LFS242 - Cloud Native Logging with Fluentd

Lab 9 – Debugging & Tuning Fluentd

Fluentd needs to be configured correctly, not only to ensure that it runs and carries events to their intended destinations, but also to ensure that it is using system resources wisely. A Fluentd instance can be deployed on a host that has resource constraints (due to another application being there or the specifications of the host), so it becomes important to ensure that the Fluentd instance only uses the resources it needs and nothing more.

There are times where Fluentd needs more resources to process events, like when Fluentd is acting as a log aggregator and processing thousands, maybe millions of requests per second. For the log aggregator role, Fluentd can be tuned to use more resources to ensure that it has the bandwidth to handle the workloads intended for it.

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

- Practice fixing nonfunctional Fluentd configurations
- Explore Fluentd's multi-worker operation mode
- Learn how to establish TLS/SSL connections between two Fluentd instances

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 1.13.2

ubuntu@labsys:~$
```

This lab will also be using Docker, so an installation of Docker will be required.

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. Debugging Configurations

Debugging Fluentd configurations is usually a matter of ensuring that the configuration file is well formed, meaning:

- All directive tags must be closed properly
- All configuration parameters required by a plugin are present

This portion of the lab will present a variety of broken, outdated, or unhealthy Fluentd configuration files. The goal will be to address any errors or warnings that Fluentd generates using the debugging tools presented in the module text, such as `-v`, the `@FLUENT_LOG` labels, and Fluentd's own output.

Create a new configuration file called `error1.conf`:

```
ubuntu@labsys:~$ mkdir ~/lab9 && cd $_

ubuntu@labsys:~/lab9$ nano error1.conf && cat $_

<source>
  @type http
  port 32999
</source>

<match>
  @type file
  path /tmp/lab9/output.txt
<match>

ubuntu@labsys:~/lab9$
```

This configuration will launch a Fluentd instance that will listen for HTTP traffic on port 32999, then send all output to a file at `/tmp/lab9/output.txt`.

Try to run this configuration with Fluentd:

```
ubuntu@labsys:~/lab9$ fluentd -c error1.conf

Traceback (most recent call last):
        15: from /usr/local/bin/fluentd:23:in `<main>'
        14: from /usr/local/bin/fluentd:23:in `load'
        13: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/bin/fluentd:15:in `<top (required)>'
        12: from /usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
        11: from /usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
        10: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/fluentd.rb:354:in `<top
(required)>'
         9: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/fluentd.rb:354:in `new'
         8: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/supervisor.rb:618:in
`initialize'
         7: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config.rb:39:in `build'
         6: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config.rb:58:in `parse'
         5: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/v1_parser.rb:33:in
`parse'
         4: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/v1_parser.rb:44:in
`parse!'
         3: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/v1_parser.rb:96:in
`parse_element'
         2: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/v1_parser.rb:96:in
`parse_element'
```

```
        1: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/v1_parser.rb:66:in
`parse_element'
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config/basic_parser.rb:92:in `parse_error!':
expected end tag '</match>' but got end of file at error1.conf line 9,8 (Fluent::ConfigParseError)
  8:   path /tmp/lab9/output.txt
  9: <match>

    --------^

ubuntu@labsys:~/lab9$
```

This one failed to start. Fluentd stated that exact nature of the error: the `match` tag at line 9 lacks a closing slash. Syntactic errors like these prevent Fluentd from starting. Correct this configuration file by adding the "/" to line 9.

```
ubuntu@labsys:~/lab9$ nano error1.conf && cat error1.conf

<source>
  @type http
  port 32999
</source>

<match>
  @type file
  path /tmp/lab9/output.txt
</match>

ubuntu@labsys:~/lab9$
```

After the correction, try to run Fluentd. Use the `-o` flag to redirect Fluentd's standard output to a file a /tmp/lab9/error1.out.

Run Fluentd with `error1.conf`. After a moment, use `CTRL C` to exit and then read /tmp/lab9/error1.out

```
ubuntu@labsys:~/lab9$ fluentd -c error1.conf -o /tmp/lab9/error1.out

^C

ubuntu@labsys:~/lab9$ tail -5 /tmp/lab9/error1.out

2022-06-20 21:15:22 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 21:15:22 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 21:15:22 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:76c"
2022-06-20 21:15:22 +0000 [info]: #0 shutting down output plugin type=:file plugin_id="object:730"
2022-06-20 21:15:22 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

By fixing the wrong syntax, Fluentd was able to start.

Create a new configuration, naming it `error2.conf`:

```
ubuntu@labsys:~$ nano error2.conf && cat $_

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

```
<source>
  @type http
</source>

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

ubuntu@labsys:~$
```

Now try to run Fluentd with `error2.conf`:

```
ubuntu@labsys:~/lab9$ fluentd -c error2.conf -o /tmp/lab9/error2.out

ubuntu@labsys:~/lab9$ cat /tmp/lab9/error2.out

2022-06-20 21:15:56 +0000 [info]: parsing config file is succeeded path="error2.conf"
2022-06-20 21:15:56 +0000 [info]: gem 'fluent-plugin-prometheus' version '2.0.3'
2022-06-20 21:15:56 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 21:15:56 +0000 [info]: Oj isn't installed, fallback to Yajl as json parser
2022-06-20 21:15:56 +0000 [error]: config error file="error2.conf" error_class=Fluent::ConfigError
error="<parse> section is required."


ubuntu@labsys:~/lab9$
```

Fluentd was unable to start. There is a missing parameter in this configuration file: a `<parse>` section (subdirective) is required for one of the plugins. However, both source plugins used in this configuration use the `<parse>` subdirective, so more information is needed.

Rerun Fluentd with this configuration and raise the verbosity with the `-v` flag:

```
ubuntu@labsys:~/lab9$ fluentd -c error2.conf -v -o /tmp/lab9/error2.out

ubuntu@labsys:~/lab9$ cat /tmp/lab9/error2.out

2022-06-20 21:15:56 +0000 [info]: parsing config file is succeeded path="error2.conf"

...

2022-06-20 21:16:23 +0000 [error]: fluent/log.rb:372:error: config error file="error2.conf"
error_class=Fluent::ConfigError error="<parse> section is required."
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/in_tail.rb:130:in `configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin.rb:187:in `configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/root_agent.rb:320:in `add_source'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/root_agent.rb:161:in `block in configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/root_agent.rb:155:in `each'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/root_agent.rb:155:in `configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/engine.rb:105:in `configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/engine.rb:80:in `run_configure'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
```

```
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/supervisor.rb:668:in `run_supervisor'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/fluentd.rb:356:in `<top (required)>'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/bin/fluentd:15:in `<top (required)>'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/usr/local/bin/fluentd:23:in `load'
  2022-06-20 21:16:23 +0000 [debug]: core_ext/kernel_require.rb:72:require:
/usr/local/bin/fluentd:23:in `<main>'


ubuntu@labsys:~/lab9$
```

There it is: with the verbosity increased to DEBUG using the `-v` flag, Fluentd now shows a stack trace of the error. The first line of the error message mentions the in_tail plugin, which provides evidence that Fluentd failed to start when it tried to load the in_tail plugin in this configuration.

Add a `<parse>` subdirective to the in_tail `<source>` directive using the Apache2 parser plugin:

```
ubuntu@labsys:~/lab9$ nano error2.conf && cat $_

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

<source>
  @type http
</source>

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

ubuntu@labsys:~/lab9$
```

Try starting Fluentd with `error2.conf`. If it stays running, use `CTRL C` to exit after running it for a moment:

```
ubuntu@labsys:~/lab9$ fluentd -c error2.conf -v -o /tmp/lab9/error2.out

^C

ubuntu@labsys:~/lab9$ cat /tmp/lab9/error2.out | grep "now running" -2

2022-06-20 21:19:27 +0000 [debug]: #0 fluent/log.rb:309:debug: tailing paths: target =
/var/log/apache2/access.log | existing =
2022-06-20 21:19:27 +0000 [info]: #0 fluent/log.rb:330:info: following tail of
/var/log/apache2/access.log
2022-06-20 21:19:27 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 21:19:28 +0000 [debug]: #0 fluent/log.rb:309:debug: fluentd main process get SIGINT
```

```
2022-06-20 21:19:28 +0000 [info]: fluent/log.rb:330:info: Received graceful stop

ubuntu@labsys:~/lab9$
```

Fluentd started successfully! Ensuring that all plugins have their required parameters and subdirectives is key to running Fluentd correctly.

Next, create a third configuration named `warning1.conf`:

```
ubuntu@labsys:~/lab9$ nano warning1.conf && cat $_

<source>
@type forward
port 24224
</source>

<match>
@type stdout
format msgpack
</match>

ubuntu@labsys:~/lab9$
```

See how Fluentd loads this one, this time running the configuration file with TRACE level debugging, using `-vv`:

```
ubuntu@labsys:~/lab9$ fluentd -c warning1.conf -vv

...

2022-06-20 21:20:18 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
  pid i ppid ë worker message 5starting fluentd worker pid=53654 ppid=53649 worker=0
  port ^  bind 0.0.0.0 message (listening port port=24224 bind="0.0.0.0"
  worker message &fluentd worker is now running worker=0
```

This configuration starts, but it has generated warnings: The configuration is using syntax from versions below v0.14 to name each plugin under the `<source>` and `<match>` directives (using type rather than @type) and also declaring the format in the `<match>` directive.

Despite this, it will still work. As the TRACE messages show, Fluentd loaded all the plugins and their default settings. You can also see how Fluentd loaded the modern equivalent for the `format msgpack` parameter, a `<format>` subdirective.

In another terminal window, send an event to Fluentd:

```
ubuntu@labsys:~$ echo '{"lfs242":"learn"}' | fluent-cat mod9.lab

ubuntu@labsys:~$
```

In the Fluentd terminal, that event should have been received:

```
...


2022-06-20 21:20:36 +0000 [trace]: #0 fluent/log.rb:287:trace: connected fluent socket
addr="127.0.0.1" port=37774
2022-06-20 21:20:36 +0000 [trace]: #0 fluent/log.rb:287:trace: accepted fluent socket
addr="127.0.0.1" port=37774
  lfs242 learn
  addr  127.0.0.1 port  message 3connected fluent socket addr="127.0.0.1" port=37774
```

```
addr 127.0.0.1 port message 2accepted fluent socket addr="127.0.0.1" port=37774
```

Fluentd recorded the connection and event acceptance events as well in trace mode!

Terminate this Fluentd instance with `CTRL C` before proceeding with the last debugging exercise:

```
...
^C

2022-06-20 21:20:55 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

Now create another configuration file with the following context, making it `debugme.conf`:

```
ubuntu@labsys:~/lab9$ nano debugme.conf && cat $_

<source>
  @type forward
  port 24000
</source>

<match **>
  @type null
</match>

<match **>
  @type file
  path /tmp/wordpress-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
</match>

ubuntu@labsys:~$
```

Now start it with `-vv` so it runs at TRACE verbosity:

```
ubuntu@labsys:~/lab9$ fluentd -c debugme.conf -vv

2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="debugme.conf"
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-formatter_pretty_json'
version '1.0.0'
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-prometheus' version
'2.0.3'
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: gem 'fluentd' version '1.14.6'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'null'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'file'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered buffer plugin 'file'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered formatter plugin 'out_file'
2022-06-20 21:21:32 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'forward'
2022-06-20 21:21:32 +0000 [warn]: fluent/log.rb:351:warn: define <match fluent.**> to capture
```

```
fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: using configuration file: <ROOT>
  <source>
    @type forward
    port 24000
  </source>
  <match **>
    @type null
  </match>
  <match **>
    @type file
    path "/tmp/wordpress-log"
    <buffer>
      timekey 60s
      timekey_wait 1m
      path "/tmp/wordpress-log"
    </buffer>
  </match>
</ROOT>
2022-06-20 21:21:32 +0000 [info]: fluent/log.rb:330:info: starting fluentd-1.14.6 pid=53667
ruby="2.7.0"

...

2022-06-20 21:21:33 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
```

So far, so good. It looks like the syntax is correct and all directives are well-formed. Fluentd has generated no warnings or errors.

In another terminal window, send events to this Fluentd instance and then flush the buffer:

```
ubuntu@labsys:~$ cd ~/lab9

ubuntu@labsys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000

ubuntu@labsys:~/lab9$ pkill -sigusr1 fluentd

ubuntu@labsys:~$
```

And check the file:

```
ubuntu@labsys:~/lab9$ ls -l /tmp/lab9/

total 20
-rw-rw-r-- 1 ubuntu ubuntu 2107 Jun 20 21:15 error1.out
-rw-rw-r-- 1 ubuntu ubuntu 9318 Jun 20 21:19 error2.out
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:15 output.txt

ubuntu@labsys:~/lab9$
```

The file isn't there - just a directory! What happened?

Check the Fluentd terminal:

```
...

2022-06-20 21:22:17 +0000 [debug]: fluent/log.rb:309:debug: fluentd supervisor process get SIGUSR1
2022-06-20 21:22:17 +0000 [debug]: #0 fluent/log.rb:309:debug: fluentd main process get SIGUSR1
```

```
2022-06-20 21:22:17 +0000 [info]: #0 fluent/log.rb:330:info: force flushing buffered events
2022-06-20 21:22:17 +0000 [info]: #0 fluent/log.rb:330:info: flushing all buffer forcedly
2022-06-20 21:22:17 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1900
2022-06-20 21:22:17 +0000 [debug]: #0 fluent/log.rb:309:debug: flushing thread: flushed

...
```

Fluentd definitely received the event - there was a connection to the Fluent socket that was accepted. It also recorded reception of the `SIGUSR1` to flush the buffers, which it managed to do successfully as well. Something's not right.

Shut down the Fluentd instance:

```
...
^C

2022-06-20 21:22:58 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

One way to debug event flow is by using the `filter_stdout` plugin. This will capture events at the point in the processing pipeline where the directive was placed, and output them to STDOUT without removing the event from the pipeline. To use it in a debugging role, place a `<filter>` directive using `@type stdout` between the existing directives in the configuration file.

Place a STDOUT filter between each of the directives in the `debugme.conf` file:

```
ubuntu@labsys:~/lab9$ nano debugme.conf && cat $_

<source>
  @type forward
  port 24000
</source>

## Debugging instrumentation
<filter>
  @type stdout
</filter>
## End Debugging instrumentation

<match **>
  @type null
</match>

## Debugging instrumentation
<filter>
  @type stdout
</filter>
## End Debugging instrumentation

<match **>
  @type file
  path /tmp/lab9/wordpress-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
</match>
```

```
ubuntu@labsys:~/lab9$
```

Now rerun Fluentd with TRACE verbosity (`-vv`):

```
ubuntu@labsys:~/lab9$ fluentd -c debugme.conf -vv

2022-06-20 21:23:30 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="debugme.conf"

...

2022-06-20 21:23:31 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=53698
ppid=53693 worker=0
2022-06-20 21:23:31 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1940
stage_size=0 queue_size=0
2022-06-20 21:23:31 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 21:23:31 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24000
bind="0.0.0.0"
2022-06-20 21:23:31 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 21:23:31 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 21:23:31 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1940
2022-06-20 21:23:31.719968154 +0000 fluent.info:
{"pid":53698,"ppid":53693,"worker":0,"message":"starting fluentd worker pid=53698 ppid=53693
worker=0"}
2022-06-20 21:23:31.720330104 +0000 fluent.debug:
{"instance":1940,"stage_size":0,"queue_size":0,"message":"buffer started instance=1940 stage_size=0
queue_size=0"}
2022-06-20 21:23:31.721412566 +0000 fluent.debug: {"message":"flush_thread actually running"}
2022-06-20 21:23:31.721650371 +0000 fluent.info: {"port":24000,"bind":"0.0.0.0","message":"listening
port port=24000 bind=\"0.0.0.0\""}
2022-06-20 21:23:31.722166274 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 21:23:31.722413954 +0000 fluent.debug: {"message":"enqueue_thread actually running"}
2022-06-20 21:23:31.722525194 +0000 fluent.trace: {"instance":1940,"message":"enqueueing all chunks
in buffer instance=1940"}
2022-06-20 21:23:37 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1940
```

A Fluentd event has been printed to STDOUT, which is a good sign. It means that events are being discarded and not being sent to the buffer file at some point in the pipeline.

In another terminal (or your working terminal), send a test event to Fluentd:

```
ubuntu@labsys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000

ubuntu@labsys:~/lab9$
```

Then check Fluentd to see if your test event was printed:

```
...

2022-06-20 21:24:25 +0000 [trace]: #0 fluent/log.rb:287:trace: connected fluent socket
addr="127.0.0.1" port=45968
2022-06-20 21:24:25.066808000 +0000 fluent.trace:
{"addr":"127.0.0.1","port":45968,"message":"connected fluent socket addr=\"127.0.0.1\" port=45968"}
2022-06-20 21:24:25 +0000 [trace]: #0 fluent/log.rb:287:trace: accepted fluent socket
```

```
addr="127.0.0.1" port=45968
2022-06-20 21:24:25.067135215 +0000 fluent.trace:
{"addr":"127.0.0.1","port":45968,"message":"accepted fluent socket addr=\"127.0.0.1\" port=45968"}
2022-06-20 21:24:25.066396720 +0000 mod9.lab: {"lfs242":"event"}

...
```

This confirms that there's something in the configuration file preventing your test events from going to your intended file.

Terminate the Fluentd session again:

```
^C

...

2022-06-20 21:24:56 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

STDOUT `<filter>` and `<match>` directives can serve as breakpoints. Now that you know that the configuration file is discarding events at some point, try to narrow it down by removing one of your debugging settings.

Remove the first `<filter>` directives you are using for debugging from the configuration file:

```
ubuntu@labsys:~/lab9$ nano debugme.conf && cat $_

<source>
  @type forward
  port 24000
</source>

<match **>
  @type null
</match>

## Debugging instrumentation
<filter>
  @type stdout
</filter>
## End Debugging instrumentation

<match **>
  @type file
  path /tmp/lab9/wordpress-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
</match>

ubuntu@labsys:~/lab9$
```

Now rerun Fluentd again:

```
ubuntu@labsys:~/lab9$ fluentd -c debugme.conf -vv

...
```

```
2022-06-20 21:25:34 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=53717
ppid=53712 worker=0
2022-06-20 21:25:34 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1920
stage_size=0 queue_size=0
2022-06-20 21:25:34 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 21:25:34 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24000
bind="0.0.0.0"
2022-06-20 21:25:34 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 21:25:34 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 21:25:34 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1920
```

The fluent.info events are gone again! So the STDOUT `<filter>` directive you removed was between the configuration file's input and the point where events were being discarded.

At this point the answer should be clear: There is a catch-all `<match>` directive in the middle that's discarding events.

Terminate the instance:

```
^C

...

2022-06-20 21:25:50 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

Remove the offending `<match>` directive from the configuration:

```
ubuntu@labsys:~/lab9$ nano debugme.conf && cat $_

<source>
  @type forward
  port 24000
</source>

## Debugging instrumentation
<filter>
  @type stdout
</filter>
## End Debugging instrumentation

<match **>
  @type file
  path /tmp/lab9/wordpress-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
</match>

ubuntu@labsys:~/lab9$
```

With those changes in place rerun Fluentd again:

```
ubuntu@labsys:~/lab9$ fluentd -c debugme.conf -vv
```

```
2022-06-20 21:26:20 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="debugme.conf"

...

2022-06-20 21:26:21 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=53732
ppid=53727 worker=0
2022-06-20 21:26:21 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1880
stage_size=0 queue_size=0
2022-06-20 21:26:21 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 21:26:21 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24000
bind="0.0.0.0"
2022-06-20 21:26:21 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 21:26:21 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1880
2022-06-20 21:26:21 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 21:26:21.766700118 +0000 fluent.info:
{"pid":53732,"ppid":53727,"worker":0,"message":"starting fluentd worker pid=53732 ppid=53727
worker=0"}
```

Now send an event to Fluentd, making sure to flush the buffer to ensure the events are written to the intended file.

```
ubuntu@labsys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000

ubuntu@labsys:~/lab9$ pkill -sigusr1 fluentd

ubuntu@labsys:~/lab9$
```

Now, check the /tmp/lab9 directory; there should now be a wordpress-log suffixed with a timestamp:

```
ubuntu@labsys:~/lab9$ ls -l /tmp/lab9/

total 32
-rw-rw-r-- 1 ubuntu ubuntu 2107 Jun 20 21:15 error1.out
-rw-rw-r-- 1 ubuntu ubuntu 9318 Jun 20 21:19 error2.out
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:15 output.txt
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:27 wordpress-log
-rw-r--r-- 1 ubuntu ubuntu 1611 Jun 20 21:27 wordpress-log.202206202126_0.log
-rw-r--r-- 1 ubuntu ubuntu 1715 Jun 20 21:27 wordpress-log.202206202127_0.log


ubuntu@labsys:~/lab9$ tail -8 /tmp/lab9/wordpress-log.202206202127_0.log

2022-06-20T21:27:02+00:00       fluent.trace    {"message":"chunk /tmp/lab9/wordpress-
log/buffer.b5e1e7c3cf9b182d958d1d1e9028d613c.log size_added: 54 new_size: 193"}
2022-06-20T21:27:05+00:00       fluent.debug    {"message":"fluentd main process get SIGUSR1"}
2022-06-20T21:27:05+00:00       fluent.info     {"message":"force flushing buffered events"}
2022-06-20T21:27:05+00:00       fluent.info     {"message":"flushing all buffer forcedly"}
2022-06-20T21:27:05+00:00       fluent.trace    {"instance":1880,"message":"enqueueing all chunks in
buffer instance=1880"}
2022-06-20T21:27:05+00:00       fluent.trace    {"instance":1880,"metadata":"#<struct
Fluent::Plugin::Buffer::Metadata timekey=1655760360, tag=nil, variables=nil,
seq=0>","message":"enqueueing chunk instance=1880 metadata=#<struct Fluent::Plugin::Buffer::Metadata
timekey=1655760360, tag=nil, variables=nil, seq=0>"}
2022-06-20T21:27:05+00:00       fluent.trace    {"instance":1880,"metadata":"#<struct
Fluent::Plugin::Buffer::Metadata timekey=1655760420, tag=nil, variables=nil,
seq=0>","message":"enqueueing chunk instance=1880 metadata=#<struct Fluent::Plugin::Buffer::Metadata
timekey=1655760420, tag=nil, variables=nil, seq=0>"}
```

```
2022-06-20T21:27:05+00:00       fluent.trace    {"instance":1880,"message":"dequeueing a chunk
instance=1880"}

ubuntu@labsys:~/lab9$
```

The file was written with the most recent user-submitted event. It did capture other Fluentd logs due to the trace verbosity, though. There's a way to fix that.

Terminate Fluentd again:

```
^C

...

2022-06-20 21:27:42 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

Now add the `<label @FLUENT_LOG>` directive to the end of the file:

```
ubuntu@labsys:~/lab9$ nano debugme.conf && cat $_

<source>
  @type forward
  port 24000
</source>

## Debugging instrumentation
<filter>
  @type stdout
</filter>
## End Debugging instrumentation

<match **>
  @type file
  path /tmp/lab9/wordpress-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
</match>

<label @FLUENT_LOG>
  <match>
    @type file
    path /tmp/lab9/fluentd.events.log
  </match>
</label>

ubuntu@labsys:~/lab9$
```

Rerun Fluentd after making those changes:

```
ubuntu@labsys:~/lab9$ fluentd -c debugme.conf -vv

2022-06-20 21:28:29 +0000 [info]: fluent/log.rb:330:info: adding match in @FLUENT_LOG pattern="**"
type="file"
```

```
...

2022-06-20 21:28:29 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0

...
```

Fluentd should now be sending its TRACE logs to another series of files prefixed with `fluentd.events.log.`, while all other events should be routed to the wordpress-log series of files.

Check the /tmp/lab9 directory to see if a buffer directory for all Fluentd events has been created:

```
ubuntu@labsys:~/lab9$ ls -l /tmp/lab9

total 44
-rw-rw-r-- 1 ubuntu ubuntu 2107 Jun 20 21:15 error1.out
-rw-rw-r-- 1 ubuntu ubuntu 9318 Jun 20 21:19 error2.out
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:28 fluentd.events.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:15 output.txt
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:29 wordpress-log
-rw-r--r-- 1 ubuntu ubuntu 1611 Jun 20 21:27 wordpress-log.202206202126_0.log
-rw-r--r-- 1 ubuntu ubuntu 1715 Jun 20 21:27 wordpress-log.202206202127_0.log
-rw-r--r-- 1 ubuntu ubuntu 5487 Jun 20 21:29 wordpress-log.202206202127_1.log

ubuntu@labsys:~/lab9$
```

There it is! Now send a test event to Fluentd and check the file to see if your wordpress-log files still receive trace messages:

```
ubuntu@labsys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000

ubuntu@labsys:~/lab9$ pkill -sigusr1 fluentd

ubuntu@labsys:~/lab9$ ls -l /tmp/lab9

total 60
-rw-rw-r-- 1 ubuntu ubuntu 2107 Jun 20 21:15 error1.out
-rw-rw-r-- 1 ubuntu ubuntu 9318 Jun 20 21:19 error2.out
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:29 fluentd.events.log
-rw-r--r-- 1 ubuntu ubuntu 9358 Jun 20 21:29 fluentd.events.log.20220620_0.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:15 output.txt
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 21:29 wordpress-log
-rw-r--r-- 1 ubuntu ubuntu 1611 Jun 20 21:27 wordpress-log.202206202126_0.log
-rw-r--r-- 1 ubuntu ubuntu 1715 Jun 20 21:27 wordpress-log.202206202127_0.log
-rw-r--r-- 1 ubuntu ubuntu 5487 Jun 20 21:29 wordpress-log.202206202127_1.log
-rw-r--r-- 1 ubuntu ubuntu   54 Jun 20 21:29 wordpress-log.202206202129_0.log

ubuntu@labsys:~/lab9$ tail -5 /tmp/lab9/wordpress-log.202206202129_0.log

2022-06-20T21:29:23+00:00       mod9.lab        {"lfs242":"event"}

ubuntu@labsys:~/lab9$
```

So far so good: the actual event log is no longer recording the trace events. Now check the log file declared in the `@FLUENT_LOG` label:

```
ubuntu@labsys:~/lab9$ tail -5 /tmp/lab9/fluentd.events.log.*.log

2022-06-20T21:29:25+00:00       fluent.trace
{"chunk":"5e1e7cc34698349bbd3ceaf6c2611e5f","delayed":false,"message":"committing write operation to
```

```
a chunk chunk=\"5e1e7cc34698349bbd3ceaf6c2611e5f\" delayed=false"}
2022-06-20T21:29:25+00:00       fluent.trace
{"instance":1940,"chunk_id":"5e1e7cc34698349bbd3ceaf6c2611e5f","metadata":"#<struct
Fluent::Plugin::Buffer::Metadata timekey=1655760540, tag=nil, variables=nil,
seq=0>","message":"purging a chunk instance=1940 chunk_id=\"5e1e7cc34698349bbd3ceaf6c2611e5f\"
metadata=#<struct Fluent::Plugin::Buffer::Metadata timekey=1655760540, tag=nil, variables=nil,
seq=0>"}
2022-06-20T21:29:25+00:00       fluent.trace
{"instance":1940,"chunk_id":"5e1e7cc34698349bbd3ceaf6c2611e5f","metadata":"#<struct
Fluent::Plugin::Buffer::Metadata timekey=1655760540, tag=nil, variables=nil,
seq=0>","message":"chunk purged instance=1940 chunk_id=\"5e1e7cc34698349bbd3ceaf6c2611e5f\"
metadata=#<struct Fluent::Plugin::Buffer::Metadata timekey=1655760540, tag=nil, variables=nil,
seq=0>"}
2022-06-20T21:29:25+00:00       fluent.trace
{"chunk":"5e1e7cc34698349bbd3ceaf6c2611e5f","message":"done to commit a chunk
chunk=\"5e1e7cc34698349bbd3ceaf6c2611e5f\""}
2022-06-20T21:29:25+00:00       fluent.trace    {"instance":1860,"metadata":"#<struct
Fluent::Plugin::Buffer::Metadata timekey=1655683200, tag=nil, variables=nil,
seq=0>","message":"enqueueing chunk instance=1860 metadata=#<struct Fluent::Plugin::Buffer::Metadata
timekey=1655683200, tag=nil, variables=nil, seq=0>"}

ubuntu@labsys:~/lab9$
```

The `@FLUENT_LOG` label can be used to run a Fluentd instance at high verbosity without compromising any existing Fluentd event destinations.

The unhealthy configurations presented in this lab are simple in comparison to real world solutions. The techniques shown here, however, can be used to achieve those solutions:

- Adjusting the verbosity (`-v` and `-vv`)
- Instrumenting event pipelines with the filter_stdout and out_stdout plugins
- Redirecting logs to system labels, like `@FLUENT_LOG` (shown here) and `@ERROR` (shown in Chapter 9)

Terminate Fluentd before proceeding with the next step:

```
^C

2022-06-20 21:42:46 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab9$
```

2. Tuning Fluentd

Fluentd can be tuned by adjusting its resource footprint on the systems it is deployed on. The amount of tuning needed depends on how many plugins are selected and how complex the processing pipelines are. Performance tuning techniques can range from ensuring that configurations are lean to adjusting Ruby environment variables that influence a Fluentd instance's overall resource footprint.

2a. Ruby Garbage Collection Optimization

Ruby garbage collection is the process that Ruby employs to reclaim unused system memory. Since Ruby acts as a wrapper for Fluentd, this process can have a major effect on Fluentd's memory utilization when it is not adjusted properly. Memory is one of Fluentd's most consumed resources because the entire configuration file is loaded into memory and many plugins (especially those that buffer into memory) may also seek to use system memory to perform their operations.

In a cloud native environment, Ruby Garbage Collection is something you can do to help optimize your Fluentd instances for smaller, cheaper environments (especially running in containers).

In this step, you will be shown how to set Ruby garbage collection parameters to affect a Fluentd instance's memory footprint:

```
ubuntu@labsys:~/lab9$ nano tuneme.conf && cat $_

<source>
  # WordPress Database
  @type forward
  port 24000
  @label wordpress
</source>

<source>
  # WordPress
  @type forward
  port 24100
  @label wordpress
</source>

<source>
  # Guestbook Database
  @type forward
  port 24200
  @label guestbook
</source>

<source>
  # Guestbook
  @type forward
  port 24300
  @label guestbook
</source>

<label wordpress>
  <match wordpress>
    @type relabel
    @label frontend
  </match>
  <match wordpress-db>
    @type relabel
    @label db
  </match>
</label>

<label guestbook>
  <match guestbook>
    @type relabel
    @label frontend
  </match>
  <match guestbook-db>
    @type relabel
    @label db
  </match>
</label>

<label frontend>
  <match **>
    @type file
    path /tmp/lab9/frontend-log
    <buffer>
      @type memory
    </buffer>
  </match>
</label>
```

```
<label db>
  <match **>
    @type file
    path /tmp/lab9/database-log
    <buffer>
      @type memory
    </buffer>
  </match>
</label>

ubuntu@labsys:~/lab9$
```

This configuration file is the same as one prepared in an earlier lab, except it has been configured to buffer events in memory. One of the tuning conventions for Fluentd is to keep configurations lean, but buffered plugins like out_file will have a direct effect on a Fluentd instance's memory footprint.

Run this configuration in Fluentd, sending its output to `/tmp/lab9/tuningconf.out` with the `-o` flag set it as a background task with an `&`:

```
ubuntu@labsys:~/lab9$ fluentd -c tuneme.conf -o /tmp/lab9/tuningconf.out &

[1] 54245

ubuntu@labsys:~/lab9$
```

Now, check the resource usage of the Fluentd instance with `top`:

```
ubuntu@labsys:~/lab9$ top -p `pgrep -d "," fluentd\|ruby`

top - 21:45:13 up  7:04,  3 users,  load average: 0.26, 0.12, 0.04
Tasks:   2 total,   0 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3865.9 total,    262.5 free,    345.7 used,   3257.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   3227.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  54245 ubuntu    20   0  305492  46716   9208 S   0.0   1.2   0:00.82 fluentd
  54250 ubuntu    20   0  643320  51312   9220 S   0.0   1.3   0:00.75 ruby2.7

q

ubuntu@labsys:~/lab9$
```

Right now, Fluentd is using about 55-56MB of memory between its supervisor and worker process and it hasn't even done anything at this point. Since it is using memory to buffer events, this instance has the potential to use up a lot of memory.

Kill the Fluentd instance:

```
ubuntu@labsys:~/lab9$ pkill -f fluentd

ubuntu@labsys:~/lab9$ # press enter

[1]+  Done                    fluentd -c tuneme.conf -o /tmp/lab9/tuningconf.out

ubuntu@labsys:~/lab9$
```

To influence Fluentd's resource usage, Ruby garbage collection parameters can be adjusted by passing environment variables to the Fluentd host.

```
ubuntu@labsys:~/lab9$ RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR=0.1 \
fluentd -c tuneme.conf -o /tmp/lab9/tuningconf.out &

[1] 54268

ubuntu@labsys:~/lab9$
```

RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR limits the factor at which Fluentd will grow to use memory. This will affect the starting size of Fluentd.

Check the resource usage of the Fluentd instance again with `top`:

```
ubuntu@labsys:~/lab9$ top -p `pgrep -d "," fluentd\|ruby`

top - 21:45:52 up  7:05,  3 users,  load average: 0.22, 0.12, 0.04
Tasks:   2 total,   0 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3865.9 total,    261.3 free,    346.9 used,   3257.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   3226.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  54268 ubuntu    20   0  305972  47228   9112 S   0.0   1.2   0:01.28 fluentd
  54273 ubuntu    20   0  643520  51536   8940 S   0.0   1.3   0:01.06 ruby2.7

q

ubuntu@labsys:~/lab9$
```

By reducing the `RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR` value, you have achieved a slight reduction in Fluentd's initial memory usage. Once this configuration is run in a more significant workload, this Ruby parameter and other garbage collection parameters should be adjusted according to the memory use of a Fluentd instance.

Be sure to clear the environment variable before moving on and kill the instance of Fluentd:

```
ubuntu@labsys:~/lab9$ pkill -f fluentd

ubuntu@labsys:~/lab9$ export RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR=

[1]+  Done                    RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR=0.1 fluentd -c tuneme.conf -o
/tmp/lab9/tuningconf.out

ubuntu@labsys:~/lab9$
```

Many container runtimes like Docker and container orchestrator like Kubernetes allow you to set environment variables to help tune in-container parameters. When used in conjunction with resource limit settings (such as those in Kubernetes), you have a lot of options to help optimize your Fluentd instances in your cloud native environments.

2b. Setting up Fluentd with Multiple Worker Processes

Next you will explore the use of multiple worker processes with Fluentd. Worker processes are what execute the directives placed inside a configuration file. As you saw in the previous step, there are two Fluentd processes, one started with the `fluentd` command and another started with `ruby2.3` with the `--under-supervisor` flag. You also see these as the ones being reloaded whenever you send a `SIGHUP` to a Fluentd instance.

By separating workloads between multiple workers, Fluentd can use additional CPU threads to perform event processing.

Use the following configuration to run a Fluentd instance:

```
ubuntu@labsys:~/lab9$ nano multiworker.conf && cat $_

<system>
  process_name aggregator
</system>

<source>
  @type forward
  port 24500
</source>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab9$
```

Run Fluentd using `multiworker.conf`. Use the `-o` flag to have the instance output to a file and `&` to run it in the background:

```
ubuntu@labsys:~/lab9$ fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out &

[1] 54296

ubuntu@labsys:~/lab9$
```

Now tail the output log:

```
ubuntu@labsys:~/lab9$ tail /tmp/lab9/multiworker.out

2022-06-20 21:47:24 +0000 [info]: adding match pattern="**" type="stdout"
2022-06-20 21:47:24 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 21:47:24 +0000 [info]: adding source type="forward"
2022-06-20 21:47:24 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 21:47:24 +0000 [info]: #0 starting fluentd worker pid=54301 ppid=54296 worker=0
2022-06-20 21:47:24 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-20 21:47:24 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 21:47:24.765607205 +0000 fluent.info:
{"pid":54301,"ppid":54296,"worker":0,"message":"starting fluentd worker pid=54301 ppid=54296
worker=0"}
2022-06-20 21:47:24.766372961 +0000 fluent.info: {"port":24500,"bind":"0.0.0.0","message":"listening
port port=24500 bind=\"0.0.0.0\""}
2022-06-20 21:47:24.766886206 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

ubuntu@labsys:~/lab9$
```

Use `ps` to check running processes:

```
ubuntu@labsys:~/lab9$ ps

    PID TTY          TIME CMD
```

```
  44150 pts/1    00:00:00 bash
  54296 pts/1    00:00:00 fluentd
  54301 pts/1    00:00:00 ruby2.7
  54305 pts/1    00:00:00 ps

ubuntu@labsys:~/lab9$
```

By running `ps`, you can see the supervisor and worker processes. The supervisor manages each of the workers, and is the main entrypoint for all system signals coming into the instance. Workers load directives in the configuration and execute them - this is what Fluentd indicates at the end of the startup sequence when it states, `fluentd worker is now running worker=0`

Use `pkill` to terminate Fluentd:

```
ubuntu@labsys:~/lab9$ pkill fluentd

ubuntu@labsys:~/lab9$ tail /tmp/lab9/multiworker.out

2022-06-20 21:47:24.765607205 +0000 fluent.info:
{"pid":54301,"ppid":54296,"worker":0,"message":"starting fluentd worker pid=54301 ppid=54296
worker=0"}
2022-06-20 21:47:24.766372961 +0000 fluent.info: {"port":24500,"bind":"0.0.0.0","message":"listening
port port=24500 bind=\"0.0.0.0\""}
2022-06-20 21:47:24.766886206 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 21:47:58 +0000 [info]: Received graceful stop
2022-06-20 21:47:58 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 21:47:58.710538224 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
stopping worker=0"}
2022-06-20 21:47:58 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 21:47:58 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 21:47:58 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:730"
2022-06-20 21:47:59 +0000 [info]: Worker 0 finished with status 0
[1]+  Done                    fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out

ubuntu@labsys:~/lab9$
```

To launch multiple workers, the workers parameter needs to be set under the `<system>` directive inside a Fluentd configuration.

Edit the configuration file, adding `workers 3` to the `<system>` directive at the top of the `multiworker.conf` file:

```
ubuntu@labsys:~/lab9$ nano multiworker.conf && cat $_

<system>
  process_name aggregator
  workers 3
</system>

<source>
  @type forward
  port 24500
</source>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab9$
```

Under the `<system>` directive, the `workers 3` parameter has been passed. This will ensure that Fluentd will spawn two additional

workers when it is started.

Now rerun Fluentd with that configuration:

```
ubuntu@labsys:~/lab9$ fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out -vv &

[1] 54312

ubuntu@labsys:~/lab9$ cat /tmp/lab9/multiworker.out | grep "fluentd worker is now running worker"

2022-06-20 21:47:24 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 21:47:24.766886206 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 21:48:32 +0000 [info]: #2 fluent/log.rb:330:info: fluentd worker is now running worker=2
2022-06-20 21:48:32 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 21:48:32.830282637 +0000 fluent.info: {"worker":2,"message":"fluentd worker is now
running worker=2"}
2022-06-20 21:48:32.831987457 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 21:48:32 +0000 [info]: #1 fluent/log.rb:330:info: fluentd worker is now running worker=1
2022-06-20 21:48:32.894363349 +0000 fluent.info: {"worker":1,"message":"fluentd worker is now
running worker=1"}

ubuntu@labsys:~/lab9$
```

Multiple workers have been launched. Notice how each worker loads its own instance of both the `in_forward` plugin and the `out_stdout`. Fluentd separates logs from each worker, by appending a number after the initial timestamp and level.

Now run `ps` to check the amount of worker processes. Remember that the Fluentd supervisor launches workers as Ruby processes:

```
ubuntu@labsys:~/lab9$ ps

    PID TTY          TIME CMD
  44150 pts/1    00:00:00 bash
  54312 pts/1    00:00:00 fluentd
  54317 pts/1    00:00:01 ruby2.7
  54318 pts/1    00:00:01 ruby2.7
  54319 pts/1    00:00:01 ruby2.7
  54330 pts/1    00:00:00 ps

ubuntu@labsys:~/lab9$
```

Send an event to Fluentd using `fluent-cat`, then check the log:

```
ubuntu@labsys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24500

ubuntu@labsys:~/lab9$ tail /tmp/lab9/multiworker.out

2022-06-20 21:49:26 +0000 [trace]: #1 fluent/log.rb:287:trace: connected fluent socket
addr="127.0.0.1" port=58758
2022-06-20 21:49:26.951760225 +0000 fluent.trace:
{"addr":"127.0.0.1","port":58758,"message":"connected fluent socket addr=\"127.0.0.1\" port=58758"}
2022-06-20 21:49:26 +0000 [trace]: #1 fluent/log.rb:287:trace: accepted fluent socket
addr="127.0.0.1" port=58758
2022-06-20 21:49:26.952046695 +0000 fluent.trace:
{"addr":"127.0.0.1","port":58758,"message":"accepted fluent socket addr=\"127.0.0.1\" port=58758"}
2022-06-20 21:49:26.951350087 +0000 mod9.lab: {"lfs242":"event"}

ubuntu@labsys:~/lab9$
```

In this case, worker #1 received and processed the incoming event, as

```
fluent.trace: {"addr":"127.0.0.1","port":58758,"message":"accepted fluent socket addr=\"127.0.0.1\" port=58758"}
```

suggests.

Workers can be selected to execute specific directives (and by extension, pipelines). This is done by declaring a `<worker N>` directive, where N is the number of the desired worker to execute the directives. This can be a single integer, or a range of integers.

Use `pkill` to terminate Fluentd:

```
ubuntu@labsys:~/lab9$ pkill fluentd

ubuntu@labsys:~/lab9$ rm -f /tmp/lab9/multiworker.out

[1]+  Done                    fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out -vv

ubuntu@labsys:~/lab9$
```

Edit the `multiworker.conf`:

```
ubuntu@labsys:~/lab9$ nano multiworker.conf && cat $_

<system>
  process_name aggregator
  workers 3
</system>

<worker 0-1>
  <source>
    @type forward
  </source>
  <match>
    @type stdout
    <format>
      @type msgpack
    </format>
  </match>
</worker>

<worker 2>
  <source>
    @type tail
    tag app
    path /tmp/lab9/app.log
    pos_file /tmp/lab9/app.log.pos
    <parse>
      @type none
    </parse>
  </source>
  <match>
    @type stdout
  </match>
</worker>

ubuntu@labsys:~/lab9$
```

The configuration above has been split: workers 0 and 1 will handle all forward traffic, while worker 2 will tail a log file. Some plugins, like

in_tail, do not support multiple workers; in order to use them in a multi-worker instance, those plugins must be configured to run on a specific worker.

Run Fluentd again using this configuration:

```
ubuntu@labsys:~/lab9$ fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out -vv &

[1] 15810

ubuntu@labsys:~/lab9$ tail /tmp/lab9/multiworker.out

2022-06-20 21:51:01 +0000 [trace]: #0 fluent/log.rb:287:trace: registered formatter plugin 'msgpack'
2022-06-20 21:51:01 +0000 [info]: #0 fluent/log.rb:330:info: adding source type="forward"
2022-06-20 21:51:01 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'forward'
2022-06-20 21:51:01 +0000 [warn]: #0 fluent/log.rb:351:warn: define <match fluent.**> to capture
fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 21:51:01 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=54347
ppid=54342 worker=0
2022-06-20 21:51:01 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24224
bind="0.0.0.0"
2022-06-20 21:51:01 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
pid K ppid F worker message 5starting fluentd worker pid=54347 ppid=54342 worker=0
port ^ bind 0.0.0.0 message (listening port port=24224 bind="0.0.0.0"
worker message &fluentd worker is now running worker=0

ubuntu@labsys:~/lab9$
```

Out of the three workers, #0 and #1 output their ready event as msgpack, while #2 reported it in the default JSON. Since each one was configured with a different set of directives and plugins, their outputs will differ greatly.

See how worker #2 handles events by printing a message into the log it is tailing:

```
ubuntu@labsys:~/lab9$ echo "LFS242 rocks!" >> /tmp/lab9/app.log

ubuntu@labsys:~/lab9$ pkill -SIGUSR1 fluentd

ubuntu@labsys:~/lab9$ tail /tmp/lab9/multiworker.out

2022-06-20 21:51:35.352946219 +0000 fluent.info: {"message":"force flushing buffered events"}
2022-06-20 21:51:35 +0000 [info]: #2 fluent/log.rb:330:info: force flushing buffered events
2022-06-20 21:51:35 +0000 [debug]: #0 fluent/log.rb:309:debug: flushing thread: flushed
message flushing thread: flushed
2022-06-20 21:51:35 +0000 [info]: #2 fluent/log.rb:330:info: flushing all buffer forcedly
2022-06-20 21:51:35.353079822 +0000 fluent.info: {"message":"flushing all buffer forcedly"}
2022-06-20 21:51:35 +0000 [debug]: #1 fluent/log.rb:309:debug: flushing thread: flushed
message flushing thread: flushed
2022-06-20 21:51:35 +0000 [debug]: #2 fluent/log.rb:309:debug: flushing thread: flushed
2022-06-20 21:51:35.354498828 +0000 fluent.debug: {"message":"flushing thread: flushed"}

ubuntu@labsys:~/lab9$
```

Events coming through a specific worker will show a number right after the log level of their message. This is how you can keep track of which worker is doing what in an environment with many workers.

This lab has taken you through several debugging exercises, introduced how Ruby Garbage collection can affect Fluentd's resource footprint, and also how to tune Fluentd to use more CPU by spawning more worker processes to handle multiple (unique or duplicate) pipelines. The next step, as with any tuning, is iteration and observation. Refer to Lab 8 to see the tools that will give you insight on how to best tune Fluentd to work within the resource requirements and constraints of your own environment .

3. Clean up

When you have finished exploring type `pkill fluentd` at the console of any Fluentd instances left running to shut them down (be patient as it will take Fluentd a moment to clean up and shutdown):

```
ubuntu@labsys:~/lab9$ pkill fluentd

ubuntu@labsys:~/lab9$ # press enter

[1]+  Done                    fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out -vv

ubuntu@labsys:~/lab9$ cd

ubuntu@labsys:~$
```

Congratulations, you have completed the Lab.