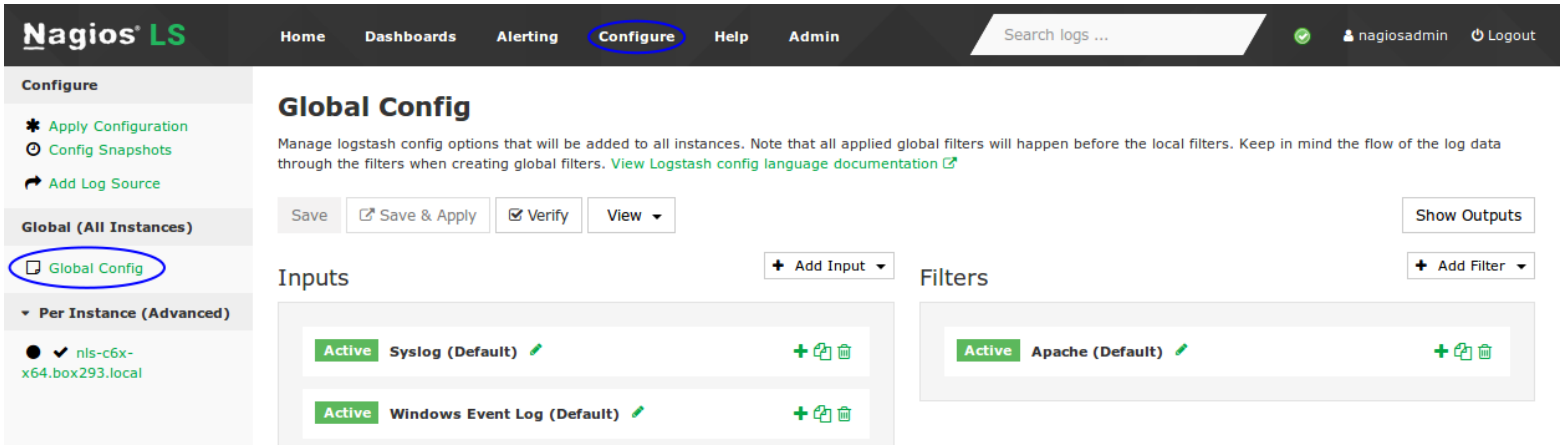
This document is intended for use by Nagios Log Server Administrators who want to customize their Nagios Log Server Filters.

What Are Filters?

Filters can be applied to messages before they are sent to Elasticsearch for indexing. They perform actions such as breaking apart messages into fields for easy searching, adding geo location information, resolving IP to DNS names and dropping messages you do not want indexed.

Filter Configuration Location

Nagios Log Server is a cluster oriented application that uses Logstash to receive and process logs. The base configuration provided with a default installation of Nagios Log Server has all of the filters defined as part of the Global Config. The Global Config is an easy way to set up the same Logstash configuration on all your instances. To access the configuration navigate to **Configure** > **Global (All Instances)** > **Global Config**.



On the Global Config page there are two main tables named Inputs and Filters *(and a third table called Outputs which is hidden)*. Inputs, Filters and Outputs are all used by Logstash to process incoming log data and do something with it, which normally is to store it in the Elasticsearch database. This document will be focusing on **Filters**.

Filters Explained

The filters are in a structured format like this:

```
<filter_type> {
     <filter_action> => [ '<matching_field>', '<matching_pattern>' ]
}
```

`<filter_type>` is the filter plugin that will be used for the `<filter_action>`. Logstash allows a large amount of possible filter types, this documentation will explore some that are useful for manipulating logs.

This example will explain how the **grok** `<filter_type>` can used for filtering. Grok is a plugin that is used by Logstash for making specific filters using regular expressions and matching. The grok documentation explains it as:

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

The grok plugin allows a lot of customization and will be used heavily in making custom filters in your Nagios Log Server configuration.

`<filter_action>` is the action that will be taken using the filter type. Common grok actions are **match**, **add_field** and **add_tag**.

<matching_field> will be the log field that is being processed. When Logstash receives logs from sources like syslog, the received data is categorized by certain fields (like the message field that contains some text about the log entry). You can choose things like hostname, severity, message and many more.

<matching_pattern> relates to the pattern that will be looked for in the <matching_field>, the data matched by the pattern(s) can be turned into new fields stored in the Elasticsearch database.

A real world example will help explain this in more detail. In this example the field message will be the <matching_field>. Here is an example of a line from an Apache error_log:

```
[Wed Oct 22 12:45:58 2014] [error] [client 192.168.5.19] File does not
exist: /usr/local/nagiosxi/html/someURL
```

This message from the Apache error_log is showing that the someURL file does not exist. It shows the time the notice was posted, the type [error], the client trying to access the someURL file, the log message and the path that generated the request error.

What is important is that the data in this line is following a structured format, every time an entry is received from this input the structure of the entry will always be the same. Identifying this structure is done by using the <matching_pattern> to break this string of text into fields. Here's the structured data broken down:

```
[Wed Oct 22 12:45:58 2014]
[error]
[client 192.168.5.19]
File does not exist: /usr/local/nagiosxi/html/someURL
```

To take control of this message and filter the whole message or parts of it you will use a grok pattern that relates to the pattern of the message.

The filter shown below will be applied to any logs with the `apache_error` program name. This is one of the two default filters that are present inside Nagios Log Server.

```
if [program] == 'apache_error' {
   grok {
        match => [ 'message', '\[(?<timestamp>%{DAY:day} %{MONTH:month} %
        {MONTHDAY} %{TIME} %{YEAR})\] \[%{WORD:class}\] \[%{WORD:originator} %
        {IP:clientip}\] %{GREEDYDATA:errmsg}' ]
   }
   mutate {
        replace => [ 'type', 'apache_error' ]
   }
}
```

At this point the focus will be on the `match` line (it wraps over three lines). The first line is explained in the Filter Conditions section and the mutate lines will be explained in the Mutate section.

Match

The line starts off like this:

```
   match => [ 'message', '
```

The `'message'` is the `<matching_field>` field that contains the data you want to break into fields, take note that it is surrounded by single quotes. You can also see that the square bracket `[` is used to begin the match definition *(filter structure)*. A single quote is then used to start the `<matching_pattern>`.

The first part of the `<matching_pattern>` is to handle the date/time stamp.

Here is the example from the log file:

```
[Wed Oct 22 12:45:58 2014]
```

Here is the pattern:

```
\[(?<timestamp>%{DAY:day} %{MONTH:month} %{MONTHDAY} %{TIME} %{YEAR})\]
```

You can see that the words in CAPITALS represent the pattern being matched, like `%{MONTHDAY}` which corresponds to `22` in the string. The words in CAPITALS is how Logstash makes it easy for you to be able to use regular expressions.

If you look at `%{MONTH:month}`, the `:month` is taking the value for `%{MONTH}` and turning it into a field that is stored in the Elasticsearch database. This `":<field_name>"` syntax is how you create new fields in the Elasticsearch database.

This section of the pattern match also needs explaining:

```
(?<timestamp>
```

The grok documentation explains this a Custom Pattern:

> *Sometimes logstash doesn't have a pattern you need. For this, 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)
```

The `?<timestamp>` is storing the whole pattern string into a field called `timestamp` that gets stored in the Elasticsearch database. The whole pattern is surrounded by round brackets `()`.

These square brackets are very important:

```
\[
\]
```

The square brackets are preceded by a backslash `\` because the backslash is telling the grok pattern to treat the square bracket as plain text *(the date/time stamp in the log file is surrounded by square brackets)*. Otherwise the square bracket will make Logstash think it's part of the filter structure and will result in an invalid config. This practice of using a backslash is called "escaping".

To summarize, the pattern for the date/time stamp is creating the following fields in the Elasticsearch database:

```
timestamp, day, month
```

The next part of the pattern is as follows:

```
\[%{WORD:class}\]
```

Which applies to this text:

```
[error]
```

The pattern WORD is an expression to say that the next pattern to match is a single word without spaces (the range is `A-Za-z0-9_`). This will be stored in the Elasticsearch database as the field called `class`. Once again the square brackets are escaped as they form part of the string.

However to make it very clear, here is that same example with the last piece of the date/time stamp pattern:

```
%{YEAR})\] \[%{WORD:class}\]
```

There is a space between the `\] \[` as the log data received also has a space:

```
2014] [error]
```

It is very important that the overall pattern you are defining is accurate, otherwise the pattern will not be matched and the log data will not be broken up into fields in the Elasticsearch database.

The next part of the pattern is as follows:

```
\[%{WORD:originator} %{IP:clientip}\]
```

Which applies to this text:

```
[client 192.168.5.19]
```

The first `WORD` will be stored in the Elasticsearch database as the field called `originator`.

The pattern `IP` is an expression that will identify an IPv4 or IPv6 address, this will be stored in the Elasticsearch database in the field called `clientip`.

The last part of the pattern is as follows:

```
%{GREEDYDATA:errmsg}' ]
```

Which applies to this text:

```
File does not exist: /usr/local/nagiosxi/html/someURL
```

The pattern `GREEDYDATA` is an expression which takes all the remaining text, it will be stored in the Elasticsearch database as the field called `errmsg`.

The single quote ends the entire `<matching_pattern>` and the square bracket `]` ends the filter structure.

That example of the match line should have helped explain how the data received by Logstash is broken up and stored in separate fields in the Elasticsearch database.

The next sections will explain the other lines in the filter example that was provided earlier.

Mutate

Mutate is another very useful plugin used in Logstash filtering. This allows you to replace or append a value in any field with a new value that may replace the whole field or add, delete or append portions of it. Here is the mutate line from earlier:

```
mutate {
     replace => [ 'type', 'apache_error' ]
}
```

In our example above we are changing the log type which would be `syslog` and replace it with `apache_error`. This allows you to differentiate between normal syslogs and apache syslogs.

Filter Conditions

An filter can be restricted to certain logs by using a simple if statement:

```
if [program] == 'apache_error' {
     grok {
     }
     mutate {
     }
}
```

Here you can see that the `[program]` must be `apache_error` for these filters to be applied.

How does the log received by Logstash know that the program is `apache_error`? When you run the setup script on your Linux server that is running Apache it will define the name as `apache_error`.

In Nagios Log Server navigate to **Configure** > **Add Log Source** and select **Apache Server**. Under the **Run the Script heading** you can see the following line of code:

```
sudo bash setup-linux.sh -s nagios_log_server -p 5544 -f
"/var/log/httpd/error_log" -t apache_error
```

You can see that **apache_error** is being defined as part of the setup script. The syslog application on the sending server will define these log entries as coming from `apache_error`.

This completes the explanation of the above example for Apache error_log messages.

Patterns

Patterns turn complex regular expressions and define them as understandable words. For example:

```
MONTHDAY = (?:(?:0[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])
GREEDYDATA  = .*
WORD = \b\w+\b
```

As you can see they are complicated and not easy to understand, but words like `MONTHDAY` are easy to understand.

If you would like to see a list of available patterns you will need to refer to the source code. Here is the source code for grok patterns:

https://github.com/elastic/logstash/blob/1.4/patterns/grok-patterns

All of the available patterns are in different files which are located here:

https://github.com/elastic/logstash/tree/1.4/patterns

Filter Configuration Options

Navigate to **Configure** > **Global (All Instances)** > **Global Config**. In the Filters table there are several pre-configured filters that come as part of Nagios Log Sever, these are called **blocks**. The blocks have several icons which are explained as follows.

Each of these blocks are Active, which is indicated by the **Active** box next to each filter

When you click the Active box the filter will be marked as **Inactive** and the next time the configuration is applied this filter will not be part of the running configuration

This allows you to save filters even if you don't want to use them right away

Clicking the box again will toggle it back to Active

- Rename
  - Allows you to rename the block
- Open / Close
  - This will expand the block to show the filter configuration in a text field
  - Please refer to the Filters Explained section of this document for more information
  - The icon will change to a hyphen when open, clicking the hyphen will collapse the block
- Copy
  - This will create a duplicate of that block
  - Allows you to easily create a new filter based off the configuration of an existing filter
- Remove
  - Delete a block when it is no longer required

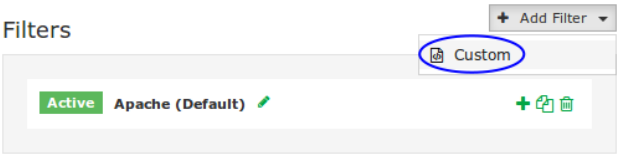
Any changes you make will not be saved until you click the **Save** button. Keep this in mind as when you navigate away from the page all of your changes will be lost.

Adding A Filter

The best way to learn how filters work is to add one and then send some log data. This could become a very complicated topic however the following example is kept simple.

The following example is going to rely on the **File Test** Input that was demonstrated in the Configuring Nagios Log Server Inputs documentation. Please return here after you have the input created, here is an example screenshot of the input.

On the **Global Config** page click the **Add Filter** drop down list and select **Custom**.



A new filter will appear at the bottom of the list of Filters. Type a unique **name** for the filter.

In the text field you will need to define the filter configuration. Paste the following example:

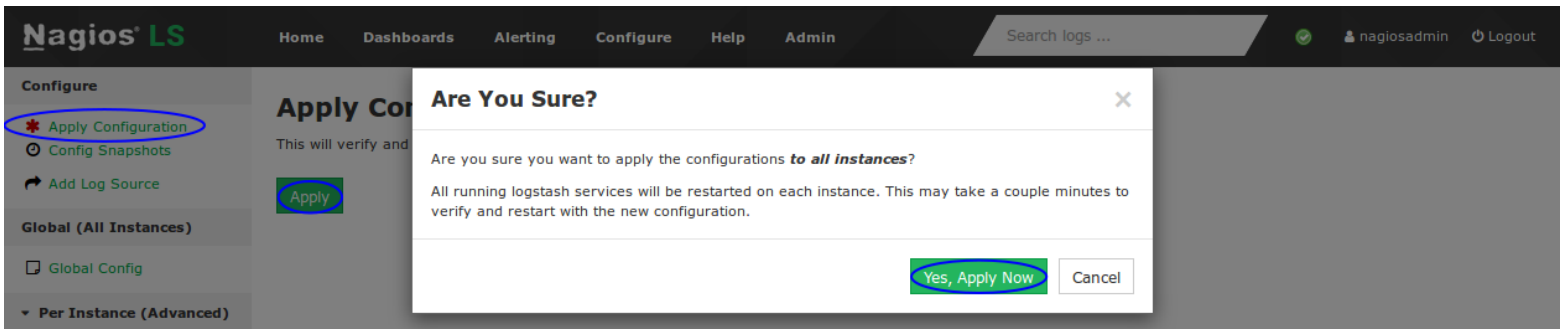
```
if [type] == 'testing' {
     grok {
          match => [ 'message', '%{WORD:first_word} %{WORD:second_word} %
          {GREEDYDATA:everything_else}' ]
     }
}
```

Once you have finished click the **Save** button.

For the new filter to become active you will need to Apply Configuration, however it is recommended that you Verify the configuration first. The next step will be to Apply Configuration to put this new filter into the running configuration so it can be tested and demonstrated.

Apply Configuration

To apply the configuration, in the left hand pane under **Configure** click **Apply Configuration**. Click the **Apply** button and then on the modal that appears click the **Yes, Apply Now** button.



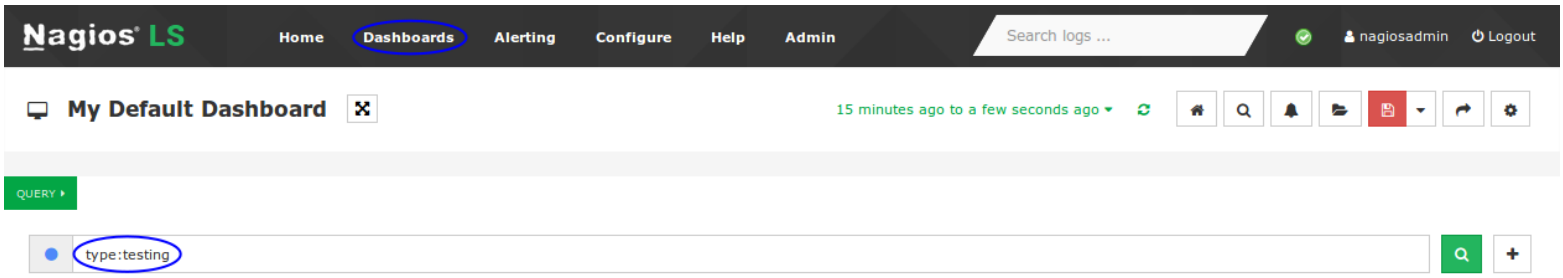
Wait while the configuration is applied. When it completes the screen will refresh with a completed message. Please proceed to the Test Filter section.

Test Filter

Using the filter example created previously, these steps will show you how to confirm the filter is working. Establish a terminal session to your Nagios Log Server instance and then execute the following command:

```
echo "This is a test log entry" >> /tmp/test.log
```

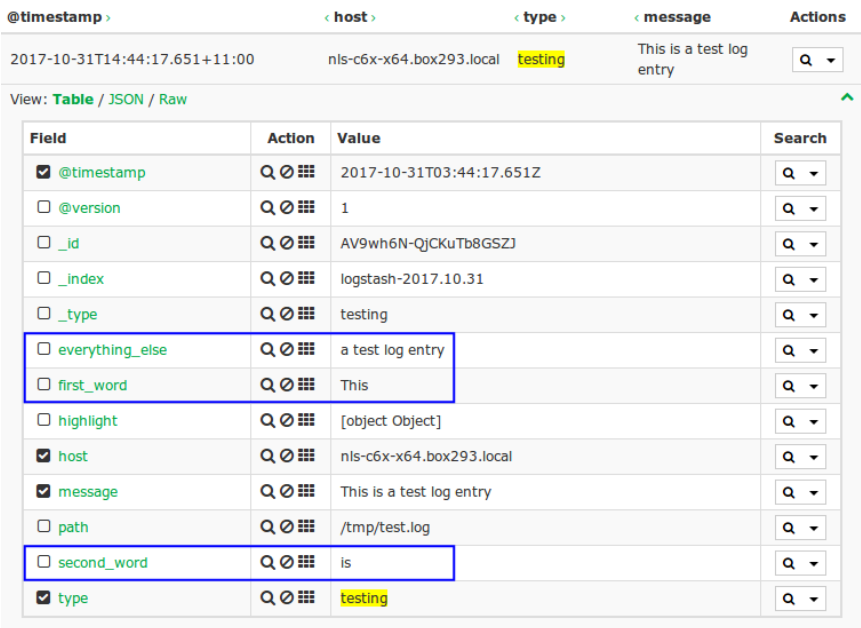
Now in Nagios Log Server open the Dashboards page and perform the query `type:testing` as per this screenshot:



The query should return one result in the ALL EVENTS panel.

Clicking on the log entry will show you the full details about the entry.

Here you can see that the filter successfully broke the `message` field into the three fields of `first_word` (this), `second_word` (is) and `everything_else` (a test log entry).

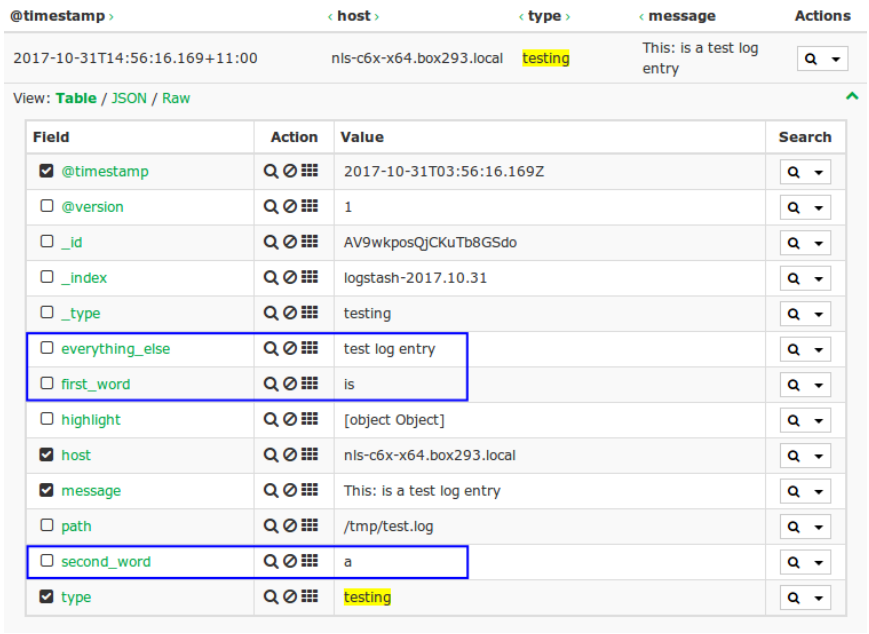


So far you can see it is relatively simple to create a filter to break apart the log data into separate fields. Earlier on it was explained that it is very important the overall pattern you are defining is accurate, otherwise the pattern will not be matched and the log data will not be broken up into fields in the Elasticsearch database.

So how do you know the pattern isn't matching and something is wrong? In your terminal session execute the following command to submit another log entry *(notice how a colon `:` was added)*:

```
echo "This: is a test log entry" >> /tmp/test.log
```

Now in Nagios Log Server on the Dashboards page perform the query `type:testing` and have a look at the new log entry.



Here you can see that the filter isn't working as expected, it broke the `message` field into the three fields of `first_word` (is), `second_word` (a) and `everything_else` (test log entry).

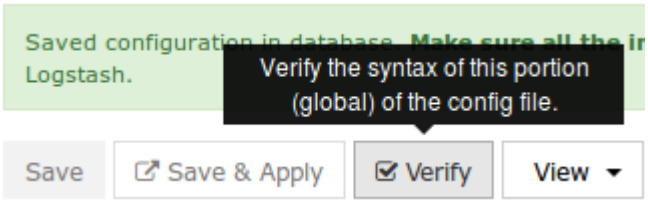
What is happening is that the colon that was part of the first word "`This:`" is causing the grok pattern to start at the "`is`" word. The grok pattern does not have a colon in it, or using a pattern syntax that matches it.

Sometimes complicated grok patterns will simply add a `tag` field with the value of `_grokparsefailure` when it does not match. When troubleshooting grok patterns, try and simplify it, remove a section and see if it works. If it does then the problem was in the section you removed.

The remainder of this documentation explains the buttons available on the **Global Config** page.

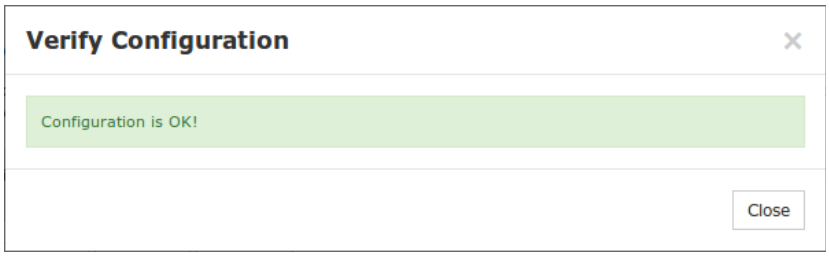
Verify

The Verify button ensures that the current saved configuration is valid. It can be useful when updating your configurations before attempting to Apply Configuration.



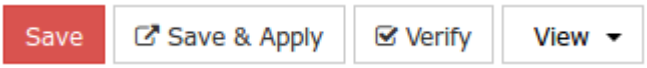
Wait while the configuration is verified.

If you do not receive a **Configuration is OK** message then you will need to fix the problem before applying the configuration.



Save vs Save & Apply

There are two separate buttons, **Save** and **Save & Apply**.

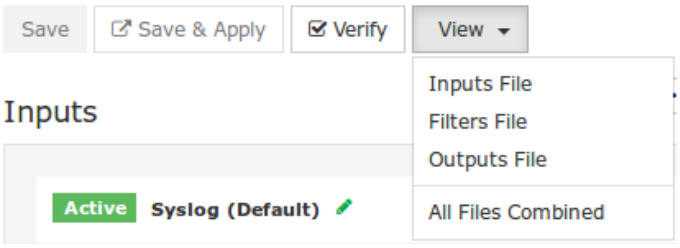


Save allows you to make changes but not make the changes become immediately active. Allows you to navigate away from the Configure screen without losing your work.

Save & Apply will save your changes and then make the changes become immediately active. This can be helpful if you changed a simple option in your config that does not need to be verified.

View

The View button allows you to view the raw Logstash configuration. When you click the button you have a choice of selecting an individual config or a combination of all the configs.



This will open a new modal window where the configuration can be viewed or copied.

External Resources

Structure Of A Logstash Configuration File:

https://www.elastic.co/guide/en/logstash/1.5/configuration-file-structure.html

Here is the source code for grok patterns:

https://github.com/elastic/logstash/blob/1.4/patterns/grok-patterns

All of the available patterns are in different files which are located here:

https://github.com/elastic/logstash/tree/1.4/patterns

Finishing Up

This completes the documentation on Configuring Filters in Nagios Log Server.

If you have additional questions or other support related questions, please visit us at our Nagios Support Forums:

https://support.nagios.com/forum

The Nagios Support Knowledgebase is also a great support resource:

https://support.nagios.com/kb