LFS242 - Cloud Native Logging with Fluentd

Lab 4 – Extending Fluentd with Plugins: Creating Pipelines with Filter Plugins

The previous lab explored the use of input and output plugins with the intent of establishing data processing pipelines with Fluentd. The "processing" part of the data processing pipeline comes in the form of `<filter>` directives and filter plugins. If the data coming from a select source needs to be modified in any way, then Fluentd's filter functionality is the way to go.

Data processing in Fluentd can come in the form of manipulating events, selectively printing or excluding them, or otherwise performing actions on them that a simple log forwarding solution would not provide. The ultimate goal of the filters within the Fluentd data processing pipeline is to extract the most value out of the data coming in from the configured sources.

This lab is designed to be completed on an Ubuntu 20.04 system. The labs install and configure software, so a cloud instance or local VM is recommended.

Objectives

- Learn how to put together `<filter>` directives for Fluentd
- Explore the grep and transformation capabilities of Filter plugins
- Learn how to use multiple `<filter>` directives together in Fluentd
- Use filters on a set of real application logs

0. Prepare the lab system

A Fluentd instance that can be freely modified is required for this lab. Create and run the following script in your VM to quickly set up Fluentd:

```
ubuntu@labsys:~$ nano fluentd-setup && cat $_

#!/bin/sh

sudo apt update
sudo apt install ruby-full ruby-dev libssl-dev libreadline-dev zlib1g-dev gcc make -y
sudo gem install bundle
sudo gem install fluentd

ubuntu@labsys:~$ chmod +x fluentd-setup

ubuntu@labsys:~$ ./fluentd-setup

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

This lab will be using Docker to run an instance of NGINX and Elasticsearch. Lab 1-B has more detailed instructions and explanations of the Docker installation process.

Install Docker with the quick installation script for Debian:

```
ubuntu@labsys:~$ wget -O - https://get.docker.com | sh

...
```

```
ubuntu@labsys:~$
```

All `docker` commands will be run with `sudo` in this lab so there is no need to set up rootless interaction.

1. Creating a Fluentd Filter directive

A `<filter>` directive is the heart of a Fluentd processing pipeline. It exists between source and `<match>` directives (though it can follow a `<match>` directive if that `<match>` directive is using a filter-like plugin). It uses the same syntax as source and `<match>` directives, with each new directive being enclosed in its own set of tags. The top tag can contain its own pattern attribute in order to capture events, just like a `<match>` directive.

To begin, prepare a very simple configuration. Since the focus of this step will be on `<filter>` directives, you will be using `fluent-cat` to create events implicitly, then view them in STDOUT.

Use the in_forward and out_stdout plugins to create a simple configuration:

```
ubuntu@labsys:~$ mkdir ~/lab4 ; cd ~/lab4

ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
@type forward
</source>

<match **>
@type stdout
</match>

ubuntu@labsys:~/lab4$
```

This will create a Fluentd instance that will listen on the default forward port, 24224, for any events passed through the Fluentd socket. This will be the perfect environment to test out new directives.

Using this configuration, run a Fluentd instance:

```
ubuntu@labsys:~/lab4$ fluentd -c ~/lab4/lab4.conf

2022-06-14 22:04:00 +0000 [info]: parsing config file is succeeded
path="/home/ubuntu/lab4/lab4.conf"
2022-06-14 22:04:00 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-14 22:04:00 +0000 [info]: Oj isn't installed, fallback to Yajl as json parser
2022-06-14 22:04:00 +0000 [warn]: define <match fluent.**> to capture fluentd logs in top level is
deprecated. Use <label @FLUENT_LOG> instead
2022-06-14 22:04:00 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
  </source>
  <match **>
    @type stdout
  </match>
</ROOT>
2022-06-14 22:04:00 +0000 [info]: starting fluentd-1.14.6 pid=201340 ruby="2.7.0"
2022-06-14 22:04:00 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c", "/home/ubuntu/lab4/lab4.conf", "--under-
supervisor"]
2022-06-14 22:04:00 +0000 [info]: adding match pattern="**" type="stdout"
2022-06-14 22:04:00 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-14 22:04:00 +0000 [info]: adding source type="forward"
```

```
2022-06-14 22:04:00 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-14 22:04:00 +0000 [info]: #0 starting fluentd worker pid=201345 ppid=201340 worker=0
2022-06-14 22:04:00 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2022-06-14 22:04:00 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-14 22:04:00.867568495 +0000 fluent.info:
{"pid":201345,"ppid":201340,"worker":0,"message":"starting fluentd worker pid=201345 ppid=201340
worker=0"}
2022-06-14 22:04:00.868321673 +0000 fluent.info: {"port":24224,"bind":"0.0.0.0","message":"listening
port port=24224 bind=\"0.0.0.0\""}
2022-06-14 22:04:00.868893353 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
```

Fluentd is now listening for inputs. Since the -d flag was not used, this terminal will continually tail Fluentd's operations. This terminal will be referred to as the Fluentd terminal from this point, and you will be asked to refer back to it to check on Fluentd's operations as commands are passed.

Open a new terminal session, which will be your working terminal, then use `fluent-cat` to send a simple message:

```
ubuntu@labsys:~$ cd lab4

ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.test

ubuntu@labsys:~/lab4$
```

The echoed JSON pair `{lfs242:hello}` will be piped into the fluent-cat tool with the event tag `lab4.test`, producing the following output in the Fluentd terminal:

```
...

2022-06-14 22:04:00.868893353 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-14 22:04:50.944347819 +0000 lab4.test: {"lfs242":"hello"}
```

Now that you confirmed the configuration is working, it is time to explore how to create a `<filter>` directive.

In your working terminal, edit the `lab4.conf` created earlier in this step:

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
@type forward
</source>

<filter **>
@type stdout
</filter>

<match>
@type stdout
</match>

ubuntu@labsys:~/lab4$
```

A new `<filter>` directive has been added to this setting, using the filter_stdout plugin. Like and directives, a plugin for a `<filter>`

directive is selected with the @type parameter. STDOUT is a simple plugin, so it does not have many other configurations to be set.

Reload the configuration by sending a SIGUSR2 to Fluentd in the working terminal:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

This will signal Fluentd to reload the configuration without terminating the session in the Fluentd terminal:

```
...

2022-06-14 22:05:48 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-14 22:05:48 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-14 22:05:48 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
```

Now that Fluentd is reloaded, answer the following questions:

- What plugins were loaded now?
- What events will the `<filter>` directive capture?
- How many times did Fluentd report that the worker is now running?

In your working terminal, run `fluent-cat` again, this time tagging the event `lab4.test.2`:

```
ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.test.2

ubuntu@labsys:~/lab4$
```

In the Fluentd terminal, check the resulting output:

```
2022-06-14 22:06:10.806555329 +0000 lab4.test.2: {"lfs242":"hello"}
2022-06-14 22:06:10.806555329 +0000 lab4.test.2: {"lfs242":"hello"}
```

Fluentd is outputting to STDOUT twice. This is because both the `<filter>` directive and the directive are outputting to STDOUT after they process an event. Using the stdout filter plugin is useful for debugging pipelines because it prints the output of an event without removing it from the processing pipeline.

There is a way to change this behavior, however, as some filter plugins can use subdirectives that call on their own plugins. In the case of the STDOUT plugin, the `<format>` subdirective and format plugins can be used.

In your working terminal, modify the `<filter>` directive in `lab4.conf` to use a format plugin:

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type forward
</source>

<filter **>
  @type stdout
  <format>
    @type msgpack
  </format>
```

```
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

In order to differentiate the STDOUT outputs, Fluentd will now use the `<format>` subdirective to change events processed by the STDOUT `<filter>` directive to use msgpack, an binary format alternative of the JSON format (which is default).

Use SIGUSR2 to reload the configuration:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

Check the Fluentd terminal now:

```
2022-06-14 22:06:40 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-14 22:06:40 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-14 22:06:40 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
```

Now send another test event:

```
ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.test.2

ubuntu@labsys:~/lab4$
```

You should now see that the final confirmation message has been drastically changed due to the use of a format plugin. The msgpack portion does not append a new line at its end, so the standard JSON event begins at the same line and character. The full msgpack event actually reads:

```
...

lfs242hello2022-06-14 22:06:55.171959722 +0000 lab4.test.2: {"lfs242":"hello"}
```

This step has introduced how to set up a basic `<filter>` directive. In the next steps, you will be guided through the use of some of the other base Fluentd filter plugins.

2. Exploring core filter Plugins

At this point, you should have a basic Fluentd `<filter>` directive sending event data to STDOUT in msgpack form. This example is useful in development, but the true power of Fluentd's processing capabilities can be explored with some of the available core plugins: filter_grep, filter_record_transform, and filter_parse.

2a. Grep an event

One way to extract the best value from incoming data is to remove irrelevant events from a data stream. While you can limit excessive logging by turning an application's log level down, Fluentd can use the grep plugin for instances where a log level cannot or should not be adjusted. Like the shell tool of the same name, it will selectively print out the desired output based on a user-provided pattern. An

application log that is written with high verbosity can retain that setting and, with Fluentd, only the events deemed important will be printed to their intended destination.

Back up your current lab4.conf into a new copy:

```
ubuntu@labsys:~/lab4$ cp lab4.conf lab4-1.conf

ubuntu@labsys:~/lab4$
```

In this step, you will simulate grepping a log for errors based on traditional log level tags. Create a configuration that reads a log and outputs it to STDOUT:

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type tail
  path /tmp/tailfile
  pos_file /tmp/tailfile.pos
    <parse>
      @type none
    </parse>
  tag lab4.step2.a
</source>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

The `<source>` directive is reading a file that is effectively unformatted, so a `<parse>` subdirective was included to ensure that no other formats would be assumed by Fluentd when reading this log. It will also record its last read position in another file, `/tmp/tailfile.pos`, and finally tag all events as `lab4.2.a`.

Reload Fluentd by sending a `SIGUSR2` signal to the process:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

In your working terminal, add some events into the tailfile with `echo` using some standard error log level tags:

```
ubuntu@labsys:~/lab4$ touch /tmp/tailfile

ubuntu@labsys:~/lab4$ echo "INFO New event" >> /tmp/tailfile

ubuntu@labsys:~/lab4$ echo "ERROR New error" >> /tmp/tailfile

ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

Fluentd should confirm that /tmp/tailfile is being tailed for content. In the Fluentd terminal, you should see activity corresponding to the new information being added to the log. If it does not, ensure that the directory specified in the configuration file is correct.

```
...

2022-06-14 22:07:46 +0000 [info]: #0 following tail of /tmp/tailfile
```

Now you know Fluentd can correctly read and parse the log. For this example, the most valuable information that this log can produce would be its errors, as the original application (manual intervention) is not storing its errors in a separate log. Rather than change the code (what the user is doing), Fluentd can be used to pipe all the valuable error information to its intended destination.

Based on the output from above, Fluentd is passing all log entries into the "message" key of the event record. This key is important, as it is typically what is used to identify which part of an event record needs to be worked on.

Create a `<filter>` directive using the grep plugin to print only the ERROR messages:

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type tail
  path /tmp/tailfile
  pos_file /tmp/tailfile.pos
    <parse>
      @type none
    </parse>
  tag lab4.2.a
</source>

<filter>
  @type grep
    <regexp>
      key message
      pattern ERROR
    </regexp>
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

This `<filter>` directive using the filter_grep plugin will use a regular expression to match any ERROR patterns in the "message" key of the event record. Remember that the "message" key was chosen because that is where Fluentd is piping all of the log entries for this event. By using the `<regexp>` subdirective for the filter_grep plugin, it will print only the keys that match the pattern (which is in this case a literal match to ERROR).

With this setting in place, send a SIGUSR2 to Fluentd to reconfigure it:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd
```

By default, a `<filter>` directive will match against all events if it is not provided a pattern. While this is useful in development, it is generally best to provide a pattern for both filter and `<match>` directives.

Now that Fluentd has been reconfigured, append the following events to the log file in your working terminal:

```
ubuntu@labsys:~/lab4$ echo "INFO New event" >> /tmp/tailfile
```

```
ubuntu@labsys:~/lab4$ echo "TRACE New event" >> /tmp/tailfile

ubuntu@labsys:~/lab4$ echo "ERROR New event" >> /tmp/tailfile
```

Check the contents of the log file; depending on how many events have been injected, your output may differ:

```
ubuntu@labsys:~/lab4$ cat /tmp/tailfile

INFO New event
ERROR New error
INFO New event
TRACE New event
ERROR New event

ubuntu@labsys:~/lab4$
```

The log level for this file is set to TRACE, which means that it would be difficult to parse the exact log events that may be most valuable; the sheer number may severely affect readability.

Check what logs Fluentd is pulling from this bloated log:

```
...

2022-06-14 22:08:21 +0000 [info]: #0 following tail of /tmp/tailfile
2022-06-14 22:08:33.360600989 +0000 lab4.2.a: {"message":"ERROR New event"}
```

Because Fluentd has been configured to use the grep plugin, only the ERROR events are being output to the user's consumption medium, in this case STDOUT. If a file were configured as a destination, then all ERROR events can be correctly routed to an easier to read ERROR log file.

You should now have a basic `<filter>` directive that will selectively print events based on your input using the filter_grep plugin.

- Using the filter_grep documentation on the Fluentd website, create a configuration that excludes all INFO events
- Find a pattern that will only print events that mention "file"

2b. Transforming Records with filter_record_transformer

Filter plugins can also transform records to introduce consistency into records from different sources, obfuscate sensitive information, or perform mathematic evaluation. This is done using the record_transformer plugin to modify an event's record, which usually contains the event's actual log or metrics data.

For simplicity, this step will continue to use `fluent-cat` to pass events into Fluentd, and send them to STDOUT.

Backup your current lab4.conf to a new copy called `lab4-2.conf` and replace its contents:

```
ubuntu@labsys:~/lab4$ cp lab4.conf lab4-2.conf

ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type forward
</source>

<filter>
  @type record_transformer
  <record>
```

```
    status filtered
  </record>
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

This configuration will use the filter_record_transformer plugin. The filter_record_transformer plugin uses the `<record>` subdirective to specify the operations that need to be performed to an event record. Each parameter under the `<record>` subdirective represents a key-value pair that will be appended to the event. The value can be a literal string or a Ruby operation, which will be explored in a later step.

Reload the Fluentd configuration with `SIGUSR2`:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

```
2022-06-14 22:09:07 +0000 [info]: Reloading new config
2022-06-14 22:09:07 +0000 [info]: Oj isn't installed, fallback to Yajl as json parser
2022-06-14 22:09:07 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
  </source>
  <filter>
    @type record_transformer
    <record>
      status filtered
    </record>
  </filter>
  <match>
    @type stdout
  </match>
</ROOT>
2022-06-14 22:09:07 +0000 [info]: shutting down input plugin type=:tail plugin_id="object:898"
2022-06-14 22:09:07 +0000 [info]: shutting down filter plugin type=:grep plugin_id="object:85c"
2022-06-14 22:09:07 +0000 [info]: shutting down output plugin type=:stdout plugin_id="object:870"

...
```

The record_transformer plugin should be loaded now, matching all event inputs.

Now pass an event to the newly reconfigured Fluentd:

```
ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.2.b
```

This should pass a simple key-value pair to Fluentd under the lab4.2.b tag. Since `fluent-cat` is being used, it will submit it to the forward plugin configured earlier.

Check the Fluentd terminal:

```
2022-06-14 22:09:43.811212367 +0000 lab4.2.b: {"lfs242":"hello","status":"filtered"}
```

As configured, the `<filter>` directive in this configuration appended the `"status":"filtered"` key-value pair to the event you just passed to Fluentd. This feature can be useful for introducing keys that destinations or users can use to perform additional operations.

Portions of a record can also be edited by calling the record itself as a value. This is done by including `${record[""]}` in the value portion of a new key.

Adjust the `<filter>` directive to add a filter pattern and another key-value pair to the `<record>` subdirective:

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
@type forward
</source>

<filter lab4**>
@type record_transformer
  <record>
    status filtered
    filter processed - ${record["lfs242"]}
  </record>
</filter>

<match>
@type stdout
</match>

ubuntu@labsys:~/lab4$
```

Now that this configuration is becoming more complex, a pattern was added to the `<filter>` directive to ensure that only the step-related commands (and not the Fluentd info commands) are passed through the filter. Also, this new configuration will add a new key ("filter") with the value ("processed - "), followed by the value of tag within the `[]`.

Reconfigure Fluentd by sending a SIGUSR2:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd
```

```
2022-06-14 22:10:14 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-14 22:10:14 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-14 22:10:14 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
```

Since a filter pattern was introduced to the `<filter>` directive, you should see that the fluent.info event was not affected by the filter; otherwise it would have appended `status":"filtered` to its event record as it did the first time this configuration was run.

Now to test an event that will actually be caught by the filter - recall that it was configured to capture events tagged `lab4**`, meaning that in order to be processed by the filter, the event must include `lab4` in its tag.

In your working terminal, run a test event using the lab4.2.b tag:

```
ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.2.b
```

```
2022-06-14 22:10:27.612681415 +0000 lab4.2.b:
{"lfs242":"hello","status":"filtered","filter":"processed - hello"}
```

The event was processed by the `<filter>` directive, as indicated by the "status: filtered" key-value. Also, note that the "filter" key has been added, with added the "processed - value". Where did `value` come from? It is actually what was passed to Fluentd! By using the `{record[""]}` template in the value, a user can insert data from the event itself and reuse it in a more valuable way.

The event itself is starting to show redundant information, as seen with the original "key : value" pair being referenced. filter_record_transformer can be configured to remove tags from records, which is useful since the `<record>` subdirective only edits or adds keys to a record.

Add another parameter, `remove_keys`, to the configuration:

```
ubuntu@labsys:~$ nano lab4.conf && cat $_

<source>
  @type forward
</source>

<filter lab4**>
  @type record_transformer
    <record>
      status filtered
      filter processed - ${record["lfs242"]}
    </record>
  remove_keys lfs242
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~$
```

`remove_keys` allows a user to exclude record content based on the keys. Multiple keys can be specified by using commas.

Reconfigure Fluentd with a SIGUSR2, and submit another example event using `fluent-cat`:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello"}' | fluent-cat lab4.2.b
```

In the Fluentd terminal, check the event.

```
2022-06-14 22:11:05.700281867 +0000 lab4.2.b: {"status":"filtered","filter":"processed - hello"}
```

The original user key-value, "lfs242":"hello", was removed from the event. The remove_keys parameter is useful for remove portions of an event that may not be appropriate to show.

There is another way to obfuscate data by directly manipulating the text. Fluentd and the filter_record_transformer plugin can leverage Ruby expressions to perform more complex transformation tasks. This can be toggled using the `enable_ruby` parameter for the filter_record_transformer plugin.

In your working terminal, send an event that passes potentially sensitive data (like a password) in its record:

```
ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello","password":"secret"}' | fluent-cat lab4.2.b
```

In the Fluentd terminal, you can see that the password is plainly visible.

```
2022-06-14 22:11:55.599118632 +0000 lab4.2.b:
{"password":"secret","status":"filtered","filter":"processed - hello"}
```

This event flow is insecure as any keys that include a "password" or some other credential should be obfuscated. Ruby has a regular expression functionality that can be used to match specific strings inside a record and select them for processing.

In your working terminal, add a configuration that replaces all the values of a "password" key with stars.

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
@type forward
</source>

<filter lab4**>
  @type record_transformer
  enable_ruby
    <record>
      status filtered
      filter processed - ${record["lfs242"]}
      password ${record["password"].gsub(/./,"*")}
    </record>
  remove_keys lfs242
</filter>

<match>
@type stdout
</match>

ubuntu@labsys:~/lab4$
```

Here, the `gsub` functionality in Ruby will be used to match everything (signaled by the `.`) inside the "password" key's value and replace it with an asterisk.

Reconfigure Fluentd with a SIGUSR2 in your working terminal, and submit an insecure event:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$ echo '{"lfs242":"hello","password":"secret"}' | fluent-cat lab4.2.b
```

Check the resulting event in the Fluentd terminal:

```
2022-06-14 22:12:27.792066473 +0000 lab4.2.b:
{"password":"******","status":"filtered","filter":"processed - hello"}
```

Since the "password" key already existed inside the event record, the filter_record_transformer plugin edited the contents of its value rather than appending it as a duplicate key. This ensures that event records can be edited without introducing additional data.

Excellent! This may not be the most secure method of obfuscation, as it is still obvious that it is a six character password; it is much better than the previous solution, though!

3. Using multiple Filters

Multiple filters can be utilized in a Fluentd configuration. Just like `<match>` directives, they are executed in the order that they are listed

inside the configuration file. Unlike `<match>` directives though, events caught by filters remain inside the processing pipeline.

For this step, you will use multiple filters to grep and edit events in a catch-all log (the same one from step 2), using the same tailfile.

Backup your config file and replace its contents with a new configuration file that follows tailfile and outputs to STDOUT:

```
ubuntu@labsys:~/lab4$ cp lab4.conf lab4-2-b.conf

ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type tail
  path /tmp/tailfile
  pos_file /tmp/tailfile.pos
    <parse>
      @type none
    </parse>
  tag lab4.3
</source>

<filter lab4**>
  @type grep
    <regexp>
      key message
      pattern login
    </regexp>
</filter>

<filter lab4**>
  @type record_transformer
  enable_ruby
    <record>
      status "obfuscated due to sensitive info"
      report ${record["message"].gsub(/password:.*/,"password:removed")}
    </record>
  remove_keys message
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

This example will utilize both the filter_grep and filter_record_transformer plugins to remove sensitive information from login errors. First, it will search for any "login" patterns - which in this case display the passwords students are using - and remove them from the stream. It will also append a status and remove it from the event.

Before proceeding, read the `<filter>` directive in this configuration to answer the following questions:

- Will the original log message be preserved in this configuration?
  - If so, under what key?
- Will there be a password key appended to the record?

Now reload Fluentd:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab4$
```

Fluentd is now ready to obfuscate sensitive information that it might receive.

In your working terminal, run an event containing sensitive information through to /tmp/tailfile:

```
ubuntu@labsys:~/lab4$ echo "ERROR login failed {"user":"student","password":"secret"}" >>
/tmp/tailfile
```

```
...

2022-06-14 22:13:00 +0000 [info]: #0 following tail of /tmp/tailfile
2022-06-14 22:13:05 +0000 [info]: #0 disable filter chain optimization because
[Fluent::Plugin::RecordTransformerFilter] uses `#filter_stream` method.
2022-06-14 22:13:05.243113121 +0000 lab4.3: {"status":"obfuscated due to sensitive
info","report":"ERROR login failed {user:student,password:removed"}
```

> The disable filter chain optimization warning is due to Fluentd being unable to optimize the filter calls to increase performance by default. Plugins that use filter_stream methods will cause Fluentd to disable it and report this error. According to the documents, that error is safe to ignore.

The event tagged with lab4.3 has been obfuscated due to sensitive information being present! Try to run other events:

```
ubuntu@labsys:~/lab4$ echo "INFO login succeeded {"user":"student","password":"secret"}" >>
/tmp/tailfile

ubuntu@labsys:~/lab4$ echo "INFO A user has successfully logged in" >> /tmp/tailfile
```

```
2022-06-14 22:13:34.401125280 +0000 lab4.3: {"status":"obfuscated due to sensitive
info","report":"INFO login succeeded {user:student,password:removed"}
```

There should be no response for the second event. This is because of the patterns in the filter match; all events with a matching tag are being pulled through the filter, and since the grep plugin is only looking for events that possess `login` in them, all others will be excluded. This can be addressed with its own set of labels and separating the pipeline, which will be explored in a later lab.

4. Creating an NGINX to Elasticsearch logging pipeline

By this point, you have explored using `<filter>` directives to manipulate simulated input streams. Now it's time to do so with an actual application.

The sample use case for this step will be the following: an NGINX deployment needs to have a pared-down log for the access events with errors. The error log provides enough information, but it needs to be in a format that is similar to the access log. Extract the errors from the access log and send them to their destination, Elasticsearch.

4a. Prepare an NGINX container

Docker will simplify the process of running NGINX, providing a pre-baked environment for NGINX to be deployed. In order to access the NGINX logs while it's running under Docker, the logs directory should be mounted to the host machine.

In your working terminal, create an nginx/log directory, then start an NGINX container:

```
ubuntu@labsys:~/lab4$ mkdir /tmp/lab4/

ubuntu@labsys:~/lab4$ mkdir /tmp/lab4/nginx

ubuntu@labsys:~/lab4$ sudo docker run --name nginx -d -v /tmp/lab4/nginx:/var/log/nginx nginx

...

a3fc86044d1e8d3f6d8bcdeaa5a0fe370fb7b81b8ac9cb764eef446a3174580f

ubuntu@labsys:~/lab4$
```

This container is named nginx for simplicity, and will run as a background daemon, so your terminal will remain free.

Since you will need to interact with it to produce events, retrieve the IP address. This IP will be used for `curl` in the coming steps:

```
ubuntu@labsys:~/lab4$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}
{{end}}' nginx

172.17.0.2

ubuntu@labsys:~/lab4$
```

Now send a `curl` request to the NGINX container's IP to see if it responds:

```
ubuntu@labsys:~/lab4$ curl 172.17.0.2

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...

ubuntu@labsys:~/lab4$
```

NGINX is now ready and prepared to receive events for Fluentd to interact with.

4b. Run Elasticsearch

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Because it uses JSON, it is a commonly used as a destination for Fluentd events; the log processing stack called "EFK" is comprised of Elasticsearch, Fluentd and Kibana (a GUI frontend for Elasticsearch data).

In your working terminal, run an Elasticsearch container:

```
ubuntu@labsys:~$ sudo docker run -d --name elasticsearch -p 9200:9200 -e "discovery.type=single-
node" docker.elastic.co/elasticsearch/elasticsearch:8.2.2

...

dabbc4214a0b5c59fcc7c93bd0a45e27e31cc4657d502f99e6bf27b5c32bb060

ubuntu@labsys:~$
```

This will open an Elasticsearch instance at port 9200. Like the NGINX container above, you will need to retrieve its IP address to interact

with it.

```
ubuntu@labsys:~/lab4$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}
{{end}}' elasticsearch

172.17.0.3

ubuntu@labsys:~/lab4$
```

The Elasticsearch 8 container requires a password and certificate in order to successfully authenticate incoming connections.

First, you need to retrieve a password for the `elastic` user by generating it using the `elasticsearch-reset-password` utility packaged with the Elasticsearch 8 container. You will need to confirm the autogeneration, so the reset password operation must be done in an interactive terminal:

```
ubuntu@labsys:~/lab4$ sudo docker exec -it elasticsearch /usr/share/elasticsearch/bin/elasticsearch-
reset-password -u elastic

WARNING: Owner of file [/usr/share/elasticsearch/config/users] used to be [root], but now is
[elasticsearch]
WARNING: Owner of file [/usr/share/elasticsearch/config/users_roles] used to be [root], but now is
[elasticsearch]
This tool will reset the password of the [elastic] user to an autogenerated value.
The password will be printed in the console.
Please confirm that you would like to continue [y/N]y


Password for the [elastic] user successfully reset.
New value: 61u+EV8Eev+t1WBV798u

ubuntu@labsys:~/lab4$
```

Save the New value of the password to an environment variable to make it easy to reference:

```
ubuntu@labsys:~/lab4$ export ESPW=61u+EV8Eev+t1WBV798u     # Use the value presented after you
changed the password.

ubuntu@labsys:~/lab4$
```

Retrieve the CA certificate from the container, which will allow any clients to successfully authenticate with Elasticsearch:

```
ubuntu@labsys:~/lab4$ sudo docker cp elasticsearch:/usr/share/elasticsearch/config/certs/http_ca.crt
.

ubuntu@labsys:~/lab4$
```

To confirm that Elasticsearch is functional, send a request to Elasticsearch using `curl` in your working terminal:

```
ubuntu@labsys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
https://172.17.0.3:9200

{
  "name" : "dabbc4214a0b",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "XIhB4drQT_2WYxn4_crSTw",
  "version" : {
```

```
    "number" : "8.2.2",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "9876968ef3c745186b94fdabd4483e01499224ef",
    "build_date" : "2022-05-25T15:47:06.259735307Z",
    "build_snapshot" : false,
    "lucene_version" : "9.1.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}

ubuntu@labsys:~/lab4$
```

Elasticsearch is up and running, so now it's time to prepare Fluentd to link Elasticsearch and NGINX.

4c. Configure Fluentd for the NGINX to Elasticsearch pipeline

Now that NGINX and Elasticsearch are up and running, it's time to prepare Fluentd to read it as a source. To simplify the configuration for this processing pipeline, you will want to parse both the access and error logs. There is a regular NGINX parser plugin, but the error log parsing is more complicated.

You will need the following Fluentd plugins:

- fluent-plugin-nginx-error-multiline which pre-packages the multi-line configuration for the NGINX error log
- fluent-plugin-elasticsearch to allow Fluentd to communicate with Elasticsearch

Install the fluent-plugin-nginx-error-multiline and fluent-plugin-elasticsearch plugins using `fluent-gem`:

```
ubuntu@labsys:~/lab4$ sudo fluent-gem install -N \
'fluent-plugin-nginx-error-multiline:~>0.2.0' \
'elasticsearch:~>8.2.2' \
'fluent-plugin-elasticsearch:~>5.2.2'

Fetching fluent-plugin-nginx-error-multiline-0.2.0.gem
Successfully installed fluent-plugin-nginx-error-multiline-0.2.0
Fetching faraday-multipart-1.0.4.gem
Fetching faraday-em_synchrony-1.0.0.gem
Fetching faraday-httpclient-1.0.1.gem
Fetching multipart-post-2.2.3.gem
Fetching faraday-em_http-1.0.0.gem
Fetching multi_json-1.15.0.gem
Fetching faraday-excon-1.1.0.gem
Fetching faraday-rack-1.0.0.gem
Fetching faraday-net_http_persistent-1.2.0.gem
Fetching faraday-net_http-1.0.1.gem
Fetching faraday-patron-1.0.0.gem
Fetching faraday-retry-1.0.3.gem
Fetching ruby2_keywords-0.0.5.gem
Fetching elasticsearch-api-8.2.2.gem
Fetching elastic-transport-8.0.1.gem
Fetching elasticsearch-8.2.2.gem
Fetching faraday-1.10.0.gem
Successfully installed multi_json-1.15.0
Successfully installed faraday-em_http-1.0.0
Successfully installed faraday-em_synchrony-1.0.0
Successfully installed faraday-excon-1.1.0
Successfully installed faraday-httpclient-1.0.1
Successfully installed multipart-post-2.2.3
```

```
Successfully installed faraday-multipart-1.0.4
Successfully installed faraday-net_http-1.0.1
Successfully installed faraday-net_http_persistent-1.2.0
Successfully installed faraday-patron-1.0.0
Successfully installed faraday-rack-1.0.0
Successfully installed faraday-retry-1.0.3
Successfully installed ruby2_keywords-0.0.5
Successfully installed faraday-1.10.0
Successfully installed elastic-transport-8.0.1
Successfully installed elasticsearch-api-8.2.2
Successfully installed elasticsearch-8.2.2
Fetching fluent-plugin-elasticsearch-5.2.2.gem
Fetching excon-0.92.3.gem
Successfully installed excon-0.92.3
Successfully installed fluent-plugin-elasticsearch-5.2.2
20 gems installed

ubuntu@labsys:~/lab4$
```

The plugins' optional Ri documentations were omitted using the `-N` flag.

Now, create a new configuration file that uses both NGINX logs as an input source:

```
ubuntu@labsys:~/lab4$ cp lab4.conf lab4-3.conf

ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type tail
  <parse>
    @type nginx
  </parse>
  path /tmp/lab4/nginx/access.log
  pos_file /tmp/lab4/nginx/access.pos
  tag nginx.access
</source>

<source>
  @type tail
  path /tmp/lab4/nginx/error.log
  pos_file /tmp/lab4/nginx/error.pos
  tag nginx.error
  <parse>
    @type nginx_error_multiline
  </parse>
</source>

<match nginx.**>
  @type elasticsearch
  host 172.17.0.3
  port 9200
  user elastic
  password "#{ENV['ESPW']}"
  scheme https
  ssl_verify false
  ca_file /home/ubuntu/lab4/http_ca.crt
</match>

ubuntu@labsys:~/lab4$
```

The access log will pipe its events into the nginx.access tag, while the error log will use the nginx.error tag for its events. Both logs will submit their outputs to Elasticsearch.

You need terminate the previous Fluentd instance if it is still running with `CTRL C`, then reload with the new configuration in order for the Elasticsearch plugin to be loaded successfully:

```
^C

...


2022-06-14 22:18:38 +0000 [info]: Worker 0 finished with status 0
```

Set the ESPW environment variable then restart the Fluentd instance:

```
ubuntu@labsys:~/lab4$ export ESPW=61u+EV8Eev+t1WBV798u

ubuntu@labsys:~/lab4$ fluentd -c ~/lab4/lab4.conf

...

2022-06-14 22:18:57 +0000 [info]: adding match pattern="nginx.**" type="elasticsearch"
2022-06-14 22:18:58 +0000 [info]: adding source type="tail"
2022-06-14 22:18:58 +0000 [info]: adding source type="tail"
2022-06-14 22:18:58 +0000 [info]: #0 starting fluentd worker pid=202417 ppid=202412 worker=0
2022-06-14 22:18:58 +0000 [info]: #0 following tail of /tmp/lab4/nginx/error.log
2022-06-14 22:18:58 +0000 [info]: #0 following tail of /tmp/lab4/nginx/access.log
2022-06-14 22:18:58 +0000 [info]: #0 fluentd worker is now running worker=0
```

Notice how the `fluent_nginx_error_multiline` plugin loaded a pre-baked format. Confirm that the logs are being tailed by Fluentd by looking for the "following tail of..." reports in the above info dump. Remember that Fluentd will not report an error if it cannot find the file: it will simply not tail the log.

It's time to test the configuration.

In your working terminal, send a curl request to NGINX to a valid and an invalid page (one that returns 404):

```
ubuntu@labsys:~$ curl http://172.17.0.2

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>

...

ubuntu@labsys:~$
```

```
ubuntu@labsys:~$ curl http://172.17.0.2/404

<html>
<head><title>404 Not Found</title></head>
...

ubuntu@labsys:~$
```

Now check to see if Elasticsearch received any events by sending a `curl` to query its indices:

```
ubuntu@labsys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
'https://172.17.0.3:9200/_cat/indices?v'

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

ubuntu@labsys:~/lab4$
```

Nothing. The Elasticsearch plugin is a buffered plugin, so Fluentd's buffers need to be flushed in order for it to deliver events to Elasticsearch.

Send a `SIGUSR1` to Fluentd to force a flush, then `curl` Elasticsearch again:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR1 fluentd

ubuntu@labsys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
'https://172.17.0.3:9200/_cat/indices?v'

health status index   uuid                   pri rep docs.count docs.deleted store.size
pri.store.size
yellow open   fluentd tt3D9llmT2-XepVD7OxpTQ   1   1          3            0       13kb
13kb

ubuntu@labsys:~/lab4$
```

The result confirms Fluentd is writing events to the `fluentd` index inside Elasticsearch.

In your working terminal, query that index with `curl`:

```
ubuntu@labsys:~/lab4$ curl -s -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
https://172.17.0.3:9200/fluentd/_search?pretty

{
  "took" : 6,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "fluentd",
        "_id" : "5OxNZIEBDyHFTUMiNlAN",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/",
```

```
          "code" : "200",
          "size" : "615",
          "referer" : "-",
          "agent" : "curl/7.68.0",
          "http_x_forwarded_for" : "-"
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "5exNZIEBDyHFTUMiNlAT",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/404",
          "code" : "404",
          "size" : "153",
          "referer" : "-",
          "agent" : "curl/7.68.0",
          "http_x_forwarded_for" : "-"
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "5uxNZIEBDyHFTUMiOVA0",
        "_score" : 1.0,
        "_source" : {
          "log_level" : "error",
          "pid" : "31",
          "tid" : "31",
          "message" : "*3 open() \"/usr/share/nginx/html/404\" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host:
\"172.17.0.2\""
        }
      }
    ]
  }
}

ubuntu@labsys:~/lab4$
```

Excellent! You now have NGINX events being sent to Elasticsearch through Fluentd. Using the output of your query to Elasticsearch, answer the following:

- How many events were recorded?
- Use `cat` to read one of the NGINX log files, and compare how that log is presented in Elasticsearch

4d. Adding Filters to the NGINX-Elasticsearch pipeline

It's time to configure the first filters. First, we'll want to ensure that the access log only transfers the actual, non-error events. This will improve readability by preventing duplicate entries from appearing on the user's buffer.

Looking at the error from the access log in your previous `curl`, you can see that the actual response code is stored with the `code` key in the access log JSON.

```
...
      {
        "_index" : "fluentd",
```

```
        "_id" : "5exNZIEBDyHFTUMiNlAT",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/404",
          "code" : "404",
          "size" : "153",
          "referer" : "-",
          "agent" : "curl/7.68.0",
          "http_x_forwarded_for" : "-"
        }
      },

...
```

Use filter grep to print all events from the access log except for any 404 patterns. These will be eventually piped to another log, so the user needs to see fewer details to effectively work with the access log.

```
ubuntu@labsys:~/lab4$ nano lab4.conf && cat $_

<source>
  @type tail
  <parse>
    @type nginx
  </parse>
  path /tmp/lab4/nginx/access.log
  pos_file /tmp/lab4/nginx/access.pos
  tag nginx.access
</source>

<source>
  @type tail
  path /tmp/lab4/nginx/error.log
  pos_file /tmp/lab4/nginx/error.pos
  tag nginx.error
  <parse>
    @type nginx_error_multiline
  </parse>
</source>

### Scrub 404 errors from access log to prevent duplicate outputs to Elasticsearch
<filter nginx.access>
  @type grep
  <exclude>
    key code
    pattern 404
  </exclude>
</filter>

<match nginx.**>
  @type elasticsearch
  host 172.17.0.3
  port 9200
  user elastic
  password "#{ENV['ESPW']}"
  scheme https
  ssl_verify false
  ca_file /home/ubuntu/lab4/http_ca.crt
```

```
</match>

ubuntu@labsys:~/lab4$
```

In this case, the `<exclude>` subdirective is used to prevent events that match the pattern issued in this subdirective from being printed out. Based on the output from earlier, you will be using the code key and the pattern 404 to match the events.

Send a `SIGUSR2` to Fluentd to reconfigure it:

```
ubuntu@labsys:~/lab4$ pkill -SIGUSR2 fluentd
```

Now that Fluentd is reconfigured, send a pair of `curl` commands, then flush the buffer to see if the filter is effective:

```
ubuntu@labsys:~$ curl http://172.17.0.2/404

...

ubuntu@labsys:~$ curl http://172.17.0.2

...

ubuntu@labsys:~$ pkill -SIGUSR1 fluentd
```

In your working terminal, `curl` Elasticsearch to see if the events were written:

```
ubuntu@labsys:~/lab4$ curl -s -u elastic:$ESPW  --cacert ~/ubuntu/lab4/http_ca.crt -k
https://172.17.0.3:9200/fluentd/_search?pretty

{
  "took" : 235,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "fluentd",
        "_id" : "5OxNZIEBDyHFTUMiNlAN",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/",
          "code" : "200",
          "size" : "615",
          "referer" : "-",
          "agent" : "curl/7.68.0",
```

```
          "http_x_forwarded_for" : "-"
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "5exNZIEBDyHFTUMiNlAT",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/404",
          "code" : "404",
          "size" : "153",
          "referer" : "-",
          "agent" : "curl/7.68.0",
          "http_x_forwarded_for" : "-"
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "5uxNZIEBDyHFTUMiOVA0",
        "_score" : 1.0,
        "_source" : {
          "log_level" : "error",
          "pid" : "31",
          "tid" : "31",
          "message" : "*3 open() \"/usr/share/nginx/html/404\" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host:
\"172.17.0.2\""
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "5-xPZIEBDyHFTUMiSVAL",
        "_score" : 1.0,
        "_source" : {
          "log_level" : "error",
          "pid" : "31",
          "tid" : "31",
          "message" : "*4 open() \"/usr/share/nginx/html/404\" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host:
\"172.17.0.2\""
        }
      },
      {
        "_index" : "fluentd",
        "_id" : "6OxPZIEBDyHFTUMiTFDV",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/",
          "code" : "200",
          "size" : "615",
          "referer" : "-",
          "agent" : "curl/7.68.0",
          "http_x_forwarded_for" : "-"
        }
```

```
      }
    ]
  }
}

ubuntu@labsys:~/lab4$
```

If your filter was effective, there should only be five events total (seen with the value of `hits` in the Elasticsearch output). Recall that Fluentd is now configured to discard any GET requests to invalid pages, so only two events should have been added.

Excellent. You have now successfully used filters to improve the readability and value of data incoming from an NGINX instance to Elasticsearch.

Cleanup

Use CTRL C to terminate any running instances of Fluentd:

```
^C

2022-06-14 22:22:49 +0000 [info]: Received graceful stop
2022-06-14 22:22:50 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-14 22:22:50 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-14 22:22:50 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:910"
2022-06-14 22:22:50 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:924"
2022-06-14 22:22:50 +0000 [warn]: #0 got incomplete line at shutdown from /tmp/lab4/nginx/error.log:
""
2022-06-14 22:22:50 +0000 [info]: #0 shutting down output plugin type=:elasticsearch
plugin_id="object:8e8"
2022-06-14 22:22:50 +0000 [info]: #0 shutting down filter plugin type=:grep plugin_id="object:8d4"
2022-06-14 22:22:50 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab4$ cp lab4.conf lab4-nginx-es.conf

ubuntu@labsys:~/lab4$
```

Now tear down any running Docker containers:

```
ubuntu@labsys:~/lab4$ sudo docker container rm $(sudo docker container stop nginx elasticsearch)

nginx
elasticsearch

ubuntu@labsys:~/lab4$
```

Congratulations, you have completed the lab!