LFS242 - Cloud Native Logging with Fluentd

Lab 2 – Configuring Fluentd

Understanding configurations is an essential part of working with Fluentd. Mastering source and match configurations will provide you with the building blocks to complicated configurations. This hands-on lab will examine the process of putting together simple source and match configurations together in a Fluentd configuration file to create a logging pipeline.

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:

- Explore the ways configuration files are selected for use by Fluentd
- Create multiple `<source>` directives that allow data input from a variety of sources
- Formulate `<match>` directives that match on various patterns
- Compare the use of buffered and unbuffered plugins in `<match>` directives

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:~$
```

1. Using Configuration Files

Fluentd functions are defined by a configuration file which is loaded before it runs. A Fluentd configuration file provides the following components:

- Event processing pipelines that can contain `<source>`, `<label>`, `<filter>`, and `<match>` directives. These define the plugins that will be loaded by Fluentd.
- An optional `<system>` directive that defines process-level parameters for that instance of Fluentd, including a global log level, process names and the amount of worker processes that will be spawned.
- Any additional directives that may accompany certain parameters, like `<worker N>` if a workers parameter was set (this will be

covered in Chapter 9).

The following steps will walk through how to run Fluentd using configuration files.

1a. Default setup: Running Fluentd on its own and using a config file at /etc/fluent/fluent.conf

Fluentd instances require a configuration file before it can start. The exception to this is running with the `--help` flag to dump the help file.

Try to run Fluentd without any arguments:

```
ubuntu@labsys:~$ fluentd

Traceback (most recent call last):
        9: from /usr/local/bin/fluentd:23:in `<main>'
        8: from /usr/local/bin/fluentd:23:in `load'
        7: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/bin/fluentd:15:in `<top (required)>'
        6: from /usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
        5: from /usr/lib/ruby/2.7.0/rubygems/core_ext/kernel_require.rb:72:in `require'
        4: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/fluentd.rb:355:in `<top
(required)>'
        3: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/supervisor.rb:747:in `configure'
        2: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config.rb:31:in `build'
        1: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config.rb:31:in `open'
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/config.rb:31:in `initialize': No such file or
directory @ rb_sysopen - /etc/fluent/fluent.conf (Errno::ENOENT)

ubuntu@labsys:~$
```

As expected, this failed because there was no configuration file present to be loaded. However, it did try to look for one at `/etc/fluent/fluent.conf`. Fluentd can generate a simple configuration file with the `--setup` flag, which will install one at `/etc/fluent` by default.

Use the `--setup` flag with Fluentd. You may need `sudo` to successfully execute it without any other flags:

```
ubuntu@labsys:~$ sudo fluentd --setup

Installed /etc/fluent/fluent.conf.

ubuntu@labsys:~$
```

Without any other flags, it has created a configuration file at `/etc/fluent/fluent.conf` (a privileged directory). If you prefer to have Fluentd write the configuration file to a local directory, follow the `--setup` flag with a path.

Have Fluentd create a fluent.conf file in your user's home directory:

```
ubuntu@labsys:~$ fluentd --setup ~/lab2

Installed /home/ubuntu/lab2/fluent.conf.

ubuntu@labsys:~$
```

Since you are specifying Fluentd to write in a non-privileged path, you do not need to run it using `sudo`.

Check the contents of the generated fluent.conf file:

```
ubuntu@labsys:~$ cat lab2/fluent.conf
```

```
# In v1 configuration, type and id are @ prefix parameters.
# @type and @id are recommended. type and id are still available for backward compatibility

## built-in TCP input
## $ echo <json> | fluent-cat <tag>
<source>
  @type forward
  @id forward_input
</source>

...

ubuntu@labsys:~$
```

The fluent.conf file provides several commented examples of various `<source>`, `<filter>`, `<match>`, and `<label>` directives. If this configuration file is run, the resulting Fluentd instance will be configured to:

- Listen for traffic from other Fluentd instances on port 24224 (the default port) using in_forward
- Listen for HTTP traffic on port 8888 using in_http
- Expose various internal Fluentd metrics on a per-plugin basis using monitor_agent, which can be accessed using `curl` to hit http://localhost:24220/api/plugins
- Use the debug_agent plugin to expose port 24230, which can be accessed using the dRuby debugging agent
- Send events that have tags starting with `debug.` to STDOUT with out_stdout
- Forward events that have tags starting with `debug.` to another Fluentd instance listening on at 192.168.0.12:24224

There are also commented out examples of:

- A `<source>` directive that tells Fluentd to listen for events on a standard Unix socket
- A `<source>` directive configured to tail an Apache2 http access log, which will tag events as `apache.access`
- A `<filter>` directive that appends the hostname and tag to the records of events tagged `apache.access`
- A `<filter>` directive that allows only events with tags starting with `apache.` containing "GET" requests to be printed. Note that the syntax is actually out of date: the filter_grep plugin is configured using `<regexp>` and `<exclude>` subdirectives.
- A `<match>` directive using the out_copy plugin to send events with tags starting with `myapp.` to a Fluentd instance at 192.168.0.13 and a file at /var/log/fluent/myapp at the same time.
- A `<match>` directive that will discard all Fluentd internal events using out_null
- A `<match>` directive that sends all other events that were not caught by other directives to a compressed file at /var/log/fluent/else
- A `<label>` directive for events that were labeled @STAGING, which will send events to a Fluentd instance as 192.168.0.101. To enable this, one of the `<source>` directives will need to have the `@label @STAGING` parameter added to them.

The plugins configured in this sample configuration file are only a small subset of the base plugins installed with Fluentd!

Try to run to Fluentd now, using `sudo` without providing any other configuration file:

```
ubuntu@labsys:~$ sudo fluentd

2022-06-20 14:50:00 +0000 [info]: parsing config file is succeeded path="/etc/fluent/fluent.conf"
2022-06-20 14:50:00 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 14:50:00 +0000 [info]: [stdout_output] Oj isn't installed, fallback to Yajl as json
parser
2022-06-20 14:50:00 +0000 [info]: adding forwarding server '192.168.0.12:24224' host="192.168.0.12"
port=24224 weight=60 plugin_id="object:758"
2022-06-20 14:50:00 +0000 [info]: [forward_output] adding forwarding server '192.168.0.11:24224'
host="192.168.0.11" port=24224 weight=60 plugin_id="forward_output"
2022-06-20 14:50:00 +0000 [warn]: [http_input] LoadError
2022-06-20 14:50:01 +0000 [info]: using configuration file: <ROOT>
  <source>
```

```
    @type forward
    @id forward_input
  </source>
  <source>
    @type http
    @id http_input
    port 8888
  </source>
  <source>
    @type monitor_agent
    @id monitor_agent_input
    port 24220
  </source>
  <source>
    @type debug_agent
    @id debug_agent_input
    bind "127.0.0.1"
    port 24230
  </source>
  <match debug.**>
    @type stdout
    @id stdout_output
  </match>
  <match system.**>
    @type forward
    @id forward_output
    <server>
      host "192.168.0.11"
    </server>
    <secondary>
      <server>
        host "192.168.0.12"
      </server>
    </secondary>
  </match>
</ROOT>
2022-06-20 14:50:01 +0000 [info]: starting fluentd-1.14.6 pid=7161 ruby="2.7.0"
2022-06-20 14:50:01 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "--under-supervisor"]
2022-06-20 14:50:01 +0000 [info]: adding match pattern="debug.**" type="stdout"
2022-06-20 14:50:01 +0000 [info]: #0 [stdout_output] Oj isn't installed, fallback to Yajl as json
parser
2022-06-20 14:50:01 +0000 [info]: adding match pattern="system.**" type="forward"
2022-06-20 14:50:01 +0000 [info]: #0 adding forwarding server '192.168.0.12:24224'
host="192.168.0.12" port=24224 weight=60 plugin_id="object:758"
2022-06-20 14:50:01 +0000 [info]: #0 [forward_output] adding forwarding server '192.168.0.11:24224'
host="192.168.0.11" port=24224 weight=60 plugin_id="forward_output"
2022-06-20 14:50:01 +0000 [info]: adding source type="forward"
2022-06-20 14:50:01 +0000 [info]: adding source type="http"
2022-06-20 14:50:01 +0000 [warn]: #0 [http_input] LoadError
2022-06-20 14:50:01 +0000 [info]: adding source type="monitor_agent"
2022-06-20 14:50:01 +0000 [info]: adding source type="debug_agent"
2022-06-20 14:50:01 +0000 [info]: #0 starting fluentd worker pid=7166 ppid=7161 worker=0
2022-06-20 14:50:01 +0000 [info]: #0 [debug_agent_input] listening dRuby
uri="druby://127.0.0.1:24230" object="Fluent::Engine" worker=0
2022-06-20 14:50:01 +0000 [info]: #0 [forward_input] listening port port=24224 bind="0.0.0.0"
2022-06-20 14:50:01 +0000 [info]: #0 fluentd worker is now running worker=0
```

Since you ran `fluentd install` earlier, there is a valid configuration file that can be found in `/etc/fluent`.

Use `CTRL C` to terminate this Fluentd session and release your terminal.

```
^C

C2022-06-20 14:50:23 +0000 [info]: Received graceful stop
2022-06-20 14:50:24 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 14:50:24 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 14:50:24 +0000 [info]: #0 shutting down input plugin type=:debug_agent
plugin_id="debug_agent_input"
2022-06-20 14:50:24 +0000 [info]: #0 shutting down input plugin type=:forward
plugin_id="forward_input"
2022-06-20 14:50:24 +0000 [info]: #0 shutting down input plugin type=:monitor_agent
plugin_id="monitor_agent_input"
2022-06-20 14:50:24 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="http_input"
2022-06-20 14:50:24 +0000 [info]: #0 shutting down output plugin type=:stdout
plugin_id="stdout_output"
2022-06-20 14:50:24 +0000 [info]: #0 shutting down output plugin type=:forward
plugin_id="forward_output"
2022-06-20 14:50:29 +0000 [warn]: #0 [forward_output] event loop does NOT exit until hard timeout.
2022-06-20 14:50:34 +0000 [warn]: #0 event loop does NOT exit until hard timeout.
2022-06-20 14:50:36 +0000 [warn]: #0 killing existing thread thread=#
<Thread:0x00005576fbc95028@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 sleep>
2022-06-20 14:50:36 +0000 [warn]: #0 thread doesn't exit correctly (killed or other reason)
plugin=Fluent::Plugin::ForwardOutput title=:event_loop thread=#<Thread:0x00005576fbc95028@event_loop
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/thread.rb:70 aborting> error=nil
2022-06-20 14:50:36 +0000 [warn]: #0 [forward_output] killing existing thread thread=#
<Thread:0x00005576fbc8f330@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 sleep>
2022-06-20 14:50:36 +0000 [warn]: #0 [forward_output] thread doesn't exit correctly (killed or other
reason) plugin=Fluent::Plugin::ForwardOutput title=:event_loop thread=#
<Thread:0x00005576fbc8f330@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 aborting> error=nil
2022-06-20 14:50:36 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~$
```

As long as a valid Fluentd configuration file is present under `/etc/fluent/`, Fluentd can be started without any other command line arguments. This can be useful for expediting the Fluentd deployment process, but it does require `sudo` or privileged access to the machine.

1b. Using command line argument --config

There may be times where a Fluentd instance will need to have a configuration selected on an ad-hoc basis, or it needs to be run under a non-privileged user for security reasons. This can be done by using the `--config` flag, which allows the user to specify a path to a configuration file.

Create a configuration file with `touch`:

```
ubuntu@labsys:~$ cd ~/lab2

ubuntu@labsys:~/lab2$ touch ~/lab2/lab2.conf

ubuntu@labsys:~/lab2$
```

By using `touch` to create `lab2.conf`, the configuration file has no directives inside it. With an empty placeholder configuration file, the resulting Fluentd instance would not do any event processing, if it starts at all.

Use the newly created `lab2.conf` with Fluentd with the `--config` flag:

```
ubuntu@labsys:~/lab2$ fluentd --config lab2.conf

2022-06-20 14:51:28 +0000 [info]: parsing config file is succeeded path="lab2.conf"
2022-06-20 14:51:28 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 14:51:28 +0000 [info]: using configuration file: <ROOT>
</ROOT>
2022-06-20 14:51:28 +0000 [info]: starting fluentd-1.14.6 pid=7204 ruby="2.7.0"
2022-06-20 14:51:28 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "--config", "lab2.conf", "--under-supervisor"]
2022-06-20 14:51:28 +0000 [info]: #0 starting fluentd worker pid=7209 ppid=7204 worker=0
2022-06-20 14:51:28 +0000 [info]: #0 fluentd worker is now running worker=0
```

It worked! Using the terminal output from that invocation, answer the following questions:

- Were any plugins loaded?
- Can Fluentd perform any functions at this point?
- What would the benefit of running a blank config file be?
- What circumstances would it be appropriate to run Fluentd with a specific configuration file like this be?

Use `CTRL C` to terminate this Fluentd session and release your terminal:

```
^C

2022-06-20 14:51:40 +0000 [info]: Received graceful stop
2022-06-20 14:51:41 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 14:51:41 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 14:51:41 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab2$
```

Being able to select configuration files gives the user the ability to diversify Fluentd's roles in their environment. There is one more way to specify what configuration file to use for Fluentd.

1c. Using FLUENT_CONF environment variable

The last method of running a Fluentd configuration file is by using the FLUENT_CONF environment variable. This method allows the user to select a specific configuration file in a deployment where a terminal might not be available, like a container.

Before proceeding, look at the `fluentd --help` output for the `--config` flag.

```
ubuntu@labsys:~/lab2$ fluentd --help | grep -- "--config"

    -c, --config PATH                config file path (default: /etc/fluent/fluent.conf)

ubuntu@labsys:~/lab2$
```

In this Ruby Gem installed instance, the `--config` path is defaulting to `/etc/fluent/fluent.conf`. In the absence of a `--config` flag, the `fluentd` command will search for a configuration file located at `/etc/fluent/fluent.conf`.

You can store a path to another configuration file in the FLUENT_CONF environment variable. td-agent, the packaged version of Fluentd, uses this method to identify which configuration file to load. As mentioned earlier, this method is also ideal for starting Fluentd containers implicitly.

Check the FLUENT_CONF variable on your terminal session:

```
ubuntu@labsys:~/lab2$ echo $FLUENT_CONF

ubuntu@labsys:~/lab2$
```

No output. The default configuration file path is set internally with the Ruby Gem installation. So, we need to define the `FLUENT_CONF` variable with the `lab2.conf` placeholder file created earlier:

```
ubuntu@labsys:~/lab2$ export FLUENT_CONF=~/lab2/lab2.conf

ubuntu@labsys:~/lab2$ echo $FLUENT_CONF

/home/ubuntu/lab2/lab2.conf

ubuntu@labsys:~/lab2$
```

FLUENT_CONF is now defined for this shell session. Remember that environment variables in Linux are unique between shell sessions, so the FLUENT_CONF environment variable needs to be set for each new one.

Check the `fluentd` command's help now:

```
ubuntu@labsys:~/lab2$ fluentd --help | grep -- "--config"

    -c, --config PATH                config file path (default: /home/ubuntu/lab2/lab2.conf)

ubuntu@labsys:~/lab2$
```

With the environment variable set, the Fluentd instance in this session now recognizes that /home/ubuntu/lab2/lab2.conf is the default configuration file, and will attempt to load it whenever the `fluentd` command is run.

Notice in all cases that the Fluentd configuration file is referred to by a full, absolute path. Fluentd does not natively support Linux variables in its configuration files, so files must be referred to by a full path whenever they need to be referenced in Fluentd.

Try running Fluentd without any flags:

```
ubuntu@labsys:~/lab2$ fluentd

2022-06-20 14:52:46 +0000 [info]: parsing config file is succeeded
path="/home/ubuntu/lab2/lab2.conf"
2022-06-20 14:52:46 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 14:52:46 +0000 [info]: using configuration file: <ROOT>
</ROOT>
2022-06-20 14:52:46 +0000 [info]: starting fluentd-1.14.6 pid=7217 ruby="2.7.0"
2022-06-20 14:52:46 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "--under-supervisor"]
2022-06-20 14:52:47 +0000 [info]: #0 starting fluentd worker pid=7222 ppid=7217 worker=0
2022-06-20 14:52:47 +0000 [info]: #0 fluentd worker is now running worker=0
```

Success!

- Where would using an environment variable to set the configuration file be most appropriate?
- Would it be possible to run more than one Fluentd instance on the same machine using more than one terminal?

Use `CTRL C` to terminate this Fluentd session and release your terminal:

```
^C

2022-06-20 14:52:57 +0000 [info]: Received graceful stop
2022-06-20 14:52:58 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 14:52:58 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 14:52:58 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab2$
```

2. Creating `<source>` directives

`<source>` directives determine the various inputs that Fluentd will listen on. The job of the `<source>` directive, and the plugins it uses, is to accept data from the configured applications and methods listed in each directive.

2a. Forward

The first directive that will be set is a `forward`. This will cause Fluentd to listen to a TCP socket at the specified port. The most common use for this type of `<source>` directive is to allow a central log aggregator instance of Fluentd to receive traffic from other Fluentd instances.

Add a simple `<source>` directive using the forward plugin to lab2.conf:

```
ubuntu@labsys:~/lab2$ nano lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

ubuntu@labsys:~/lab2$
```

Configuring the in_forward plugin is fairly simple: declare a port on which the Fluentd instance will listen to. Any other Fluentd instances that need to communicate with this one must be configured to send traffic to the declared port, which in this example is port 31604.

Start Fluentd:

```
ubuntu@labsys:~/lab2$ fluentd

2022-06-20 14:53:30 +0000 [info]: parsing config file is succeeded
path="/home/ubuntu/lab2/lab2.conf"
2022-06-20 14:53:30 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 14:53:30 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
    port 31604
  </source>
</ROOT>
2022-06-20 14:53:30 +0000 [info]: starting fluentd-1.14.6 pid=7227 ruby="2.7.0"
2022-06-20 14:53:30 +0000 [info]: spawn command to main:  cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "--under-supervisor"]
2022-06-20 14:53:31 +0000 [info]: adding source type="forward"
2022-06-20 14:53:31 +0000 [info]: #0 starting fluentd worker pid=7232 ppid=7227 worker=0
2022-06-20 14:53:31 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
2022-06-20 14:53:31 +0000 [info]: #0 fluentd worker is now running worker=0
```

Once started, Fluentd begins listening on all addresses at port 31604. To test this configuration, `fluent-cat` will allow a user to manually assign a tag and send a message to an open Fluentd TCP socket, like the one opened by the in_forward plugin.

Open another terminal, then use `fluent-cat` to send a message to the Fluentd forward socket:

```
ubuntu@labsys:~$ echo '{"lfs242":"hello"}'| fluent-cat mod2.lab -p 31604 --json

ubuntu@labsys:~$
```

In the original Fluentd terminal (where Fluentd is currently running), you should observe the following event after sending the message via fluent-cat:

```
...

2022-06-20 14:53:31 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 14:53:59 +0000 [warn]: #0 no patterns matched tag="mod2.lab"
```

The warning printed at the end means that the `<source>` directive is functional, but cannot route the traffic anywhere because there are no destinations configured. For the purposes of this step, that is okay. `<match>` directives will be added to this configuration in the following steps.

2b. Adding another source: HTTP

Multiple `<source>` directives can be defined inside a Fluentd configuration. This allows very granular control over input streams, as well as the ability to diversify input streams by specifying different protocols and methods to listen on.

Append the `lab2.conf` file with a new `<source>` directive:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

ubuntu@labsys:~$
```

This `<source>` directive will use the `in_http` plugin, which will turn the Fluentd instance into an HTTP endpoint listening for RESTFUL on the specified port. Setting up a source like this is good for capturing web-based events.

By enclosing different `<source>` directives between their own `<source>` tags, more than one input source can be listed inside a single Fluentd configuration file.

In order to reload the configuration, send a `SIGUSR2` to the running Fluentd instance to gracefully restart the Fluentd process.

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

ubuntu@labsys:~$
```

In the Fluentd terminal, the restart event can be observed:

```
2022-06-20 14:55:57 +0000 [info]: Reloading new config
2022-06-20 14:55:57 +0000 [warn]: LoadError
2022-06-20 14:55:57 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
    port 31604
  </source>
  <source>
    @type http
    port 32767
  </source>
</ROOT>
2022-06-20 14:55:57 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:730"
2022-06-20 14:55:57 +0000 [info]: adding source type="forward"
2022-06-20 14:55:57 +0000 [info]: adding source type="http"
2022-06-20 14:55:57 +0000 [warn]: #0 LoadError
2022-06-20 14:55:57 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 14:55:57 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:730"
2022-06-20 14:55:57 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 14:55:57 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

Take a close look at the output after the restart:

- What happens during a restart event?
- How many sources are loaded now?

Fluentd should now be configured to listen for http requests on port 32767.

In the working terminal, try to use curl to post a message to it:

```
ubuntu@labsys:~$ curl -X POST -d 'json={"lfs242":"hello from http"}'
http://localhost:32767/mod2.http

ubuntu@labsys:~$
```

An warning should be generated in the Fluentd terminal's output:

```
2022-06-20 14:56:55 +0000 [warn]: #0 no patterns matched tag="mod2.http"
```

Once again, a `no patterns matched` warning is generated, as there are still no `<match>` directives configured. However, the tag that is presented shows that Fluentd was able to capture the request sent by the curl command.

Each `<source>` directive's plugin has its own way of dealing with incoming data traffic; not all plugins can handle all types of traffic.

Try to use fluent-cat against port 32767, sending the same message as the curl request:

```
ubuntu@labsys:~$ echo '{"lfs242":"hello from fluent-cat"}' | fluent-cat cat -p 32767

ubuntu@labsys:~$
```

The request was sent and the terminal did not hang. **Check the output of the Fluentd terminal**:

```
2022-06-20 14:57:07 +0000 [warn]: #0 unexpected error error="Could not parse data entirely (0 !=
36)"
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin/in_http.rb:407:in `<<'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin/in_http.rb:407:in `on_read'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin/in_http.rb:296:in `block in on_server_connect'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/server.rb:630:in `on_read_without_connection'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/cool.io-
1.7.1/lib/cool.io/io.rb:123:in `on_readable'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/cool.io-
1.7.1/lib/cool.io/io.rb:186:in `on_readable'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/cool.io-
1.7.1/lib/cool.io/loop.rb:88:in `run_once'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/cool.io-
1.7.1/lib/cool.io/loop.rb:88:in `run'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/event_loop.rb:93:in `block in start'
  2022-06-20 14:57:07 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:78:in `block in thread_create'
```

Fluentd was able to capture the event, and successfully received it. However, when it came time to create an event for processing, the http plugin failed to parse that input and generated an error. The point of configuring multiple sources is to ensure that Fluentd can process data streams from an expected target, and nothing else.

2c. Tailing Files: Further `<source>` directive Configuration

So far, this Fluentd instance is configured to take inputs from TCP sockets on 31604 and REST requests on 32767. To complete the scope for this instance, one more `<source>` directive should be added to track logs stored inside a path.

Prepare a directory that is commonly writable, such as `tmp`:

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

ubuntu@labsys:~$ touch /tmp/fluent/applogs
```

To ensure that Fluentd is able to recognize files within that directory, a dummy file has been placed inside the directory.

Since the intent for this `<source>` directive is the track the contents of log files under this directory, the in_tail plugin can be used here.

Add the following syntax to the lab2.conf:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*
```

```
</source>

ubuntu@labsys:~$
```

The new `<source>` directive will invoke the in_tail plugin, and have it look in the /tmp/fluent directory.

- Based on the above configuration, what files will be tailed under the /tmp/fluent directory?
- What does the tail plugin tell you about the way this plugin will work?

Send a `SIGUSR2` to the running Fluentd instance to load the new configuration:

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

ubuntu@labsys:~$
```

Fluentd should restart with the new configuration.

```
2022-06-20 15:03:12 +0000 [info]: Reloading new config
2022-06-20 15:03:13 +0000 [warn]: LoadError
2022-06-20 15:03:13 +0000 [error]: Failed to reload config file: <parse> section is required.
```

The configuration failed, and the Fluentd instance did not reload (but it is still running)

- Before it failed to reload, how many plugins were loaded after the SIGUSR2 was sent?

What went wrong? According to the output, the `in_tail` plugin requires an additional directive before it can function. In this case, a `<parse>` subdirective is required but there are no defaults set for the in_tail plugin.

A `<parse>` subdirective's primary function is to allow a configured input plugin to turn incoming data into key-value pairs carried inside an event's record. in_tail does not have a default parse plugin configured, so one must be provided.

Revise the configuration file according to the feedback from Fluentd:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*
  <parse>
    @type none
  </parse>
</source>

ubuntu@labsys:~/lab2$
```

The tail `<source>` directive is now using the parse_none plugin, which will take all incoming data and pair it with a user-defined key ("message" by default) inside an event’s record.

Send a `SIGUSR2` to the running Fluentd instance to load the new configuration:

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

ubuntu@labsys:~$
```

In the terminal that Fluentd failed to start in, check if Fluentd reloaded:

```
...

2022-06-20 15:04:36 +0000 [info]: Reloading new config
2022-06-20 15:04:36 +0000 [warn]: LoadError
2022-06-20 15:04:36 +0000 [error]: config error in:
<source>
  @type tail
  path "/tmp/fluent/*"
  <parse>
    @type none
    unmatched_lines
  </parse>
</source>

2022-06-20 15:04:36 +0000 [error]: Failed to reload config file: 'tag' parameter is required
```

It still didn’t work; this time a `tag` parameter is required, according to Fluentd.

- What is different about this crash compared to the previous one?
- Were any plugins loaded before this error was generated?

The `tag` identifies an event inside the Fluentd routing engine, allowing it to be processed by `<filter>` or `<match>` directives. The in_tail plugin does not provide a default tag, so one must be specified. Without one, this `<source>` directive cannot create any events for incoming data.

Add a `tag` parameter to the tail `<source>` directive:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*
  <parse>
    @type none
  </parse>
  tag local.logs
</source>
```

```
ubuntu@labsys:~/lab2$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab2$
```

All events produced with data collected from files under /tmp/fluent will now bear the tag `local.logs`.

Rerun Fluentd now:

```
2022-06-20 15:05:24 +0000 [info]: Reloading new config
2022-06-20 15:05:24 +0000 [warn]: LoadError
2022-06-20 15:05:24 +0000 [warn]: 'pos_file PATH' parameter is not set to a 'tail' source.
2022-06-20 15:05:24 +0000 [warn]: this parameter is highly recommended to save the position to
resume tailing.
2022-06-20 15:05:24 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
    port 31604
  </source>
  <source>
    @type http
    port 32767
  </source>
  <source>
    @type tail
    path "/tmp/fluent/*"
    tag "local.logs"
    <parse>
      @type "none"
      unmatched_lines
    </parse>
  </source>
</ROOT>
2022-06-20 15:05:24 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 15:05:24 +0000 [info]: shutting down input plugin type=:http plugin_id="object:76c"
2022-06-20 15:05:24 +0000 [info]: adding source type="forward"
2022-06-20 15:05:24 +0000 [info]: adding source type="http"
2022-06-20 15:05:24 +0000 [warn]: #0 LoadError
2022-06-20 15:05:24 +0000 [info]: adding source type="tail"
2022-06-20 15:05:24 +0000 [warn]: #0 'pos_file PATH' parameter is not set to a 'tail' source.
2022-06-20 15:05:24 +0000 [warn]: #0 this parameter is highly recommended to save the position to
resume tailing.
2022-06-20 15:05:24 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:05:24 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:7a8"
2022-06-20 15:05:24 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:7bc"
2022-06-20 15:05:24 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:05:24 +0000 [info]: #0 following tail of /tmp/fluent/applogs
2022-06-20 15:05:24 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

Now it's running! Though it looks like Fluentd is recommending another parameter: pos_file, which allow Fluentd to resume reading a file should it go down.

In another terminal, add the pos_file parameter to the in_tail `<source>` directive:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
```

```
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

ubuntu@labsys:~$
```

After adding this parameter, send a `SIGUSR2` to Fluentd:

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

ubuntu@labsys:~$
```

Fluentd should report:

```
...

2022-06-20 15:06:30 +0000 [info]: Reloading new config
2022-06-20 15:06:30 +0000 [warn]: LoadError
2022-06-20 15:06:30 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type forward
    port 31604
  </source>
  <source>
    @type http
    port 32767
  </source>
  <source>
    @type tail
    path "/tmp/fluent/*"
    tag "local.logs"
    pos_file "/tmp/fluent/lab2.tail.pos"
    <parse>
      @type "none"
      unmatched_lines
    </parse>
  </source>
</ROOT>
2022-06-20 15:06:30 +0000 [info]: shutting down input plugin type=:tail plugin_id="object:7f8"
2022-06-20 15:06:30 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:7d0"
2022-06-20 15:06:30 +0000 [info]: shutting down input plugin type=:http plugin_id="object:7e4"
2022-06-20 15:06:30 +0000 [info]: adding source type="forward"
2022-06-20 15:06:30 +0000 [info]: adding source type="http"
2022-06-20 15:06:30 +0000 [warn]: #0 LoadError
2022-06-20 15:06:30 +0000 [info]: adding source type="tail"
2022-06-20 15:06:30 +0000 [info]: #0 shutting down fluentd worker worker=0
```

```
2022-06-20 15:06:30 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:974"
2022-06-20 15:06:30 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:99c"
2022-06-20 15:06:30 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:988"
2022-06-20 15:06:31 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:06:31 +0000 [info]: #0 following tail of /tmp/fluent/lab2.tail.pos
2022-06-20 15:06:31 +0000 [info]: #0 following tail of /tmp/fluent/applogs
2022-06-20 15:06:31 +0000 [warn]: #0 no patterns matched tag="local.logs"
2022-06-20 15:06:31 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

- Did you get any indication that any files are being tracked under the /tmp/fluent/ directory?

Now that it's up and running, it's time to test whether the in_tail plugin can parse the files under this directory.

Append an encouraging statement to the end of /tmp/fluent/applogs:

```
ubuntu@labsys:~$ echo 'Initializing' >> /tmp/fluent/applogs

ubuntu@labsys:~$ echo 'Logger setup successful!' >> /tmp/fluent/applogs

ubuntu@labsys:~$
```

And check the Fluentd terminal:

```
...

2022-06-20 15:07:01 +0000 [warn]: #0 no patterns matched tag="local.logs"
2022-06-20 15:07:06 +0000 [warn]: #0 no patterns matched tag="local.logs"
```

This tail setup is very permissive, reading all files under the directory due to the wildcard in the `path` argument in the directive.

Try to create another file inside that directory:

```
ubuntu@labsys:~$ touch /tmp/fluent/error.log

ubuntu@labsys:~$ echo "Fluentd-tailed Error log" > /tmp/fluent/error.log

ubuntu@labsys:~$
```

In fact, this setup may be *too* permissive: Look back at the Fluentd startup messages, and you will see it is tracking the .pos file!

```
...

2022-06-20 15:08:31 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:08:31 +0000 [warn]: #0 no patterns matched tag="local.logs"

...
```

After a short pause, it is clear that this `<source>` directive is too permissive in its current state. Think back to the original intent of this `<source>` directive: to track *log* files stored in this directory. Being clear about the purpose of `<source>` directives is key to preparing good Fluentd configurations.

Reconfigure Fluentd to only look for valid log files, which in this case will be any files ending in `.log`

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

ubuntu@labsys:~$
```

By appending `*.log` to the path, Fluentd will now specifically look for all files with the `.log` extension.

Sending a `SIGUSR2` to reconfigure the Fluentd instance:

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

ubuntu@labsys:~$
```

```
...

2022-06-20 15:09:24 +0000 [info]: shutting down input plugin type=:tail plugin_id="object:834"
2022-06-20 15:09:24 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:80c"
2022-06-20 15:09:24 +0000 [info]: shutting down input plugin type=:http plugin_id="object:820"
2022-06-20 15:09:24 +0000 [info]: adding source type="forward"
2022-06-20 15:09:24 +0000 [info]: adding source type="http"
2022-06-20 15:09:24 +0000 [warn]: #0 LoadError
2022-06-20 15:09:24 +0000 [info]: adding source type="tail"
2022-06-20 15:09:24 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:09:24 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:ab4"
2022-06-20 15:09:24 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:adc"
2022-06-20 15:09:24 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:ac8"
2022-06-20 15:09:24 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:09:24 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:09:24 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

Excellent, Fluentd is now only tailing the `error.log` created earlier in this step.

In your working terminal, send the following echo commands to append data the tracked log files:

```
ubuntu@labsys:~$ echo 'Should not be tracked' >> /tmp/fluent/applogs

ubuntu@labsys:~$ echo 'An error has occurred: Operation completed successfully' >>
/tmp/fluent/error.log
```

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

Only one event should have been logged after the second echo command going to the `error.log`.

And check the Fluentd terminal:

```
2022-06-20 15:10:13 +0000 [warn]: #0 no patterns matched tag="local.logs"
```

When working with `<source>` directives, much of the configuration effort will go into satisfying the requirements of the intended plugin. Both core and third party plugins will have their own specific configurations, so be sure to refer to the Fluentd and plugin-specific documentation for any required arguments. `<source>` directives should be written to maximize the value of the data from the given source.

Using the terminal output from previous runs, try to answer the following questions:

- What plugins are loaded when this Fluentd configuration file is used?
- Are these plugins loaded in any particular order?
- Is there any benefit to being able to run Fluentd configurations with only `<source>` directives?

After following these instructions, you should now have a multi-source Fluentd configuration. It's time to address the `no patterns matched` warnings that each of these `<source>` directives have been reporting.

Keep the current Fluentd instance running for the next step.

3. Configuring outputs with `<match>` directives

Output configurations are the end of a data processing pipeline in Fluentd. They take events created by the input plugins and send them to their intended destinations. `<match>` directives can take effect immediately after an event is ingested into Fluentd. They can also be placed after a `<filter>` directive to allow any intermediate processing to occur.

3a. Setting `<match>` directives

The `<match>` directive configures an output plugin that will send data to an intended destination, which can be an external datastore, a file, or back into Fluentd for further processing. The @type parameter within a `<match>` directive will select plugins with the `out_` prefix.

The formatting of the `<match>` directive is different in that the opening tag can include a pattern. This pattern determines what event tags will be processed by that `<match>` directive. If no pattern is specified, all events will be processed by that `<match>` directive.

Append a new `<match>` directive to the end of `lab2.conf`:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

<match>
  @type stdout
</match>

ubuntu@labsys:~$
```

The first `<match>` directive for this Fluentd instance should now be in place! This `<match>` directive will process all events bearing any tag.

Based on the above `<match>` directive, answer these questions:

- What plugin will be called?
- From the plugin that's been selected, what can you infer will happen to matching events?

In your working terminal, reconfigure Fluentd by sending a SIGUSR2 to its process:

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

ubuntu@labsys:~$
```

Check the Fluentd terminal for the restart:

```
...

2022-06-20 15:11:47 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:848"
2022-06-20 15:11:47 +0000 [info]: shutting down input plugin type=:http plugin_id="object:85c"
2022-06-20 15:11:47 +0000 [info]: shutting down input plugin type=:tail plugin_id="object:870"
2022-06-20 15:11:47 +0000 [info]: adding match pattern="**" type="stdout"
2022-06-20 15:11:47 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 15:11:47 +0000 [info]: adding source type="forward"
2022-06-20 15:11:47 +0000 [info]: adding source type="http"
2022-06-20 15:11:47 +0000 [warn]: #0 LoadError
2022-06-20 15:11:47 +0000 [info]: adding source type="tail"
2022-06-20 15:11:47 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:11:47 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:c44"
2022-06-20 15:11:47 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:c6c"
2022-06-20 15:11:47 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:c58"
2022-06-20 15:11:47 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:11:47 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 15:11:47 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:11:47 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

All events, including internal Fluentd log events (noted by the deprecation notice in the output), are now being processed by the `<match>` directive and sent to STDOUT.

> N.B. The

> `[warn]: #0 define <match fluent.**> to capture Fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead`
> warning is an indication that Fluentd is trying to send internal events to the output. This method is now deprecated, and now users must specifically declare a label to capture Fluentd internal events.

In your working terminal, test your `<source>` directives by sending `fluent-cat`, `curl`, and `echo` to each of them:

```
ubuntu@labsys:~$ echo '{"lfs242":"hello"}'| fluent-cat mod2.lab -p 31604 --json

ubuntu@labsys:~$ curl -X POST -d 'json={"lfs242":"hello from http"}'
http://localhost:32767/mod2.http

ubuntu@labsys:~$ echo 'An error has occurred: Done' >> /tmp/fluent/error.log

ubuntu@labsys:~$
```

Since time was taken to properly configure each of the sources, a corresponding event should have been logged to STDOUT.

Check the Fluentd terminal:

```
...

2022-06-20 15:11:47 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:11:47 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
2022-06-20 15:12:35.498560681 +0000 mod2.lab: {"lfs242":"hello"}
2022-06-20 15:12:38.509906200 +0000 mod2.http: {"lfs242":"hello from http"}
2022-06-20 15:12:41.754525102 +0000 local.logs: {"message":"An error has occurred: Done"}
```

Sending events to STDOUT is useful for displaying the results of a logging pipeline immediately, which makes it a good configuration development and debugging tool. Still, it might not be appropriate for this `<match>` directive to process all events, so it's time to narrow its scope.

3b. Working with patterns

In the previous step, you configured a `<match>` directive that would capture all events bearing any tag. This behavior can be controlled with the use of a pattern defined inside the opening `<match>` tag of a `<match>` directive. Patterns are the most important part of a `<match>` directive. Understanding how to create effective patterns will ensure that Fluentd will always be able to capture the data that's valuable to you.

Patterns match on event tags, which are generated when an event is produced by an input plugin. By default, these event tags are not visible inside an event, but they can be made available using an `<inject>` subdirective that some plugins, including out_stdout, can use.

In your working terminal, modify the stdout `<match>` directive so that it injects the event tag to an event's record:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
```

```
</source>

<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

<match **>
  @type stdout
  <inject>
    tag_key event_tag
  </inject>
</match>

ubuntu@labsys:~$
```

This will make the `<match>` directive show the internal tag that was assigned to an event after it was received by the Fluentd data processing pipeline. The methods used in this lab so far have allowed the user to determine the tag, so this output is more useful when setting up `<match>` directive patterns for application-generated tags.

In your working terminal, send a SIGUSR2 and reconfigure Fluentd:

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

ubuntu@labsys:~$
```

```
...

2022-06-20 15:13:47 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:13:47 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 15:13:47 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:13:47 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

After the reconfiguration, answer the following questions:

- How does the new event abide by the new configurations?

The Fluentd documentation provides the following guidelines for setting tag formats:

> Fluentd accepts all non-period characters as a part of a tag. However, since the tag is sometimes used in a different context by output destinations (e.g., table name, database name, key name, etc.), it is strongly recommended that you stick to the lower-case alphabets, digits and underscore, e.g., ^[a-z0-9_]+$.
>
> - https://docs.fluentd.org/v1.0/articles/config-file

This tag format suggestion can be carried into creating patterns for `<match>` directives. Additionally, there are a handful of wildcards that can be used to generalize or narrow the scope of a `<match>` directive.

- `*` matches a single portion of a tag

- `**` matches zero to many portions of a tag (this is the default if no pattern is specified)
- `{}` can be used to specify a series of tags
  - `*` and `**` wildcards can be used inside these brackets
- Multiple patterns can be used inside a `<match>` directive's opening tag, and Fluentd will try to match any one of them.

In your working terminal, modify the STDOUT `<match>` directive to only capture events with tags starting with `mod2.`:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>

<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

<match mod2.*>
  @type stdout
  <inject>
    tag_key event_tag
  </inject>
</match>

ubuntu@labsys:~$
```

Reconfigure Fluentd with a SIGUSR2 **from your working terminal**

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

ubuntu@labsys:~$
```

And check the Fluentd terminal:

```
...

2022-06-20 15:14:56 +0000 [info]: adding match pattern="mod2.*" type="stdout"
2022-06-20 15:14:56 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 15:14:57 +0000 [info]: adding source type="forward"
2022-06-20 15:14:57 +0000 [info]: adding source type="http"
2022-06-20 15:14:57 +0000 [warn]: #0 LoadError
2022-06-20 15:14:57 +0000 [info]: adding source type="tail"
2022-06-20 15:14:57 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:14:57 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:f64"
2022-06-20 15:14:57 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:f78"
```

```
2022-06-20 15:14:57 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:f8c"
2022-06-20 15:14:57 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:f3c"
2022-06-20 15:14:57 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:14:57 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:14:57 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

The warning about events tagged with `fluent.info` is no longer being generated, which is a good sign that the new pattern is working. You can also see that Fluentd acknowledged the `mod2.*` pattern to the STDOUT `<match>` directive.

In your working terminal, send the three test commands from earlier steps to see the effect:

```
ubuntu@labsys:~$ echo '{"lfs242":"hello"}'| fluent-cat mod2.lab -p 31604 --json

ubuntu@labsys:~$ curl -X POST -d 'json={"lfs242":"hello from http"}'
http://localhost:32767/mod2.http

ubuntu@labsys:~$ echo 'An error has occurred: Done' >> /tmp/fluent/error.log

ubuntu@labsys:~$
```

And check the Fluentd terminal to see what events were processed:

```
...

2022-06-20 15:15:31.941939362 +0000 mod2.lab: {"lfs242":"hello","event_tag":"mod2.lab"}
2022-06-20 15:15:34.732836358 +0000 mod2.http: {"lfs242":"hello from http","event_tag":"mod2.http"}
2022-06-20 15:15:37 +0000 [warn]: #0 no patterns matched tag="local.logs"
```

Only two events from the `fluent-cat` and `curl` command were matched, as there were the only tags matching `mod2.*`.

- Try to use a combination of the other wildcard types in another `<match>` directive
- How can you change the in_tail `<source>` directive to work with the <match mod2.*> directive?

3c. Buffered plugins: Writing out to files with out_file

The `out_stdout` plugin used in the previous steps is an example of an unbuffered output plugin, which simply forwards the resulting event and record without enqueueing it into a buffer. Most production oriented tasks will want to use buffered plugins, which send processed events to a queue that will wait for a configured size or time limit before sending the aggregated grouping of results, or chunk, to its destination.

In your working terminal, append a new `<match>` directive to `lab2.conf` using out_file, a buffered plugin:

```
ubuntu@labsys:~$ nano ~/lab2/lab2.conf && cat $_

<source>
  @type forward
  port 31604
</source>

<source>
  @type http
  port 32767
</source>
```

```
<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag local.logs
  pos_file /tmp/fluent/lab2.tail.pos
</source>

<match mod2.*>
  @type stdout
  <inject>
    tag_key event_tag
  </inject>
</match>

<match **>
  @type file
  path /tmp/fluent/aggregated-logs
</match>

ubuntu@labsys:~$
```

This `<match>` directive will write out processed events to a file found under the `/tmp/fluent/aggregated-logs` directory, and capture any other events that do not have `mod2.` in their tag.

In your working terminal, send a SIGUSR2 to reconfigure Fluentd:

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

ubuntu@labsys:~$
```

```
...

2022-06-20 15:17:00 +0000 [info]: adding match pattern="**" type="file"
2022-06-20 15:17:00 +0000 [info]: adding source type="forward"
2022-06-20 15:17:00 +0000 [info]: adding source type="http"
2022-06-20 15:17:00 +0000 [warn]: #0 LoadError
2022-06-20 15:17:00 +0000 [info]: adding source type="tail"
2022-06-20 15:17:00 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:17:00 +0000 [info]: #0 shutting down input plugin type=:forward
plugin_id="object:10cc"
2022-06-20 15:17:00 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:10e0"
2022-06-20 15:17:00 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:10f4"
2022-06-20 15:17:00 +0000 [info]: #0 shutting down output plugin type=:stdout
plugin_id="object:10a4"
2022-06-20 15:17:01 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 15:17:01 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 15:17:01 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:17:01 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

A new `<match>` directive has been detected, using the `file` type.

In your working terminal, check if `/tmp/fluent/aggregated-logs` is now present in the /tmp/fluent folder:

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

total 16
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 15:17 aggregated-logs
-rw-rw-r-- 1 ubuntu ubuntu   60 Jun 20 15:10 applogs
-rw-rw-r-- 1 ubuntu ubuntu  137 Jun 20 15:15 error.log
-rw-r--r-- 1 ubuntu ubuntu   56 Jun 20 15:17 lab2.tail.pos

ubuntu@labsys:~$
```

Now check the files under the aggregated-logs directory:

```
ubuntu@labsys:~$ ls -l /tmp/fluent/aggregated-logs/

total 0

ubuntu@labsys:~$
```

Files in the `/aggregated-logs/` directory you created are the out_file plugin's buffer files: temporary destinations for events before they are written to an actual log file. Any events sent to these buffer files will remain there until Fluentd flushes the buffer. The `<match>` directive was open, matching all patterns and tags, so `fluent.info` tagged events will also be caught and output to the `buffer.\*.log` file

Send Fluentd more events using the test commands from earlier, except be sure to modify the tag to `file.mod2.*`:

```
ubuntu@labsys:~$ echo '{"lfs242":"hello"}'| fluent-cat file.mod2.lab -p 31604 --json

ubuntu@labsys:~$ curl -X POST -d 'json={"lfs242":"hello from http"}'
http://localhost:32767/file.mod2.http

ubuntu@labsys:~$ echo 'An error has occurred: Done' >> /tmp/fluent/error.log

ubuntu@labsys:~$
```

In the Fluentd terminal, check for events:

```
2022-06-20 15:17:01 +0000 [info]: #0 following tail of /tmp/fluent/error.log
2022-06-20 15:17:01 +0000 [info]: #0 listening port port=31604 bind="0.0.0.0"
```

Since you passed different tags with each of the test events, there should be no events sent to STDOUT. The previous `<match>` directive is still operational, however, so any events whose tag matches `mod2.*` will be output to STDOUT.

Now cat the aggregated-log file:

```
ubuntu@labsys:~$ ls -l /tmp/fluent/aggregated-logs/

total 8
-rw-r--r-- 1 ubuntu ubuntu 208 Jun 20 15:18 buffer.b5e1e29d1d104a693df1a35613a3184ab.log
-rw-r--r-- 1 ubuntu ubuntu  79 Jun 20 15:18 buffer.b5e1e29d1d104a693df1a35613a3184ab.log.meta

ubuntu@labsys:~$ cat /tmp/fluent/aggregated-logs/buffer.*.log

2022-06-20T15:18:18+00:00       file.mod2.lab   {"lfs242":"hello"}
2022-06-20T15:18:21+00:00       file.mod2.http  {"lfs242":"hello from http"}
```

```
2022-06-20T15:18:24+00:00       local.logs      {"message":"An error has occurred: Done"}

ubuntu@labsys:~$
```

The other events that did not match the STDOUT filter were recorded into this buffer log file. Notice, though, that the event that was sent to STDOUT is not present in the buffer log file. This is because `<match>` directives capture and output events in the order they are listed in the configuration file. Once an event is captured by a `<match>` directive and output, it is no longer a part of the processing data stream. If that previous `<match>` directive was set to match all tags with the `**` pattern, then this second `<match>` directive would never have come into effect.

Take a look at that name, though: buffer indicates that this is not the final destination of the log file. Since no buffer options were configured for this `<match>` directive, the default buffer behavior will be used, and will be written out after about a day or so. You can force Fluentd to flush the buffers, both in memory and on the local filesystem, manually by sending a `SIGUSR1`.

In your working terminal, send a `-SIGUSR1` to Fluentd to flush the buffer:

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

```
2022-06-20 15:19:40 +0000 [info]: #0 force flushing buffered events
2022-06-20 15:19:40 +0000 [info]: #0 flushing all buffer forcedly
2022-06-20 15:20:01 +0000 [info]: #0 following tail of /tmp/fluent/aggregated-logs.20220620_0.log
```

Flushing the buffers will take all events written in the buffer and publish them to a file, removing the old files once the flush is complete. List the aggregated-logs directory again to see:

```
ubuntu@labsys:~$ ls -l /tmp/fluent/aggregated-logs

total 8
-rw-r--r-- 1 ubuntu ubuntu 379 Jun 20 15:20 buffer.b5e1e2a33f3e1c8d50327e84f30d4b496.log
-rw-r--r-- 1 ubuntu ubuntu  79 Jun 20 15:20 buffer.b5e1e2a33f3e1c8d50327e84f30d4b496.log.meta

ubuntu@labsys:~$
```

A new set of buffer files have taken the previous file's place, but the console output of the Fluentd instance indicated that a new file was created (and tailed).

List the contents of the /tmp/fluent directory:

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

total 20
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 15:20 aggregated-logs
-rw-r--r-- 1 ubuntu ubuntu  208 Jun 20 15:19 aggregated-logs.20220620_0.log
-rw-rw-r-- 1 ubuntu ubuntu   60 Jun 20 15:10 applogs
-rw-rw-r-- 1 ubuntu ubuntu  165 Jun 20 15:18 error.log
-rw-r--r-- 1 ubuntu ubuntu  133 Jun 20 15:20 lab2.tail.pos

ubuntu@labsys:~$
```

When the Fluentd buffer is flushed, the old buffer file is removed (as you have seen) and the actual log file is created. The actual log file's name is a combination of:

- `aggregated-logs.log` - the value of the out_file plugin's `path` parameter

- `20190512` - the current date of the machine (default YYYYMMDD), at the time the buffer was flushed
- `0` - the number of the buffer flush that created the file

Now check the contents of that file:

```
ubuntu@labsys:~$ cat /tmp/fluent/aggregated-logs.*.log

2022-06-20T15:18:18+00:00       file.mod2.lab   {"lfs242":"hello"}
2022-06-20T15:18:21+00:00       file.mod2.http  {"lfs242":"hello from http"}
2022-06-20T15:18:24+00:00       local.logs      {"message":"An error has occurred: Done"}

ubuntu@labsys:~$
```

When a buffered plugin receives and event, it will not immediately output to the file that was specified in the `<match>` directive. Instead, it will wait until the buffer is flushed (automatically or manually) before writing to the file.

You have now seen buffered output plugins in action. This buffering behavior, while slow by default, gives Fluentd robustness against data loss. If the example Fluentd instance were to terminate unexpectedly, any events stored in its buffer files will be sent again as soon as it returns. This minimizes the chance of data loss due to an unexpected failure in Fluentd.

Cleanup

Use CTRL C to terminate any running instances of Fluentd:

```
^C

2021-04-14 22:07:42 +0000 [info]: Received graceful stop
2021-04-14 22:07:43 +0000 [info]: #0 fluentd worker is now stopping worker=0
2021-04-14 22:07:43 +0000 [info]: #0 shutting down fluentd worker worker=0
2021-04-14 22:07:43 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:11f8"
2021-04-14 22:07:43 +0000 [info]: #0 shutting down input plugin type=:forward
plugin_id="object:13d8"
2021-04-14 22:07:43 +0000 [info]: #0 shutting down input plugin type=:http plugin_id="object:11e4"
2021-04-14 22:07:43 +0000 [info]: #0 shutting down output plugin type=:stdout
plugin_id="object:1194"
2021-04-14 22:07:43 +0000 [info]: #0 shutting down output plugin type=:file plugin_id="object:11bc"
2021-04-14 22:07:43 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab2$ cd

ubuntu@labsys:~$
```

Congratulations, you have completed the Lab.