LFS242 - Cloud Native Logging with Fluentd

Lab 5 – Working with Parser and Formatter plugins: processing Apache2 log data

In order for Fluentd to work with multiple sources, it must have the flexibility to read and write different formats. These differences could be as simple as the order of information in an application's log or as complex as being in an entirely different format altogether (such as JSON or CSV). Parser and formatter plugins allow other Fluentd plugins to either ingest (parse) or output (format) events according to the needs of the destination.

Parser and formatter plugins are not called directly by the directive they are used under; rather, they are called by certain compatible plugins. As of version 1.0 (or 0.14 which preceded it), they are called under their own subdirectives, `<parse>` and `<format>` respectively.

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 parse Apache2 logs with input and filter parser plugins
- Reformat Apache2 log events using output format plugins

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 use Docker to run an instance of Apache Webserver. Lab 1-B has more detailed information on 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.

This lab also uses a local installation of Apache2 webserver in conjunction with Docker. Using your virtual machine's package manager, install apache2 (or httpd, depending on distribution):

```
ubuntu@labsys:~$ sudo apt install apache2 -y

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-
ldap libjansson4 liblua5.2-0 ssl-cert

...

Setting up apache2 (2.4.41-4ubuntu3.11) ...

...

ubuntu@labsys:~$
```

You should now be ready to perform the steps described in the lab.

1. Exploring Parser Plugins

When Fluentd encounters an unfamiliar format, it uses the default parser formats. This might not be ideal, depending on the requirements for users on the destination end. To cope with this, Fluentd has the can use a set of plugins known as parser plugins to provide the frameworks needed for Fluentd to correctly sift out important data from sources.

Parser plugins are called with the `<parse>` subdirective, which use their own @type parameter to select a plugin. This lab will guide you through some simple parser use cases.

In this example, you will parse Apache webserver logs. These logs provide information about requests made to the webserver, showing the host, user and what method was invoked to make the request. Fluentd cannot parse the Apache log format on its own, but since they are commonly used together, Apache-specific plugins with format expressions provided are readily available.

1a. Using manual regular expressions

The most basic parsing strategy is to use regular expressions. This is the most flexible of the parser plugins, but it can also be the most challenging to set up depending on the user's familiarity with the source's event log format.

Regular expressions can be used to determine formats by calling the `regexp` plugin, so create a `<parse>` subdirective under the source:

```
ubuntu@labsys:~$ mkdir ~/lab5 ; cd $_

ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_

<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type regexp
```

```
    expression /^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +(?
<path>[^ ]*) +\S*)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)" "(?<agent>[^\"]*)")?$/
    time_format %d/%b/%Y:%H:%M:%S %z
  </parse>
</source>

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

ubuntu@labsys:~/lab5$
```

The regular expression used was provided by the Fluentd documentation. When setting formats manually with regular expressions, the first source to go to is the monitored application's official documentation.

Launch a Fluentd instance using `lab5.conf`. Since Fluentd is not being run as a daemon, this terminal will be locked. This window will be referred to as "the Fluentd terminal" for the remainder of this lab.

```
ubuntu@labsys:~/lab5$ fluentd -c lab5.conf

2022-06-20 16:09:57 +0000 [info]: parsing config file is succeeded path="lab5.conf"
2022-06-20 16:09:57 +0000 [info]: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 16:09:57 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 16:09:57 +0000 [info]: Oj isn't installed, fallback to Yajl as json parser
2022-06-20 16:09:57 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type tail
    path "/var/log/apache2/access.log"
    pos_file "/tmp/lab5/apache_access.pos"
    tag "apache2.access"
    <parse>
      @type "regexp"
      expression /^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +(?
<path>[^ ]*) +\S*)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)" "(?<agent>[^\"]*)")?$/
      time_format "%d/%b/%Y:%H:%M:%S %z"
      unmatched_lines
    </parse>
  </source>
  <match apache**>
    @type stdout
  </match>
</ROOT>
2022-06-20 16:09:57 +0000 [info]: starting fluentd-1.14.6 pid=45787 ruby="2.7.0"
2022-06-20 16:09:57 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c", "lab5.conf", "--under-supervisor"]
2022-06-20 16:09:57 +0000 [info]: adding match pattern="apache**" type="stdout"
2022-06-20 16:09:57 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 16:09:57 +0000 [info]: adding source type="tail"
2022-06-20 16:09:57 +0000 [info]: #0 starting fluentd worker pid=45792 ppid=45787 worker=0
2022-06-20 16:09:57 +0000 [info]: #0 following tail of /var/log/apache2/access.log
2022-06-20 16:09:57 +0000 [info]: #0 fluentd worker is now running worker=0
```

Launch another terminal session; this will be the working terminal.

Now that Fluentd is configured and running, run a `curl` against this local session of Apache:

```
ubuntu@labsys:~$ cd ~/lab5
```

```
ubuntu@labsys:~/lab5$ curl -I http://localhost

HTTP/1.1 200 OK
Date: Mon, 20 Jun 2022 16:11:00 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Mon, 20 Jun 2022 16:06:36 GMT
ETag: "2aa6-5e1e349debac8"
Accept-Ranges: bytes
Content-Length: 10918
Vary: Accept-Encoding
Content-Type: text/html

ubuntu@labsys:~/lab5$ curl -I http://localhost

...

ubuntu@labsys:~/lab5$
```

In the Fluentd terminal, you can see the results of the curl:

```
2022-06-20 16:10:34.000000000 +0000 apache2.access: {"host":"127.0.0.1","user":"-
","method":"HEAD","path":"/","code":"200","size":"255","referer":"-","agent":"curl/7.68.0"}
2022-06-20 16:11:00.000000000 +0000 apache2.access: {"host":"127.0.0.1","user":"-
","method":"HEAD","path":"/","code":"200","size":"255","referer":"-","agent":"curl/7.68.0"}
```

Now check the same event in the event log itself using `tail`:

```
ubuntu@labsys:~/lab5$ tail /var/log/apache2/access.log

127.0.0.1 - - [20/Jun/2022:16:10:34 +0000] "HEAD / HTTP/1.1" 200 255 "-" "curl/7.68.0"
127.0.0.1 - - [20/Jun/2022:16:11:00 +0000] "HEAD / HTTP/1.1" 200 255 "-" "curl/7.68.0"

ubuntu@labsys:~/lab5$
```

Comparing the outputs between the two logs (Fluentd event and the log itself), answer the following questions:

- Who made the request?
- From where was the request made?
- What tag was assigned to the event? Was this something that was set by the user?
- What is the most significant difference between the two logs?
- Remove the `<parse>` subdirective from the `<source>` directive; how did it turn out in Fluentd?

The key take-away is that while Fluentd already provides a measure of standardization when it comes to publishing events, using parser plugins enables additional processing for event data by separating incoming data into new key-value pairs, which other directives can process.

1b. Application Specific Parser Plugins

As mentioned earlier, some applications are so common used in conjunction with Fluentd that they have built-in plugins already installed. These plugins can simplify the process of configuring a file, improve reusability, and provide a more consistent approach for Fluentd to read events correctly.

Replace the `@regexp` plugin with `@apache2` in the `<source>` directive's `<parse>` subdirective:

```
ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_
```

```
<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type apache2
  </parse>
</source>

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

ubuntu@labsys:~/lab5$
```

The parser_apache2 plugin utilizes the same exact regular expression to parse the access log. The advantage of this plugin is that it simplifies the configuration, and removes the need for a user to fully understand the intricacies of the formats to be effective providers.

Use a SIGUSR2 sent by the `pkill` command on the Fluentd instance to reconfigure it:

```
ubuntu@labsys:~/lab5$ pkill -SIGUSR2 fluentd
```

Once Fluentd is reloaded, rerun the `curl` command against the localhost:

```
ubuntu@labsys:~/lab5$ curl -I http://localhost

...

ubuntu@labsys:~/lab5$
```

Then, check Fluentd:

```
2022-06-20 16:13:56.000000000 +0000 apache2.access:
{"host":"127.0.0.1","user":null,"method":"HEAD","path":"/","code":200,"size":255,"referer":null,"age
nt":"curl/7.68.0"}
```

The `parser_apache2` plugin uses a similar regular expression as the one used previously, so the resulting output log to STDOUT was formatted similarly (though some of the values are different).

Many plugins were written for specifically to simplify the Fluentd configuration process. Filter plugins incorporate a regular expression like parsers in order to sequester or remove certain events.

1c. Using parser plugins with `<filter>` directives

In some cases, Fluentd may not have the ability to directly access the logs for an application. This is the case in the Docker version of httpd, where the container images are not set to write logs to the container's filesystem, so it would not be possible to simply mount the log directory and read the logs from there.

This step will utilize Docker's Fluentd logging engine to extract the logs, then parse them so they are sent to their destination (in this case, STDOUT) in the same format as a local installation of Apache2 would.

First add a `<source>` directive using in_forward:

```
ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_
```

```
<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type apache
  </parse>
</source>

<source>
  @type forward
</source>

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

ubuntu@labsys:~/lab5$
```

This allows Fluentd to capture events on Port 24224, which is the default for Docker's Fluentd logging engine (and both the `in_forward` and `out_forward` plugins as well). The Fluentd logging engine sends Docker log traffic as Fluentd events in the same way that `fluent-cat` or another Fluentd instance using the `out_forward` plugin would.

Reconfigure Fluentd with a SIGUSR2:

```
ubuntu@labsys:~/lab5$ pkill -SIGUSR2 fluentd
```

Fluentd is now listening for other Fluentd event traffic on port 24224.

Run the httpd (formal name for Apache2) Docker image:

```
ubuntu@labsys:~/lab5$ sudo docker run -d -p 8080 \
-v /tmp/lab5/pages:/usr/local/apache2/htdocs/ \
--log-driver fluentd \
--log-opt tag={{.Name}} \
--name apache-cont httpd:2.4

...

c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7

ubuntu@labsys:~/lab5$
```

- `-v /tmp/lab5/pages:/usr/local/apache2/htdocs/` will mount the local directory /lab5/pages to the container's htdocs
- `--log-driver fluentd` tells Docker to forward container logs to a Fluentd listener on port 24224
- `--log-opt tag={{.Name}}` sets the tag of Docker events to use the name of the containers
- `--name apache-cont` sets a user-provided name as the tag for the container's events

Since the Fluentd logging engine is in use, Docker should have forwarded events directly to Fluentd over port 24224.

Check the Fluentd terminal:

```
2022-06-20 16:16:44.000000000 +0000 apache-cont:
{"container_id":"c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7","container_name":
"/apache-cont","source":"stderr","log":"AH00558: httpd: Could not reliably determine the server's
fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress
```

```
this message"}
2022-06-20 16:16:44.000000000 +0000 apache-cont:
{"container_id":"c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7","container_name":
"/apache-cont","source":"stderr","log":"AH00558: httpd: Could not reliably determine the server's
fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress
this message"}
2022-06-20 16:16:44.000000000 +0000 apache-cont: {"container_name":"/apache-
cont","source":"stderr","log":"[Mon Jun 20 16:16:44.766221 2022] [mpm_event:notice] [pid 1:tid
140594006998336] AH00489: Apache/2.4.54 (Unix) configured -- resuming normal
operations","container_id":"c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7"}
2022-06-20 16:16:44.000000000 +0000 apache-cont: {"log":"[Mon Jun 20 16:16:44.766439 2022]
[core:notice] [pid 1:tid 140594006998336] AH00094: Command line: 'httpd -D
FOREGROUND'","container_id":"c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7","cont
ainer_name":"/apache-cont","source":"stderr"}
```

- Where is the container sending httpd logs?
- Which part of the Fluentd event for the container has the actual httpd log content?

In order to `curl`, retrieve the container's IP address. While it is running on the same hardware, the httpd Docker container is running on a completely separate network namespace from the host, so `curl`ing localhost will only call on the locally installed apache2 instance.

In your working terminal, use `docker inspect` to retrieve the IP address of the apache container:

```
ubuntu@labsys:~/lab5$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}
{{end}}' apache-cont

172.17.0.2

ubuntu@labsys:~/lab5$
```

Setting the `--name` flag for the container to name it apache-cont allows you to refer to the container by a simpler name rather than the 64-character container ID to retrieve the container IP.

Send a curl event to the containerized httpd instance:

```
ubuntu@labsys:~/lab5$ curl -I http://172.17.0.2

HTTP/1.1 200 OK
Date: Mon, 20 Jun 2022 16:17:32 GMT
Server: Apache/2.4.54 (Unix)
Content-Type: text/html;charset=ISO-8859-1

ubuntu@labsys:~/lab5$
```

The httpd container is a more barebones setup compared to the apache2 package installed earlier in this lab, so the page retrieved is much simpler in comparison.

Check how Fluentd formatted the event:

```
2022-06-20 16:17:32.000000000 +0000 apache-cont:
{"container_id":"c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7","container_name":
"/apache-cont","source":"stdout","log":"172.17.0.1 - - [20/Jun/2022:16:17:32 +0000] \"HEAD /
HTTP/1.1\" 200 -"}
```

That is a verbose event; there may be a need to ensure that only the httpd log itself is being piped to its intended destination.

In this case, ensure that all httpd log entries are sent to the destination in a consistent format. The in_forward Fluentd plugin, which is handling the input, does not support any parser plugins. To retrieve the logs and format them to the proper Fluentd httpd format, the filter_parser plugin needs to perform the parsing after the source ingestion. This is why the `<parse>` subdirective can be used in all of the major directives in a Fluentd configuration.

In your working terminal, use the `apache2` parser plugin in a `<filter>` directive (which is also using the filter_parser plugin):

```
ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_

<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type apache
  </parse>
</source>

<source>
  @type forward
</source>

<filter apache-cont>
  @type parser
  key_name log
  <parse>
    @type apache2
  </parse>
</filter>

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

ubuntu@labsys:~/lab5$
```

This `filter_parser` plugin format the incoming events to match the format provided by the `<parse>` subdirective below it. This enables parsing for pipelines where a parser cannot be used with the a `<source>` directive.

Setting the event tag to the name of the container (which you provided as apache-cont) enables you to use a known substring as the `<filter>` directive's pattern. The answer to an earlier question noted that the actual log content from the container is attached to the `log` key in the event record, so it is selected as the `key_name` for parsing.

Reconfigure Fluentd with `pkill` in your working terminal:

```
ubuntu@labsys:~/lab5$ pkill -SIGUSR2 fluentd
```

```
...

2022-06-20 16:23:19 +0000 [info]: adding filter pattern="apache-cont" type="parser"
2022-06-20 16:23:19 +0000 [info]: adding match pattern="apache**" type="stdout"
2022-06-20 16:23:19 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 16:23:19 +0000 [info]: adding source type="tail"
2022-06-20 16:23:19 +0000 [info]: adding source type="forward"
2022-06-20 16:23:19 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 16:23:19 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:898"
2022-06-20 16:23:19 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:8ac"
```

```
2022-06-20 16:23:19 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:870"
2022-06-20 16:23:19 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 16:23:19 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2022-06-20 16:23:19 +0000 [info]: #0 following tail of /var/log/apache2/access.log
```

You can see that Fluentd now loads a filter that will process events tagged apache-cont, the name of your httpd container.

Rerun `curl` on the container IP and localhost:

```
ubuntu@labsys:~/lab5$ curl -I http://172.17.0.2 ; curl -I http://localhost

...

ubuntu@labsys:~/lab5$
```

And check the Fluentd event:

```
2022-06-20 16:23:50.000000000 +0000 apache-cont:
{"host":"172.17.0.1","user":null,"method":"HEAD","path":"/","code":200,"size":null,"referer":null,"a
gent":null}
2022-06-20 16:23:50.000000000 +0000 apache2.access: {"host":"127.0.0.1","user":"-
","method":"HEAD","path":"/","code":"200","size":"255","referer":"-","agent":"curl/7.68.0"}
```

Despite the two different tags providing event data from two different configurations (one from docker, tagged apache-cont, and the other a local installation, tagged apache2.access), their log outputs mostly look the same.

2. Exploring Format Plugins

The `<format>` directive allows a `<filter>` or `<match>` directive to output an incoming event in a different manner. They are called by filter and output plugins; which typically emit data to a destination (STDOUT and out_file, for example). The rule of thumb for format plugins is: if the output or filter plugin supports text formatting, the `<format>` subdirective can be used.

2a. Changing Event Metadata

Some use cases may require a specific format for time stamps or other intrinsic metadata that is included with an event. Fluentd ships with the out_file plugin, which allow a user to include certain metadata, such as timestamps or event tags, into outgoing events.

In your working terminal, add a `<format>` subdirective using the formatter_out_file plugin to the `<match>` directive in `lab5.conf`:

```
ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_

<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type apache2
  </parse>
</source>

<source>
  @type forward
</source>

<filter apache-cont>
```

```
  @type parser
  key_name log
  <parse>
    @type apache2
  </parse>
</filter>

<match apache**>
  @type stdout
  <format>
    @type out_file
    delimiter “COMMA”(‘,’)
    output_tag true
    output_time true
    time_format %Y%m%dT%H%M%S%z
  </format>
</match>

ubuntu@labsys:~/lab5$
```

The formatter_out_file plugin allows a user to determine the full output of an event meant for a destination system that does not have a preconfigured output format. So far, you have been using STDOUT which formats all events into STDOUT format. The reconfigured `<match>` directive will do the following:

- The `time_format %y/%m/%d - %H:%M:%S` parameter specifies a Ruby strftime (string format for time) expression that will change the event's timestamp
- Set `output_time true` and `output_tag true` to ensure the event time and tag respectively are printed with the event
- `delimiter “COMMA”(‘,’)` ensures all parts of the event are separated by a specific kind of delimiter, like a comma

Reload Fluentd with `pkill -SIGUSR2`:

```
ubuntu@labsys:~/lab5$ pkill -SIGUSR2 fluentd
```

At default verbosity, a formatter plugin is not listed in the startup data dump, though it may report an error when it tries to reload.

Use `curl` on localhost to see the results of this time format:

```
ubuntu@labsys:~/lab5$ curl -I http://localhost

...

ubuntu@labsys:~/lab5$
```

In the Fluentd terminal, the output of the event should have been changed accordingly:

```
20220620T162510+0000,apache2.access,
{"host":"127.0.0.1","user":null,"method":"HEAD","path":"/","code":200,"size":255,"referer":null,"age
nt":"curl/7.68.0"}
```

- How is the time different?
- Using the information on https://docs.ruby-lang.org/en/2.4.0/Time.html#method-i-strftime, change the time format.

Format plugins, in conjunction with filter and match plugins, can be used to ensure that events will conform to the required formats for their intended destinations.

2b. Changing Event log formats

Just as the metadata of an event can be changed with a format plugin, so too can the format of the record itself.

For this step, assume that the intended destination requires a specific form of JSON that is easily readable - pretty. In order to do this, you will utilize a third party plugin: formatter_pretty_json. For more information on this plugin, see its official GitHub page: https://github.com/mia-0032/fluent-plugin-formatter_pretty_json

Use `fluent-gem` to install `formatter_pretty_json` plugin, which will make JSON output from Fluentd more human readable:

```
ubuntu@labsys:~/lab5$ sudo fluent-gem install fluent-plugin-formatter_pretty_json -N -v 1.0.0

Fetching fluent-plugin-formatter_pretty_json-1.0.0.gem
Successfully installed fluent-plugin-formatter_pretty_json-1.0.0
1 gem installed

ubuntu@labsys:~/lab5$
```

In your working terminal, add the `formatter_pretty_json` as the format plugin for the Fluentd configuration:

```
ubuntu@labsys:~/lab5$ nano lab5.conf && cat $_

<source>
  @type tail
  path /var/log/apache2/access.log
  pos_file /tmp/lab5/apache_access.pos
  tag apache2.access
  <parse>
    @type apache
  </parse>
</source>

<source>
  @type forward
</source>

<filter apache-cont>
  @type parser
  key_name log
  <parse>
    @type apache2
  </parse>
</filter>

<match apache**>
  @type stdout
  <format>
    @type pretty_json
  </format>
</match>

ubuntu@labsys:~/lab5$
```

All JSON output from events processed by the `<match>` directive in this configuration will be presented in a human-readable JSON map.

In your working terminal, reconfigure Fluentd with a `SIGHUP` to fully reload the process and ensure Fluentd is using the new gem:

```
ubuntu@labsys:~/lab5$ pkill -SIGHUP fluentd
```

```
2022-06-20 16:27:12 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 16:27:12 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 16:27:12 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:af0"
2022-06-20 16:27:12 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:b04"
2022-06-20 16:27:12 +0000 [info]: #0 shutting down filter plugin type=:parser plugin_id="object:ab4"
2022-06-20 16:27:12 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:ac8"
2022-06-20 16:27:12 +0000 [error]: Worker 0 finished unexpectedly with status 0
2022-06-20 16:27:13 +0000 [info]: adding filter pattern="apache-cont" type="parser"
2022-06-20 16:27:13 +0000 [info]: adding match pattern="apache**" type="stdout"
2022-06-20 16:27:13 +0000 [info]: adding source type="tail"
2022-06-20 16:27:13 +0000 [info]: adding source type="forward"
2022-06-20 16:27:13 +0000 [info]: #0 starting fluentd worker pid=46114 ppid=45787 worker=0
2022-06-20 16:27:13 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2022-06-20 16:27:13 +0000 [info]: #0 following tail of /var/log/apache2/access.log
2022-06-20 16:27:13 +0000 [info]: #0 fluentd worker is now running worker=0
```

Send a `curl` command to the Apache2 instance at localhost:

```
ubuntu@labsys:~/lab5$ curl -I http://localhost

...

ubuntu@labsys:~/lab5$
```

Now see the format of the resulting Apache / httpd event in the Fluentd terminal:

```
{
  "host": "127.0.0.1",
  "user": "-",
  "method": "HEAD",
  "path": "/",
  "code": "200",
  "size": "255",
  "referer": "-",
  "agent": "curl/7.68.0"
}
```

There are many other plugins available for Fluentd that can influence the outgoing event format, such as msgpack or CSV.

Using parser and formatter plugins in your Fluentd configurations ensures that any data going into Fluentd can be manipulated to the fullest extent (by using parser plugins to break that data down into separate event record keys) and be output in formats acceptable to other applications in a stack (by using formatter plugins to influence the final event format).

Cleanup

Now tear down any running Docker containers:

```
ubuntu@labsys:~/lab5$ sudo docker container rm $(sudo docker container stop apache-cont)

apache-cont

ubuntu@labsys:~/lab5$
```

Use CTRL C to terminate any running instances of Fluentd:

```
^C

2022-06-20 16:28:06 +0000 [info]: Received graceful stop
2022-06-20 16:28:07 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 16:28:07 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 16:28:07 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:76c"
2022-06-20 16:28:07 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:780"
2022-06-20 16:28:07 +0000 [info]: #0 shutting down filter plugin type=:parser plugin_id="object:730"
2022-06-20 16:28:07 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:744"
2022-06-20 16:28:08 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab5$
```

Congratulations, you have completed the lab!