

LFS242 - Cloud Native Logging with Fluentd

Fluentd is a cross platform open-source data collector. It is written primarily in the Ruby programming language with several components developed in C for performance. Fluentd can be used to create a unified logging layer, allowing users to unify data collection and consumption across a wide array of systems and services.

Fluentd can be installed and configured for a variety of environments. Lab 1 is organized into three separate parts, each involves the installation and configuration steps necessary to get started with Fluentd in a different environment:

- A. Linux
- B. Docker
- C. Kubernetes

Lab 1-A – Installing and configuring Fluentd on Linux

In this lab you will get a chance to acquire and run Fluentd on Linux. This lab is designed to be completed on an Ubuntu 20.04 system, if you have an appropriate system available you can perform the lab exercises on that system. The labs do install and configure software, for that reason using a disposable system, such as a cloud instance or local VM, is recommended. The labs can be performed on a system with 2GB of memory but a system with 4GB (or more) will provide the best most responsive experience.

Objectives

- Learn how to install Fluentd directly on a Linux system.
- Understand basic system tuning techniques that can improve the performance of Fluentd
- Use Fluentd in a simple log forwarding use case

1. Preparing the host

Fluentd can run on most cloud based server instances with little effort. However there are several standard configuration tasks generally performed on production servers involved in logging or log forwarding. For example log data is typically time stamped to help with performance and problem analysis. Ensuring that the logging node has a synchronized clock is a basic step in any production setting. In large heavily used systems there may also be a need to interact with many log files/streams concurrently. Setting the number of file descriptors supported by the system is another common chore.

In this step we will complete several tasks to prepare our lab system to properly forward logs with Fluentd. As you work through the lab, you may notice that the prompt in the lab examples is different from your lab system prompt. This is expected, as long as your prompt shows your " `user name @ your hostname : your current working directory` \$" you are in good shape.

1.a. Ensure the system clock is synchronized

To ensure that our logging activity includes proper time stamps we will need to synchronize our host system's clock with a central time authority. The Network Time Protocol (NTP) is typically used for this purpose. We can ensure that the network time protocol daemon is active on our system using the `timedatectl` command.

Start your lab system and open a shell prompt, then check the NTP daemon on your lab system:

```
ubuntu@labsys:~$ sudo timedatectl

          Local time: Wed 2022-06-15 15:39:07 UTC
        Universal time: Wed 2022-06-15 15:39:07 UTC
              RTC time: Wed 2022-06-15 15:39:06
            Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

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

The `timedatectl` command should report that NTP is on and synchronized. If it does not, you can enable the service with `sudo timedatectl set-ntp on`.

1.b. Increase the file descriptor limits

Next let's check the file descriptor limit on the system with `ulimit -n`. The `ulimit` command allows us to display and set various user limits. The `-n` switch displays the maximum number of open file descriptors allowed. Check the current file descriptor limit:

```
ubuntu@labsys:~$ ulimit -n

1024

ubuntu@labsys:~$
```

Fluentd recommends a file descriptor limit of 65,536 for large systems. The value shown in the example above, 1024, is too small for most purposes and should be increased.

To permanently increase the user file descriptor limit on the system you can update the `nofile` setting in the `/etc/security/limits.conf` file. This file contains a soft and hard limit for the number of files that can be opened. Display the current contents of the `/etc/security/limits.conf` file:

```
ubuntu@labsys:~$ cat /etc/security/limits.conf

# /etc/security/limits.conf
#
#Each line describes a limit for a user in the form:
#
#<domain>          <type> <item> <value>
#
#Where:
#<domain> can be:
#      - a user name
#      - a group name, with @group syntax
#      - the wildcard *, for default entry
#      - the wildcard %, can be also used with %group syntax,
#        for maxlogin limit
#      - NOTE: group and wildcard limits are not applied to root.
#        To apply a limit to the root user, <domain> must be
#        the literal username root.
#
#<type> can have the two values:
#      - "soft" for enforcing the soft limits
#      - "hard" for enforcing hard limits
#
#<item> can be one of the following:
#      - core - limits the core file size (KB)
#      - data - max data size (KB)
#      - fsize - maximum filesize (KB)
#      - memlock - max locked-in-memory address space (KB)
#      - nofile - max number of open file descriptors
#      - rss - max resident set size (KB)
#      - stack - max stack size (KB)
#      - cpu - max CPU time (MIN)
#      - nproc - max number of processes
#      - as - address space limit (KB)
```

```

# - maxlogins - max number of logins for this user
# - maxsyslogins - max number of logins on the system
# - priority - the priority to run user process with
# - locks - max number of file locks the user can hold
# - sigpending - max number of pending signals
# - msgqueue - max memory used by POSIX message queues (bytes)
# - nice - max nice priority allowed to raise to values: [-20, 19]
# - rtprio - max realtime priority
# - chroot - change root to directory (Debian-specific)
#
#<domain>      <type> <item>      <value>
#
#*              soft   core         0
#root           hard   core         100000
#*              hard   rss          10000
#@student       hard   nproc        20
#@faculty       soft   nproc        20
#@faculty       hard   nproc        50
#ftp            hard   nproc        0
#ftp            -      chroot        /ftp
#@student       -      maxlogins    4

# End of file

ubuntu@labsys:~$

```

Per the comments in the file (if you don't see comments in your file you can refer to the example), entries are of the form: "
`<domain> <type> <item> <value>`". These are the values we will use for our entries:

- Domain: This is the user or group account to affect, we will run Fluentd as root so our setting will be `root`.
- Type: This can be hard or soft, soft limits can be adjusted by a user to anything between 0 and the hard limit, we'll set the hard and soft limit to the same value, which means we'll need two entries, one `soft` and one `hard`.
- Item: This is the specific limit to set, for us `nofile`.
- Value: This is the actual limit, for us, 64k in decimal: `65536`

Our new entries will look like this:

```

root soft nofile 65536
root hard nofile 65536

```

Add the new hard and soft limits to the end of the limits file:

```

ubuntu@labsys:~$ sudo nano /etc/security/limits.conf && sudo tail $_

#@faculty       soft   nproc        20
#@faculty       hard   nproc        50
#ftp            hard   nproc        0
#ftp            -      chroot        /ftp
#@student       -      maxlogins    4

root soft nofile 65536
root hard nofile 65536

# End of file

ubuntu@labsys:~$

```

N.B. The `$_` tells the shell to refer to the target of the previous command, in this case `/etc/security/limits.conf`

After you have the `limits.conf` updated, reboot the system so that the changes take effect:

```
ubuntu@labsys:~$ sudo reboot  
...
```

When the system restarts, log back in. To check the file descriptor limit for root we will need to switch to the root user temporarily. Once logged in, switch to the root user and check the file limit:

```
ubuntu@labsys:~$ sudo su  
root@labsys:/home/ubuntu# ulimit -n  
65536  
root@labsys:/home/ubuntu# exit  
exit  
ubuntu@labsys:~$
```

You may be wondering why we did not just use `sudo ulimit -n`. If you try that command you will see that it fails. This is because `ulimit` is a shell command not a program in the filesystem you can run. In the example we use `sudo` to run the `su` command which changes our identity to root and starts a root shell. Then we issue the `ulimit -n` command in the root shell, to see the limit for the root user.

Try displaying the limit for your non root user:

```
ubuntu@labsys:~$ ulimit -n  
1024  
ubuntu@labsys:~$
```

As you can see the normal user's limit is unchanged. When configuring Fluentd, be sure that you have set the limits for the actual user Fluentd will run under.

1.c. Tune network settings

Fluentd is often used to aggregate log data from one or more sources and then to forward that data on to one or more destinations. These sources and destinations can be remote, creating network traffic. To ensure that Fluentd network activity is as efficient as possible we can tune various network settings.

One common problem with TCP/IP based networking is that some of the defaults make sense for slow, buggy, 1970s era networks, the networks in existence when TCP/IP was invented. Modern, fast, largely reliable, digital networks have very different characteristics. In particular, by default, a disconnected TCP session will cause future connections using the same resources to fail for a period of time to ensure that the previous connection can be properly shutdown.

For systems that connect and disconnect a lot, this can cause a meaningful delay in the time it takes to transfer data, due to the wait time needed between connections. By enabling the Linux `net.ipv4.tcp_tw_reuse` setting, we ensure that a Linux system can reuse an existing connection in the TIME-WAIT state when needed. Modern TCP connections use timestamps to ensure that duplicate connections

are not created.

Another problem systems with many network connections may run into is a lack of available ports. The default port range provided by many systems is less than 30,000. This may seem like a lot but if you consider a server that supports 10,000 passive users, each requiring three connections to the host, you are already out of ports if your limit is 30,000. Fortunately this limit can be increased to over 50,000 with the Linux `net.ipv4.ip_local_port_range` setting. If you are running Fluentd on a busy network server this may be a setting worth adjusting.

The status of the `net.ipv4.tcp_tw_reuse` and `net.ipv4.ip_local_port_range` settings are provided in Linux through the proc filesystem. Display the settings on your lab system:

```
ubuntu@labsys:~$ cat /proc/sys/net/ipv4/tcp_tw_reuse
2
ubuntu@labsys:~$ cat /proc/sys/net/ipv4/ip_local_port_range
32768    60999
ubuntu@labsys:~$
```

Though these settings are listed under ipv4 they actually affect both IPv4 and IPv6. In the example the reuse setting is set to enable for loopback traffic only (2) and the port range is limited to about 30,000 (32768 - 60999). To make permanent changes to these values we need to modify the sysctl.conf file in the /etc directory.

All of the settings in the default file in the example are commented out. We will add the following two options to the file:

- `net.ipv4.tcp_tw_reuse = 1`
- `net.ipv4.ip_local_port_range = 10240 65535`

You will need to use sudo to edit the sysctl.conf file. Add the two settings in the list above to the bottom of your sysctl.conf file:

```
ubuntu@labsys:~$ sudo nano /etc/sysctl.conf && sudo tail $_
#####
# Magic system request Key
# 0=disable, 1=enable all, >1 bitmask of sysrq functions
# See https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html
# for what other values do
#kernel.sysrq=438

net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
ubuntu@labsys:~$
```

These settings turn on TCP port reuse and give us a port range of 10240 - 65535 to work with.

Like the user limits, you can reboot the system to cause the changes to take affect, however, most sysctl settings can also be modified on a running system by writing to the proc filesystem or using the sysctl command. Run the sysctl command with the -p switch to apply the sysctl.conf file changes:

```
ubuntu@labsys:~$ sudo sysctl -p
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
```

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

Verify that the changes are in effect:

```
ubuntu@labsys:~$ cat /proc/sys/net/ipv4/tcp_tw_reuse
1
ubuntu@labsys:~$ cat /proc/sys/net/ipv4/ip_local_port_range
10240 65535
ubuntu@labsys:~$
```

Perfect!

While there are many other network settings we could configure, these are some of the most important for efficient Fluentd operations.

2. Installing Fluentd on Linux

There are various ways to install Fluentd on a host system. Some of the options include:

- Distribution package managers, tools like `yum`, `apt`, `zypper`, `brew` and `choco`
- Release tarballs
- Source code
- The Ruby library manager, `gem`

Distribution package managers work well and are highly integrated with their target system, however they tend to install older versions of software. The other approaches allow you to access the latest release of Fluentd easily. Perhaps the simplest of these is the Ruby Gem approach, which we'll use next.

2.a. Install Ruby on the host system

It is a good idea to update the package indexes on the host to ensure that the latest package information is used. Update the package cache on your lab system as follows:

```
ubuntu@labsys:~$ sudo apt update
...
Fetched 23.3 MB in 4s (6538 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
ubuntu@labsys:~$
```

Before we install Fluentd through the Ruby Gem package manager, we will need to install Ruby and Ruby Gem.

```
ubuntu@labsys:~$ sudo apt install ruby-full ruby-dev -y
...
Setting up ruby2.7-dev:amd64 (2.7.0-5ubuntu1.7) ...
Setting up ruby-dev:amd64 (1:2.7+1) ...
Setting up ruby-full (1:2.7+1) ...
```

```
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.7) ...
Processing triggers for man-db (2.9.1-1) ...
```

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

Ruby is an interpreted programming language and, while fast for a scripting language, it is not as fast as a compiled language. To eliminate bottlenecks in Ruby based programs, developers often create Ruby libraries in compiled languages like C. Fluentd includes C based components to ensure maximum performance.

To make use of the Fluentd C libraries the Gem installer will need to compile them. For this to work you need to have C programming language tools installed locally, `make` and `gcc` in particular.

Some additional libraries you will require to properly install Ruby include:

- GNU Readline to provide allow users to edit command lines as they are typed in
- OpenSSL dev library to provide TLS support
- zLib to implement the deflate compression method in gzip/pkzip

Install `libssl-dev`, `libreadline-dev`, `zlib1g-dev`, to your lab system (you may already have `make` and `gcc` installed.):

```
ubuntu@labsys:~$ sudo apt install -y libssl-dev libreadline-dev zlib1g-dev gcc make -y

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-9 gcc-9 gcc-9-base libasan5 libatomic1
  libbinutils libc-dev-bin libc6 libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0
  libctf0 libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0 libmpc3 libncurses-dev libquadmath0
  libssl1.1 libtsan0 libubsan1 linux-libc-dev manpages-dev
Suggested packages:
  binutils-doc cpp-doc gcc-9-locales gcc-multilib autoconf automake libtool flex bison gdb gcc-doc
  gcc-9-multilib gcc-9-doc glibc-doc ncurses-doc readline-doc libssl-doc
  make-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cpp cpp-9 gcc gcc-9 gcc-9-base libasan5
  libatomic1 libbinutils libc-dev-bin libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0
  libctf0 libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0 libmpc3 libncurses-dev libquadmath0
  libreadline-dev libssl-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev
  zlib1g-dev
The following packages will be upgraded:
  libc6 libssl1.1
2 upgraded, 33 newly installed, 0 to remove and 63 not upgraded.

...

ubuntu@labsys:~$
```

Ruby is now installed under `/home/ubuntu/.rbenv/versions/2.7.0`. Next, we need to install Bundler to provide a consistent environment for Ruby projects and ensure dependencies are met by all other gems. Before we install gem, you need to use rbenv rehash to ensure your shell can reach the gem tool using a rbenv shims:

```
ubuntu@labsys:~$ sudo gem install bundle

Fetching bundle-0.0.1.gem
Successfully installed bundle-0.0.1
Parsing documentation for bundle-0.0.1
```

```
Installing ri documentation for bundle-0.0.1
Done installing documentation for bundle after 0 seconds
1 gem installed

ubuntu@labsys:~$
```

We're now ready to install Fluentd.

2.b. Install the Fluentd Ruby Gem

With Ruby setup, the official Fluentd Ruby Gem can be installed using the Ruby Gem library manager. Installable Ruby libraries are known as Gems. By default the Gem tool will build and install the executable Gems, including all of its dependencies and documentation. You can use the `-N` switch to skip the documentation installation on production systems that have no need for it.

The `--local` switch can be used to install Gems locally in the current user's directory. This is good for working with systems where you do not have root access:

Install Fluentd with the gem librarian:

```
ubuntu@labsys:~$ sudo gem install fluentd

Fetching concurrent-ruby-1.1.10.gem
Fetching http_parser.rb-0.8.0.gem
Fetching strptime-0.2.5.gem
Fetching msgpack-1.5.2.gem
Fetching sigdump-0.2.4.gem
Fetching serverengine-2.3.0.gem
Fetching yajl-ruby-1.4.3.gem
Fetching fluentd-1.14.6.gem
Fetching tzinfo-data-1.2022.1.gem
Fetching tzinfo-2.0.4.gem
Fetching cool.io-1.7.1.gem
Building native extensions. This could take a while...
Successfully installed msgpack-1.5.2
Building native extensions. This could take a while...
Successfully installed yajl-ruby-1.4.3
Building native extensions. This could take a while...
Successfully installed cool.io-1.7.1
Successfully installed sigdump-0.2.4
Successfully installed serverengine-2.3.0
Building native extensions. This could take a while...
Successfully installed http_parser.rb-0.8.0
Successfully installed concurrent-ruby-1.1.10
Successfully installed tzinfo-2.0.4
Successfully installed tzinfo-data-1.2022.1
Building native extensions. This could take a while...
Successfully installed strptime-0.2.5
Successfully installed fluentd-1.14.6
Parsing documentation for msgpack-1.5.2
Installing ri documentation for msgpack-1.5.2
Parsing documentation for yajl-ruby-1.4.3
Installing ri documentation for yajl-ruby-1.4.3
Parsing documentation for cool.io-1.7.1
Installing ri documentation for cool.io-1.7.1
Parsing documentation for sigdump-0.2.4
Installing ri documentation for sigdump-0.2.4
Parsing documentation for serverengine-2.3.0
Installing ri documentation for serverengine-2.3.0
Parsing documentation for http_parser.rb-0.8.0
unknown encoding name "chunked\r\n\r\n25" for ext/ruby_http_parser/vendor/http-parser-
```



```
java/tools/parse_tests.rb, skipping
Installing ri documentation for http_parser.rb-0.8.0
Parsing documentation for concurrent-ruby-1.1.10
Installing ri documentation for concurrent-ruby-1.1.10
Parsing documentation for tzinfo-2.0.4
Installing ri documentation for tzinfo-2.0.4
Parsing documentation for tzinfo-data-1.2022.1
Installing ri documentation for tzinfo-data-1.2022.1
Parsing documentation for strptime-0.2.5
Installing ri documentation for strptime-0.2.5
Parsing documentation for fluentd-1.14.6
Installing ri documentation for fluentd-1.14.6
Done installing documentation for msgpack, yajl-ruby, cool.io, sigdump, serverengine,
http_parser.rb, concurrent-ruby, tzinfo, tzinfo-data, strptime, fluentd after 15 seconds
11 gems installed

ubuntu@labsys:~$
```

With the gems installed, the `fluentd` command should be available. Use the `--help` switch to list the available Fluentd command line arguments:

```
ubuntu@labsys:~$ fluentd --help

Usage: fluentd [options]
  -s, --setup [DIR=/etc/fluent]  install sample configuration file to the directory
  -c, --config PATH              config file path (default: /etc/fluent/fluent.conf)
      --dry-run                  Check fluentd setup is correct or not
      --show-plugin-config=PLUGIN [DEPRECATED] Show PLUGIN configuration and exit(ex:
input:dummy)
  -p, --plugin DIR              add plugin directory

...

ubuntu@labsys:~$
```

Great, Fluentd is now ready to run!

2.c. Explore the Fluentd installation

To test out Fluentd we'll create a simple Fluentd configuration and then try processing some log data. Fluentd reads a configuration file (typically `fluent.conf`) at startup to determine which log sources to process and where to send them. The Fluentd command supplies a `--setup` flag that can be used to create a sample configuration file.

Generate a Fluentd configuration file template:

```
ubuntu@labsys:~$ fluentd --setup ./fluent

Installed ./fluent/fluent.conf.

ubuntu@labsys:~$
```

This creates a Fluentd directory with a pre-populated configuration file called `fluent.conf`.

Inspect the `fluent` folder:

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

```
total 8
-rw-rw-r-- 1 ubuntu ubuntu 2696 Jun 15 15:56 fluent.conf
drwxrwxr-x 2 ubuntu ubuntu 4096 Jun 15 15:56 plugin

ubuntu@labsys:~$
```

A plugin directory has been created alongside a fluent.conf file, though it is currently empty:

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

total 0
```

Fluentd is more of a framework than an actual application. As a framework Fluentd uses data source plugins to read log data, filter plugins to process log data and output plugins to forward log data to particular targets. For example, you might want to read data from syslog, filter out everything but errors, and forward it to Elasticsearch. As we will see in this and future labs, plugins are a valuable part of the Fluentd ecosystem.

Display the sample configuration file Fluentd generated:

```
ubuntu@labsys:~$ cat fluent/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>

## built-in UNIX socket input
#<source>
#  @type unix
#</source>

# HTTP input
# http://localhost:8888/<tag>?json=<json>
<source>
  @type http
  @id http_input

  port 8888
</source>

## File input
## read apache logs with tag=apache.access
#<source>
#  @type tail
#  format apache
#  path /var/log/httpd-access.log
#  tag apache.access
#</source>

## Mutating event filter
## Add hostname and tag fields to apache.access tag events
#<filter apache.access>
#  @type record_transformer
#  <record>
```

```

#   hostname ${hostname}
#   tag ${tag}
# </record>
#</filter>

## Selecting event filter
## Remove unnecessary events from apache prefixed tag events
#<filter apache.**>
#   @type grep
#   include1 method GET # pass only GET in 'method' field
#   exclude1 message debug # remove debug event
#</filter>

# Listen HTTP for monitoring
# http://localhost:24220/api/plugins
# http://localhost:24220/api/plugins?type=TYPE
# http://localhost:24220/api/plugins?tag=MYTAG
<source>
  @type monitor_agent
  @id monitor_agent_input

  port 24220
</source>

# Listen DRb for debug
<source>
  @type debug_agent
  @id debug_agent_input

  bind 127.0.0.1
  port 24230
</source>

## match tag=apache.access and write to file
#<match apache.access>
#   @type file
#   path /var/log/fluent/access
#</match>

## match tag=debug.** and dump to console
<match debug.**>
  @type stdout
  @id stdout_output
</match>

# match tag=system.** and forward to another fluent server
<match system.**>
  @type forward
  @id forward_output

  <server>
    host 192.168.0.11
  </server>
  <secondary>
    <server>
      host 192.168.0.12
    </server>
  </secondary>
</match>

## match tag=myapp.** and forward and write to file
#<match myapp.**>

```

```
# @type copy
# <store>
#   @type forward
#   buffer_type file
#   buffer_path /var/log/fluent/myapp-forward
#   retry_limit 50
#   flush_interval 10s
#   <server>
#     host 192.168.0.13
#   </server>
# </store>
# </store>
# @type file
# path /var/log/fluent/myapp
# </store>
#</match>
```

```
## match fluent's internal events
#<match fluent.**>
# @type null
#</match>
```

```
## match not matched logs and write to file
#<match **>
# @type file
# path /var/log/fluent/else
# compress gz
#</match>
```

```
## Label: For handling complex event routing
#<label @STAGING>
# <match system.**>
#   @type forward
#   @id staging_forward_output
#   <server>
#     host 192.168.0.101
#   </server>
# </match>
#</label>
```

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

N.B. This config is just an example and does not require any additional resources

There's a lot to know about configuring Fluentd. Essentially all of the features of Fluentd are exposed through its configuration file. Fluentd configurations will be covered in depth throughout the course but we can start with the basics.

Fluentd configuration files are XML documents with support for `#` prefixed comments. Top level XML tags, known as directives, include:

- `<source>` - defines a data source
- `<match>` - defines a data pattern to match and send to a destination
- `<filter>` - defines a data pattern to match and process in some way
- `<label>` - groups filters and outputs (match) directives into a pipeline

These directives include a `@type` parameter which specifies the plugin to use. For example:

```
<match **>
  @type file
```

```
path /var/log/fluent/else
compress gz
</match>
```

The configuration stanza above matches all log messages (`<match **>`) and outputs them to the file plugin (`@type file`). The plugin will be provided with the parameters `path /var/log/fluent/else` , which specifies the output directory, and `compress gz` , which requests that the output be gzipped. Parameters that begin with `@` , like `@type` are known as system parameters and are recognized and used by Fluentd. Other parameters, like `path` , are only used by the plugin, though plugins can see the system parameters as well.

Part of mastering Fluentd is simply learning all of the plugins and their parameters. The `file` plugin is built-in to Fluentd, as are many of the most used plugins. Other plugins must be installed to be used. We'll try working with third party plugins in a later lab.

- What types of plugins are used in the sample config file above?

3. Running Fluentd with a simple configuration

You can pass Fluentd a configuration file on the command line using the `--config` or `-c` switch. Run Fluentd with the config file generated in the last step and with the `-vv` (very verbose) switch:

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

2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="./fluent/fluent.conf"
2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330:info: gem 'fluentd' version '1.14.6'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'stdout'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered formatter plugin 'stdout'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered formatter plugin 'json'
2022-06-15 16:00:20 +0000 [info]: [stdout_output] Oj isn't installed, fallback to Yajl as json
parser
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'forward'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered sd plugin 'static'
2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330:info: adding forwarding server
'192.168.0.12:24224' host="192.168.0.12" port=24224 weight=60 plugin_id="object:758"
2022-06-15 16:00:20 +0000 [debug]: fluent/log.rb:309:debug: rebuilding weight array lost_weight=0
2022-06-15 16:00:20 +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"
2022-06-15 16:00:20 +0000 [debug]: [forward_output] rebuilding weight array lost_weight=0
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'forward'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'in_http'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'http'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'msgpack'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'json'
2022-06-15 16:00:20 +0000 [warn]: [http_input] LoadError
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'monitor_agent'
2022-06-15 16:00:20 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'debug_agent'
2022-06-15 16:00:20 +0000 [debug]: fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330: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>
2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330:info: starting fluentd-1.14.6 pid=6515
ruby="2.7.0"
2022-06-15 16:00:20 +0000 [info]: fluent/log.rb:330:info: spawn command to main: cmdline=
["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "--config",
"./fluent/fluent.conf", "-vv", "--under-supervisor"]
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding match pattern="debug.**"
type="stdout"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered output plugin 'stdout'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered formatter plugin 'stdout'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered formatter plugin 'json'
2022-06-15 16:00:21 +0000 [info]: #0 [stdout_output] Oj isn't installed, fallback to Yajl as json
parser
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding match pattern="system.**"
type="forward"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered output plugin 'forward'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered sd plugin 'static'
2022-06-15 16:00:21 +0000 [info]: #0 fluent/log.rb:330:info: adding forwarding server
'192.168.0.12:24224' host="192.168.0.12" port=24224 weight=60 plugin_id="object:758"
2022-06-15 16:00:21 +0000 [debug]: #0 fluent/log.rb:309:debug: rebuilding weight array lost_weight=0
2022-06-15 16:00:21 +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"
2022-06-15 16:00:21 +0000 [debug]: #0 [forward_output] rebuilding weight array lost_weight=0
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding source type="forward"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'forward'
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding source type="http"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'in_http'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'http'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'msgpack'
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'json'
2022-06-15 16:00:21 +0000 [warn]: #0 [http_input] LoadError
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding source type="monitor_agent"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin

```

```
'monitor_agent'
2022-06-15 16:00:21 +0000 [info]: fluent/log.rb:330:info: adding source type="debug_agent"
2022-06-15 16:00:21 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'debug_agent'
2022-06-15 16:00:21 +0000 [debug]: #0 fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-15 16:00:21 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=6520
ppid=6515 worker=0
2022-06-15 16:00:21 +0000 [debug]: #0 [forward_output] buffer started instance=1860 stage_size=0
queue_size=0
2022-06-15 16:00:21 +0000 [info]: #0 [debug_agent_input] listening dRuby
uri="druby://127.0.0.1:24230" object="Fluent::Engine" worker=0
2022-06-15 16:00:21 +0000 [debug]: #0 [forward_output] flush_thread actually running
2022-06-15 16:00:21 +0000 [debug]: #0 [monitor_agent_input] listening monitoring http server on
http://0.0.0.0:24220/api/plugins for worker0
2022-06-15 16:00:21 +0000 [debug]: #0 [forward_output] enqueue_thread actually running
2022-06-15 16:00:21 +0000 [trace]: #0 [forward_output] enqueueing all chunks in buffer instance=1860
2022-06-15 16:00:21 +0000 [debug]: #0 [monitor_agent_input] Start webrick HTTP server listening
2022-06-15 16:00:21 +0000 [debug]: #0 [http_input] listening http bind="0.0.0.0" port=8888
2022-06-15 16:00:21 +0000 [info]: #0 [forward_input] listening port port=24224 bind="0.0.0.0"
2022-06-15 16:00:21 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-15 16:00:22 +0000 [trace]: #0 [forward_output] sending heartbeat host="192.168.0.11"
port=24224 heartbeat_type=:transport
2022-06-15 16:00:22 +0000 [debug]: #0 [forward_output] connect new socket
2022-06-15 16:00:22 +0000 [trace]: #0 fluent/log.rb:287:trace: sending heartbeat host="192.168.0.12"
port=24224 heartbeat_type=:transport
2022-06-15 16:00:22 +0000 [debug]: #0 fluent/log.rb:309:debug: connect new socket
2022-06-15 16:00:27 +0000 [trace]: #0 [forward_output] enqueueing all chunks in buffer instance=1860
```

Examine the Fluentd output and answer these questions:

- What is the path to the config file Fluentd is using?
- What port is the Fluentd "forwarding server" using?
- How many plugins are registered for use? Are all of these mentioned in your config file?

Fluentd, like Elasticsearch and many other popular log storage and processing systems, is designed to work with JSON formatted data. One of the first tasks involved in ingesting data is formatting it in JSON if it is not already.

We can test our Fluentd instance by sending it a simple JSON message using the fluent-cat utility. Open a second command line shell on your lab system and try running fluent-cat:

```
ubuntu@labsys:~$ echo '{"json":"message"}' | fluent-cat debug.test
ubuntu@labsys:~$
```

The fluent-cat tool comes with Fluentd and is helpful for testing and debugging. The example echoes a JSON message to fluent-cat, which tags it with the string "debug.test" and delivers it to Fluentd via the default Fluentd port on the localhost interface, 127.0.0.1:24224. The matcher `<match debug.**>` picks up the message because the message was given a matching tag, "debug.test". The matcher then sends the message to STDOUT as directed, `@type STDOUT`.

Fluentd should output the message to STDOUT in the second original terminal:

```
2022-06-15 16:01:04 +0000 [trace]: #0 [forward_input] connected fluent socket addr="127.0.0.1"
port=55612
2022-06-15 16:01:04 +0000 [trace]: #0 [forward_input] accepted fluent socket addr="127.0.0.1"
port=55612
2022-06-15 16:01:04.338419615 +0000 debug.test: {"json":"message"}
```

- Issue the debug message several more times; what is different about each message?

4. Clean up

When you have finished exploring type `ctrl c` at the console of any Fluentd instances left running to shut them down (it will take Fluentd a moment to stop all active plugins and shutdown):

```
...

Ctrl c

2022-06-15 16:01:20 +0000 [debug]: #0 fluent/log.rb:309:debug: fluentd main process get SIGINT
2022-06-15 16:01:20 +0000 [info]: fluent/log.rb:330:info: Received graceful stop
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: fluentd main process get SIGTERM
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: getting start to shutdown main
process
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now stopping worker=0
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down fluentd worker worker=0
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on input plugin
type=:forward plugin_id="forward_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on input plugin
type=:http plugin_id="http_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on input plugin
type=:monitor_agent plugin_id="monitor_agent_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on input plugin
type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on output plugin
type=:stdout plugin_id="stdout_output"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling stop on output plugin
type=:forward plugin_id="forward_output"
2022-06-15 16:01:21 +0000 [trace]: #0 [forward_output] enqueueing all chunks in buffer instance=1860
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown input plugin
type=:monitor_agent plugin_id="monitor_agent_input"
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down input plugin
type=:monitor_agent plugin_id="monitor_agent_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown input plugin
type=:forward plugin_id="forward_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown input plugin
type=:http plugin_id="http_input"
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down input plugin type=:http
plugin_id="http_input"
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down input plugin
type=:forward plugin_id="forward_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown input plugin
type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down input plugin
type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown output plugin
type=:forward plugin_id="forward_output"
2022-06-15 16:01:21 +0000 [trace]: #0 [forward_output] enqueueing all chunks in buffer instance=1860
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down output plugin
type=:forward plugin_id="forward_output"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: preparing shutdown output plugin
type=:stdout plugin_id="stdout_output"
2022-06-15 16:01:21 +0000 [info]: #0 fluent/log.rb:330:info: shutting down output plugin
type=:stdout plugin_id="stdout_output"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on input
plugin type=:forward plugin_id="forward_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on input
plugin type=:http plugin_id="http_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on input
plugin type=:monitor_agent plugin_id="monitor_agent_input"
```



```

2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on input
plugin type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on output
plugin type=:stdout plugin_id="stdout_output"
2022-06-15 16:01:21 +0000 [debug]: #0 fluent/log.rb:309:debug: calling after_shutdown on output
plugin type=:forward plugin_id="forward_output"
2022-06-15 16:01:26 +0000 [warn]: #0 [forward_output] event loop does NOT exit until hard timeout.
2022-06-15 16:01:31 +0000 [warn]: #0 fluent/log.rb:351:warn: event loop does NOT exit until hard
timeout.
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing input plugin type=:forward
plugin_id="forward_input"
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing input plugin type=:http
plugin_id="http_input"
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing input plugin
type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing input plugin
type=:monitor_agent plugin_id="monitor_agent_input"
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing output plugin type=:forward
plugin_id="forward_output"
2022-06-15 16:01:31 +0000 [debug]: #0 fluent/log.rb:309:debug: closing output plugin type=:stdout
plugin_id="stdout_output"
2022-06-15 16:01:31 +0000 [debug]: #0 [forward_output] closing buffer instance=1860
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on input plugin
type=:forward plugin_id="forward_input"
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on input plugin
type=:http plugin_id="http_input"
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on input plugin
type=:monitor_agent plugin_id="monitor_agent_input"
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on input plugin
type=:debug_agent plugin_id="debug_agent_input"
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on output plugin
type=:stdout plugin_id="stdout_output"
2022-06-15 16:01:33 +0000 [debug]: #0 fluent/log.rb:309:debug: calling terminate on output plugin
type=:forward plugin_id="forward_output"
2022-06-15 16:01:33 +0000 [warn]: #0 fluent/log.rb:351:warn: killing existing thread thread=#
<Thread:0x000055ef041e7fa0@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 sleep>
2022-06-15 16:01:33 +0000 [warn]: #0 fluent/log.rb:351:warn: thread doesn't exit correctly (killed
or other reason) plugin=Fluent::Plugin::ForwardOutput title=:event_loop thread=#
<Thread:0x000055ef041e7fa0@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 aborting> error=nil
2022-06-15 16:01:33 +0000 [warn]: #0 [forward_output] killing existing thread thread=#
<Thread:0x000055ef041e7b18@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 sleep>
2022-06-15 16:01:33 +0000 [warn]: #0 [forward_output] thread doesn't exit correctly (killed or other
reason) plugin=Fluent::Plugin::ForwardOutput title=:event_loop thread=#
<Thread:0x000055ef041e7b18@event_loop /var/lib/gems/2.7.0/gems/fluentd-
1.14.6/lib/fluent/plugin_helper/thread.rb:70 aborting> error=nil
2022-06-15 16:01:33 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~$

```

In this lab we installed Fluentd, configured it, and tested basic functionality. We'll learn more about Fluentd in the labs ahead.

Congratulations, you have completed the Lab.

LFS242 - Cloud Native Logging with Fluentd

Fluentd is a cross platform open-source data collector. It is written primarily in the Ruby programming language with several components developed in C for performance. Fluentd can be used to create a unified logging layer, allowing users to unify data collection and consumption across a wide array of systems and services.

Fluentd can be installed and configured for a variety of environments. Lab 1 is organized into three separate parts, each involves the installation and configuration steps necessary to get started with Fluentd in a different environment:

- A. Linux
- B. Docker
- C. Kubernetes

Lab 1-B – Running Fluentd in a Docker environment

In this lab you will get a chance to run Fluentd using the Docker container runtime. This lab can be completed on the same lab system used for Lab 1-A or on a new machine. Instructions for configuring a suitable lab machine can be found in that lab.

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. The use of Linux containers to deploy applications is called containerization. Containers can greatly simplify the process of application deployment, increasing deployment modularity and reliability. Containerized applications, like all applications, emit log data that needs to be processed and collated to allow developers and administrators to detect trends and research problems. In this lab we'll see how Fluentd can play a key role in the log management of containerized applications.

Objectives

- Learn how to install Docker CE.
- Learn how to run Fluentd in a Docker environment.
- Learn how to process logs from containerized applications with Fluentd.

1. Prepare the lab system

If you have not already, log back into your lab system. Before we begin working with containers, make sure that any Fluentd instances or other programs started in prior labs are stopped.

2. Install Docker

Docker is available in a commercial Enterprise Edition (Docker EE) and a free open source Community Edition (Docker CE), though the core docker container runtime functionality is the same in both. We will install Docker CE on our lab systems. The docker web site provides an easy to use installation script that we can run to install docker.

In a new shell on your lab system, run the get.docker.com script to install Docker CE:

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

--2022-06-15 16:58:21-- https://get.docker.com/
Resolving get.docker.com (get.docker.com)... 13.32.208.79, 13.32.208.39, 13.32.208.37, ...
Connecting to get.docker.com (get.docker.com)|13.32.208.79|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20009 (20K) [text/plain]
Saving to: 'STDOUT'

-
100%
[=====>]
19.54K --.-KB/s in 0s
```

2022-06-15 16:58:21 (56.9 MB/s) - written to stdout [20009/20009]

```
# Executing docker install script, commit: b2e29ef7a9a89840d2333637f7d1900a83e7153f
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https ca-
certificates curl >/dev/null
+ sudo -E sh -c mkdir -p /etc/apt/keyrings && chmod -R 0755 /etc/apt/keyrings
+ sudo -E sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | gpg --dearmor --yes -o
/etc/apt/keyrings/docker.gpg
+ sudo -E sh -c chmod a+r /etc/apt/keyrings/docker.gpg
+ sudo -E sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu focal stable" > /etc/apt/sources.list.d/docker.list
+ sudo -E sh -c apt-get update -qq >/dev/null
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommends
docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-scan-plugin >/dev/null
+ version_gte 20.10
+ [ -z ]
+ return 0
+ sudo -E sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-extras
>/dev/null
+ sudo -E sh -c docker version
```

Client: Docker Engine - Community

```
Version:      20.10.17
API version:   1.41
Go version:    go1.17.11
Git commit:    100c701
Built:         Mon Jun  6 23:02:57 2022
OS/Arch:       linux/amd64
Context:       default
Experimental:  true
```

Server: Docker Engine - Community

```
Engine:
Version:      20.10.17
API version:   1.41 (minimum version 1.12)
Go version:    go1.17.11
Git commit:    a89b842
Built:         Mon Jun  6 23:01:03 2022
OS/Arch:       linux/amd64
Experimental:  false
containerd:
Version:      1.6.6
GitCommit:    10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
runc:
Version:      1.1.2
GitCommit:    v1.1.2-0-ga916309
docker-init:
Version:      0.19.0
GitCommit:    de40ad0
```

=====

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

```
dockerd-rootless-setuptool.sh install
```

Visit <https://docs.docker.com/go/rootless/> to learn about rootless mode.

To run the Docker daemon as a fully privileged service, but granting non-root

users access, refer to <https://docs.docker.com/go/daemon-access/>

WARNING: Access to the remote API on a privileged Docker daemon is equivalent to root access on the host. Refer to the 'Docker daemon attack surface' documentation for details: <https://docs.docker.com/go/attack-surface/>

=====

ubuntu@labsys:~\$

The command above sends the [get.docker.com](https://docs.docker.com/go/daemon-access/) install script to a pipe (`|`) connected to a Bourne shell (`sh`). The Bourne shell runs the script, adding the Docker package repository and key to the local apt cache, then installing docker from packages.

At the bottom of the output the script reports the docker version installed and provides instructions on running Docker in rootless mode. Normally the dockerd daemon runs as root and can run arbitrary containers as root on the host. This makes users who have access to docker very powerful. By default only root and members of the docker group can run docker commands.

Install Docker in rootless mode. First, install the `uidmap` package from apt:

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

...

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

Next, run the Docker rootless mode installation script:

```
ubuntu@labsys:~$ dockerd-rootless-setuptool.sh install
```

```
[INFO] Creating /home/ubuntu/.config/systemd/user/docker.service
[INFO] starting systemd service docker.service
+ systemctl --user start docker.service
+ sleep 3
+ systemctl --user --no-pager --full status docker.service
• docker.service - Docker Application Container Engine (Rootless)
  Loaded: loaded (/home/ubuntu/.config/systemd/user/docker.service; disabled; vendor preset:
enabled)
  Active: active (running) since Wed 2022-06-15 17:06:48 UTC; 3s ago
  Docs: https://docs.docker.com/go/rootless/
  Main PID: 20459 (rootlesskit)
  CGroup: /user.slice/user-1000.slice/user@1000.service/docker.service
          └─20459 rootlesskit --net=slirp4netns --mtu=65520 --slirp4netns-sandbox=auto --
slirp4netns-seccomp=auto --disable-host-loopback --port-driver=builtin --copy-up=/etc --copy-up=/run
--propagation=rslave /usr/bin/dockerd-rootless.sh
          └─20470 /proc/self/exe --net=slirp4netns --mtu=65520 --slirp4netns-sandbox=auto --
slirp4netns-seccomp=auto --disable-host-loopback --port-driver=builtin --copy-up=/etc --copy-up=/run
--propagation=rslave /usr/bin/dockerd-rootless.sh
          └─20486 slirp4netns --mtu 65520 -r 3 --disable-host-loopback --enable-sandbox --enable-
seccomp 20470 tap0
          └─20493 dockerd
          └─20508 containerd --config /run/user/1000/docker/containerd/containerd.toml --log-
level info

Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514638655Z"
level=warning msg="Your kernel does not support CPU realtime scheduler"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514674527Z"
level=warning msg="Your kernel does not support cgroup blkio weight"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.514685253Z"
level=warning msg="Your kernel does not support cgroup blkio weight_device"
```

```

Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.515295476Z"
level=info msg="Loading containers: start."
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.639161573Z"
level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option
--bip can be used to set a preferred IP address"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.706986336Z"
level=info msg="Loading containers: done."
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.725144035Z"
level=warning msg="Not using native diff for overlay2, this may cause degraded performance for
building images: running in a user namespace" storage-driver=overlay2
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.725845647Z"
level=info msg="Docker daemon" commit=a89b842 graphdriver(s)=overlay2 version=20.10.17
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.726124312Z"
level=info msg="Daemon has completed initialization"
Jun 15 17:06:49 ip-172-31-9-135 dockerd-rootless.sh[20493]: time="2022-06-15T17:06:49.812766614Z"
level=info msg="API listen on /run/user/1000/docker.sock"
+ DOCKER_HOST=unix:///run/user/1000/docker.sock /usr/bin/docker version
Client: Docker Engine - Community
 Version:      20.10.17
 API version:  1.41
 Go version:   go1.17.11
 Git commit:   100c701
 Built:        Mon Jun  6 23:02:57 2022
 OS/Arch:     linux/amd64
 Context:     default
 Experimental: true

Server: Docker Engine - Community
 Engine:
  Version:      20.10.17
  API version:  1.41 (minimum version 1.12)
  Go version:   go1.17.11
  Git commit:   a89b842
  Built:        Mon Jun  6 23:01:03 2022
  OS/Arch:     linux/amd64
  Experimental: false
 containerd:
  Version:      1.6.6
  GitCommit:    10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:      1.1.2
  GitCommit:    v1.1.2-0-ga916309
 docker-init:
  Version:      0.19.0
  GitCommit:    de40ad0
+ systemctl --user enable docker.service
Created symlink /home/ubuntu/.config/systemd/user/default.target.wants/docker.service →
/home/ubuntu/.config/systemd/user/docker.service.
[INFO] Installed docker.service successfully.
[INFO] To control docker.service, run: `systemctl --user (start|stop|restart) docker.service`
[INFO] To run docker.service on system startup, run: `sudo loginctl enable-linger ubuntu`

[INFO] Creating CLI context "rootless"
Successfully created context "rootless"

[INFO] Make sure the following environment variables are set (or add them to ~/.bashrc):

export PATH=/usr/bin:$PATH
export DOCKER_HOST=unix:///run/user/1000/docker.sock

ubuntu@labsys:~$

```

Finally, modify the `PATH` and `DOCKER_HOST` variables as suggested at the end of the script:

```
ubuntu@labsys:~$ export PATH=/usr/bin:$PATH

ubuntu@labsys:~$ export DOCKER_HOST=unix:///run/user/1000/docker.sock

ubuntu@labsys:~$
```

2.a. Verify the Docker installation

Check the docker version to insure the docker command line client is properly installed:

```
ubuntu@labsys:~$ docker --version

Docker version 20.10.17, build 100c701

ubuntu@labsys:~$
```

You can get command line help with the docker help switch. Try it:

```
ubuntu@labsys:~$ docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default "/home/ubuntu/.docker")
  -c, --context string  Name of the context to use to connect to the daemon (overrides
DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug           Enable debug mode
  -H, --host list       Daemon socket(s) to connect to
  -l, --log-level string Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default
"info")
  --tls                Use TLS; implied by --tlsverify
  --tlscacert string   Trust certs signed only by this CA (default
"/home/ubuntu/.docker/ca.pem")
  --tlscert string      Path to TLS certificate file (default "/home/ubuntu/.docker/cert.pem")
  --tlskey string       Path to TLS key file (default "/home/ubuntu/.docker/key.pem")
  --tlsverify          Use TLS and verify the remote
  -v, --version        Print version information and quit

...

ubuntu@labsys:~$
```

Both of the above commands invoke the docker command line program but neither connect to the docker daemon. To test the full docker installation run the docker info command:

```
ubuntu@labsys:~$ docker info

Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Docker Buildx (Docker Inc., v0.8.2-docker)
```

```
compose: Docker Compose (Docker Inc., v2.6.0)
scan: Docker Scan (Docker Inc., v0.17.0)
```

Server:

```
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 0
Server Version: 20.10.17
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: false
  userxattr: true
Logging Driver: json-file
Cgroup Driver: none
Cgroup Version: 1
Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
runc version: v1.1.2-0-ga916309
init version: de40ad0
Security Options:
  seccomp
   Profile: default
  rootless
Kernel Version: 5.13.0-1022-aws
Operating System: Ubuntu 20.04.4 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 3.775GiB
Name: ip-172-31-9-135
ID: 26TG:DAZN:RLQL:OL2A:WPMF:WTOX:5P2N:YMST:T46U:WOQ2:ROH4:PBYN
Docker Root Dir: /home/ubuntu/.local/share/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false
```

WARNING: Running in rootless-mode without cgroups. To enable cgroups in rootless-mode, you need to boot the system in cgroup v2 mode.

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

N.B. You can safely ignore any warnings regarding cgroups - this lab will not be using any container options related to them.

This command causes the docker client to connect to dockerd and output various pieces of information associated with the dockerd daemon.

Look through the output of the info command and answer these questions:

- What logging driver is dockerd using?
- A registry, in docker terms, is a network server from which container images can be downloaded, which "Registry" is docker using as a default?
- The docker daemon (dockerd) is sometimes referred to as the docker server, what is the docker server version?
- In docker, plugins extend the functionality of the basic dockerd services, what log plugins are available?

3. Running Fluentd in a container

In this step we'll run the latest public containerized release of Fluentd from the docker hub registry. Fluentd is a Cloud Native Computing Foundation open source project with source code available on GitHub under the Fluent organization: <https://github.com/fluent>. The Fluentd project community also hosts prebuilt Fluentd docker images on docker hub under the Fluent organization: <https://hub.docker.com/u/fluent>.

As we have seen in previous labs, using Fluentd in most settings will involve running it with a specific configuration file and the necessary third party plugins (if any). By default, containerized applications can only read and write files from within their isolated container filesystem. Thus two applications running in separate containers on the same host will not see each other's files. In particular, `/fluent/etc/fluent.conf` in one container is a completely separate file from `/fluent/etc/fluent.conf` in another container and both are separate from `/fluent/etc/fluent.conf` on the host.

We can, however, run a stock Fluentd container with a custom configuration file by mapping the configuration file from a host path into the container using a container "volume". "Volumes" (in the vocabulary of containers) refer to storage mapped into the container from the outside world. Such storage can be as simple as a directory on the host computer's hard disk, or as complex as a dynamically mounted iSCSI, network attached, block storage device. For our Fluentd container purposes, mapping in a host path containing our Fluentd configuration file will work perfectly.

Let's start by creating a simple Fluentd configuration file on the host that we can later map into our Fluentd container with a volume. Create the following `docker.conf` Fluentd configuration in the `fluent/` folder:

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

ubuntu@labsys:~$ nano ~/fluent/docker.conf && $_

<source>
  @type http
  port 24220
  bind 0.0.0.0
</source>

<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>

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

ubuntu@labsys:~$
```

- What type of input plugins are being used in this configuration?
- What ports will Fluentd be listening on?
- What type of output plugins are being used in this configuration?

Now run the fluent organization Fluentd stable release container with our `~/fluent` directory mapped into the container `/fluent/etc`

directory:

```
ubuntu@labsys:~$ docker run -d -p 24220:24220 -p 24224:24224 \
-v $HOME/ fluent:/fluentd/etc -e FLUENTD_CONF=docker.conf --name fluentd fluent/fluentd:v1.14-1

Unable to find image 'fluent/fluentd:v1.14-1' locally
v1.14-1: Pulling from fluent/fluentd
6097bfa160c1: Pull complete
6f3790a59bdb: Pull complete
10979cec8705: Pull complete
f8fd29633127: Pull complete
4847fc0f8623: Pull complete
Digest: sha256:3749047d00b40bd5854df8dd8dd036aaf4ea3e12c2d4ed3c3ee2b329abfd6ddf
Status: Downloaded newer image for fluent/fluentd:v1.14-1
120249ab5531c4d94c3e2970519329567540b2bc00c7b2396597b4a8c5e04da4

ubuntu@labsys:~$
```

Let's review the parts of the docker command we used to launch our Fluentd instance:

- **docker** - this is the docker command line program
- **run** - the docker run subcommand runs a new container, downloading the container image if it is not found locally
- **-p 24220:24220** - the port switch (-p) maps a port from the host to the container
- **-p 24224:24224** - the port switch can be used multiple times to create multiple port mappings
- **-v \$HOME/ fluent:/fluentd/etc** - the volume switch (-v) maps a path on the host to a path in the container
- **-e FLUENTD_CONF=docker.conf** - the environment switch (-e) creates an environment variable inside the container
- **--name fluentd** - provides a user-defined name for the image, making it easy to refer to this container
- **fluent/fluentd:<tag>** - this is the image to run in organization/image:tag format (tags are often used to pull different versions of the same image)

Our configuration file asks Fluentd to listen on ports 24220 and 24224 which we have mapped to the host in this example so that clients outside of the Fluentd container can reach Fluentd. We have also mapped our fluent directory on the host into the container's /fluent/etc directory. This is the default directory that Fluentd will inspect for configuration files. By setting the FLUENTD_CONF environment variable to docker.conf we have effectively asked the containerized Fluentd to use the /fluent/etc/docker.conf configuration file, which maps to ~/fluent/docker.conf on the host.

Since the docker **-d** flag is daemonizing the container, so you need to use **docker logs** to view its output as it runs in the background.

```
ubuntu@labsys:~$ docker logs fluentd

2021-07-22 18:14:32 +0000 [info]: parsing config file is succeeded path="/fluentd/etc/docker.conf"
2021-07-22 18:14:32 +0000 [info]: gem 'fluentd' version '1.13.2'
2021-07-22 18:14:32 +0000 [warn]: define <match fluent.**> to capture fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2021-07-22 18:14:32 +0000 [info]: using configuration file: <ROOT>
<source>
  @type http
  port 24220
  bind "0.0.0.0"
</source>
<source>
  @type forward
  port 24224
  bind "0.0.0.0"
</source>
<match **>
```

```

    @type stdout
  </match>
</ROOT>
2021-07-22 18:14:32 +0000 [info]: starting fluentd-1.13.2 pid=8 ruby="2.7.3"
2021-07-22 18:14:32 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby", "-Eascii-8bit:ascii-8bit", "/usr/bin/fluentd", "-c", "/fluentd/etc/docker.conf", "-p", "/fluentd/plugins", "-under-supervisor"]
2021-07-22 18:14:33 +0000 [info]: adding match pattern="*" type="stdout"
2021-07-22 18:14:33 +0000 [info]: adding source type="http"
2021-07-22 18:14:33 +0000 [info]: adding source type="forward"
2021-07-22 18:14:33 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2021-07-22 18:14:33 +0000 [info]: #0 starting fluentd worker pid=17 ppid=8 worker=0
2021-07-22 18:14:33 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2021-07-22 18:14:33 +0000 [info]: #0 fluentd worker is now running worker=0
2021-07-22 18:14:33.128930680 +0000 fluent.info: {"pid":17,"ppid":8,"worker":0,"message":"starting fluentd worker pid=17 ppid=8 worker=0"}
2021-07-22 18:14:33.129215627 +0000 fluent.info: {"port":24224,"bind":"0.0.0.0","message":"listening port port=24224 bind=\"0.0.0.0\""}
2021-07-22 18:14:33.130459979 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now running worker=0"}

ubuntu@labsys:~$

```

Examine the containerized Fluentd output.

- Identify the log lines that report docker downloading the Fluentd container image
- Locate the log output that reports the Fluentd configuration file in use
- What version of Fluentd is running in the container?

Now that we have Fluentd running, let's test it! We'll use `curl` to send a simple HTTP message to the containerized Fluentd instance. Recall that we configured Fluentd to listen for HTTP traffic on interface 0.0.0.0 (all interfaces) and port 24220. We can use the `curl` command to POST a simple JSON message to localhost:24220. Run the following command in a new shell:

```

ubuntu@labsys:~$ curl -X POST -d 'json={"hi":"mom"}' http://localhost:24220/test

ubuntu@labsys:~$

```

In the Fluentd terminal, note the stdout output of the posted message:

```

ubuntu@labsys:~$ docker logs fluentd | tail -5

2022-06-15 17:38:58 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-15 17:38:58.258981881 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting fluentd worker pid=16 ppid=7 worker=0"}
2022-06-15 17:38:58.259342336 +0000 fluent.info: {"port":24224,"bind":"0.0.0.0","message":"listening port port=24224 bind=\"0.0.0.0\""}
2022-06-15 17:38:58.260795935 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now running worker=0"}
2022-06-15 17:39:41.169533020 +0000 test: {"hi":"mom"}

ubuntu@labsys:~$

```

- How did the route in the `curl` command (`/test`) manifest in the container output?
- Try this `curl` command: `curl -X POST -d 'json={"hi":"mom"}' http://localhost:24220/test/ing`
- Try this `curl` command:
`curl -X POST -d 'json={"hi":"mom", "severity":"superbad"}' http://localhost:24220/test/ing`

- Experiment with some other curl commands of your own

4. Using Fluentd to process container log data

Now that we have Fluentd up and running in a container, imagine we need to run multiple containers on this machine and that we'd like all of their log output to be collected and processed by the Fluentd instance we are already running.

Recall that in the configuration we used in the previous step we configured the following source:

```
<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>
```

This source forwards traffic inbound on port 24224. Docker's own integrated Fluentd log driver plugin forwards traffic to localhost:24224 by default. This means that as long as we have some Fluentd instance listening on localhost:24224 (which we do), any docker container using the Fluentd log driver plugin will send log output to that container. In the world of containers, any text written to stdout or stderr is piped to docker and delivered to the terminal (if any) and the configured log driver. By default Docker logs container output to files in /var/lib/docker, however we can select a different default log driver and/or specify a specific log driver for each container we run.

The docker run `--log-driver=fluentd` flag tells Docker to send output from that container to Fluentd. Try it by running an nginx container set to log using the Fluentd docker plugin:

```
ubuntu@labsys:~$ docker run -P -d --log-driver=fluentd --name nginx nginx

Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
33847f680f63: Pull complete
dbb907d5159d: Pull complete
8a268f30c42a: Pull complete
b10cf527a02d: Pull complete
c90b090c213b: Pull complete
1f41b2f2bf94: Pull complete
Digest: sha256:8f335768880da6baf72b70c701002b45f4932acae8d574dedfddaf967fc3ac90
Status: Downloaded newer image for nginx:latest
0f21b957ab5a0e58a19b08d0fe6a21294b4590142a67903adf5b9d04dfbedfa3

ubuntu@labsys:~$
```

Let's examine the docker command used:

- `docker` - the docker command line program
- `run` - creates a new container, downloading the image if it is not present
- `-P` - maps all of the container's exposed ports to the host (80 and 443 in this case)
- `-d` - run the container detached, in the background
- `--log-driver=fluentd` - sets the log driver plugin to Fluentd
- `--name nginx` - to name the nginx container with something easy to reference
- `nginx` - specifies the container image to run, in this case using the default registry (hub.docker.com), the default organization (none, aka official), the nginx repository and the default tag ("latest")

List the currently running docker containers with `docker container ls`:

```
ubuntu@labsys:~$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
663b62907d68	nginx	"/docker-entrypoint..."	7 minutes ago	Up 7 minutes
120249ab5531	fluent/fluentd:v1.14-1	"tini -- /bin/entryp..."	9 minutes ago	Up 9 minutes

```

PORTS
NAMES
663b62907d68  nginx          "/docker-entrypoint..."  7 minutes ago  Up 7 minutes
0.0.0.0:49153->80/tcp, :::49153->80/tcp
nginx
120249ab5531  fluent/fluentd:v1.14-1  "tini -- /bin/entryp..."  9 minutes ago  Up 9 minutes
0.0.0.0:24220->24220/tcp, :::24220->24220/tcp, 5140/tcp, 0.0.0.0:24224->24224/tcp, :::24224->24224/tcp  fluentd

ubuntu@labsys:~$

```

The container running the `nginx` image has been assigned the localhost port `49153`, which maps to port 80 in that container. This means that the container should be accessible through `curl` on localhost:49153.

Try to `curl` nginx on `localhost`:

N.B. Make sure you check the ports assignments on your terminal, the port your container receives may be different!

```

ubuntu@labsys:~$ curl http://localhost:49153

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

ubuntu@labsys:~$

```

The `curl` should return the default nginx landing page. The nginx instance inside the container logs all web hits to stdout. Take a look at the Fluentd container's terminal:

```

ubuntu@labsys:~$ docker logs fluentd | tail -5

2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"log":"2022/06/15 17:40:37 [notice] 1#1:
getrlimit(RLIMIT_NOFILE):
1048576:1048576","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","
container_name":"/nginx","source":"stderr"}

```

```

2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"source":"stderr","log":"2022/06/15 17:40:37
[notice] 1#1: start worker
processes","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","contai
ner_name":"/nginx"}
2022-06-15 17:40:37.000000000 +0000 663b62907d68:
{"container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_name":
"/nginx","source":"stderr","log":"2022/06/15 17:40:37 [notice] 1#1: start worker process 32"}
2022-06-15 17:40:37.000000000 +0000 663b62907d68: {"source":"stderr","log":"2022/06/15 17:40:37
[notice] 1#1: start worker process
33","container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_nam
e":"/nginx"}
2022-06-15 17:48:24.000000000 +0000 663b62907d68:
{"container_id":"663b62907d687a931831aeabc65c6591da35acf540f9c300341fc82603f4b5cc","container_name":
"/nginx","source":"stdout","log":"172.17.0.1 - - [15/Jun/2022:17:48:24 +0000] \"GET / HTTP/1.1\" 200
615 \"-\" \"curl/7.68.0\" \"-\"\""}

ubuntu@labsys:~$

```

Fluentd received all logged events (including application-level events reported by NGINX) from the NGINX container via Docker. The Fluentd forward plugin requires log input in a particular format. Fortunately the built-in Docker Fluentd plugin automatically formats container output for Fluentd while enriching it with container metadata, like the container ID and name.

Using information from the lab vm, answer the following questions:

- Where did the logged event come from?
- What is the actual event text?

5. Cleanup

As a final step we'll shutdown and delete all of the containers we started in this lab. We can send the container ids to the `docker container rm` command to remove them, but first you need to stop the containers:

```

ubuntu@labsys:~$ docker container stop fluentd nginx

fluentd
nginx

ubuntu@labsys:~$

```

Now remove the containers using Docker. This removes them entirely, freeing up your system for later:

```

ubuntu@labsys:~$ docker container rm fluentd nginx

fluentd
nginx

ubuntu@labsys:~$

```

Verify that all of the containers have been removed:

```

ubuntu@labsys:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES

ubuntu@labsys:~$

```

Perfect!

In this lab we ran Fluentd in a Docker container and used it to manage log output from other containers. We'll learn a lot more about Fluentd in the labs ahead.

Congratulations, you have completed the Lab.

LFS242 - Cloud Native Logging with Fluentd

Fluentd is a cross platform open-source data collector. It is written primarily in the Ruby programming language with several components developed in C for performance. Fluentd can be used to create a unified logging layer, allowing users to unify data collection and consumption across a wide array of systems and services.

Fluentd can be installed and configured for a variety of environments. Lab 1 is organized into three separate parts, each involves the installation and configuration steps necessary to get started with Fluentd in a different environment:

- A. Linux
- B. Docker
- C. Kubernetes

Lab 1-C – Running Fluentd in a Kubernetes environment

In this lab you will get a chance to run Fluentd using the Kubernetes container orchestration system. This lab is designed to be independent of Lab 1-A and Lab 1-B and can be run on a separate virtual machine, though you can run these steps on same machine as Lab 1-B.

Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the Kubernetes community.

In this lab we will see how Fluentd can play a key role in the log management of containerized applications running under Kubernetes.

Objectives

- Learn how to install a simple Kubernetes test cluster.
- Learn how to run Fluentd on Kubernetes.
- Learn how to process logs from containerized applications with Fluentd on Kubernetes.

1. Prepare the lab system

If you have not already, log in to your lab system and ensure that any Fluentd instances or other programs started in prior labs are stopped. Also stop any other containers you have on the system.

In order to work with Fluentd on Kubernetes we will need to install Kubernetes. Kubernetes requires a container runtime like Docker to operate. If you have not yet installed Docker on your lab system, do so now using the Docker quick-start script:

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

...

ubuntu@labsys:~$
```

If your lab system has less than 4GB of memory you will get faster performance if you increase it to 4GB or more. To increase the memory you will need to shut down the virtual machine (`$ sudo shutdown -h now`), increase the memory through the hypervisor (VMwarePlayer/Workstation/Fusion, Virtual Box, etc.) and then restart the system.

Computer operating systems have traditionally used disk space as a place to swap out less used pages of memory when the system is low on memory. These swap/page files/volumes worked well in older computers which operated with very small amounts of memory. Modern systems are not typically as constrained.

Kubernetes is designed to make efficient use of a computer cluster, so adding more memory is as easy as adding another computer. Disk

access is much slower than actual memory access, so Kubernetes expects memory swap to be disabled.

Turn off memory swapping on your system:

N.B If you are running this lab on a cloud instance, this step may not be necessary:

```
ubuntu@labsys:~$ sudo swapoff -a
ubuntu@labsys:~$
```

The `swapoff` command disables memory swapping to a particular file or device and the `-a` switch disables "all". If your system is not configured for swapping this command does nothing.

If swap is configured it will be reenabled upon your next reboot. To disable swap permanently update your file system table configuration file, commenting out any volumes listed as "swap" (and there may be none):

```
ubuntu@labsys:~$ cat /etc/fstab

# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options>          <dump> <pass>
# / was on /dev/sda1 during installation
UUID=ae4d6013-3015-4619-a301-77a55030c060 /          ext4      errors=remount-ro 0      1
# swap was on /dev/sda5 during installation
UUID=70f4d3ab-c8a1-48f9-bf47-2e35e4d4275f none          swap      sw          0      0

ubuntu@labsys:~$ sudo nano /etc/fstab && cat $_

# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point> <type> <options>          <dump> <pass>
# / was on /dev/sda1 during installation
UUID=ae4d6013-3015-4619-a301-77a55030c060 /          ext4      errors=remount-ro 0      1
# swap was on /dev/sda5 during installation
# UUID=70f4d3ab-c8a1-48f9-bf47-2e35e4d4275f none          swap      sw          0      0

ubuntu@labsys:~$
```

2. Kubernetes Pre-Installation Steps

This lab will have you install Kubernetes using the standard `kubeadm` tool. Before running the Kubernetes installation, you need to install `kubeadm` and its dependencies as well as prepare Docker to interact with Kubernetes.

2a. Install the packages

To begin, update your package index:

```
ubuntu@labsys:~$ sudo apt-get update
```



```
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu focal InRelease
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Reading package lists... Done

ubuntu@labsys:~$
```

Kubernetes packages are installed via https, so install `apt-https-transport` (it may already be installed):

```
ubuntu@labsys:~$ sudo apt-get install -y apt-transport-https

Reading package lists... Done
Building dependency tree
Reading state information... Done
apt-transport-https is already the newest version (2.0.8).
0 upgraded, 0 newly installed, 0 to remove and 61 not upgraded.

ubuntu@labsys:~$
```

Next add the Kubernetes package repository to the local system's package index. To ensure that apt can verify the Kubernetes packages add the repository key:

```
ubuntu@labsys:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

OK

ubuntu@labsys:~$
```

Now add the Kubernetes package URL to the apt sources list:

```
ubuntu@labsys:~$ echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

ubuntu@labsys:~$
```

Great. Now that we have added the Kubernetes package repository update the local package index to include the packages that can be found there:

```
ubuntu@labsys:~$ sudo apt-get update

...

Get:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease [9383 B]
Get:7 https://packages.cloud.google.com/apt kubernetes-xenial/main amd64 Packages [56.5 kB]
Fetched 65.9 kB in 1s (92.7 kB/s)
Reading package lists... Done

ubuntu@labsys:~$
```

You should see apt report updates from `kubernetes-xenial`.

Now we can install the core Kubernetes packages:

- kubelet - the node manager for Kubernetes
- kubeadm - the Kubernetes cluster control plane installer
- kubectl - the Kubernetes admin command line tool

Install the core packages:

N.B. To ensure lab stability the lab instructions will use Kubernetes 1.24.0. You may omit the `=1.24.0-00` parts of the command or substitute it with a later or earlier version.

```
ubuntu@labsys:~$ sudo apt-get install -y kubelet=1.24.0-00 kubeadm=1.24.0-00 kubectl=1.24.0-00

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables kubernetes-cni socat
Suggested packages:
  nftables
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubectl kubelet kubernetes-cni socat
0 upgraded, 8 newly installed, 0 to remove and 61 not upgraded.

...

Setting up conntrack (1:1.4.5-2) ...
Setting up kubectl (1.24.0-00) ...
Setting up ebtables (2.0.11-3build1) ...
Setting up socat (1.7.3.3-2) ...
Setting up cri-tools (1.24.0-00) ...
Setting up kubernetes-cni (0.8.7-00) ...
Setting up kubelet (1.24.0-00) ...
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service →
/lib/systemd/system/kubelet.service.
Setting up kubeadm (1.24.0-00) ...
Processing triggers for man-db (2.9.1-1) ...

ubuntu@labsys:~$
```

2b. Prepare the container runtime

Next, you need to complete the following steps to enable Docker as the container runtime for Kubernetes:

- Configure Docker to use systemd as the cgroup driver
- Install `cri-dockerd`, the interface that allows Docker to communicate with the Kubelet.

Create a file at `/etc/docker/daemon.json` with the following contents, then restart Docker:

```
ubuntu@labsys:~$ sudo mkdir -p /etc/docker

ubuntu@labsys:~$ sudo nano /etc/docker/daemon.json && cat $_

{
  "exec-opts": ["native.cgroupdriver=systemd"]
}
```

```
ubuntu@labsys:~$ sudo systemctl restart docker

ubuntu@labsys:~$
```

Next, install the `cri-dockerd` interface, which consists of a binary, a systemd unit file, and a socket definition.

Download the latest `cri-dockerd` binary:

```
ubuntu@labsys:~$ wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.2.2/cri-dockerd-0.2.2.amd64.tgz

...

ubuntu@labsys:~$
```

Unpack the contents of the tarball and move them to your machine's `/usr/bin` directory:

```
ubuntu@labsys:~$ tar xvf cri-dockerd-0.2.2.amd64.tgz

cri-dockerd/
cri-dockerd/cri-dockerd

ubuntu@labsys:~$ sudo mv cri-dockerd /usr/bin/

ubuntu@labsys:~$
```

Next, set up the systemd unit which actually runs the `cri-dockerd` binary:

```
ubuntu@labsys:~$ sudo nano /etc/systemd/system/cri-docker.service && sudo cat $_
```

```
[Unit]
Description=CRI Interface for Docker Application Container Engine
Documentation=https://docs.mirantis.com
After=network-online.target firewall.service docker.service
Wants=network-online.target
Requires=cri-docker.socket

[Service]
Type=notify
ExecStart=/usr/bin/cri-dockerd/cri-dockerd --container-runtime-endpoint fd:// --network-plugin=cni -
-cni-bin-dir=/opt/cni/bin --cni-cache-dir=/var/lib/cni/cache --cni-conf-dir=/etc/cni/net.d
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

StartLimitBurst=3
StartLimitInterval=60s

LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

TasksMax=infinity
Delegate=yes
```

```
KillMode=process

[Install]
WantedBy=multi-user.target
```

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

This systemd unit includes options that instruct `cri-dockerd` to use a container network interface (CNI) plugin to provide IP addresses to workloads orchestrated by Kubernetes.

After that, set up the socket which the Kubelet uses to communicate with the `cri-dockerd` service:

```
ubuntu@labsys:~$ sudo nano /etc/systemd/system/cri-docker.socket && sudo cat $_
```

```
[Unit]
Description=CRI Docker Socket for the API
PartOf=cri-docker.service

[Socket]
ListenStream=%t/cri-dockerd.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker

[Install]
WantedBy=sockets.target
```

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

Finally, enable both the `cri-dockerd` socket and service:

```
ubuntu@labsys:~$ sudo systemctl daemon-reload

ubuntu@labsys:~$ sudo systemctl enable cri-docker.service

Created symlink /etc/systemd/system/multi-user.target.wants/cri-docker.service →
/etc/systemd/system/cri-docker.service.

ubuntu@labsys:~$ sudo systemctl enable --now cri-docker.socket

Created symlink /etc/systemd/system/sockets.target.wants/cri-docker.socket →
/etc/systemd/system/cri-docker.socket.

ubuntu@labsys:~$
```

The Kubelet can now communicate with Docker through the `cri-dockerd` interface.

Great, we are ready to install a Kubernetes cluster.

3. Configure a Kubernetes cluster

The `kubeadm` command has an `init` subcommand we can use to setup a Kubernetes cluster on our lab system. Run `kubeadm init` with the `--kubernetes-version 1.24.0` option to declare which version of Kubernetes to use and `--cri-socket=unix:///run/cri-dockerd.sock` option to declare which container runtime to use:

```

ubuntu@labsys:~$ sudo kubeadm init --kubernetes-version 1.24.0 --cri-socket=unix:///run/cri-dockerd.sock

[init] Using Kubernetes version: v1.24.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [labsys.kubernetes.kubernetes.default.kubernetes.default.svc.kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.9.34]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [labsys localhost] and IPs [172.31.9.34 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [labsys localhost] and IPs [172.31.9.34 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 12.503704 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node labsys as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node labsys as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: y70mwc.6rm1i1hccyt42uuq
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

```

```
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client
certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the root user, you can run:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.31.9.34:6443 --token y70mwc.6rm1i1hccyt42uuq \
--discovery-token-ca-cert-hash
sha256:6db8ab9bcf87aa456bee75d86fd3d7ba6203225b935198ef02fef5ca4452cf60

ubuntu@labsys:~$
```

The initialization process creates a Kubernetes control plane node, which runs all of the components that make up a minimal but functional Kubernetes deployment. We will follow the suggestions `kubeadm init` makes after the installation completes in the coming steps.

To communicate with the cluster we will need credentials. Kubeadm generates administrator credentials and stores them in the file `/etc/kubernetes/admin.conf`. Per the kubeadm comments, copy the admin credentials over to your user account so that you can administer the cluster without sudo:

```
ubuntu@labsys:~$ mkdir -p $HOME/.kube

ubuntu@labsys:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

ubuntu@labsys:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

ubuntu@labsys:~$
```

To make kubectl suggest commands and resource names to us, enable bash completion on your machine:

```
ubuntu@labsys:~$ sudo apt install bash-completion

...

ubuntu@labsys:~$ source <(kubectl completion bash)

ubuntu@labsys:~$ echo "source <(kubectl completion bash)" >> ~/.bashrc

ubuntu@labsys:~$
```

Now use the kubectl tool to get the status of your cluster:

```
ubuntu@labsys:~$ kubectl cluster-info

Kubernetes control plane is running at https://labsys:6443
CoreDNS is running at https://labsys:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

ubuntu@labsys:~$
```

Great, Kubernetes is running. There are still a few things we need to do before using our cluster.

Kubernetes control plane nodes initialized by `kubeadm init` are not expected to run regular workloads. By default, kubeadm configures a control plane node with taints, or reservations, which prevent regular application containers from being able to run on it. Since we will use our one node for all kinds of workloads, remove the taints by removing the `node-role.kubernetes.io/master` and `node-role.kubernetes.io/control-plane` keys with `kubectl taint`:

```
ubuntu@labsys:~$ kubectl taint nodes --all node-role.kubernetes.io/master- node-
role.kubernetes.io/control-plane-

node/ubuntu untainted

ubuntu@labsys:~$
```

Be careful not to skip the `-` at the end of each key. The minus specified that the taint should be removed as opposed to added.

As a final step we need to add a software defined networking solution to support our container networking needs. Kubernetes supports a wide range of networking solutions through the popular CNI, Container Networking Interface. We'll use the Weave CNI solution on our cluster. Now that we have a Kubernetes cluster running, we can actually install Weave as a container workload. Run the following command:

```
ubuntu@labsys:~$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |
base64 | tr -d '\n')"

serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created

ubuntu@labsys:~$
```

This command may generate a deprecation warning about `kubectl version --short`. The manifest is still correctly retrieved and this warning can be ignored.

The above command installs weave-net as a DaemonSet after creating its required security roles and bindings. Weave Net deploys all of its components using a single declarative specification file in YAML format.

Kubernetes packages containers in "pods". List all of the pods running on your Kubernetes cluster:

```
ubuntu@labsys:~$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-6d4b75cb6d-15hs6	1/1	Running	0	4m20s
kube-system	coredns-6d4b75cb6d-shf17	1/1	Running	0	4m20s
kube-system	etcd-labsys	1/1	Running	0	4m25s
kube-system	kube-apiserver-labsys	1/1	Running	0	4m25s
kube-system	kube-controller-manager-labsys	1/1	Running	0	4m25s
kube-system	kube-proxy-nnc9h	1/1	Running	0	4m21s
kube-system	kube-scheduler-labsys	1/1	Running	0	4m25s
kube-system	weave-net-gflnt	2/2	Running	1 (46s ago)	53s

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

If you see something like the output above, your cluster is ready to use. If, for example, the `coredns` pods are not yet running, you may need to give the `weave-net` pods more time to start. DNS will not be enabled until the node is ready, which happens after a network plugin is configured.

4. Creating a Fluentd ConfigMap

There are many platform related tools administrators want or need to run on every node in their environment. For example, the Weave SDN agent has to be running on every node in our Kubernetes cluster for containers to be able to communicate with each other. Kubernetes has a special deployment type, known as a DaemonSet, which runs one copy of a given pod on potentially every system in the cluster.

DaemonSets are also a great way to deploy Fluentd. If you want to have Fluentd running as a log forwarding agent on many nodes in your cluster a Fluentd DaemonSet is the right choice. DaemonSets operate at the control plane level so when new nodes are added to the cluster, all of the DaemonSet pods are automatically launched on the node. This greatly simplifies cluster maintenance.

In this step we will deploy Fluentd using a DaemonSet. Before we begin consider the Fluentd configuration file. In prior labs we passed Fluentd a configuration file from the local host's filesystem. This is not going to work well in a Kubernetes cluster. Imagine copying a configuration file to hundreds of hosts and then trying to audit and maintain all of them. Not a great prospect.

Fortunately Kubernetes provides a resource known as ConfigMaps. A ConfigMap is a control plane managed resource that pods can access from anywhere in the cluster. ConfigMaps can contain any kind of typical configuration data including, data used for environment variables, passed on command lines, or stored in configuration files mapped into containers. If we save our Fluentd configuration file as a ConfigMap we can use it with every Fluentd pod in the cluster.

To initialize our ConfigMap we can start by simply creating a normal Fluentd configuration file:

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

```
ubuntu@labsys:~$ nano ~/fluent/fluentd-kube.conf && cat $_
```

```
<source>
  @type http
  port 24220
  bind 0.0.0.0
</source>

<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>

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



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

Next, create a Kubernetes manifest that constructs a ConfigMap using the data in the config file. We can use the `kubectl create configmap` command to do this. Try it:

```
ubuntu@labsys:~$ kubectl create configmap fluentd-config --from-file ~/fluent/fluentd-kube.conf
configmap/fluentd-config created

ubuntu@labsys:~$
```

`kubectl` formulates an API request to create a ConfigMap containing the contents of the `~/fluent/fluentd-kube.conf` file, then sends it to the API server.

Use `kubectl get` to list your config map, then use `kubectl describe` to get a summary of its contents:

```
ubuntu@labsys:~$ kubectl get configmap

NAME              DATA  AGE
fluentd-config    1      6s
kube-root-ca.crt  1      6m22s

ubuntu@labsys:~$ kubectl describe configmap fluentd-config

Name:          fluentd-config
Namespace:     default
Labels:        <none>
Annotations:   <none>

Data
====
fluentd-kube.conf:
----
<source>
  @type http
  port 24220
  bind 0.0.0.0
</source>

<source>
  @type forward
  port 24224
  bind 0.0.0.0
</source>

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

BinaryData
====

Events:  <none>

ubuntu@labsys:~$
```

Looks good. The config file contents are stored under the ConfigMap's `fluentd-kube.conf` data key. Now we can run a Fluentd DaemonSet that uses the ConfigMap for its configuration.

5. Creating a Fluentd DaemonSet

With Kubernetes running and our ConfigMap created we're ready to craft a Fluentd DaemonSet manifest. Kubernetes manifests are coded as YAML files (<https://yaml.org/>). Create the following Kubernetes manifest:

```
ubuntu@labsys:~$ nano ~/fluent/fluentd-ds.yaml && cat $_
```

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-ds
  labels:
    k8s-app: fluentd-logging
    version: v1
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-logging
  template:
    metadata:
      labels:
        k8s-app: fluentd-logging
        version: v1
    spec:
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
        - key: node-role.kubernetes.io/control-plane
          effect: NoSchedule
      terminationGracePeriodSeconds: 30
      containers:
        - name: fluentd-ds
          image: fluent/fluentd:v1.14.6-1.1
          resources:
            limits:
              memory: 200Mi
          env:
            - name: FLUENTD_CONF
              value: "fluentd-kube.conf"
          volumeMounts:
            - name: fluentd-conf
              mountPath: /fluentd/etc
      volumes:
        - name: fluentd-conf
          configMap:
            name: fluentd-config
```

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

While this is not a Kubernetes course, it's worth examining the various components of the DaemonSet specification we have defined for our Fluentd Kubernetes pods. We'll examine the manifest block by block.

The first block includes general Kubernetes configuration used with all Kubernetes resource types:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-ds
  labels:
    k8s-app: fluentd-logging
    version: v1
```

The `apiVersion` and `kind` keys together tell Kubernetes what type of resource we are creating. The `metadata` key provides descriptive information for our resource. The `name` is required and `labels` allow us to add arbitrary identifiers to the resource that we can use when filtering large lists of resources in big clusters.

The next section is the DaemonSet specification:

```
spec:
  template:
    metadata:
      labels:
        k8s-app: fluentd-logging
        version: v1
```

This is the beginning of the specification for our DaemonSet. A DaemonSet maintains a template for a pod that the cluster will create on (potentially) every node. The pod template can have labels but not a name, since the template will be used to create many pods. DaemonSet pod names are generated automatically and given the DaemonSet's name as a prefix.

The next block defines tolerations:

```
tolerations:
- key: node-role.kubernetes.io/master
  effect: NoSchedule
- key: node-role.kubernetes.io/control-plane
  effect: NoSchedule
```

When we configured our Kubernetes cluster we removed the all taints from our one and only control plane node. In a normal cluster the control plane would not run application pods, however we probably do want to support log forwarding on our control plane nodes. We can allow operational work loads, like Fluentd, to run on all nodes in the cluster by "tolerating" any taints that are defined. In the code above we specify that our DaemonSet pods should be allowed to run on nodes with the taints having the `node-role.kubernetes.io/master` and `node-role.kubernetes.io/control-plane` keys with the effect `NoSchedule`. This is the standard taint assigned to control plane nodes with most installation tools.

Next we set a termination grace period for the pods in the DaemonSet:

```
terminationGracePeriodSeconds: 30
```

Normally Kubernetes will kill a container that does not exit within 10 seconds of the initial SIGTERM. As we have seen, Fluentd can take 10-15 seconds to shutdown after the cancel (^C) signal is sent. This setting extends the window Kubernetes allows for container shutdown.

Next we specify the container that will run in the pod:

```
containers:
- name: fluentd-ds
  image: fluent/fluentd:v1.14.6-1.1
  resources:
    limits:
```

```
memory: 200Mi
```

This block specifies the container image to run, `fluent/fluentd:v1.14.6-1.1`, which will automatically be pulled from DockerHub. It also tells Kubernetes that our container should be limited to 200MB of memory, keeping the logging system from eating up too much of the node's memory. Limits can not be exceeded.

N.B. To ensure lab stability the image tag has been set to the latest available as of this publication. You may substitute the tag with the latest Fluentd version available, viewable on <https://hub.docker.com/r/fluent/fluentd>

The next block of YAML sets up the Fluentd configuration file:

```
env:
  - name: FLUENTD_CONF
    value: "fluentd-kube.conf"
volumeMounts:
  - name: fluentd-conf
    mountPath: /fluentd/etc
```

Here we declare an environment variable `FLUENTD_CONF` set to `fluentd-kube.conf`. Then we mount a volume named `fluentd-conf` on the container path `/fluentd/etc`, the default location where Fluentd looks for its configuration file. This environment variable must be declared in full uppercase.

The last block defines the `fluentd-conf` volume mentioned above:

```
volumes:
  - name: fluentd-conf
    configMap:
      name: fluentd-config
```

In the volume specification above we use the previously created ConfigMap to populate the `fluentd-conf` volume. Since the ConfigMap specifies the contents of a `fluentd-kube.conf` file, the container mount path `/fluentd/etc` will be populated with the ConfigMap's `fluentd-kube.conf` file. If you want to update the Fluentd configuration, you can create a new ConfigMap (with a name like `fluentd-config-v2`) with the updated file and change the name in this section. Doing so will automatically trigger an update to the Fluentd DaemonSet pods.

N.B. You can also update the existing ConfigMap, but that may affect your ability to roll back to a known-good version of the previous configuration if required. You will need to restart the Fluentd processes in every pod manually by sending an appropriate signal to the processes or deleting the pods (which will be automatically recreated).

Now, create the DaemonSet:

```
ubuntu@labsys:~$ kubectl apply -f ~/fluent/fluentd-ds.yaml
daemonset.apps/fluentd-ds created
ubuntu@labsys:~$
```

List the DaemonSets on your system:

```
ubuntu@labsys:~$ kubectl get ds
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
fluentd-ds	1	1	1	1	1	<none>	4s

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

Since we did not specify a namespace, Kubernetes runs the DaemonSet pods in the configured namespace `default`. In a production setting you might run Fluentd in the `kube-system` namespace or another namespace.

Within a few moments, a Fluentd pod prefixed `fluentd-ds` should be available. List the pods in the default namespace:

```
ubuntu@labsys:~$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
fluentd-ds-sj2xx	1/1	Running	0	8s

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

To see the activity of the Fluentd pod, use the `kubectl logs` command to view its logs:

```
ubuntu@labsys:~$ kubectl logs -l k8s-app=fluentd-logging
```

```
2022-06-14 16:15:38 +0000 [info]: starting fluentd-1.3.2 pid=7 ruby="2.5.2"
2022-06-14 16:15:38 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby", "-Eascii-8bit:ascii-8bit", "/usr/bin/fluentd", "-c", "/fluentd/etc/fluentd-kube.conf", "-p",
"/fluentd/plugins", "--under-supervisor"]
2022-06-14 16:15:38 +0000 [info]: gem 'fluentd' version '1.3.2'
2022-06-14 16:15:38 +0000 [info]: adding match pattern="*" type="stdout"
2022-06-14 16:15:38 +0000 [info]: adding source type="http"
2022-06-14 16:15:38 +0000 [info]: adding source type="forward"
2022-06-14 16:15:38 +0000 [info]: #0 starting fluentd worker pid=17 ppid=7 worker=0
2022-06-14 16:15:38 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2022-06-14 16:15:38 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-14 16:15:38.614288045 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now running worker=0"}
```

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

Referring to the pod by the label `k8s-app=fluentd-logging` allows you to see the logs for the pod without knowing any specific pod names. If you have multiple DaemonSet pods, the logs from all pods will be displayed. Based on the output, Fluentd is running and so far everything looks good!

6. Testing the Fluentd DaemonSet

Try to curl the Fluentd HTTP port on localhost:

```
ubuntu@labsys:~$ curl -X POST -d 'json={"Hi":"Mom"}' http://localhost:24220/fluentd
```

```
curl: (7) Failed to connect to localhost port 24220: Connection refused
```

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

This does not work because applications running within Kubernetes pods are separated from the host's network, instead running on the configured "pod network", in our case Weave. To interact with this Fluentd instance we either need to rerun it with its container ports mapped to host ports or access it from another pod within the cluster's pod network.

To communicate with a pod on the pod network we will need to get its IP address. Get a listing of pods with `kubectl get pods` and

the `-o wide` option to find the Fluentd pod IP:

```
ubuntu@labsys:~$ kubectl get pod -o wide

NAME                READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE
READINESS GATES
fluentd-ds-dzml9    1/1     Running   0           100s   10.32.0.4    labsys     <none>
<none>

ubuntu@labsys:~$
```

Now use `kubectl run` to create a test client pod. Run an interactive pod to test our Fluentd DaemonSet with:

```
ubuntu@labsys:~$ kubectl run test-client --image centos:8 -it

If you don't see a command prompt, try pressing enter.

[root@test-client /]#
```

The command above runs a new pod named `test-client` with a container that uses the `centos:8` container image. The container launches with the `-i` and `-t` switches. The `t` creates a tty in the container and the `i` redirects the current shell's input to the container so that we can issue commands from within it.

From inside the container try to post some data to Fluentd, using the Fluentd pod's IP address retrieved by `kubectl get pods -o wide`:

```
[root@test-client /]# curl -X POST -d 'json={"Hi":"Mom"}' http://10.32.0.4:24220/test

[root@test-client /]#
```

Looks good. Exit the test container:

```
[root@test-client /]# exit

exit
Session ended, resume using 'kubectl attach test-client -c test-client -i -t' command when the pod
is running

ubuntu@labsys:~$
```

As the exit message reports, the pod is still running after we leave the shell and we can reattach if we like.

Now redisplay the log output of the Fluentd pod:

```
ubuntu@labsys:~$ kubectl logs -l k8s-app=fluentd-logging | tail -5

2022-06-14 16:15:38 +0000 [info]: #0 starting fluentd worker pid=17 ppid=7 worker=0
2022-06-14 16:15:38 +0000 [info]: #0 listening port port=24224 bind="0.0.0.0"
2022-06-14 16:15:38 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-14 16:15:38.614288045 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-14 16:16:25.081212507 +0000 test: {"Hi":"Mom"}

ubuntu@labsys:~$
```

Great! Our Fluentd pod is operating perfectly.

To make it easier to communicate with the Fluentd pods (whose IP addresses can change, especially if they are reconfigured), it is ideal to create a corresponding Service for the Fluentd DaemonSet pods. The Service's job is to provide a single, stable network identity which routes traffic to a set of pods it represents.

Create the service definition:

```
ubuntu@labsys:~$ nano ~/fluent/fluentd-ds-svc.yaml && cat $_

apiVersion: v1
kind: Service
metadata:
  labels:
    app: fluentd
    name: fluentd
spec:
  ports:
    - name: 80-24220
      port: 80                      # The service will listen for requests on this port
      protocol: TCP
      targetPort: 24220             # The service sends traffic it receives to these container ports on
the target pods
  selector:
    k8s-app: fluentd-logging      # Establishes which pods the service submits traffic to
    type: ClusterIP

ubuntu@labsys:~$
```

When created, this service will have a virtual IP allocated to it. That virtual IP will listen on port 80 and then submit traffic to the Fluentd container at port 24220 (which is where we have a http endpoint listening for traffic).

Apply the service and check what IP address it gets:

```
ubuntu@labsys:~$ kubectl apply -f ~/fluent/fluentd-ds-svc.yaml

service/fluentd created

ubuntu@labsys:~$ kubectl get service

NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
fluentd       ClusterIP   10.99.73.8   <none>        80/TCP     11s
kubernetes    ClusterIP   10.96.0.1    <none>        443/TCP    70m

ubuntu@labsys:~$
```

Now use `kubectl exec` to test the cluster virtual IP (a.k.a. Cluster IP) at port 80:

N.B. Be sure to use the Cluster IP shown by your `kubectl get service` output.

```
ubuntu@labsys:~$ kubectl exec test-client -- curl -X POST -d '{"From":"Service"}'
http://10.99.73.8:80/test

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100    17      0     0  100    17      0   5666  --:--:--  --:--:--  --:--:-- 17000
```

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

In addition to the virtual IP, the service also receives a DNS name from `coredns` which is formatted as:
`servicename.namespace.svc.clusterdomain`.

Try the test again, but this time with the DNS name. In this case, the DNS name is `fluentd.default.svc.cluster.local`:

```
ubuntu@labsys:~$ kubectl exec test-client -- curl -X POST -d 'json={"Via":"DNS"}'  
http://fluentd.default.svc.cluster.local:80/test
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	17	0	0	100	17	0	680
							--:--:-- --:--:-- --:--:-- 708

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

Finally, confirm that Fluentd has been receiving the requests you have been making:

```
ubuntu@labsys:~$ kubectl logs -l k8s-app=fluentd-logging | tail -5
```

```
2022-06-14 16:15:38 +0000 [info]: #0 fluentd worker is now running worker=0  
2022-06-14 16:15:38.614288045 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now  
running worker=0"}  
2022-06-14 16:16:25.081212507 +0000 test: {"Hi":"Mom"}  
2022-06-14 16:25:54.975264042 +0000 test: {"From":"Service"}  
2022-06-14 16:28:02.070436227 +0000 test: {"Via":"DNS"}
```

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

Now your workloads can easily access Fluentd on Kubernetes! Fluentd pod configurations vary widely with the needs of each Kubernetes cluster but our example here demonstrates a number of key features and configuration approaches.

7. Cleanup

To return the cluster to its initial state, delete the `test-client` pod and Fluentd DaemonSet:

```
ubuntu@labsys:~$ kubectl delete pod test-client
```

```
pod "test-client" deleted
```

```
ubuntu@labsys:~$ kubectl get daemonset
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
fluentd-ds	1	1	1	1	1	<none>	7m

```
ubuntu@labsys:~$ kubectl delete ds fluentd-ds
```

```
daemonset.apps "fluentd-ds" deleted
```

```
ubuntu@labsys:~$ kubectl get daemonset,pod
```

```
No resources found in default namespace.
```

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

Optionally, you may uninstall Kubernetes using the following commands:


```
ubuntu@labsys:~$ sudo kubeadm reset
```

```
[reset] WARNING: Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.
[reset] Are you sure you want to proceed? [y/N]: y
[preflight] Running pre-flight checks
W0728 16:00:56.719433 47758 removeetcdmember.go:79] [reset] No kubeadm config, using etcd pod spec
to get data directory
[reset] No etcd config found. Assuming external etcd
[reset] Please, manually reset etcd to prevent further issues
[reset] Stopping the kubelet service
[reset] Unmounting mounted directories in "/var/lib/kubelet"
[reset] Deleting contents of config directories: [/etc/kubernetes/manifests /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf
/etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf
/etc/kubernetes/scheduler.conf]
[reset] Deleting contents of stateful directories: [/var/lib/kubelet /var/lib/dockershim
/var/run/kubernetes /var/lib/cni]
```

The reset process does not clean CNI configuration. To do so, you must remove /etc/cni/net.d

The reset process does not reset or clean up iptables rules or IPVS tables.
If you wish to reset iptables, you must do so manually by using the "iptables" command.

If your cluster was setup to utilize IPVS, run `ipvsadm --clear` (or similar)
to reset your system's IPVS tables.

The reset process does not clean your kubeconfig files and you must remove them manually.
Please, check the contents of the `$HOME/.kube/config` file.

```
ubuntu@labsys:~$ sudo rm -r /etc/cni/net.d
```

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

Congratulations, you have completed the Lab.

LFS242 - Cloud Native Logging with Fluentd

Lab 1-A – Installing and configuring Fluentd on Linux

- What types (plugins) are used in the sample config file above?
 - in_forward, in_http, in_unix, monitoring_agent, filter_record_transformer, filter_grep, debug_agent, out_file, out_stdout, out_forward, out_copy, out_null
- What is the path to the config file Fluentd is using?
 - /etc/fluent/fluent.conf
- What port is the Fluentd "forwarding server" using?
 - 24224 by default, used when no `port` parameter is specified
- How many plugins are registered for use? Are all of these mentioned in your config file?
 - Six total are in uncommented directives
- Issue the debug message several more times; what is different about each message?
 - The timestamp, which bears microsecond precision

Lab 1-B – Running Fluentd in a Docker environment

- What logging driver is dockerd using?
 - json_file
- A registry, in docker terms, is a network server from which container images can be downloaded, which "Registry" is docker using as a default?
 - <https://index.docker.io/v1/>
- The docker daemon (dockerd) is sometimes referred to as the docker server, what is the docker server version?
 - 20.10.7
- In docker, plugins extend the functionality of the basic dockerd services, what log plugins are available?
 - awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
- What type of input plugins are being used in this configuration?
 - in_forward and in_http
- What ports will Fluentd be listening on?
 - Port 24220 for HTTP traffic and 24224 for Fluentd traffic
- What type of output plugins are being used in this configuration?
 - out_stdout
- Identify the log lines that report docker downloading the Fluentd container image
 - Lines like `b73585fcd1ae: Pull complete` indicate that parts of the Fluentd container image are being downloaded
- Locate the log output that reports the Fluentd configuration file in use
 - `2021-07-22 18:14:32 +0000 [info]: parsing config file is succeeded`
`path="/fluentd/etc/docker.conf"`
- What version of Fluentd is running in the container?
 - Version 1.13.2

- How did the route in the curl command (`/test`) manifest in the container output?
 - As a plain JSON map, resulting in `{"hi":"mom"}`
- Try this curl command: `curl -X POST -d 'json={"hi":"mom"}' http://localhost:24220/test/ing`
 - ``2021-07-22 18:19:12.022070131 +0000 test.ing: {"hi":"mom"}```
- Try this curl command:
 `curl -X POST -d 'json={"hi":"mom", "severity":"superbad"}' http://localhost:24220/test/ing`
 - `2021-07-22 18:19:29.542448298 +0000 test.ing: {"hi":"mom","severity":"superbad"}`
- Where did the logged event come from?
 - The nginx container.
- What is the actual event text?
 - `172.17.0.1 - - [22/Jul/2021:18:21:54 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.68.0" "-"`

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 `'
  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 `'
  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 `'
  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 to 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 forcibly
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 an 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.

LFS242 - Cloud Native Logging with Fluentd

Lab 2 – Configuring Fluentd

- Were there any plugins loaded?
 - No. Since there were no directives written in the configuration file, no plugins were loaded.
- Can Fluentd perform any functions at this point?
 - No. Without any directives configuring any plugins for event processing, this configuration is effectively nonfunctional.
- What would the benefit of running a blank configuration file be?
 - If Fluentd is able to start at all, it shows that all of its dependencies are fulfilled and that Fluentd itself is healthy.
- What circumstances would it be appropriate to run Fluentd with a specific configuration file like this be?
 - Development
 - Debugging
 - Testing a Fluentd deployment on a specific host
 - Reusing known good configuration files originating from other systems
- Where would using an environment variable to set the configuration file be most appropriate?
 - Containers and pre-packaged deployments benefit the most from configuration files defined in environment variables.
- Would it be possible to run more than one Fluentd instance on the same machine using more than one terminal?
 - Yes, as long as their ports do not conflict on the host.
- What happens during a restart event?
 - Fluentd receives the signal, terminates the worker and then restarts with the new configuration.
- How many sources are loaded now?
 - 2 sources are now loaded: one forward and the other http.
- Based on the above configuration, what files will be tailed under the /tmp/fluent directory?
 - All of them, since a wildcard was put into place.
- What does the tail plugin tell you about the way this plugin will work?
 - It will use the tail function to create an event from an entry within a specific file.
- Before it failed to reload, how many plugins were loaded after the SIGHUP was sent?
 - Two: The in_forward and in_http plugins were after Fluentd was restarted and before it crashed.
- What is different about this crash compared to the previous one?
 - It is much shorter, barely identifying that the optional OJ json parser was missing before crashing.
- Were any plugins loaded before this error was generated?
 - Not this time, no.
- Did you get any indication that any files are being tracked under the /tmp/fluent/ directory?
 - Yes, Fluentd reported that it is tailing the following files: `/tmp/fluent/lab2.tail.pos` and `/tmp/fluent/applogs`
- What plugins are loaded when this Fluentd configuration file is used?
 - in_forward, in_http, in_tail, parse_none
- Are these plugins loaded in any particular order?
 - Yes, the plugins are loaded in the order that they are listed in the configuration file as long as Fluentd successfully parses it at runtime.
- Is there any benefit to being able to run Fluentd configurations with only `<source>` directives?

- Only for developing and testing new inputs, otherwise no.
- What plugin will be called?
 - out_stdout
- From the plugin that's been selected, what can you infer will happen to matching events?
 - Events should be sent to the host's standard output.
- What event tags will be caught by this `<match>` directive?
 - All event tags, since a double wildcard was used.
- How does the new event abide by the new configurations?
 - Events printed to STDOUT now have the "event_tag" key-value pair inside them, which was the name given in the configuration.
- Try to use a combination of the other wildcard types in another `<match>` directive
 - An example:

```
<match *.lab2 lfs242.*>
  @type stdout
</match>
```

- How can you change the in_tail `<source>` directive to work with the `<match mod2.*>` directive?
 - The tail `<source>` directive will need to be changed to send a tag with mod2.*

```
<source>
  @type tail
  path /tmp/fluent/*.log
  <parse>
    @type none
  </parse>
  tag mod2.tailed
  pos_file /tmp/fluent/lab2.tail.pos
</source>
```


LFS242 - Cloud Native Logging with Fluentd

Lab 3 – Extending Fluentd with Plugins: Working with Input and Output Plugins

The core of Fluentd's power and functionality comes from its plugins. In the previous lab, `<source>` and `<match>` directives were set up using a selection of core plugins that come with Fluentd. While functional for many use cases, they require specific configurations in order to work efficiently with other programs. This is where Fluentd's pluggable ecosystem shines: For inputs, outputs and even filtering cases that need to work with application specific syntax, users are free to add functionality with a single plugin that replaces otherwise complicated configurations.

One of the main advantages of plugins is that the log processing configuration for the intended application is abstracted. This means knowledge of application-specific syntax for is not required for interaction, so enabling logging does not require a subject matter expert for that application.

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

- Install 3rd party plugins for Fluentd
- Configure Fluentd to work with a third party application
- Establish a simple application-to-application logging pipeline between MongoDB and Nginx

0. Prepare the lab system

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

Install Docker with the quick installation script:

```
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.

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. Preparing a MongoDB database

This lab will use MongoDB as a destination datastore for Fluentd events. MongoDB is an open source, document-oriented NoSQL database designed with both scalability and developer agility in mind. MongoDB was selected for this example as it is a JSON key-value store that will work well with Fluentd.

Open a new terminal and prepare a directory on the host to act as a locally mounted volume for the MongoDB container that will store this lab's data:

```
ubuntu@labsys:~$ mkdir -p /tmp/mongodb/lab3
ubuntu@labsys:~$
```

Use Docker to run a MongoDB container, using the `-d` switch to run the container in the background:

```
ubuntu@labsys:~$ sudo docker run -d --name mongodb -v /tmp/mongodb/lab3:/data/db
bitnami/mongodb:4.2.21

...

13e144044db5a40b21eb68f70447e133aca0843556357fe75f607382ab8531f
ubuntu@labsys:~$
```

There is some information you need about the MongoDB instance, particularly the network connection information.

Use `docker logs` to see the MongoDB information:

```
ubuntu@labsys:~$ sudo docker logs mongodb --tail 5

2022-06-20T15:35:01.851+0000 I STORAGE [initandlisten] Flow Control is enabled on this deployment.
2022-06-20T15:35:01.853+0000 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/bitnami/mongodb/data/db/diagnostic.data'
2022-06-20T15:35:01.855+0000 I NETWORK [listener] Listening on /opt/bitnami/mongodb/tmp/mongodb-27017.sock
2022-06-20T15:35:01.855+0000 I NETWORK [listener] Listening on 0.0.0.0
2022-06-20T15:35:01.855+0000 I NETWORK [listener] waiting for connections on port 27017

ubuntu@labsys:~$
```

Port 27017 is being used by MongoDB, but you still need the IP address since this MongoDB instance is running on the Docker network.

Use `docker inspect`, get the IP address of the MongoDB instance. This will be used later to connect Fluentd to MongoDB:

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

172.17.0.2

ubuntu@labsys:~$
```

Using the IP of the container, use `curl` to send an HTTP request to the MongoDB instance. Remember to use the MongoDB port for the curl command, since that is what is currently open:

```
ubuntu@labsys:~$ curl $(sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongodb):27017
```

It looks like you are trying to access MongoDB over HTTP on the native driver port.

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

MongoDB expects client connections to hook to port 27017, so any traffic other than a database client will be bounced. However, it does return a response, so it means that MongoDB can be reached.

Check the MongoDB log again and you should see a corresponding error being generated:

```
ubuntu@labsys:~$ sudo docker logs mongodb --tail 5
```

```
2022-06-20T15:35:01.855+0000 I NETWORK [listener] Listening on 0.0.0.0
2022-06-20T15:35:01.855+0000 I NETWORK [listener] waiting for connections on port 27017
2022-06-20T15:35:40.333+0000 I NETWORK [listener] connection accepted from 172.17.0.1:45378 #1 (1 connection now open)
2022-06-20T15:35:40.333+0000 I NETWORK [conn1] Error receiving request from client: ProtocolError: Client sent an HTTP request over a native MongoDB connection. Ending connection from 172.17.0.1:45378 (connection id: 1)
2022-06-20T15:35:40.333+0000 I NETWORK [conn1] end connection 172.17.0.1:45378 (0 connections now open)
```

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

MongoDB is able to bounce our request to that specific port and its container IP, indicating that it is ready to accept connections from the clients and protocols that it expects.

In order to store data, MongoDB will need to have a database created. Each database hosts a number of collections, which are similar to tables in other database systems. Within the collection, records are stored as documents in a binary JSON-like format known as BSON.

Use `docker container exec -it` to interact with the MongoDB container:

```
ubuntu@labsys:~$ sudo docker container exec -it mongodb bash
```

```
I have no name!@13e1440044db:/$
```

The MongoDB container gave itself an interesting name it seems! Once inside, run the MongoDB shell tool to continue configuring it.

Open a client session by running `mongo` within the container:

```
I have no name!@893a8e38f2bf:/$ mongo
```

```
MongoDB shell version v4.2.21
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("1d59768f-3af3-49ea-ad46-230c831cc9b5") }
MongoDB server version: 4.2.21
---
```

Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product

```
improvements and to suggest MongoDB products and deployment options to you.
```

```
To enable free monitoring, run the following command: db.enableFreeMonitoring()  
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()  
---  
>
```

Using the terminal output, answer the following questions:

- What version of MongoDB is in use?
- Where can more comprehensive documentation be acquired?
- Are there any additional configuration steps being suggested by MongoDB?

Now that a MongoDB shell session is active, manual operations can be performed against this MongoDB instance.

Create a database for Fluentd:

```
> use fluentd  
  
switched to db fluentd
```

No concrete naming convention is prescribed in the MongoDB documentation, but there is some consensus that names should be short. In this lab, all MongoDB objects will attempt to use single words in lower case.

Create a collection called "lab3":

```
> db.createCollection("lab3")  
  
{ "ok" : 1 }
```

Keep this terminal window open, we will be using it later to confirm the integration with MongoDB. All of this activity should be tracked by the MongoDB terminal session.

In a new working terminal, check the MongoDB container logs to see the recent output:

```
ubuntu@labsys:~$ sudo docker logs mongodb --tail 3  
  
2022-06-20T15:36:32.893+0000 I NETWORK [conn2] received client metadata from 127.0.0.1:35188  
conn2: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version:  
"4.2.21" }, os: { type: "Linux", name: "PRETTY_NAME="Debian GNU/Linux 10 (buster)"" }, architecture:  
"x86_64", version: "Kernel 5.13.0-1022-aws" } }  
2022-06-20T15:37:04.962+0000 I STORAGE [conn2] createCollection: fluentd.lab3 with generated UUID:  
36365864-f542-4599-a4dc-b44407cfaf15 and options: {}  
2022-06-20T15:37:04.973+0000 I INDEX [conn2] index build: done building index _id_ on ns  
fluentd.lab3  
  
ubuntu@labsys:~$
```

- What format is the connection event being presented in?
- What is the final name of this lab's datastore inside MongoDB?

MongoDB is now ready to accept events from Fluentd!

2. Prepare Fluentd for interaction with MongoDB

With MongoDB prepared, it is time to use configure Fluentd to communicate with it. These steps are a representation of a typical Fluentd processing pipeline development workflow, which consists of:

- Picking a destination datastore
- Selecting, acquiring (or creating) the necessary plugins to handle the datastore
- Creating a Fluentd configuration for the datastore
- Executing the pipeline

2a. Installing 3rd party Fluentd plugins with `fluent-gem`

For this lab, plugins will be installed using the `gem` commands installed with Ruby and Fluentd, specifically the `fluent-gem`, which is a wrapper around the standard `gem` command. Both function identically, but this lab will be using `fluent-gem`.

```
ubuntu@labsys:~$ fluent-gem --help
```

RubyGems is a sophisticated package manager for Ruby. This is a basic help message containing pointers to more information.

Usage:

```
gem -h/--help
gem -v/--version
gem command [arguments...] [options...]
```

Examples:

```
gem install rake
gem list --local
gem build package.gemspec
gem help install
```

Further help:

gem help commands	list all 'gem' commands
gem help examples	show some examples of usage
gem help gem_dependencies	gem dependencies file guide
gem help platforms	gem platforms guide
gem help <COMMAND>	show help on COMMAND (e.g. 'gem help install')
gem server	present a web page at http://localhost:8808/ with info about installed gems

Further information:

<http://guides.rubygems.org>

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

The `list` option can be used to retrieve all installed gems. Try it out:

```
ubuntu@labsys:~$ fluent-gem list
```

```
*** LOCAL GEMS ***
```

```
benchmark (default: 0.1.0)
bigdecimal (default: 2.0.0)
bundle (0.0.1)
bundler (default: 2.1.2)
```

```
...
```

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

Since Fluentd was installed as a Ruby gem in this example, it is listed as one of the local gems.

- What other gems are present on the system?
- Of the gems installed, identify which ones were installed implicitly. Why would other gems be present?

The list option can also be used to search its configured remote sources for plugins, which by default will include [Rubygems.org](https://rubygems.org).

Using the `--remote` flag, search for gems available. It will list **all** of the available plugins, so pipe the output to `tail` :

```
ubuntu@labsys:~$ fluent-gem list --remote | tail -5

zzot-zzot-semi-static (0.0.1)
zzsharedlib (0.0.7)
zzutil (0.0.4)
Zzzzz (0.0.1)
zzzzzz (0.0.3)

ubuntu@labsys:~$
```

The output is likely too long for a regular terminal. By appending additional arguments after the `--remote` flag, the search can be narrowed. A majority of the plugins that are registered on the Fluentd plugin website can be found under `fluent-plugin` .

Search for `fluent-plugin` on the remote gem repository (pipe the output to `head` to only show the first five entries):

```
ubuntu@labsys:~$ fluent-gem list --remote fluent-plugin | head -5

ach-fluent-plugin-sentry (0.0.18)
adp-fluent-plugin-graphite (0.1.4)
adp-fluent-plugin-kinesis (0.0.2)
apptuit-fluent-plugin (0.1.3)
async-fluent-plugin-azureeventhubs (0.0.1)

ubuntu@labsys:~$
```

Without piping the output to `head` , it's a long list, showing all of the Fluentd plugins that can be found both on Rubygems and the Fluentd plugin registry. Now that you know how to use `fluent-gem` to search for plugins, it is time to load up a MongoDB plugin.

Search for any `fluent-plugin-mongo` entries in the remote repository:

```
ubuntu@labsys:~$ fluent-gem list --remote fluent-plugin-mongo

*** REMOTE GEMS ***

fluent-plugin-mongo (1.5.0)
fluent-plugin-mongo-slow-query (0.1.1)
fluent-plugin-mongo-typed (0.1.0)
fluent-plugin-mongokpi (0.0.2)
fluent-plugin-mongostat (0.0.2)

ubuntu@labsys:~$
```

That should have pared the long list of Fluentd plugins on the [Rubygems.org](https://rubygems.org) registry to ones that match the use case presented in this lab. To truly understand where to look for the Fluentd plugin that best fits the intended use case, search the Fluentd plugin registry at <https://www.fluentd.org/plugins>.

Install the Fluentd `mongo` plugin using `fluent-gem` . The `-N` flag will prevent `fluent-gem` from installing the optional Ri documentation for the plugin:

```
ubuntu@labsys:~$ sudo fluent-gem install fluent-plugin-mongo -N -v 1.5.0
```

```
Fetching fluent-plugin-mongo-1.5.0.gem
Fetching mongo-2.6.4.gem
Fetching bson-4.15.0.gem
Building native extensions. This could take a while...
Successfully installed bson-4.15.0
Successfully installed mongo-2.6.4
Successfully installed fluent-plugin-mongo-1.5.0
3 gems installed

ubuntu@labsys:~$
```

This will install the MongoDB Fluentd plugin, which will allow Fluentd to interact with MongoDB by providing input and output plugins.

Since it is a plugin hosted on [Rubygems.org](https://rubygems.org), it can be retrieved using a `fluent-gem install` command. The `fluent-gem install` command will download the Fluentd plugin gem from [Rubygems.org](https://rubygems.org) and place it inside the GEM PATH.

To begin exploring a plugin installation, retrieve the GEM PATH using `gem env` or `fluent-gem env`:

```
ubuntu@labsys:~$ fluent-gem env
```

```
RubyGems Environment:
- RUBYGEMS VERSION: 3.1.2
- RUBY VERSION: 2.7.0 (2019-12-25 patchlevel 0) [x86_64-linux-gnu]
- INSTALLATION DIRECTORY: /var/lib/gems/2.7.0
- USER INSTALLATION DIRECTORY: /home/ubuntu/.gem/ruby/2.7.0
- RUBY EXECUTABLE: /usr/bin/ruby2.7
- GIT EXECUTABLE: /usr/bin/git
- EXECUTABLE DIRECTORY: /usr/local/bin
- SPEC CACHE DIRECTORY: /home/ubuntu/.gem/specs
- SYSTEM CONFIGURATION DIRECTORY: /etc
- RUBYGEMS PLATFORMS:
  - ruby
  - x86_64-linux
- GEM PATHS:
  - /var/lib/gems/2.7.0
  - /home/ubuntu/.gem/ruby/2.7.0
  - /usr/lib/ruby/gems/2.7.0
  - /usr/share/rubygems-integration/2.7.0
  - /usr/share/rubygems-integration/all
  - /usr/lib/x86_64-linux-gnu/rubygems-integration/2.7.0
- GEM CONFIGURATION:
  - :update_sources => true
  - :verbose => true
  - :backtrace => false
  - :bulk_threshold => 1000
- REMOTE SOURCES:
  - https://rubygems.org/
- SHELL PATH:
  - /usr/local/sbin
  - /usr/local/bin
  - /usr/sbin
  - /usr/bin
  - /sbin
  - /bin
  - /usr/games
  - /usr/local/games
  - /snap/bin
```

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

- What version of Ruby gems was installed on the system?
- What sources are registered?

The output of interest here are the GEM PATHs. These GEM PATHs are where any gems retrieved by `gem` (and `fluent-gem`) will be installed.

Check the first directory path under GEM PATHs. In this example it is `/var/lib/gems/2.7.0`:

```
ubuntu@labsys:~$ ls -l /var/lib/gems/2.7.0

total 24
drwxr-xr-x  2 root root 4096 Jun 20 14:43 build_info
drwxr-xr-x  2 root root 4096 Jun 20 15:39 cache
drwxr-xr-x 14 root root 4096 Jun 20 14:43 doc
drwxr-xr-x  3 root root 4096 Jun 20 14:43 extensions
drwxr-xr-x 17 root root 4096 Jun 20 15:39 gems
drwxr-xr-x  2 root root 4096 Jun 20 15:39 specifications

ubuntu@labsys:~$
```

The folders contained inside this path are standard Ruby gem folders, which assist, enable and record Ruby operations that are out of the scope of this lab. The `gems` folder is the most relevant subdirectory: Fluentd plugins installed by `fluent-gem` are unpacked into the `gems` directory.

```
ubuntu@labsys:~$ ls -l /var/lib/gems/2.7.0/gems/

total 60
drwxr-xr-x  5 root root 4096 Jun 20 15:39 bson-4.15.0
drwxr-xr-x  2 root root 4096 Jun 20 14:43 bundle-0.0.1
drwxr-xr-x  4 root root 4096 Jun 20 14:43 concurrent-ruby-1.1.10
drwxr-xr-x  6 root root 4096 Jun 20 14:43 cool.io-1.7.1
drwxr-xr-x  5 root root 4096 Jun 20 15:39 fluent-plugin-mongo-1.5.0
drwxr-xr-x  9 root root 4096 Jun 20 14:43 fluentd-1.14.6
drwxr-xr-x  8 root root 4096 Jun 20 14:43 http_parser.rb-0.8.0
drwxr-xr-x  5 root root 4096 Jun 20 15:39 mongo-2.6.4
drwxr-xr-x  8 root root 4096 Jun 20 14:43 msgpack-1.5.2
drwxr-xr-x  6 root root 4096 Jun 20 14:43 serverengine-2.3.0
drwxr-xr-x  3 root root 4096 Jun 20 14:43 sigdump-0.2.4
drwxr-xr-x  5 root root 4096 Jun 20 14:43 strptime-0.2.5
drwxr-xr-x  3 root root 4096 Jun 20 14:43 tzinfo-2.0.4
drwxr-xr-x  3 root root 4096 Jun 20 14:43 tzinfo-data-1.2022.1
drwxr-xr-x 10 root root 4096 Jun 20 14:43 yajl-ruby-1.4.3

ubuntu@labsys:~$
```

All of the gem files are placed into directories here.

Take a look inside the `fluent-plugin-mongo` folder:

```
ubuntu@labsys:~$ ls -l /var/lib/gems/2.7.0/gems/fluent-plugin-mongo-1.5.0/

total 52
-rw-r--r--  1 root root   88 Jun 20 15:39 AUTHORS
-rw-r--r--  1 root root 6336 Jun 20 15:39 ChangeLog
-rw-r--r--  1 root root   75 Jun 20 15:39 Gemfile
```



```
-rw-r--r-- 1 root root 8872 Jun 20 15:39 README.rdoc
-rw-r--r-- 1 root root 318 Jun 20 15:39 Rakefile
-rw-r--r-- 1 root root 6 Jun 20 15:39 VERSION
drwxr-xr-x 2 root root 4096 Jun 20 15:39 bin
-rw-r--r-- 1 root root 1112 Jun 20 15:39 fluent-plugin-mongo.gemspec
drwxr-xr-x 3 root root 4096 Jun 20 15:39 lib
drwxr-xr-x 3 root root 4096 Jun 20 15:39 test

ubuntu@lab3sys:~$
```

In this directory, there are three subdirectories among the other files:

- **lib** contains all of the plugins themselves - the ruby scripts
- **test** contains development or test versions of the plugins
- **bin** contains code that the ruby scripts may call upon or require. This is required for gem executables to run.

The plugins that Fluentd will use are contained under the lib folder, where they are nested in a couple of folders:

```
ubuntu@lab3sys:~$ ls -l /var/lib/gems/2.7.0/gems/fluent-plugin-mongo-1.5.0/lib/fluent/plugin/

total 36
-rw-r--r-- 1 root root 5098 Jun 20 15:39 in_mongo_tail.rb
-rw-r--r-- 1 root root 780 Jun 20 15:39 logger_support.rb
-rw-r--r-- 1 root root 1093 Jun 20 15:39 mongo_auth.rb
-rw-r--r-- 1 root root 13074 Jun 20 15:39 out_mongo.rb
-rw-r--r-- 1 root root 1434 Jun 20 15:39 out_mongo_replset.rb

ubuntu@lab3sys:~$
```

Each of these Ruby scripts are the heart of a Fluentd plugin: if a validly formatted Ruby script is placed in this folder, or another folder such as `/etc/fluent/plugin/`, then Fluentd can use them.

Now that you have seen the specific ruby scripts, you should now know where to find what exact plugins are available to by Fluentd.

Prepare a new Fluentd configuration file, `mongo.conf`

```
ubuntu@lab3sys:~$ mkdir ~/lab3 ; cd $_

ubuntu@lab3sys:~/lab3$ nano ~/lab3/mongo.conf && cat $_

<source>
  @type tail
  path /tmp/mongotrack
  pos_file /tmp/mongotrack.pos
  <parse>
    @type none
  </parse>
  tag mongo.lab3
</source>

<match mongo.**>
  @type mongo
  host 172.17.0.2
  port 27017
  database fluentd
  collection lab3
  <inject>
    time_key time
  </inject>
```

```
</match>
```

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

The intent of this configuration is:

- Fluentd will be tailing simple lines in a file called `mongotrack` under `/tmp/`, tagging messages under `mongo.lab3`
- Using the `mongo` output plugin, Fluentd will direct any events tagged `mongo.**` to the MongoDB container listening on port 27017 (in this case, any message tagged `mongo.lab3`)
- All log output will be directed to the `lab3` collection under the `fluentd` database created earlier

2b. Start Fluentd using the `mongo.conf`

With the configuration using the newly installed MongoDB plugin now in place, it is time to start Fluentd. This instance will be monitored with trace-level logging to STDOUT using the `-vv` option, to provide more visibility into the events.

Run Fluentd using the `mongo.conf` prepared in the lab step and with TRACE verbosity using the `-vv` flag:

```
ubuntu@labsys:~/lab3$ fluentd --config mongo.conf -vv
```

```
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="mongo.conf"
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: gem 'fluentd' version '1.14.6'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'mongo'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-20 15:47:15 +0000 [debug]: fluent/log.rb:309:debug: Setup mongo configuration: mode = normal
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'regexp'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'multiline'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'tail'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'none'
2022-06-20 15:47:15 +0000 [debug]: fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: using configuration file: <ROOT>
  <source>
    @type tail
    path "/tmp/mongotrack"
    pos_file "/tmp/mongotrack.pos"
    tag "mongo.lab3"
    <parse>
      @type "none"
      unmatched_lines
    </parse>
  </source>
  <match mongo.**>
    @type mongo
    host "172.17.0.2"
    port 27017
    database "fluentd"
    collection "lab3"
    buffer_chunk_limit 8m
    <inject>
      time_key "time"
    </inject>
  </match>
</ROOT>
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: starting fluentd-1.14.6 pid=44180
ruby="2.7.0"
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: spawn command to main: cmdline=
["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "--config", "mongo.conf",
```

```

"-vv", "--under-supervisor"]
2022-06-20 15:47:16 +0000 [info]: fluent/log.rb:330:info: adding match pattern="mongo.***"
type="mongo"
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered output plugin 'mongo'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: Setup mongo configuration: mode =
normal
2022-06-20 15:47:16 +0000 [info]: fluent/log.rb:330:info: adding source type="tail"
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'regexp'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'multiline'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'tail'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'none'
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 15:47:16 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=44185
ppid=44180 worker=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1860
stage_size=0 queue_size=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: tailing paths: target = | existing =
2022-06-20 15:47:16 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860

```

- What plugins were loaded by the above configuration?

Fluentd is now up and running, using the custom `mongo.conf` to forward input from `/tmp/mongotrack` into MongoDB. Keep this terminal active, as we will be using it later. You ran Fluentd with TRACE verbosity to better see some of the buffering behavior happening live. This may flood the Fluentd terminal as it logs events.

3. Test the MongoDB forwarding settings

Time to test whether Fluentd can successfully forward events to MongoDB.

Open a new terminal and use `echo` to append data to the `/tmp/mongotrack` file that Fluentd is tailing:

```

ubuntu@lab3sys:~$ cd ~/lab3
ubuntu@lab3sys:~/lab3$ touch /tmp/mongotrack
ubuntu@lab3sys:~/lab3$ echo "First message" >> /tmp/mongotrack
ubuntu@lab3sys:~/lab3$

```

In the Fluentd terminal, your append should have triggered some activity, as `/tmp/mongotrack` has been created and is now tracked:

```

...
2022-06-20 15:48:16 +0000 [debug]: #0 fluent/log.rb:309:debug: tailing paths: target =
/tmp/mongotrack | existing =
2022-06-20 15:48:16 +0000 [info]: #0 fluent/log.rb:330:info: following tail of /tmp/mongotrack
2022-06-20 15:48:16 +0000 [trace]: #0 fluent/log.rb:287:trace: writing events into buffer
instance=1860 metadata_size=1
2022-06-20 15:48:16 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860

```

...

Since MongoDB is a buffered output plugin, it will not write its results into the destination until the configured buffer size and/or time limit has been breached. Therefore, it will take some time before the event is actually written into MongoDB. In this configuration it will take about 60 seconds before Fluentd will flush its results and write the event to MongoDB.

You can accelerate this process by sending a `SIGUSR1` to Fluentd with `pkill -SIGUSR1 fluentd`:

```
ubuntu@labsys:~/lab3$ echo "Hello MongoDB!" >> /tmp/mongotrack
ubuntu@labsys:~/lab3$ pkill -SIGUSR1 fluentd
ubuntu@labsys:~/lab3$
```

Check the Fluentd terminal about 60 seconds after you submit the `echo` command from earlier or after you send a `SIGUSR1` signal:

```
...
2022-06-20 15:48:49 +0000 [trace]: #0 fluent/log.rb:287:trace: writing events into buffer
instance=1860 metadata_size=1
2022-06-20 15:48:49 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
2022-06-20 15:48:55 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
2022-06-20 15:48:57 +0000 [debug]: fluent/log.rb:309:debug: fluentd supervisor process get SIGUSR1
2022-06-20 15:48:57 +0000 [debug]: #0 fluent/log.rb:309:debug: fluentd main process get SIGUSR1
2022-06-20 15:48:57 +0000 [info]: #0 fluent/log.rb:330:info: force flushing buffered events
2022-06-20 15:48:57 +0000 [info]: #0 fluent/log.rb:330:info: flushing all buffer forcedly
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing chunk instance=1860
metadata=#<struct Fluent::Plugin::Buffer::Metadata timekey=nil, tag="mongo.lab3", variables=nil,
seq=0>
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: dequeueing a chunk instance=1860
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: chunk dequeued instance=1860
metadata=#<struct Fluent::Plugin::Buffer::Metadata timekey=nil, tag="mongo.lab3", variables=nil,
seq=0>
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: trying flush for a chunk
chunk="5e1e30850d5fdfa24d62f138319ac7cf"
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: adding write count instance=1840
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: executing sync write
chunk="5e1e30850d5fdfa24d62f138319ac7cf"
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: write operation done, committing
chunk="5e1e30850d5fdfa24d62f138319ac7cf"
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: committing write operation to a chunk
chunk="5e1e30850d5fdfa24d62f138319ac7cf" delayed=false
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: purging a chunk instance=1860
chunk_id="5e1e30850d5fdfa24d62f138319ac7cf" metadata=#<struct Fluent::Plugin::Buffer::Metadata
timekey=nil, tag="mongo.lab3", variables=nil, seq=0>
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: chunk purged instance=1860
chunk_id="5e1e30850d5fdfa24d62f138319ac7cf" metadata=#<struct Fluent::Plugin::Buffer::Metadata
timekey=nil, tag="mongo.lab3", variables=nil, seq=0>
2022-06-20 15:48:57 +0000 [trace]: #0 fluent/log.rb:287:trace: done to commit a chunk
chunk="5e1e30850d5fdfa24d62f138319ac7cf"
2022-06-20 15:48:58 +0000 [debug]: #0 fluent/log.rb:309:debug: flushing thread: flushed
2022-06-20 15:49:00 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
2022-06-20 15:49:05 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
```

```
instance=1860
2022-06-20 15:49:11 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
2022-06-20 15:49:16 +0000 [debug]: #0 fluent/log.rb:309:debug: tailing paths: target =
/tmp/mongotrack | existing = /tmp/mongotrack

...
```

- What did Fluentd do before it wrote to MongoDB?

In your working terminal, Use `docker logs` to check the MongoDB container log:

```
ubuntu@lab3sys:~/lab3$ sudo docker logs mongodb --tail 5

...

2022-06-20T15:47:16.889+0000 I NETWORK [conn3] end connection 172.17.0.1:45380 (1 connection now open)
2022-06-20T15:47:16.889+0000 I NETWORK [listener] connection accepted from 172.17.0.1:45382 #4 (2 connections now open)
2022-06-20T15:47:16.890+0000 I NETWORK [conn4] received client metadata from 172.17.0.1:45382
conn4: { driver: { name: "mongo-ruby-driver", version: "2.6.4" }, os: { type: "linux", name: "linux-gnu", architecture: "x86_64" }, platform: "2.7.0, x86_64-linux-gnu, x86_64-pc-linux-gnu" }
2022-06-20T15:48:57.218+0000 I NETWORK [listener] connection accepted from 172.17.0.1:45384 #5 (3 connections now open)
2022-06-20T15:48:57.219+0000 I NETWORK [conn5] received client metadata from 172.17.0.1:45384
conn5: { driver: { name: "mongo-ruby-driver", version: "2.6.4" }, os: { type: "linux", name: "linux-gnu", architecture: "x86_64" }, platform: "2.7.0, x86_64-linux-gnu, x86_64-pc-linux-gnu" }

ubuntu@lab3sys:~/lab3$
```

You can see that Fluentd connected to MongoDB with the mongo-ruby-driver.

In the MongoDB container terminal, use the mongo shell to see if any documents has been recorded into the `fluentd` database prepared earlier:

Switch to the `fluentd` database and find all entries under the `lab3` collection:

```
ubuntu@lab3sys:~/lab3$ sudo docker container exec -it mongodb bash

I have no name!@13e1440044db:/$ mongo

...

> use fluentd

switched to db fluentd

> db.lab3.find()

{ "_id" : ObjectId("62b096e977ca8aac992ea936"), "message" : "First message", "time" : ISODate("2022-06-20T15:48:16.894Z") }
{ "_id" : ObjectId("62b096e977ca8aac992ea937"), "message" : "Hello MongoDB!", "time" : ISODate("2022-06-20T15:48:49.159Z") }

>
```

You have successfully configured Fluentd to read a file and forward events into MongoDB! Keep this MongoDB instance alive.

4. Application Logging to MongoDB

Now that Fluentd is now forwarding to MongoDB, it is time to create an application-to-application logging pipeline, a common use case for Fluentd as a log forwarder and aggregator. This example will use an NGINX container, locally mount its log directory, and forward its logs to MongoDB.

4a. Preparing MongoDB for the new application

As before, a collection within a database is required to hold individual documents in MongoDB.

In the MongoDB terminal, prepare a database and collections for the two common NGINX logs, `access.log` and `error.log`:

```
> use nginx

switched to db nginx

> db.createCollection("access")

{ "ok" : 1 }

> db.createCollection("error")

{ "ok" : 1 }
```

Confirm that the collections have been created:

```
> db.runCommand( { listCollections: 1.0 } )

{
  "cursor" : {
    "id" : NumberLong(0),
    "ns" : "nginx.$cmd.listCollections",
    "firstBatch" : [
      {
        "name" : "error",
        "type" : "collection",
        "options" : {
          "readOnly" : false,
          "uuid" : UUID("449214ee-f744-4a5b-a17c-e93a00ab675f")
        }
      },
      {
        "name" : "access",
        "type" : "collection",
        "options" : {
          "readOnly" : false,
          "uuid" : UUID("449214ee-f744-4a5b-a17c-e93a00ab675f")
        }
      }
    ]
  },
  "info" : {
    "readOnly" : false,
    "uuid" : UUID("449214ee-f744-4a5b-a17c-e93a00ab675f")
  },
  "idIndex" : {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "nginx.error"
  }
}
```

```

        "info" : {
          "readOnly" : false,
          "uuid" : UUID("883900c9-3726-43d5-a5ce-aec17dab0c2c")
        },
        "idIndex" : {
          "v" : 2,
          "key" : {
            "_id" : 1
          },
          "name" : "_id_",
          "ns" : "nginx.access"
        }
      }
    ]
  },
  "ok" : 1
}
>

```

Take a note of the details in MongoDB, particularly the NS:

- What NS values are listed?
- What standards do the NS values conform to (other than MongoDB conventions)?

Finally, run `find()` command on both the error and access collections to see what data are there. They should return no output, as they are currently empty:

```

> db.error.find()
>
> db.access.find()
>

```

MongoDB now has a place to store NGINX log events for both the access and error logs. Continue to keep this terminal session up and running.

4b. Run a NGINX docker container with the log directory mounted locally

For this exercise, NGINX will be set up to store its logs onto its host machine's filesystem. This will allow a Fluentd instance running on the same system to read and parse those logs.

In your working terminal, run an NGINX container and mount the logs directory to the host so Fluentd can access them.

```

> quit()

I have no name!@13e1440044db:/$ exit

exit

ubuntu@lab3sys:~/lab3$ mkdir -p /tmp/nginx/log

ubuntu@lab3sys:~/lab3$ sudo docker run --name nginx -d -v /tmp/nginx/log:/var/log/nginx nginx

...

46f1675745e7b7749c7800e8cedc887fdae24a791dfb109f6865fa5c21f8b1b6

```

```
ubuntu@lab3sys:~/lab3$
```

This NGINX container has been run as a daemon with the `-d` flag, so this terminal can still be used. Once started, the NGINX container should place a pair of logs into its log directory, which is found under `/var/log/nginx` in the container. Since the `/tmp/nginx/log` folder created earlier was mounted to that directory, the NGINX container will write files to your machine's `/tmp` directory.

Confirm that the local directory `/tmp/nginx/log` is mounted properly:

```
ubuntu@lab3sys:~/lab3$ ls -l /tmp/nginx/log/

total 4
-rw-r--r-- 1 root root  0 Jun 20 15:53 access.log
-rw-r--r-- 1 root root 503 Jun 20 15:53 error.log

ubuntu@lab3sys:~/lab3$
```

Excellent, the container has successfully mounted a local NGINX log directory into its own filesystem and is writing all NGINX logs to that directory.

Use `docker inspect` to retrieve the container IP, then verify NGINX functionality by using `curl` to send a GET request:

```
ubuntu@lab3sys:~/lab3$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx

172.17.0.3

ubuntu@lab3sys:~/lab3$ curl 172.17.0.3

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

ubuntu@lab3sys:~/lab3$
```

A NGINX container is now functional on the lab system. Its logs are currently human readable, and susceptible to a data loss disaster on the host; it would be ideal to move its logs over into another system, such as MongoDB, to increase the durability of its log data.

4c. Prepare a Fluentd config to forward events from the NGINX logs to MongoDB

There is already a configuration file with a working `<match>` directive for MongoDB from this lab's previous section, so use it to further configure Fluentd.

In your working terminal, create a copy of the `mongo.conf` from earlier steps and update it with NGINX-specific directives:

```
ubuntu@labsys:~/lab3$ cp mongo.conf nginx-fluentd-mongo.conf

ubuntu@labsys:~/lab3$ nano nginx-fluentd-mongo.conf && cat $_

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

<source>
  @type tail
  path /tmp/nginx/log/error.log
  pos_file /tmp/nginx/error.pos
  tag mongo.nginx.error
  <parse>
    @type multiline
    format_firstline /\d{4}/\d{2}/\d{2} \d{2}:\d{2}:\d{2} \[(?<pid>\d+).(?(?<tid>\d+): /
    format1 /^(?<time>\d{4}/\d{2}/\d{2} \d{2}:\d{2}:\d{2}) \[(?<log_level>\w+)\] (?<pid>\d+).
    (?<tid>\d+): (?<message>.*)/
  </parse>
</source>

<match mongo.nginx.access.**>
  @type mongo
  host 172.17.0.2
  port 27017
  database nginx
  collection access
  <inject>
    time_key time
  </inject>
</match>

<match mongo.nginx.error.**>
  @type mongo
  host 172.17.0.2
  port 27017
  database nginx
  collection error
  <inject>
    time_key time
  </inject>
</match>

ubuntu@labsys:~/lab3$
```

There are the changes added to allow NGINX events to be sent to MongoDB:

For the `<source>` directives:

- The type is set to `tail` so that the logs will be read from the newest line
- The `nginx` parser plugin is used to process the default NGINX access log
- A multiline format is used to parse the error log; the example above was pulled from the Fluentd documentation.
- The container's local mounted log directory `/tmp/nginx/log` and files are referenced by full directory path
- The `tag` attributes have been assigned using the `mongo.database.collection` format

For the `<match>` directives:

- Each source is referenced by the tag
- The database is set to `nginx`
- The collection is set to either `error` or `access` according to the source tag
- All other match configurations have been carried over from the original `mongo.conf` prepared earlier in the lab

Now that the configuration has been prepared for this event logging pipeline, it's time to reload Fluentd. Since a new configuration file will be used, the Fluentd instance will need to be fully terminated.

In the Fluentd terminal, use `Ctrl C` to shut down the currently running Fluentd instance:

```
^C
...
2022-06-20 15:54:59 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0
ubuntu@lab3sys:~/lab3$
```

Then restart Fluentd using the `nginx-mongodb.conf`:

```
ubuntu@lab3sys:~/lab3$ fluentd --config nginx-fluentd-mongo.conf

2022-06-20 15:55:11 +0000 [info]: parsing config file is succeeded path="nginx-fluentd-mongo.conf"
2022-06-20 15:55:11 +0000 [info]: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 15:55:11 +0000 [info]: gem 'fluentd' version '1.14.6'

...

2022-06-20 15:55:12 +0000 [info]: starting fluentd-1.14.6 pid=44501 ruby="2.7.0"
2022-06-20 15:55:12 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "--config", "nginx-fluentd-mongo.conf", "--under-supervisor"]
2022-06-20 15:55:12 +0000 [info]: adding match pattern="mongo.nginx.access.*)" type="mongo"
2022-06-20 15:55:13 +0000 [info]: adding match pattern="mongo.nginx.error.*)" type="mongo"
2022-06-20 15:55:13 +0000 [info]: adding source type="tail"
2022-06-20 15:55:13 +0000 [info]: adding source type="tail"
2022-06-20 15:55:13 +0000 [info]: #0 starting fluentd worker pid=44506 ppid=44501 worker=0
2022-06-20 15:55:13 +0000 [info]: #0 following tail of /tmp/nginx/log/error.log
2022-06-20 15:55:13 +0000 [info]: #0 following tail of /tmp/nginx/log/access.log
2022-06-20 15:55:13 +0000 [info]: #0 fluentd worker is now running worker=0
```

Fluentd is now ready to forward access and error events from NGINX logs to MongoDB!

4d. Interact with NGINX to test log event forwarding

In your working terminal, run `wget` against the NGINX container's IP address. This will simulate a typical access scenario that NGINX would log.

```

ubuntu@labsys:~/lab3$ wget -qO - http://172.17.0.3

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

ubuntu@labsys:~/lab3$

```

That performed a quick GET request against the container, which should have generated an event in the NGINX log.

In your working terminal, flush the Fluentd buffer by sending a `kill -SIGUSR1` :

```

ubuntu@labsys:~/lab3$ kill -SIGUSR1 fluentd

ubuntu@labsys:~/lab3$

```

In the MongoDB terminal, query the `access` collection. The recent access event should now be visible:

```

ubuntu@labsys:~/lab3$ sudo docker container exec -it mongodb bash

I have no name!@13e1440044db:/$ mongo

> use nginx

switched to db nginx

> db.access.find()

{ "_id" : ObjectId("62b0989a77ca8aadda7f8dec"), "remote" : "172.17.0.1", "host" : "-", "user" : "-",
"method" : "GET", "path" : "/", "code" : "200", "size" : "615", "referer" : "-", "agent" :
"Wget/1.20.3 (linux-gnu)", "http_x_forwarded_for" : "-", "time" : ISODate("2022-06-20T15:55:56Z") }

>

```

Excellent! Fluentd was able to read the NGINX access log, parse its format, and send it directly to MongoDB!

```

> quit()

```

```
I have no name!@13e1440044db:/$ exit

exit

ubuntu@labsys:~/lab3$
```

In addition to the access log input, `<source>` directive for errors written to the `error.log` was also prepared. This file actually captures error events in a different format than the access log, which is put through a format plugin.

In your working terminal, try running `wget` against a non-existent page a couple of times:

```
ubuntu@labsys:~/lab3$ wget -O - http://172.17.0.3/404

--2022-06-20 15:57:46-- http://172.17.0.3/404
Connecting to 172.17.0.3:80... connected.
HTTP request sent, awaiting response... 404 Not Found
2022-06-20 15:57:46 ERROR 404: Not Found.

ubuntu@labsys:~/lab3$ wget -O - http://172.17.0.3/404

...

ubuntu@labsys:~/lab3$
```

Flush the buffers once again with `kill -SIGUSR1` :

```
ubuntu@labsys:~/lab3$ kill -sigusr1 fluentd

ubuntu@labsys:~/lab3$
```

And now query the `access` and `error` collections in the MongoDB terminal:

```
ubuntu@labsys:~/lab3$ sudo docker container exec -it mongodb mongo

> use nginx

> db.access.find()

{ "_id" : ObjectId("62b0989a77ca8aadda7f8dec"), "remote" : "172.17.0.1", "host" : "-", "user" : "-",
"method" : "GET", "path" : "/", "code" : "200", "size" : "615", "referer" : "-", "agent" :
"Wget/1.20.3 (linux-gnu)", "http_x_forwarded_for" : "-", "time" : ISODate("2022-06-20T15:55:56Z") }
{ "_id" : ObjectId("62b0990c77ca8aadda7f8ded"), "remote" : "172.17.0.1", "host" : "-", "user" : "-",
"method" : "GET", "path" : "/404", "code" : "404", "size" : "153", "referer" : "-", "agent" :
"Wget/1.20.3 (linux-gnu)", "http_x_forwarded_for" : "-", "time" : ISODate("2022-06-20T15:57:46Z") }
{ "_id" : ObjectId("62b0990c77ca8aadda7f8dee"), "remote" : "172.17.0.1", "host" : "-", "user" : "-",
"method" : "GET", "path" : "/404", "code" : "404", "size" : "153", "referer" : "-", "agent" :
"Wget/1.20.3 (linux-gnu)", "http_x_forwarded_for" : "-", "time" : ISODate("2022-06-20T15:58:00Z") }

> db.error.find()

{ "_id" : ObjectId("62b0990c77ca8aadda7f8def"), "log_level" : "error", "pid" : "30", "tid" : "30",
"message" : "*3 open() \"/usr/share/nginx/html/404\" failed (2: No such file or directory), client:
172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host: \"172.17.0.3\", \"time\" :
ISODate(\"2022-06-20T15:57:46Z\") }

>
```

A NGINX instance is now forwarding access and error log output to MongoDB through Fluentd!

Cleanup

Use CTRL C to terminate any running instances of Fluentd:

```
^C

2022-06-20 15:59:36 +0000 [info]: Received graceful stop
2022-06-20 15:59:37 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 15:59:37 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 15:59:37 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:794"
2022-06-20 15:59:37 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:7a8"
2022-06-20 15:59:37 +0000 [info]: #0 shutting down output plugin type=:mongo plugin_id="object:730"
2022-06-20 15:59:38 +0000 [info]: #0 shutting down output plugin type=:mongo plugin_id="object:76c"
2022-06-20 15:59:38 +0000 [info]: Worker 0 finished with status 0

ubuntu@lab3sys:~/lab3$
```

Now tear down any running Docker containers:

```
> quit()

ubuntu@lab3sys:~/lab3$ sudo docker container rm $(sudo docker container stop mongodb nginx)

mongodb
nginx

ubuntu@lab3sys:~/lab3$
```

Congratulations, you have completed the lab!

LFS242 - Cloud Native Logging with Fluentd

Lab 3 – Extending Fluentd with Plugins: Working with Input and Output Plugins

- What version of MongoDB is in use?
 - MongoDB server version: 4.0.9
- Where can more comprehensive documentation be acquired?
 - <http://docs.mongodb.org/>
- Are there any additional configuration steps being suggested by MongoDB?
 - None at this time
- What format is the connection event being presented in?
 - A JSON map
- What is the final name of this lab's datastore inside MongoDB?
 - fluentd.lab3
- What other gems are present on the system?
 - See the output of `fluent-gem list`
- Of the gems installed, identify which ones were installed implicitly. Why would other gems be present?
 - Only `fluentd (1.13.2.2)` was installed by the user. All other gems were pulled in as dependencies during the initial installation
- What version of Ruby gems was installed on the system?
 - RUBYGEMS VERSION: 3.1.2
- What sources are registered?
 - REMOTE SOURCES:
 - <https://rubygems.org/>
- What plugins were loaded by the above configuration?
 - `mongo`, `memory`, `tail`, `regexp`, `multiline`
- What did Fluentd do before it wrote to MongoDB?
 - It enqueued the chunk
- What NS values are listed?
 - "ns" : "nginx.\$cmd.listCollections"
 - "ns" : "nginx.error"
 - "ns" : "nginx.access"
- What standards do the NS values conform to (other than MongoDB conventions)?
 - JSON key-value pairs

LFS242 - Cloud Native Logging with Fluentd

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

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

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

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

Objectives

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

0. Prepare the lab system

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

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

#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

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

Install Docker with the quick installation script for Debian:

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

...
```

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

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

1. Creating a Fluentd Filter directive

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

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

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

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

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

<source>
@type forward
</source>

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

ubuntu@labsys:~/lab4$
```

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

Using this configuration, run a Fluentd instance:

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

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



```

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

```

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

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

```

ubuntu@labsys:~$ cd lab4

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

ubuntu@labsys:~/lab4$

```

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

```

...

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

```

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

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

```

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

<source>
@type forward
</source>

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

<match>
@type stdout
</match>

ubuntu@labsys:~/lab4$

```

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

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

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

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

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

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

Now that Fluentd is reloaded, answer the following questions:

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

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

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

In the Fluentd terminal, check the resulting output:

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

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

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

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

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

<source>
  @type forward
</source>

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

```
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$
```

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

Use SIGUSR2 to reload the configuration:

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

ubuntu@labsys:~/lab4$
```

Check the Fluentd terminal now:

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

Now send another test event:

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

ubuntu@labsys:~/lab4$
```

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

```
...
[{"lfs242":"hello"}]2022-06-14 22:06:55.171959722 +0000 lab4.test.2: {"lfs242":"hello"}
```

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

2. Exploring core filter Plugins

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

2a. Grep an event

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

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

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

```
ubuntu@lab4sys:~/lab4$ cp lab4.conf lab4-1.conf
ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ nano lab4.conf && cat $_

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

<match>
  @type stdout
</match>

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR2 fluentd
ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ touch /tmp/tailfile
ubuntu@lab4sys:~/lab4$ echo "INFO New event" >> /tmp/tailfile
ubuntu@lab4sys:~/lab4$ echo "ERROR New error" >> /tmp/tailfile
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR2 fluentd
ubuntu@lab4sys:~/lab4$
```

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

...

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

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

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

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

```
ubuntu@lab4sys:~/lab4$ nano lab4.conf && cat $_
```

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

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

```
<match>
  @type stdout
</match>
```

```
ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR2 fluentd
```

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

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

```
ubuntu@lab4sys:~/lab4$ echo "INFO New event" >> /tmp/tailfile
```

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

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

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

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

ubuntu@labsys:~/lab4$
```

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

Check what logs Fluentd is pulling from this bloated log:

```
...

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

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

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

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

2b. Transforming Records with filter_record_transformer

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

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

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

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

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

<source>
  @type forward
</source>

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

```

    status filtered
  </record>
</filter>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab4$

```

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

Reload the Fluentd configuration with `SIGUSR2` :

```

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

ubuntu@labsys:~/lab4$

```

```

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

...

```

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

Now pass an event to the newly reconfigured Fluentd:

```

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

```

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

Check the Fluentd terminal:

```

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

```

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

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

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

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

<source>
@type forward
</source>

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

<match>
@type stdout
</match>

ubuntu@labsys:~/lab4$
```

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

Reconfigure Fluentd by sending a SIGUSR2:

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

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

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

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

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

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

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


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

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

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

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

<source>
  @type forward
</source>

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

<match>
  @type stdout
</match>

ubuntu@labsys:~$
```

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

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

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

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

In the Fluentd terminal, check the event.

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

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

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

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

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

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

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

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

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

```
ubuntu@lab4sys:~/lab4$ nano lab4.conf && cat $_

<source>
@type forward
</source>

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

<match>
@type stdout
</match>

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@lab4sys:~/lab4$ echo '{"lfs242":"hello","password":"secret"}' | fluent-cat lab4.2.b
```

Check the resulting event in the Fluentd terminal:

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

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

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

3. Using multiple Filters

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

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

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

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

```
ubuntu@lab4sys:~/lab4$ cp lab4.conf lab4-2-b.conf

ubuntu@lab4sys:~/lab4$ nano lab4.conf && cat $_

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

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

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

<match>
  @type stdout
</match>

ubuntu@lab4sys:~/lab4$
```

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

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

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

Now reload Fluentd:

```
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR2 fluentd

ubuntu@lab4sys:~/lab4$
```

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

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

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

...

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

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

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

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

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

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

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

4. Creating an NGINX to Elasticsearch logging pipeline

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

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

4a. Prepare an NGINX container

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

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

```
ubuntu@lab4sys:~/lab4$ mkdir /tmp/lab4/

ubuntu@lab4sys:~/lab4$ mkdir /tmp/lab4/nginx

ubuntu@lab4sys:~/lab4$ sudo docker run --name nginx -d -v /tmp/lab4/nginx:/var/log/nginx nginx

...

a3fc86044d1e8d3f6d8bcdeaa5a0fe370fb7b81b8ac9cb764eef446a3174580f

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx

172.17.0.2

ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ curl 172.17.0.2

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

ubuntu@lab4sys:~/lab4$
```

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

4b. Run Elasticsearch

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

In your working terminal, run an Elasticsearch container:

```
ubuntu@lab4sys:~$ sudo docker run -d --name elasticsearch -p 9200:9200 -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:8.2.2

...

dabbcb4214a0b5c59fcc7c93bd0a45e27e31cc4657d502f99e6bf27b5c32bb060

ubuntu@lab4sys:~$
```

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

with it.

```
ubuntu@lab4sys:~/lab4$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' elasticsearch

172.17.0.3

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ sudo docker exec -it elasticsearch /usr/share/elasticsearch/bin/elasticsearch-reset-password -u elastic

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

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

ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ export ESPW=61u+EV8Eev+t1WBV798u      # Use the value presented after you
changed the password.

ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ sudo docker cp elasticsearch:/usr/share/elasticsearch/config/certs/http_ca.crt
.

ubuntu@lab4sys:~/lab4$
```

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

```
ubuntu@lab4sys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
https://172.17.0.3:9200

{
  "name" : "dabbc4214a0b",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "XIhB4drQT_2wYxn4_crSTw",
  "version" : {
```

```

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

ubuntu@lab4$

```

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

4c. Configure Fluentd for the NGINX to Elasticsearch pipeline

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

You will need the following Fluentd plugins:

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

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

```

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

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

```

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

ubuntu@labsys:~/lab4$
```

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

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

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

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

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

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

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

ubuntu@labsys:~/lab4$
```


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

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

```
^C
...

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

Set the ESPW environment variable then restart the Fluentd instance:

```
ubuntu@lab4sys:~/lab4$ export ESPW=61u+EV8Eev+t1WBV798u

ubuntu@lab4sys:~/lab4$ fluentd -c ~/lab4/lab4.conf

...

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

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

It's time to test the configuration.

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

```
ubuntu@lab4sys:~$ curl http://172.17.0.2

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

...

ubuntu@lab4sys:~$
```

```
ubuntu@lab4sys:~$ curl http://172.17.0.2/404

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

ubuntu@lab4sys:~$
```

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

```
ubuntu@lab4sys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
'https://172.17.0.3:9200/_cat/indices?v'

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

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ pkill -SIGUSR1 fluentd

ubuntu@lab4sys:~/lab4$ curl -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
'https://172.17.0.3:9200/_cat/indices?v'

health status index      uuid                                pri rep docs.count docs.deleted store.size pri.store.size
yellow open    fluentd tt3D9llmT2-XepVD70xpTQ      1   1         3             0        13kb
13kb

ubuntu@lab4sys:~/lab4$
```

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

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

```
ubuntu@lab4sys:~/lab4$ curl -s -u elastic:$ESPW --cacert ~/ubuntu/lab4/http_ca.crt -k
https://172.17.0.3:9200/fluentd/_search?pretty

{
  "took" : 6,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "fluentd",
        "_id" : "50xNZIEBDyHFTUMinlAN",
        "_score" : 1.0,
        "_source" : {
          "remote" : "172.17.0.1",
          "host" : "-",
          "user" : "-",
          "method" : "GET",
          "path" : "/"
        }
      }
    ]
  }
}
```

```

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

ubuntu@lab4sys:~/lab4$

```

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

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

4d. Adding Filters to the NGINX-Elasticsearch pipeline

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

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

```
### Scrub 404 errors from access log to prevent duplicate outputs to Elasticsearch
```

```

<filter nginx.access>
  @type grep
  <exclude>
    key code
    pattern 404
  </exclude>
</filter>

```

```

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

```

```
</match>
```

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

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

Send a `SIGUSR2` to Fluentd to reconfigure it:

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

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

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

```
...
```

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

```
...
```

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

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

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

```
{  
  "took" : 235,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 1,  
    "successful" : 1,  
    "skipped" : 0,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : {  
      "value" : 5,  
      "relation" : "eq"  
    },  
    "max_score" : 1.0,  
    "hits" : [  
      {  
        "_index" : "fluentd",  
        "_id" : "50xNZIEBDyHFTUMinlAN",  
        "_score" : 1.0,  
        "_source" : {  
          "remote" : "172.17.0.1",  
          "host" : "-",  
          "user" : "-",  
          "method" : "GET",  
          "path" : "/",  
          "code" : "200",  
          "size" : "615",  
          "referer" : "-",  
          "agent" : "curl/7.68.0",
```

```

    "http_x_forwarded_for" : "-"
  }
},
{
  "_index" : "fluentd",
  "_id" : "5exNZIEBDyHFTUMiNlAT",
  "_score" : 1.0,
  "_source" : {
    "remote" : "172.17.0.1",
    "host" : "-",
    "user" : "-",
    "method" : "GET",
    "path" : "/404",
    "code" : "404",
    "size" : "153",
    "referer" : "-",
    "agent" : "curl/7.68.0",
    "http_x_forwarded_for" : "-"
  }
},
{
  "_index" : "fluentd",
  "_id" : "5uxNZIEBDyHFTUMiOVA0",
  "_score" : 1.0,
  "_source" : {
    "log_level" : "error",
    "pid" : "31",
    "tid" : "31",
    "message" : "*3 open() \"/usr/share/nginx/html/404\" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host:
\"172.17.0.2\"\""
  }
},
{
  "_index" : "fluentd",
  "_id" : "5-xPZIEBDyHFTUMiSVAL",
  "_score" : 1.0,
  "_source" : {
    "log_level" : "error",
    "pid" : "31",
    "tid" : "31",
    "message" : "*4 open() \"/usr/share/nginx/html/404\" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host:
\"172.17.0.2\"\""
  }
},
{
  "_index" : "fluentd",
  "_id" : "60xPZIEBDyHFTUMiTfDV",
  "_score" : 1.0,
  "_source" : {
    "remote" : "172.17.0.1",
    "host" : "-",
    "user" : "-",
    "method" : "GET",
    "path" : "/",
    "code" : "200",
    "size" : "615",
    "referer" : "-",
    "agent" : "curl/7.68.0",
    "http_x_forwarded_for" : "-"
  }
}

```

```
}  
]  
}  
}  
}  
  
ubuntu@labsys:~/lab4$
```

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

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

Cleanup

Use CTRL C to terminate any running instances of Fluentd:

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

Now tear down any running Docker containers:

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

Congratulations, you have completed the lab!

LFS242 - Cloud Native Logging with Fluentd

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

- What plugins were loaded now?
 - forward, stdout
- What events will the `<filter>` directive capture?
 - All of them, because of the double wildcard (**) pattern in the opening
- How many times did Fluentd report that the worker is now running?
 - Once. It tried to report twice but internal Fluentd events are no longer being captured without a label.
- Using the `filter_grep` documentation on the Fluentd website, create a configuration that excludes all INFO events

```
<filter>
  @type grep
  <exclude>
    key message
    pattern INFO
  </exclude>
</filter>
```

- Find a pattern that will only print events that mention "file"

```
@type grep
<regexp>
  key message
</regexp>
  pattern file
</filter>
```

- Will the original log message be preserved in this configuration? If so, under what key?
 - Yes, it will be nested under the "report" key.
- Will there be a password key appended to the record?
 - No. There is no "password" parameter being declared under the `<record>` subdirective, so no "password" key will be appended or edited.
- How many events were recorded?
 - Three. Two GET requests and one error.
- Use `cat` to read one of the NGINX log files, and compare how that log is presented in Elasticsearch.
 - NGINX log file

```
ubuntu@labsys:~/lab4$ cat /tmp/lab4/nginx/error.log
```

```
2021/07/22 20:04:02 [error] 31#31: *4 open() "/usr/share/nginx/html/404" failed (2: No such file or
directory), client: 172.17.0.1, server: localhost, request: "GET /404 HTTP/1.1", host: "172.17.0.2"
```

- Elasticsearch

```
...
{
```



```
"_index" : "fluentd",
"_type" : "_doc",
"_id" : "ViHRz3oBbInj6EVHmnXs",
"_score" : 1.0,
"_source" : {
  "log_level" : "error",
  "pid" : "31",
  "tid" : "31",
  "message" : "*4 open() \"/usr/share/nginx/html/404\" failed (2: No such file or directory),
client: 172.17.0.1, server: localhost, request: \"GET /404 HTTP/1.1\", host: \"172.17.0.2\"\"
}

...
```

LFS242 - Cloud Native Logging with Fluentd

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

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

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

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

Objectives

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

0. Prepare the lab system

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

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

#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

...

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

This lab will use Docker to run an instance of Apache Webserver. Lab 1-B has more detailed information on the Docker installation process.

Install Docker with the quick installation script for Debian:

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

...
```

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

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

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

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

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-
  ldap libjansson4 liblua5.2-0 ssl-cert
...
Setting up apache2 (2.4.41-4ubuntu3.11) ...
...
ubuntu@labsys:~$
```

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

1. Exploring Parser Plugins

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

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

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

1a. Using manual regular expressions

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

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

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

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

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

```

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

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

ubuntu@labsys:~/lab5$

```

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

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

```

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

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

```

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

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

```

ubuntu@labsys:~$ cd ~/lab5

```

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

HTTP/1.1 200 OK
Date: Mon, 20 Jun 2022 16:11:00 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Mon, 20 Jun 2022 16:06:36 GMT
ETag: "2aa6-5e1e349deba8"
Accept-Ranges: bytes
Content-Length: 10918
Vary: Accept-Encoding
Content-Type: text/html

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

...

ubuntu@lab5$
```

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

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

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

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

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

ubuntu@lab5$
```

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

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

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

1b. Application Specific Parser Plugins

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

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

```
ubuntu@lab5$ nano lab5.conf && cat $_
```

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

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

ubuntu@lab5sys:~/lab5$
```

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

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

```
ubuntu@lab5sys:~/lab5$ kill -SIGUSR2 fluentd
```

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

```
ubuntu@lab5sys:~/lab5$ curl -I http://localhost

...

ubuntu@lab5sys:~/lab5$
```

Then, check Fluentd:

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

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

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

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

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

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

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

```
ubuntu@lab5sys:~/lab5$ nano lab5.conf && cat $_
```

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

```
<source>
  @type forward
</source>
```

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

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

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

Reconfigure Fluentd with a SIGUSR2:

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

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

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

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

...

c0a73062c976531f975605861bc65f92293a5cf5fb5357828ad3d23256a297d7

ubuntu@lab5:~/lab5$
```

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

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

Check the Fluentd terminal:

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

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

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

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

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

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

172.17.0.2

ubuntu@labsys:~/lab5$
```

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

Send a curl event to the containerized httpd instance:

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

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

ubuntu@labsys:~/lab5$
```

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

Check how Fluentd formatted the event:

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

```

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

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

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

```
ubuntu@lab5sys:~/lab5$ nano lab5.conf && cat $_

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

<source>
  @type forward
</source>

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

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

ubuntu@lab5sys:~/lab5$
```

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

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

Reconfigure Fluentd with `pkill` in your working terminal:

```
ubuntu@lab5sys:~/lab5$ pkill -SIGUSR2 fluentd
```

```
...

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

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

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

Rerun `curl` on the container IP and localhost:

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

And check the Fluentd event:

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

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

2. Exploring Format Plugins

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

2a. Changing Event Metadata

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

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

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

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

<source>
  @type forward
</source>

<filter apache-cont>
```

```

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

<match apache**>
  @type stdout
  <format>
    @type out_file
    delimiter "COMMA"(',')
    output_tag true
    output_time true
    time_format %Y%m%dT%H%M%S%Z
  </format>
</match>

ubuntu@lab5:~/lab5$

```

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

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

Reload Fluentd with `pkill -SIGUSR2` :

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

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

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

```

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

...

ubuntu@lab5:~/lab5$

```

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

```

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

```

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

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

2b. Changing Event log formats

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

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

https://github.com/mia-0032/fluent-plugin-formatter_pretty_json

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

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

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

ubuntu@lab5:~/lab5$
```

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

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

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

<source>
  @type forward
</source>

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

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

ubuntu@lab5:~/lab5$
```

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

Cleanup

Now tear down any running Docker containers:

```

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

```

Use CTRL C to terminate any running instances of Fluentd:

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

ubuntu@lab5:~/lab5$
```

Congratulations, you have completed the lab!

LFS242 - Cloud Native Logging with Fluentd

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

- Who made the request?
 - The `curl` client
- From where was the request made?
 - A local address, noted as `:::1`
- What tag was assigned to the event? Was this something that was set by the user?
 - `apache2.access` was set by the user as the `tag` parameter for the `<source>` directive tailing the Apache log
- What is the most significant difference between the two logs?
 - There are headers attached to each of the pieces of the log in the Fluentd event that are not present in the original.
- Remove the parse subdirective from the source directive; how did it turn out in Fluentd?
 - Fluentd didn't reload - the `<parse>` subdirective is required for the `in_tail` plugin
- Where is the container sending httpd logs?
 - The container is sending the httpd logs to the container's standard output, which is then forwarded to Fluentd
- Which part of the Fluentd event for the container has the actual httpd log content?
 - Actual log content is attached to the "log" key.
- How is the time different?
 - The time is formatted in a short date format (YY/MM/DD) without any milliseconds.
- Using the information on <https://docs.ruby-lang.org/en/2.4.0/Time.html#method-i-strftime>, change the time format.
 - One example being: `time_format %Y%m%dT%H%M%S%z` which will produce the following event:

```
20210722T202032+0000,apache2.access,
{"host":"127.0.0.1","user":null,"method":"GET","path":"/","code":200,"size":11173,"referer":null,"agent":"curl/7.68.0"}
```

LFS242 - Cloud Native Logging with Fluentd

Lab 6 – Organizing complex configurations with labels and includes

Fluentd configurations can become complex as directives are added to a configuration file (especially if Fluentd is being used as a log aggregator to handle many inputs from multiple sources and outputs to multiple destinations). One way of organizing Fluentd configurations is through the use of labels. Labels group and directives together into processing pipelines using the label directive. Other directives can forward messages to a given labeled pipeline by name.

In addition to labels, entirely separate configuration files can be used to separate directives from each other. By using the `@include` parameter in a Fluentd configuration file, the directives contained in a remote configuration file can be loaded as though they were part of the original configuration file. Handling scenarios with multiple data pipelines can be greatly simplified through the use of labels and `@include` statements.

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

- Use Fluentd as a log aggregator and processor for multiple container logs
- Utilize labels to create multiple pipelines within a Fluentd configuration file
- Create modularized Fluentd configurations using `@include` statements

0. Prepare the lab system

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

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

#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

This lab will be using Docker to run instances of Redis and Apache, 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. Utilizing Labels to organize data processing pipelines in Fluentd

Labels allow a user to send events to specific groups of `<filter>` and `<match>` directives. Labels are initially set in `<source>` directives and are plugins available that allow labels to be changed for additional routing. Events that are routed through a label are only processed by the directives nested under a `<label>` directive whose pattern matches their set label.

Labels are a way of establishing pipelines without having to worry about the order of directives in the configuration file.

Fluentd will be used to manage several docker application logs, so to start, create a simple source/match configuration consisting of all forwards:

```
ubuntu@labsys:~$ mkdir ~/lab6 ; cd $_
ubuntu@labsys:~/lab6$ nano lab6.conf && cat $_

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

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

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

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

<match *>
  @type stdout
</match>

ubuntu@labsys:~$
```

Remember each of the ports listed under the `<source>` directives.

This configuration will allow the Docker containers in the next step to be started using Fluentd as the log engine. Docker requires a Fluentd instance listening on 24244 by default in order to start using it.

Launch Fluentd using this configuration:

```
ubuntu@labsys:~/lab6$ fluentd -c lab6.conf
```

```

2022-06-20 17:36:22 +0000 [info]: parsing config file is succeeded path="lab6.conf"

...

2022-06-20 17:36:22 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 17:36:22.953966251 +0000 fluent.info:
{"pid":49891,"ppid":49886,"worker":0,"message":"starting fluentd worker pid=49891 ppid=49886
worker=0"}
2022-06-20 17:36:22.954702599 +0000 fluent.info: {"port":24300,"bind":"0.0.0.0","message":"listening
port port=24300 bind=\"0.0.0.0\""}
2022-06-20 17:36:22.955711108 +0000 fluent.info: {"port":24200,"bind":"0.0.0.0","message":"listening
port port=24200 bind=\"0.0.0.0\""}
2022-06-20 17:36:22.956658285 +0000 fluent.info: {"port":24100,"bind":"0.0.0.0","message":"listening
port port=24100 bind=\"0.0.0.0\""}
2022-06-20 17:36:22.958331974 +0000 fluent.info: {"port":24000,"bind":"0.0.0.0","message":"listening
port port=24000 bind=\"0.0.0.0\""}
2022-06-20 17:36:22.958616379 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

```

There is now a Fluentd instance listening on the ports configured. Make sure to use these ports when launching the containers.

1. Launching containerized applications

For this lab, two different applications will be launched. One will be a WordPress instance backed by MariaDB, which is a fork of MySQL created by the original developers of MySQL. The second application will be a guestbook application from the Kubernetes documentation, found here: <https://kubernetes.io/docs/tutorials/stateless-application/guestbook/>. All of these will be run under Docker, and you will be interacting with them to generate events.

In a new working terminal, deploy the WordPress instance using the ports selected in the Fluentd configuration file:

```

ubuntu@lab6sys:~$ cd ~/lab6

ubuntu@lab6sys:~/lab6$ sudo docker run --name wordpress-db \
-d \
-e MYSQL_ROOT_PASSWORD=mypassword \
-e MYSQL_DATABASE=wpdb \
-e MYSQL_USER=wpuser \
-e MYSQL_PASSWORD=wppassword \
--log-driver fluentd --log-opt tag={{.Name}} --log-opt fluentd-address=localhost:24000 \
mariadb:10.7.4-focal

...

b9be61b57968443cd3edfc2708eb58b36147ca57d7ff216d8475baa9b44c6bd6

ubuntu@lab6sys:~$

```

- The WordPress database container will run MariaDB
- It will be named `wordpress-db` and run as a daemon in detached mode
- Each `-e` variable is needed to set the root credentials and database for MariaDB to function correctly
- The `--log-driver` and `--log-opt` s flags will ensure that it will forward traffic to Fluentd on port 24000

This will handle the backend database for WordPress, launch it now:

```

ubuntu@lab6sys:~/lab6$ sudo docker run --name wordpress \
-d \

```

```
-P --link wordpress-db \
-e WORDPRESS_DB_USER=wpuser \
-e WORDPRESS_DB_PASSWORD=wppassword \
-e WORDPRESS_DB_NAME=wpdb \
-e WORDPRESS_DB_HOST=wordpress-db \
--log-driver fluentd --log-opt tag={{.Name}} --log-opt fluentd-address=localhost:24100 \
wordpress:6.0.0-apache

...

439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926

ubuntu@labsys:~/lab6$
```

In order to establish a connection with the WordPress database container, the `--link` flag is used so that the WordPress container will see the wordpress-db container as a valid host on the internal container network.

- On which port will WordPress send its logs to Fluentd?

Now that WordPress is deployed, it's time to deploy another web application: a simple PHP-based Guestbook application. This application working alongside WordPress, and will provide a good example of how Fluentd can manage logs from two separate services.

Deploy the Redis database and frontend guestbook instances as containers:

```
ubuntu@labsys:~/lab6$ sudo docker network create guestbook

ubuntu@labsys:~/lab6$ sudo docker run --name guestbook-db \
-d \
-p 6379 \
--log-driver fluentd --log-opt tag={{.Name}} --log-opt fluentd-address=localhost:24200 \
--network guestbook \
k8s.gcr.io/redis:e2e

...

43cfc54a187d86384b591cef0cdad66e258e33a59d7fe3a7d3a9400cfb656db

ubuntu@labsys:~/lab6$ sudo docker run --name guestbook \
-d -P \
-e GET_HOSTS_FROM=env \
-e REDIS_MASTER_SERVICE_HOST=guestbook-db \
-e REDIS_SLAVE_SERVICE_HOST=guestbook-db \
--log-driver fluentd --log-opt tag={{.Name}} --log-opt fluentd-address=localhost:24300 \
--network guestbook \
gcr.io/google-samples/gb-frontend:v4

...

92afd9985ebb087d956c42fd7207d67f451e3851dea45e7fdfb50f3717c6ee89

ubuntu@labsys:~/lab6$
```

The Guestbook application should now be deployed on the virtual machine, sending application data to Fluentd listeners taking traffic on ports 24200 and 24300. Each of the containers will output all their log traffic to their local STDOUT buffers, which are picked up by Fluentd and sent as standardized JSON events.

This lab will require some interaction with both WordPress and the Guestbook. These interactions can be performed on the virtual machine's host using the local IP of the VM itself or the external IP address if using a cloud instance.

Collect the IP address of the virtual machine. For local VMs use the following command to look up the local ethernet IP address:

For cloud instances, simply use the same external IP used to SSH into the machine.

```
ubuntu@labsys:~/lab6$ hostname -I  
labsys 172.17.0.1  
ubuntu@labsys:~/lab6$
```

Then, using `docker ps`, find out which ports are mapped to each of the frontend services:

```
ubuntu@labsys:~/lab6$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED
92afd9985ebb	gcr.io/google-samples/gb-frontent:v4	"apache2-foreground"	guestbook	46 seconds ago
Up 45 seconds	0.0.0.0:49165->80/tcp, :::49165->80/tcp			
43cfc54a187d	k8s.gcr.io/redis:e2e	"redis-server /etc/r..."	guestbook-db	About a minute ago
Up About a minute	0.0.0.0:49164->6379/tcp, :::49164->6379/tcp			
439b40dd421f	wordpress:6.0.0-apache	"docker-entrypoint.s..."	wordpress	2 minutes ago
Up 2 minutes	0.0.0.0:49163->80/tcp, :::49163->80/tcp			
b9be61b57968	mariadb:10.7.4-focal	"docker-entrypoint.s..."	wordpress-db	2 minutes ago
Up 2 minutes	3306/tcp			

```
ubuntu@labsys:~/lab6$
```

The Guestbook frontend is mapped to port `49165` on the host machine, which points to port `80` on the container. The WordPress frontend is mapped to port `49163`, which maps to port `80` of its own container. Your resulting ports may vary, so moving forward be sure to select the ports that are shown in your terminal.

In a browser on either the local machine or the host machine, connect to each frontend using

`http://<VM External IP>:<Wordpress Container Port>` and
`http://<VM External IP>:<Guestbook Container Port>` to access WordPress and the Guestbook respectively.

34.222.173.218:49153/wp-admin/install.php



English (United States)
Afrikaans
العربية
العربية المغربية
অসমীয়া
Azərbaycan dili
گۆنئی آذربایجان

The WordPress Frontend should send you to an installation page that asks for a language selection. There is no need to proceed with the installation at this time.

Guestbook

Messages

Submit

The Guestbook should present a page with a simple text box for Guestbook entries with a submission button. There is no need to add any entries at this time, but feel free to do so to test functionality.

If you look back in the Fluentd terminal, each of the application containers should have sent their output to Fluentd's terminal:

```
...

2022-06-20 17:37:09.000000000 +0000 wordpress-db:
{"container_id":"b9be61b57968443cd3edfc2708eb58b36147ca57d7ff216d8475baa9b44c6bd6","container_name":
"/wordpress-db","source":"stderr","log":"2022-06-20 17:37:09 0 [Note] mariadb: ready for
connections."}

...

2022-06-20 17:37:11.000000000 +0000 wordpress:
{"container_id":"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926","container_name":
"/wordpress","source":"stderr","log":"[Mon Jun 20 17:37:11.691271 2022] [core:notice] [pid 1]
AH00094: Command line: 'apache2 -D FOREGROUND'"}

...

2022-06-20 17:38:17.000000000 +0000 guestbook-db: {"container_name":"/guestbook-
db","source":"stdout","log":"[1] 20 Jun 17:38:17.102 * The server is now ready to accept connections
on port 6379","container_id":"43cfc54a187d86384b591cefc0cdad66e258e33a59d7fe3a7d3a9400cfb656db"}

...

2022-06-20 17:38:43.000000000 +0000 guestbook:
{"container_id":"92afd9985ebb087d956c42fd7207d67f451e3851dea45e7fdfb50f3717c6ee89","container_name":
"/guestbook","source":"stdout","log":"[Mon Jun 20 17:38:43.453776 2022] [mpm_prefork:notice] [pid 1]
AH00163: Apache/2.4.10 (Debian) PHP/5.6.20 configured -- resuming normal operations"}

...
```

The multi-application environment is now ready to go with two applications - WordPress and a Guestbook - up and running.

2. Organizing pipelines with labels

Labels are a way of organizing directives inside a configuration file outside of to using tags. They are typically set inside a `<source>` directive, but they can be set at any point in a pipeline as long as a plugin supports the `@label` parameter.

Once a label is assigned to a set of events, it will be routed to a `<label>` directive with a pattern that matches the set label. Each `<label>` directive contains its own set of `<filter>` and `<match>` directives that are only executed against events that are captured by the `<label>` directive. Once captured, standard tag-based processing occurs.

`<label>` directives can be placed within any Fluentd configuration file; since events possessing a certain label are routed directly to a matching `<label>` directive, any other directives that precede the `<label>` directive are ignored.

2a. Organizing pipelines with labels

To begin, add @label parameters to each of the forward `<source>` directives:

```
ubuntu@labsys:~/lab6$ nano lab6.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>

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

<label wordpress>
<match **>
@type file
path /tmp/lab6/wordpress-log
<buffer>
timekey 60s
timekey_wait 1m
</buffer>
</match>
</label>

<label guestbook>
<match **>
@type file
path /tmp/lab6/guestbook-log
<buffer>
timekey 60s
timekey_wait 1m
</buffer>
</match>
</label>

ubuntu@labsys:~/lab6$
```

Once a label is established, a matching `<label>` directive **must** be present inside the configuration file. Attempting to start a configuration file using a label while omitting the matching `<label>` directive will prevent Fluentd from starting or reconfiguring. This is one of the few cases outside of a syntax error that Fluentd will not start due to a configuration omission.

Each of these `<label>` directives is configured to send log data to separate logs based on each application: one label will send all WordPress traffic from the database and frontend to `/tmp/wordpress-log` and all Guestbook database and Frontend traffic to `/tmp/guestbook-log`.

For this lab, each of the files will be written out every minute based on the `timekey_wait` option.

Send a `SIGUSR2` to the Fluentd instance to reconfigure it:

```
ubuntu@labsys:~/lab6$ pkill -SIGUSR2 fluentd
```

Check the Fluentd terminal to make sure that it reconfigured correctly:

```
...

2022-06-20 17:42:04 +0000 [info]: adding match in wordpress pattern="*" type="file"
2022-06-20 17:42:04.719798649 +0000 fluent.info: {"pattern":"*","type":"file","message":"adding
match in wordpress pattern=\"*\" type=\"file\""}
2022-06-20 17:42:04.751787989 +0000 fluent.info: {"pattern":"*","type":"file","message":"adding
match in guestbook pattern=\"*\" type=\"file\""}
2022-06-20 17:42:04 +0000 [info]: adding match in guestbook pattern="*" type="file"
2022-06-20 17:42:04.782443714 +0000 fluent.info: {"pattern":"*","type":"stdout","message":"adding
match pattern=\"*\" type=\"stdout\""}
2022-06-20 17:42:04 +0000 [info]: adding match pattern="*" type="stdout"
2022-06-20 17:42:04.807073789 +0000 fluent.info: {"message":"Oj isn't installed, fallback to Yajl as
json parser"}
2022-06-20 17:42:04 +0000 [info]: #0 Oj isn't installed, fallback to Yajl as json parser
2022-06-20 17:42:04.808101578 +0000 fluent.info: {"type":"forward","message":"adding source
type=\"forward\""}
2022-06-20 17:42:04 +0000 [info]: adding source type="forward"
2022-06-20 17:42:04.811695196 +0000 fluent.info: {"type":"forward","message":"adding source
type=\"forward\""}
2022-06-20 17:42:04 +0000 [info]: adding source type="forward"
2022-06-20 17:42:04.814970888 +0000 fluent.info: {"type":"forward","message":"adding source
type=\"forward\""}
2022-06-20 17:42:04 +0000 [info]: adding source type="forward"
2022-06-20 17:42:04.818252747 +0000 fluent.info: {"type":"forward","message":"adding source
type=\"forward\""}
2022-06-20 17:42:04 +0000 [info]: adding source type="forward"
2022-06-20 17:42:04 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 17:42:04 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 17:42:04 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:76c"
2022-06-20 17:42:04 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:794"
2022-06-20 17:42:04 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:780"
2022-06-20 17:42:04 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:730"
2022-06-20 17:42:05 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 17:42:05 +0000 [warn]: #0 define <match fluent.*> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 17:42:05 +0000 [info]: #0 listening port port=24300 bind="0.0.0.0"
2022-06-20 17:42:05 +0000 [info]: #0 listening port port=24200 bind="0.0.0.0"
2022-06-20 17:42:05 +0000 [info]: #0 listening port port=24100 bind="0.0.0.0"
2022-06-20 17:42:05 +0000 [info]: #0 listening port port=24000 bind="0.0.0.0"
```

If you see `adding match in wordpress pattern...`, then it succeeded. Since Fluentd was configured to send events over to a set of files rather than STDOUT, it will not be outputting any container-specific data.

Now that Fluentd is reconfigured, check the /tmp directory to see where the logs should be written to:

```
ubuntu@lab6:~/lab6$ ls -l /tmp/lab6

total 8
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:42 guestbook-log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:25 wordpress-log

ubuntu@lab6:~/lab6$
```

A pair of directories, guestbook-log and wordpress-log are now present.

- Are those directories where the final log files will be made?

2b. Generating Events from WordPress and the Guestbook:

It's time to create some events for Fluentd to process.

WordPress

In your browser window, click **Continue** in the WordPress Frontend GUI to proceed with the installation:

Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password

Strong

Important: You will need this password to log in. Please store it in a secure location.

Your Email
Double-check your email address before continuing.

Search Engine Visibility ☒ Discourage search engines from indexing this site
It is up to search engines to honor this request.

- Site Title: **LFS242**
- Username: **student**
- Your email: **student@lfs242.linuxfoundation.org**
- Discourage search engines from indexing this site
- Copy the password to your clipboard, or set something simple to remember

Then click **Install WordPress** when all fields are filled out:

Success!

WordPress has been installed. Thank you, and enjoy!

Username

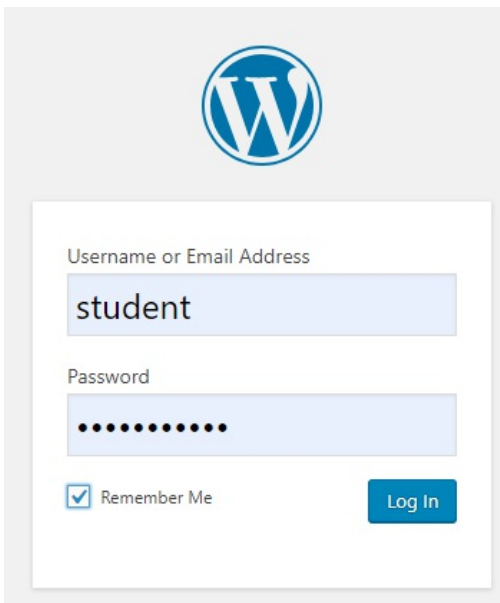
student

Password

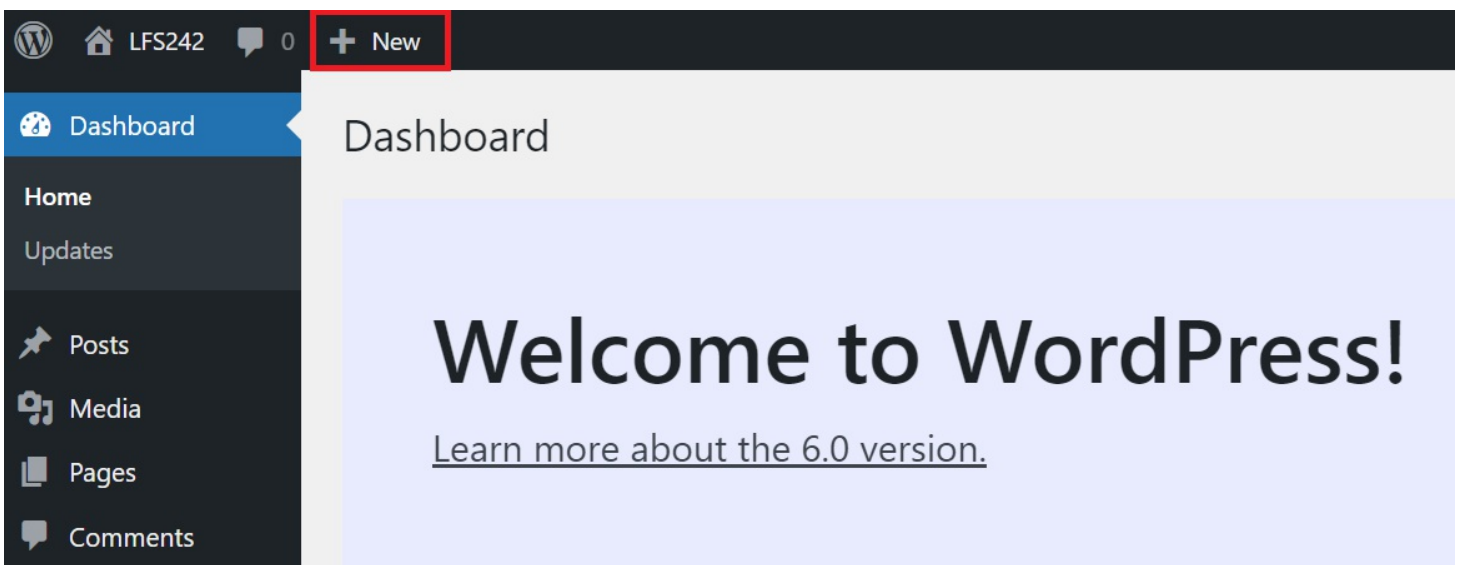
Your chosen password.

Log In

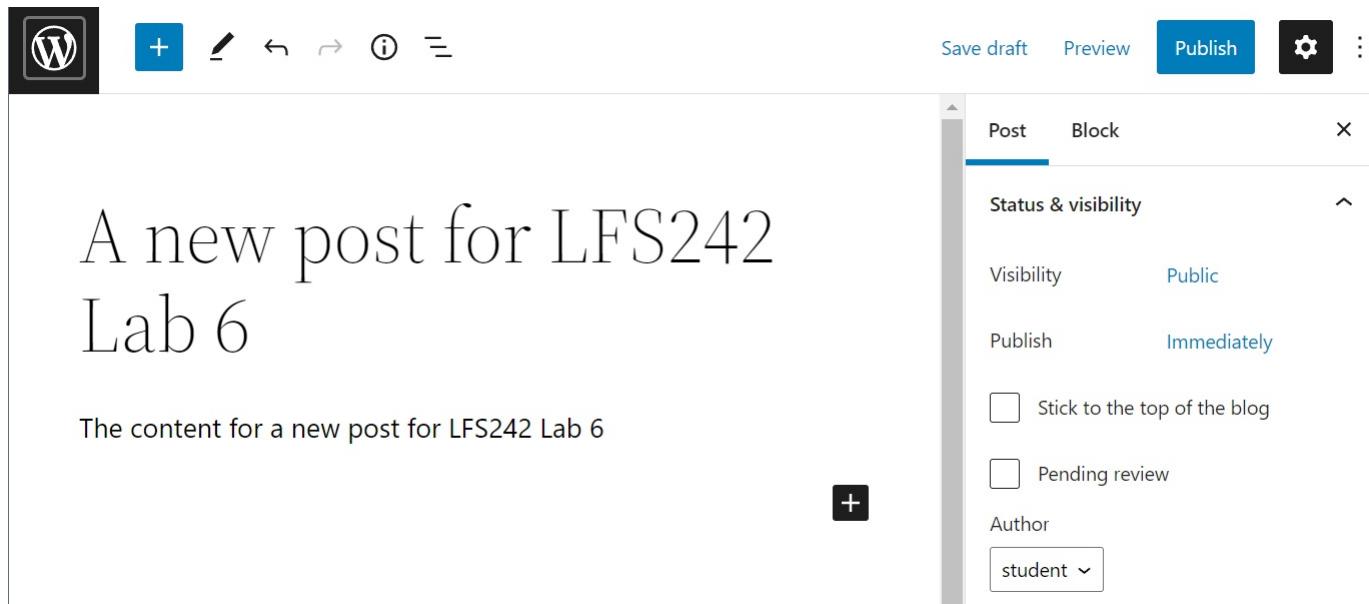
Login to the WordPress dashboard using the username and password you set:

A screenshot of the WordPress login form. At the top is the WordPress logo. Below it is a white box containing the login fields. The first field is labeled "Username or Email Address" and contains the text "student". The second field is labeled "Password" and contains a series of dots. Below the password field is a checkbox labeled "Remember Me" which is checked. To the right of the checkbox is a blue "Log In" button.

On the WordPress dashboard, click "+ New" at the top bar:

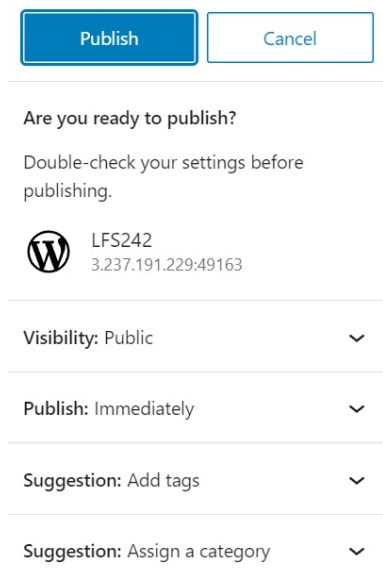
A screenshot of the WordPress dashboard. The top bar is dark grey and contains the WordPress logo, a home icon, the text "LFS242", a comment icon with "0", and a "+ New" button which is highlighted with a red box. The left sidebar is dark grey and contains a "Dashboard" button (highlighted with a blue bar), "Home", "Updates", "Posts", "Media", "Pages", and "Comments". The main content area is light grey and contains the text "Dashboard" and a large blue box with the text "Welcome to WordPress!" and a link "[Learn more about the 6.0 version.](\"#\")".

Close any dialogues that pop up, then add a title to the new post with some content:



The image shows the WordPress post editor interface. At the top, there's a toolbar with icons for adding blocks, editing, undo, redo, help, and a menu. On the right, there are buttons for 'Save draft', 'Preview', and 'Publish', along with a settings gear icon. The main content area displays the title 'A new post for LFS242 Lab 6' and the text 'The content for a new post for LFS242 Lab 6'. A small '+' icon is visible below the text. On the right sidebar, the 'Post' tab is selected, showing 'Status & visibility' settings: 'Visibility' is set to 'Public', 'Publish' is set to 'Immediately', and there are checkboxes for 'Stick to the top of the blog' and 'Pending review'. The 'Author' is set to 'student'.

When finished, click Publish... in the top right corner:



The image shows the 'Publish' confirmation dialog box. It has two buttons at the top: 'Publish' and 'Cancel'. Below them, it asks 'Are you ready to publish?' and advises to 'Double-check your settings before publishing.' The user's profile is shown as 'LFS242' with the ID '3,237,191,229:49163'. There are four expandable sections for settings: 'Visibility: Public', 'Publish: Immediately', 'Suggestion: Add tags', and 'Suggestion: Assign a category', each with a dropdown arrow.

And finally, click Publish to create your post:

A new post for LFS242 Lab 6 is now live.

What's next?

Post address

http://3.237.191.229:49163...

Copy

View Post

Add New Post

Guestbook

In the text box on the Guestbook page, type a message:

Guestbook

I am a guestbook entry!

Submit

Click Submit:

Guestbook

Messages

Submit

I am a guestbook entry!

Before you check the logs in the /tmp/lab6 directory, you may need to trigger more activity by performing actions inside the frontend Browser instances.

You can also force a buffer flush by sending a **SIGUSR1** to Fluentd, then check the directory:

```
ubuntu@lab6:~/lab6$ pkill -SIGUSR1 fluentd
```

```
ubuntu@lab6:~/lab6$ ls -l /tmp/lab6
```

```
total 96
```

```
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:45 guestbook-log
-rw-r--r-- 1 ubuntu ubuntu 472 Jun 20 17:45 guestbook-log.202206201745_0.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:45 wordpress-log
-rw-r--r-- 1 ubuntu ubuntu 33197 Jun 20 17:45 wordpress-log.202206201743_0.log
-rw-r--r-- 1 ubuntu ubuntu 37301 Jun 20 17:45 wordpress-log.202206201744_0.log
-rw-r--r-- 1 ubuntu ubuntu 4732 Jun 20 17:45 wordpress-log.202206201745_0.log
```

```
ubuntu@labsys:~/lab6$
```

Some new `.log` files have been written. Since the `file` output plugin is a buffered plugin, events are sent to a buffer that will periodically write out to the actual files after a given time or size requirement is met, which is configured by the `<match>` directive's `<buffer>` subdirective.

Inspect the new logs using `tail`, replacing the `log.*.log` with one printed in the previous step:

```
ubuntu@labsys:~/lab6$ tail -2 /tmp/lab6/wordpress-log.202206201743_0.log

2022-06-20T17:43:42+00:00      wordpress      {"log":"99.149.248.237 - - [20/Jun/2022:17:43:42
+0000] \"GET /wp-admin/admin-ajax.php?action=dashboard-
widgets&widget=dashboard_primary&pagenow=dashboard HTTP/1.1\" 200 981
\"http://3.237.191.229:49163/wp-admin/\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
Safari/537.36\"\", \"container_id\":\"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926\", \"
container_name\":\"/wordpress\", \"source\":\"stdout\"}
2022-06-20T17:43:50+00:00      wordpress      {"source":"stdout", "log":"127.0.0.1 - -
[20/Jun/2022:17:43:50 +0000] \"OPTIONS * HTTP/1.0\" 200 126 \"-\" \"Apache/2.4.53 (Debian)
PHP/7.4.30 (internal dummy
connection)\"\", \"container_id\":\"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926\", \"co
ntainer_name\":\"/wordpress\"}

ubuntu@labsys:~/lab6$
```

```
ubuntu@labsys:~/lab6$ tail -2 /tmp/lab6/guestbook-log.202206201745_0.log

2022-06-20T17:45:38+00:00      guestbook      {"log":"99.149.248.237 - - [20/Jun/2022:17:45:38
+0000] \"GET /guestbook.php?cmd=set&key=messages&value=,I%20am%20a%20guestbook%20entry! HTTP/1.1\"
200 255 \"http://3.237.191.229:49165/\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
Safari/537.36\"\", \"container_id\":\"92afd9985ebb087d956c42fd7207d67f451e3851dea45e7fdfb50f3717c6ee89\", \"
container_name\":\"/guestbook\", \"source\":\"stdout\"}

ubuntu@labsys:~/lab6$
```

Excellent! Each application is now having its logs sent to a unified log of their own. Fluentd is routing them based on the user-configurable label rather than the tag.

2c. Changing Labels with Relabel

Using the `relabel` output plugin, events can be assigned another tag in the middle of a processing pipeline without rewriting or removing the tag from the pipeline. This can be useful for changing the routing of labeled events, or aggregating logs that initially came under different tags.

In this example configuration, use the `relabel` plugin to separate the logs by purpose - by frontend and database:

```
ubuntu@labsys:~/lab6$ nano lab6.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/lab6/frontend-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
  </match>
</label>

<label db>
  <match **>

```

```
@type file
path /tmp/lab6/database-log
<buffer>
  timekey 60s
  timekey_wait 1m
</buffer>
</match>
</label>

ubuntu@labsys:~/lab6$
```

The `relabel` plugin outputs an event to another label and does not remove that event from the processing stream. Like the regular label parameter, it must be paired with a valid `<label>` directive once it is established inside a configuration file in order for Fluentd to reload correctly.

Reconfigure Fluentd with a `SIGUSR2`. Restart the `wordpress-db` and `guestbook-db` containers to ensure they reconnect and send events:

```
ubuntu@labsys:~/lab6$ kill -SIGUSR2 fluentd

ubuntu@labsys:~/lab6$ sudo docker restart wordpress-db guestbook-db

wordpress-db
guestbook-db

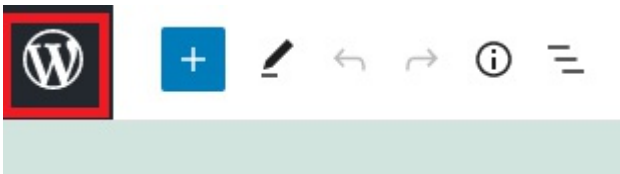
ubuntu@labsys:~/lab6$
```

```
...

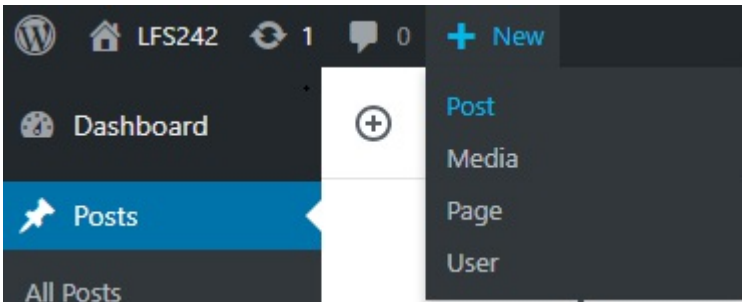
2022-06-20 17:56:44 +0000 [info]: adding match in wordpress pattern="wordpress" type="relabel"
2022-06-20 17:56:44 +0000 [info]: adding match in wordpress pattern="wordpress-db" type="relabel"
2022-06-20 17:56:44 +0000 [info]: adding match in guestbook pattern="guestbook" type="relabel"
2022-06-20 17:56:44 +0000 [info]: adding match in guestbook pattern="guestbook-db" type="relabel"
2022-06-20 17:56:44 +0000 [info]: adding match in frontend pattern="*" type="file"
2022-06-20 17:56:44 +0000 [info]: adding match in db pattern="*" type="file"
2022-06-20 17:56:44 +0000 [info]: adding source type="forward"
2022-06-20 17:56:44 +0000 [info]: adding source type="forward"
2022-06-20 17:56:44 +0000 [info]: adding source type="forward"
2022-06-20 17:56:44 +0000 [info]: adding source type="forward"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 17:56:44 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:9d8"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:9ec"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:a00"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:a14"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down output plugin type=:file plugin_id="object:938"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down output plugin type=:file plugin_id="object:974"
2022-06-20 17:56:44 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:9b0"
2022-06-20 17:56:45 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 17:56:45 +0000 [info]: #0 listening port port=24300 bind="0.0.0.0"
2022-06-20 17:56:45 +0000 [info]: #0 listening port port=24200 bind="0.0.0.0"
2022-06-20 17:56:45 +0000 [info]: #0 listening port port=24100 bind="0.0.0.0"
2022-06-20 17:56:45 +0000 [info]: #0 listening port port=24000 bind="0.0.0.0"
```

Confirm that you see `type=relabel` after Fluentd has started before proceeding. To test the new configuration, perform some activity on the WordPress and Guestbook frontend instances.

In WordPress, click the WordPress logo to return to the WordPress dashboard:



Click **+ New** in the top dashboard and click "Post":



Add a title, content, then publish a new post:



In the Guestbook, add an event by typing text in the box and clicking "Submit":

The next step needed activity!

Guestbook

Messages

Submit

I am a guestbook entry!

The next step needed activity!

Now flush the Fluentd buffers so ensure the events have been written.

```
ubuntu@labsys:~/lab6$ pkill -SIGUSR1 fluentd
```

Check the log directory, /tmp/lab6/ now:

```
ubuntu@labsys:~/lab6$ ls -l /tmp/lab6

total 164
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:58 database-log
-rw-r--r-- 1 ubuntu ubuntu 16458 Jun 20 17:58 database-log.202206201756_0.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:58 frontend-log
-rw-r--r-- 1 ubuntu ubuntu 9729 Jun 20 17:58 frontend-log.202206201757_0.log
-rw-r--r-- 1 ubuntu ubuntu 1743 Jun 20 17:58 frontend-log.202206201758_0.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:55 guestbook-log
-rw-r--r-- 1 ubuntu ubuntu 472 Jun 20 17:45 guestbook-log.202206201745_0.log
-rw-r--r-- 1 ubuntu ubuntu 1215 Jun 20 17:55 guestbook-log.202206201753_0.log
drwxr-xr-x 2 ubuntu ubuntu 4096 Jun 20 17:56 wordpress-log
-rw-r--r-- 1 ubuntu ubuntu 33197 Jun 20 17:45 wordpress-log.202206201743_0.log
-rw-r--r-- 1 ubuntu ubuntu 37301 Jun 20 17:45 wordpress-log.202206201744_0.log
-rw-r--r-- 1 ubuntu ubuntu 4732 Jun 20 17:45 wordpress-log.202206201745_0.log
-rw-r--r-- 1 ubuntu ubuntu 459 Jun 20 17:48 wordpress-log.202206201746_0.log
-rw-r--r-- 1 ubuntu ubuntu 459 Jun 20 17:50 wordpress-log.202206201748_0.log
-rw-r--r-- 1 ubuntu ubuntu 459 Jun 20 17:52 wordpress-log.202206201750_0.log
-rw-r--r-- 1 ubuntu ubuntu 459 Jun 20 17:54 wordpress-log.202206201752_0.log
-rw-r--r-- 1 ubuntu ubuntu 459 Jun 20 17:56 wordpress-log.202206201754_0.log

ubuntu@labsys:~/lab6$
```

There should now be **database-log** and **frontend-log** directories, which is where the buffers for those logs will be placed before they are written to actual log files.

Inspect one of the resulting log files (try using the latest one):

```
ubuntu@labsys:~/lab6$ tail -2 /tmp/lab6/frontend-log.202206201758_0.log

...

2022-06-20T17:58:47+00:00      guestbook
{"container_id":"92afd9985ebb087d956c42fd7207d67f451e3851dea45e7fdfb50f3717c6ee89","container_name":
"/guestbook","source":"stdout","log":"99.149.248.237 - - [20/Jun/2022:17:58:47 +0000] \"GET
/guestbook.php?
cmd=set&key=messages&value=,I%20am%20a%20guestbook%20entry!,The%20next%20step%20needed%20activity!
HTTP/1.1\" 200 255 \"http://3.237.191.229:49165/\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36\""}
2022-06-20T17:58:56+00:00      wordpress
{"container_id":"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926","container_name":
"/wordpress","source":"stdout","log":"99.149.248.237 - - [20/Jun/2022:17:58:56 +0000] \"POST /wp-
admin/admin-ajax.php HTTP/1.1\" 200 563 \"http://3.237.191.229:49163/wp-admin/post-new.php\"
\"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
Safari/537.36\""}
2022-06-20T17:58:56+00:00      wordpress
{"container_name":"/wordpress","source":"stdout","log":"99.149.248.237 - - [20/Jun/2022:17:58:56
+0000] \"POST /wp-json/wp/v2/posts/8/autosaves?_locale=user HTTP/1.1\" 200 1518
\"http://3.237.191.229:49163/wp-admin/post-new.php\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
Safari/537.36\"","container_id":"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926"}

ubuntu@labsys:~/lab6$
```

```
ubuntu@labsys:~/lab6$ tail -5 /tmp/lab6/database-log.202206201756_0.log

...
```



```

2022-06-20T17:56:58+00:00      guestbook-db
{"container_id":"43cfc54a187d86384b591cefc0cdad66e258e33a59d7fe3a7d3a9400cfb656db","container_name":
"/guestbook-db","source":"stdout","log":"[1] 20 Jun 17:56:58.056 * The server is now ready to accept
connections on port 6379"}
2022-06-20T17:56:58+00:00      wordpress-db      {"log":"2022-06-20 17:56:58 0 [Note] InnoDB: 128
rollback segments are
active.", "container_id":"b9be61b57968443cd3edfc2708eb58b36147ca57d7ff216d8475baa9b44c6bd6", "containe
r_name":"/wordpress-db", "source":"stderr"}

...

ubuntu@labsys:~/lab6$

```

Looks like those are the latest logs being written to - no other logs have been written to either the previous guestbook-log or wordpress-log files from the previous configuration. The combination of using labels and relabeling allow the user to fully influence the way events are routed in addition to setting tags.

Try the following:

- Reconfigure the database-log output to send events in msgpack
 - Add an event to the Guestbook to trigger activity here
- Using a filter, try to print only the log portion of events being sent to the frontend-log file
 - Interact with either the WordPress or guestbook frontends to generate events

3. Unifying separate configuration files using Include statements

@include statements can be used to load directives from separate configuration files. In this step, the Fluentd configuration made in this lab will be divided into separate files. By doing this, the known-good configurations listed in them can be reused at a later time. It will also greatly simplify the process of reconfiguring Fluentd, allowing new configuration files to be loaded without having to terminate the instance.

Create a file containing all of the sources:

```

ubuntu@labsys:~/lab6$ nano sources.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>

ubuntu@labsys:~/lab6$
```

Note that the source file is still utilizing labels, so one of the configuration files in the intended chain of source files needs to include matching `<label>` directives for each of those sources.

Create different configuration files based on each label:

```
ubuntu@labsys:~/lab6$ nano wordpress.conf && cat $_

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

ubuntu@labsys:~/lab6$
```

```
ubuntu@labsys:~/lab6$ nano guestbook.conf && cat $_

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

ubuntu@labsys:~/lab6$
```

In a previous step, it was shown that Fluentd can be started with configuration files that lack complete pipelines as long as all syntax requirements are met.

It is time to unify each of the files created above.

Modify the original configuration file to use the `@include` parameter to load all of these configuration files together:

```
ubuntu@labsys:~/lab6$ cp lab6.conf lab6-1.conf

ubuntu@labsys:~/lab6$ nano lab6.conf && cat $_

@include /home/ubuntu/lab6/sources.conf

@include /home/ubuntu/lab6/wordpress.conf
@include /home/ubuntu/lab6/guestbook.conf
```

```

<label frontend>
  <match **>
    @type file
    path /tmp/lab6/frontend-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
  </match>
</label>

<label db>
  <match **>
    @type file
    path /tmp/lab6/database-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
  </match>
</label>

ubuntu@labsys:~/lab6$

```

Before proceeding with the reconfiguration, answer the following questions:

- Why are a pair of `<label>` directives still inside the "master" configuration file?
- If those `<label>` directives were removed, would Fluentd still correctly reconfigure? Why or why not?

Send a `SIGUSR2` to Fluentd to have it reload the configuration file:

```

ubuntu@labsys:~/lab6$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/lab6$

```

```

ubuntu@labsys:~/lab6$ fluentd -c lab6.conf

2022-06-20 18:07:18 +0000 [info]: Reloading new config
2022-06-20 18:07:18 +0000 [info]: using configuration file: <ROOT>
<source>
  @type forward
  port 24000
  @label wordpress
</source>
<source>
  @type forward
  port 24100
  @label wordpress
</source>
<source>
  @type forward
  port 24200
  @label guestbook
</source>
<source>
  @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/lab6/frontend-log"
    <buffer>
      timekey 60s
      timekey_wait 1m
      path "/tmp/lab6/frontend-log"
    </buffer>
  </match>
</label>
<label db>
  <match **>
    @type file
    path "/tmp/lab6/database-log"
    <buffer>
      timekey 60s
      timekey_wait 1m
      path "/tmp/lab6/database-log"
    </buffer>
  </match>
</label>
</ROOT>

...

```

When Fluentd parses the @include parameters in a configuration file, it will load all of the directives into memory at runtime.

Restart the wordpress-db and guestbook-db containers to force them to reconnect to Fluentd:

```

ubuntu@labsys:~/lab6$ sudo docker restart guestbook-db wordpress-db

guestbook-db
wordpress-db

ubuntu@labsys:~/lab6$

```

Add a new blog post to WordPress and another guestbook entry, then flush the logs:

```

ubuntu@labsys:~/lab6$ pkill -sigusr1 fluentd

```

```

ubuntu@labsys:~/lab6$ tail -2 /tmp/lab6/database-log.202206201807_0.log

2022-06-20T18:07:59+00:00      wordpress-db      {"source":"stderr","log":"Version: '10.7.4-MariaDB-
1:10.7.4+maria~focal' socket: '/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary
distribution","container_id":"b9be61b57968443cd3edfc2708eb58b36147ca57d7ff216d8475baa9b44c6bd6","con
tainer_name":"/wordpress-db"}
2022-06-20T18:07:59+00:00      wordpress-db
{"container_id":"b9be61b57968443cd3edfc2708eb58b36147ca57d7ff216d8475baa9b44c6bd6","container_name":
"/wordpress-db","source":"stderr","log":"2022-06-20 18:07:59 0 [Note] InnoDB: Buffer pool(s) load
completed at 220620 18:07:59"}

ubuntu@labsys:~/lab6$ tail -1 /tmp/lab6/frontend-log.202206201808_0.log

2022-06-20T18:08:56+00:00      wordpress
{"container_id":"439b40dd421f1322072553c44871824813b1d2331365cf80195d1e779a79e926","container_name":
"/wordpress","source":"stdout","log":"99.149.248.237 - - [20/Jun/2022:18:08:56 +0000] \"GET /wp-
json/wp/v2/templates/twentytwentytwo//single?context=edit&_locale=user HTTP/1.1\" 200 3880
\"http://3.237.191.229:49163/wp-admin/post.php?post=10&action=edit\" \"Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36\""}

ubuntu@labsys:~/lab6$

```

This setup, which is made of separate files imported by `@include` parameter, should function identically to the previous configuration. By separating configurations into many separate files, easier troubleshooting and configuration modularity can be achieved.

The `@include` parameter can also be used against web URLs, so anything uploaded to a network registry or even GitHub can be reused if the Fluentd instance attempting to use those files can communicate with the remote server or host. Incorporating separate, established Fluentd configurations with the `@include` parameter makes the process of maintaining and reusing them much easier.

Now that you have worked with both labels and include in this lab, try to perform more complex configurations:

- Try using a wildcard `*` to simplify the primary Fluentd configuration file

Cleanup

To prepare for any subsequent labs, please shut down any existing instances using the following commands:

Tear down any running Docker containers:

```

ubuntu@labsys:~/lab6$ sudo docker container rm \
$(sudo docker container stop wordpress-db wordpress guestbook-db guestbook)

wordpress-db
wordpress
guestbook-db
guestbook

ubuntu@labsys:~/lab6$

```

Use `CTRL C` to terminate any locally running instances of Fluentd:

```

^C
...

2022-06-20 18:11:00 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab6$ cd

```

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

Congratulations, you have completed the lab!

LFS242 - Cloud Native Logging with Fluentd

Lab 6 – Organizing complex configurations with labels and includes

- On which port will WordPress send its logs to Fluentd?
 - Port 24100
- Are those directories where the final log files will be made?
 - No, those are the buffer directories where events will be stored until Fluentd flushes the buffer to write the files.
- Reconfigure the database-log output to send events in msgpack

```
<label db>
  <match **>
    @type file
    path /tmp/lab6/database-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
    <format>
      @type msgpack
    </format>
  </match>
</label>
```

- Using a filter, try to print only the log portion of events being sent to the frontend-log file

```
<label frontend>
  <filter **>
    @type record_transformer
    remove_keys container_id,container_name,source,
  </filter>
  <match **>
    @type file
    path /tmp/lab6/frontend-log
    <buffer>
      timekey 60s
      timekey_wait 1m
    </buffer>
  </match>
</label>
```

- Why are a pair of `<label>` directives still inside the "master" configuration file?
 - wordpress.conf and guestbook.conf contain relabel plugins that send events to the `db` and `frontend` labels. Neither of those files have a `<label>` directive to inside them, so they are placed inside the main configuration file.
- If those `<label>` directives were removed, would Fluentd still correctly reconfigure? Why or why not?
 - Fluentd would not start because a `@label` parameter always needs a matching `<label>` directive present.
- Try using a wildcard `*` to simplify the primary Fluentd configuration file

```
@include /home/ubuntu/lab6/*.conf

<label frontend>
```

```
<match **>
  @type file
  path /tmp/lab6/frontend-log
  <buffer>
    timekey 60s
    timekey_wait 1m
  </buffer>
</match>
</label>

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


LFS242 - Cloud Native Logging with Fluentd

Lab 7 – Using Multiple Fluentd Instances, Configuring High Availability and Testing Failover in Fluentd

Fluentd is capable of supporting multiple instances using the `in_forward` and `out_forward` plugins, allowing many Fluentd instances to communicate with each other. This allows many smaller instances of Fluentd to send logs to a centralized log aggregator for further processing. In addition to forwarding and aggregation, additional instances of Fluentd can be employed to make the logging pipeline more robust.

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

- Stand up a logging pipeline consisting of a log forwarder and log aggregator
- Explore the failover behavior of a multi-Fluentd instance deployment
- Learn how to set up a Highly Available Fluentd deployment

0. Prepare the lab system

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

```
ubuntu@labsys:~$ nano fluentd-setup && cat $_
#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

This lab will also be using Docker to run additional Fluentd instances, 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. Setting Up a log forwarder & log aggregator

One of the simplest cases of using multiple instances of Fluentd is in a forwarder-aggregator setup. The log forwarder instance is intended to be deployed near to or on the same node as the application it is collecting logs from, and is typically light weight in nature. Processing is shifted to a log aggregator instance, which listens for traffic from log forwarders. Since log aggregators will typically perform the processing, they tend to be larger instances and may even be deployed on nodes provisioned just for Fluentd.

To begin, create a log aggregator Fluentd configuration:

```
ubuntu@labsys:~$ mkdir ~/lab7 && cd $_
ubuntu@labsys:~/lab7$ mkdir aggregator1 && cd $_
ubuntu@labsys:~/lab7/aggregator1$ nano aggregator1.conf && cat $_

<system>
  process_name aggregator1
</system>

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

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab7/aggregator1$
```

The log aggregator in this example will resemble the Fluentd instances set up in previous labs. Log aggregators will host the majority (if not all) of the processing power within a multi-Fluentd deployment, so this is where the more complex `<filter>` and `<match>` directives will be hosted.

Run an instance of this log aggregator with Docker, making sure to mount the aggregator1 directory to make your configuration file available to the container:

```
ubuntu@labsys:~/lab7/aggregator1$ sudo docker run -p 24500:24500 -d \
--name aggregator1 \
-v $HOME/lab7/aggregator1:/fluentd/etc \
-e FLUENTD_CONF=aggregator*.conf fluent/fluentd:v1.14.6-1.1

...

0d7a050297f1d8109c8f3f82c697554a650828e40ceee32bd7c655073b79ba23

ubuntu@labsys:~/lab7/aggregator1$ sudo docker logs aggregator1 --tail 5

2022-06-20 18:16:25 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-20 18:16:25 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 18:16:25.283589771 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-20 18:16:25.283930039 +0000 fluent.info: {"port":24500,"bind":"0.0.0.0","message":"listening
port port=24500 bind=\"0.0.0.0\""}
2022-06-20 18:16:25.284440933 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

ubuntu@labsys:~/lab7/aggregator1$
```

Since this log aggregator is set up to output to STDOUT, run it without the `-d` flag to keep an eye on its output. This terminal will be locked.

- Feel free to set up the aggregator to output to a file or other destination
- Add additional filters for this aggregator
- To make changes, edit the aggregator1.conf and send the SIGHUP using `docker kill --signal=HUP aggregator1`

Once the aggregator is set up, it is ready to receive events.

Fluentd instances communicate with each other through the use of the `forward` input and output plugins over the network. In order for a forwarder-aggregator Fluentd setup to function correctly, all Fluentd instances must be able to communicate with each other. To start, the IP address is needed to send traffic to this log aggregator.

Open a new terminal session so that you can continue working. In your working terminal, retrieve the IP address of the aggregator container:

```
ubuntu@labsys:~/lab7/aggregator1$ cd ~/lab7

ubuntu@labsys:~/lab7$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' aggregator1

172.17.0.2

ubuntu@labsys:~/lab7$
```

The next piece in a multi-instance Fluentd deployment is a log forwarder. It is important for this instance to be as light weight as possible - typical log forwarders should have few, or no, filters configured to avoid taking resources from their primary source program.

It's time to set up a forwarder. In another terminal, create a simple configuration:

```
ubuntu@labsys:~$ mkdir ~/lab7/forwarder && cd $_

ubuntu@labsys:~/lab7/forwarder$ nano forwarder.conf && cat $_

<system>
  process_name forwarder
</system>

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

<match>
  @type forward
  <server>
    name aggregator1
    host 172.17.0.2
    port 24500
  </server>
</match>

ubuntu@labsys:~/lab7/forwarder$
```

The heart of the log forwarder Fluentd instance is the `out_forward` plugin. This plugin allows one or more servers to be specified as event destinations - the destination servers must be reachable by a hostname, FQDN, or IP address and be configured with an active `in_forward` `<source>` directive.

To target the docker aggregator1, the IP address of the container, and the forward port was included.

A local installation of Fluentd will be used to run the log forwarder:

```
ubuntu@labsys:~/lab7/forwarder$ fluentd -c forwarder.conf

2022-06-20 18:17:49 +0000 [info]: parsing config file is succeeded path="forwarder.conf"
2022-06-20 18:17:49 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 18:17:49 +0000 [info]: adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:730"
2022-06-20 18:17:49 +0000 [warn]: define <match fluent.**> to capture fluentd logs in top level is
deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 18:17:49 +0000 [info]: using configuration file: <ROOT>
  <system>
    process_name "forwarder"
  </system>
  <source>
    @type forward
    port 32767
  </source>
  <match>
    @type forward
    <server>
      name "aggregator1"
      host "172.17.0.2"
      port 24500
    </server>
  </match>
</ROOT>
2022-06-20 18:17:49 +0000 [info]: starting fluentd-1.14.6 pid=51762 ruby="2.7.0"
2022-06-20 18:17:49 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby2.7", "-Eascii-
8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c", "forwarder.conf", "--under-supervisor"]
2022-06-20 18:17:49 +0000 [info]: adding match pattern="*" type="forward"
2022-06-20 18:17:49 +0000 [info]: #0 adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:730"
2022-06-20 18:17:49 +0000 [info]: adding source type="forward"
2022-06-20 18:17:49 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 18:17:49 +0000 [info]: #0 starting fluentd worker pid=51767 ppid=51762 worker=0
2022-06-20 18:17:49 +0000 [info]: #0 listening port port=32767 bind="0.0.0.0"
2022-06-20 18:17:49 +0000 [info]: #0 fluentd worker is now running worker=0
```

This will also lock this terminal, so open another one to continue working.

If the correct IP (or hostname) and port were provided, the log forwarder instance should confirm that it found the log aggregator instance:

```
2022-06-20 18:17:49 +0000 [info]: #0 adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:730"
```

There is an additional setting in the output that was not included in the original setup - weight. The weight of a log forwarder determines the distribution for load balancing. This functionality will be explored later in this lab.

Now that both a log aggregator and log forwarder have been configured, it's time to test it.

In your working terminal, send a basic message to the log forwarder:

```
ubuntu@labsys:~/lab7$ echo '{"json":"message"}' | fluent-cat aggregate.me --port 32767
```

```
ubuntu@labsys:~/lab7$
```

In the forwarder terminal, see if there was any activity:

```
...
```

```
2022-06-20 18:17:49 +0000 [info]: #0 listening port port=32767 bind="0.0.0.0"
2022-06-20 18:17:49 +0000 [info]: #0 fluentd worker is now running worker=0
```

There was none recorded. Very light weight forwarders that only have a `<match>` directive will not have any additional outputs to STDOUT (unless configured for a `<filter>` directive using the filter_stdout plugin).

Check the aggregator1 terminal:

```
ubuntu@labsys:~/lab7$ sudo docker logs aggregator1 --tail 5

2022-06-20 18:16:25.284440933 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 18:17:49.965756189 +0000 fluent.info:
{"pid":51767,"ppid":51762,"worker":0,"message":"starting fluentd worker pid=51767 ppid=51762
worker=0"}
2022-06-20 18:17:49.966932466 +0000 fluent.info: {"port":32767,"bind":"0.0.0.0","message":"listening
port port=32767 bind=\"0.0.0.0\""}
2022-06-20 18:17:49.967525910 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 18:18:43.672084950 +0000 aggregate.me: {"json":"message"}

ubuntu@labsys:~/lab7$
```

After a short wait, the event will display in the `aggregator1` log output. When a `<match>` directive with the forward plugin is set up, events are sent directly to the server or servers that are configured.

Notice that there was a delay. The `out_forward` plugin is a buffered plugin, so it stored the events into a temporary buffer in disk or memory before sending them to their destination. This can be adjusted using `<buffer>` subdirectives under the forward match configuration.

2. Failover Scenario: Aggregator loss

Earlier in the lab, it was mentioned that in order for a forwarder-aggregator architecture to work correctly, both instances must be able to communicate with each other. What happens if the aggregator goes down?

Use `docker stop` on the aggregator1 terminal to stop the container:

```
ubuntu@labsys:~/lab7$ sudo docker stop aggregator1

aggregator1

ubuntu@labsys:~/lab7$
```

This container will remain on your system, so later in the lab you will only need to use `docker start` to get it running again.

However, the forwarder no longer has a destination to send its events to.

Try to send some events to it in your working terminal, flushing the buffer after sending it:

```
ubuntu@labsys:~/lab7$ echo '{"alive":"yes"}' | fluent-cat aggregate.me --port 32767
```

```
ubuntu@labsys:~/lab7$ pkill -SIGUSR1 fluentd
```

After some time, the forwarder will realize that its aggregator is down:

```
...
2022-06-20 18:21:05 +0000 [info]: #0 force flushing buffered events
2022-06-20 18:21:05 +0000 [info]: #0 flushing all buffer forcedly
2022-06-20 18:21:22 +0000 [warn]: #0 failed to flush the buffer. retry_times=0 next_retry_time=2022-06-20 18:21:24 +0000 chunk="5e1e52a8da14c865466db311acfb6f0f" error_class=Errno::EHOSTUNREACH
error="No route to host - connect(2) for \"172.17.0.2\" port 24500"
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/socket.rb:62:in `initialize'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/socket.rb:62:in `new'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/socket.rb:62:in `socket_create_tcp'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:410:in `create_transfer_socket'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward/connection_manager.rb:46:in `call'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward/connection_manager.rb:46:in `connect'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:807:in `connect'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:676:in `send_data'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:365:in `block in write'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward/load_balancer.rb:46:in `block in select_healthy_node'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward/load_balancer.rb:37:in `times'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward/load_balancer.rb:37:in `select_healthy_node'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/service_discovery/manager.rb:108:in `select_service'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin_helper/service_discovery.rb:82:in `service_discovery_select_service'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:365:in `write'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/output.rb:1179:in `try_flush'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/out_forward.rb:353:in `try_flush'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/output.rb:1500:in `flush_thread_run'
2022-06-20 18:21:22 +0000 [warn]: #0 /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/plugin/output.rb:499:in `block (2 levels) in start'
2022-06-20 18:21:22 +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'
2022-06-20 18:21:31 +0000 [warn]: #0 detached forwarding server 'aggregator1' host="172.17.0.2" port=24500 phi=16.14318557817386 phi_threshold=16
2022-06-20 18:21:31 +0000 [warn]: #0 failed to flush the buffer. retry_times=1 next_retry_time=2022-06-20 18:21:34 +0000 chunk="5e1e52a8da14c865466db311acfb6f0f" error_class=Errno::EHOSTUNREACH
error="No route to host - connect(2) for \"172.17.0.2\" port 24500"
...
```

It will continue to retry the send until it makes contact with a log aggregator again.

Bring the log aggregator container back online. Since `docker stop` was used to shut it down, use `docker container start` to restore the log aggregator and `docker container attach` to return to its terminal:

```
ubuntu@lab7$ sudo docker container start aggregator1
aggregator1
ubuntu@lab7$
```

Eventually, the forwarder will make contact once the log aggregator instance is operational:

```
...

2022-06-20 18:22:10 +0000 [warn]: #0 recovered forwarding server 'aggregator1' host="172.17.0.2"
port=24500
```

Once the next retry time is reached, all of the buffered events will be sent to the log aggregator in one chunk:

```
...

2022-06-20 18:22:28 +0000 [warn]: #0 retry succeeded. chunk_id="5e1e52a8da14c865466db311acfb6f0f"
```

The log aggregator should have received the original event, and any other events that were buffered in its absence:

```
ubuntu@lab7$ sudo docker logs aggregator1 --tail 10

2022-06-20 18:21:05.495140251 +0000 fluent.info: {"message":"flushing all buffer forcedly"}
2022-06-20 18:21:22.549751002 +0000 fluent.warn: {"retry_times":0,"next_retry_time":"2022-06-20
18:21:24 +0000","chunk":"5e1e52a8da14c865466db311acfb6f0f","error":"#<Errno::EHOSTUNREACH: No route
to host - connect(2) for \"172.17.0.2\" port 24500>","message":"failed to flush the buffer.
retry_times=0 next_retry_time=2022-06-20 18:21:24 +0000 chunk=\"5e1e52a8da14c865466db311acfb6f0f\"
error_class=Errno::EHOSTUNREACH error=\"No route to host - connect(2) for \\\"172.17.0.2\\\" port
24500\\\""}

...

2022-06-20 18:22:10.383419640 +0000 fluent.warn:
{"host":"172.17.0.2","port":24500,"message":"recovered forwarding server 'aggregator1'
host=\"172.17.0.2\" port=24500"}
2022-06-20 18:22:28.939801239 +0000 fluent.warn:
{"chunk_id":"5e1e52a8da14c865466db311acfb6f0f","message":"retry succeeded.
chunk_id=\"5e1e52a8da14c865466db311acfb6f0f\""}

```

The fact that the `out_forward` plugin is a buffered plugin is advantageous because, in the event that a log aggregator destination goes down, the data is not lost. Due to how loose the log aggregator setup is, all of the other retry events were captured and recorded to the log aggregator as well. Periodic retries are one way to guard against data loss and the ability to save events within a buffer means that, with a small time delay, events are not lost.

Knowing that buffering and retrying are involved with failover with Fluentd, try the following:

- Use **Ctrl C** on the forwarder first, send to an event to it, then restore the forwarder. What happened?
 - What behavior was observed?
 - Was the observed behavior similar or different to what happened when a log aggregator fails?
- Send an event to the forwarder, **then** use **Ctrl C** to shut it down before it sends events to the log aggregator.
 - Restore the forwarder after a minute or so.
 - Did the log aggregator ever receive the event?

3. Exploring High Availability

In the previous step, a single log aggregator went down and events were subject to a delay before they could be received. Time sensitive data may require little to no downtime.

This is a case that can be solved by adding another log aggregator instance for high availability.

Open another terminal and create a configuration for a second log aggregator:

```
ubuntu@labsys:~$ mkdir ~/lab7/aggregator2

ubuntu@labsys:~/lab7/aggregator2$ nano ~/lab7/aggregator2/aggregator2.conf && cat $_

<system>
  process_name aggregator2
</system>

<source>
  @type forward
  port 25500
</source>

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab7/aggregator2$
```

Since this log aggregator is intended to be used as a backup for the primary aggregator, it will be identically configured to the first one.

This lab shows a single node setup, meaning that a separate port needs to be declared for this setup to be functional. In some environments, the same port may be used if different nodes will host aggregator instances.

Run this log aggregator under Docker:

```
ubuntu@labsys:~/lab7/aggregator2$ sudo docker run -p 25500:25500 -d --name aggregator2 -v
$HOME/lab7/aggregator2:/fluentd/etc \
-e FLUENTD_CONF=aggregator*.conf fluent/fluentd:v1.14.6-1.1

ccce3f7d4dd85016bbff250500eb138fe6ca21e0c55622835e5e601f1b61f1e5

ubuntu@labsys:~/lab7$ sudo docker logs aggregator2 --tail 5

2022-06-20 18:29:36 +0000 [info]: #0 listening port port=25500 bind="0.0.0.0"
2022-06-20 18:29:36 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 18:29:36.529998794 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-20 18:29:36.530408174 +0000 fluent.info: {"port":25500,"bind":"0.0.0.0","message":"listening
port port=25500 bind=\"0.0.0.0\""}

```



```
2022-06-20 18:29:36.530880378 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now running worker=0"}
```

```
ubuntu@labsys:~/lab7$
```

As before, the forwarder will need a valid address in order to communicate with the second log aggregator. This can be an IP address, a DNS name, or even a service name if you are running Fluentd within some service mesh-backed system (like Kubernetes).

In your working terminal, use `docker inspect` to retrieve the new Fluentd instance's IP address:

```
ubuntu@labsys:~/lab7$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' aggregator2
```

```
172.17.0.3
```

```
ubuntu@labsys:~/lab7$
```

The aggregator instance should now be functional. It's time to make the forwarder aware that there is a backup instance of Fluentd available for failover.

```
ubuntu@labsys:~/lab7$ nano ~/lab7/forwarder/forwarder.conf && cat $_
```

```
<system>
  process_name forwarder
</system>
```

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

```
<match>
  @type forward
  <server>
    name aggregator1
    host 172.17.0.2
    port 24500
  </server>
  <server>
    name aggregator2
    host 172.17.0.3
    port 25500
    standby true
  </server>
</match>
```

```
ubuntu@labsys:~/lab7$
```

- A second `<server>` subdirective has been added to the forward `<match>` directive
- The `standby true` parameter is in place to inform the forwarder that aggregator2 is a backup instance

Send a `SIGUSR2` to the forwarder to reconfigure it:

```
ubuntu@labsys:~/lab7$ pkill -SIGUSR2 fluentd
```

```
pkill: killing pid 51964 failed: Operation not permitted
pkill: killing pid 52132 failed: Operation not permitted
```

```
ubuntu@labsys:~/lab7$
```

pkill is trying to restart the containerized Fluentd processes running in your containers but fails to do so. You can safely ignore those warnings.

```
2022-06-20 18:30:35 +0000 [info]: Reloading new config
2022-06-20 18:30:35 +0000 [info]: adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:780"
2022-06-20 18:30:35 +0000 [info]: adding forwarding server 'aggregator2' host="172.17.0.3"
port=25500 weight=60 plugin_id="object:780"
2022-06-20 18:30:35 +0000 [info]: using configuration file: <ROOT>
<system>
  process_name forwarder
</system>
<source>
  @type forward
  port 32767
</source>
<match>
  @type forward
  <server>
    name "aggregator1"
    host "172.17.0.2"
    port 24500
  </server>
  <server>
    name "aggregator2"
    host "172.17.0.3"
    port 25500
    standby true
  </server>
</match>
</ROOT>
2022-06-20 18:30:35 +0000 [info]: shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 18:30:35 +0000 [info]: shutting down output plugin type=:forward plugin_id="object:730"
2022-06-20 18:30:35 +0000 [info]: adding match pattern="*" type="forward"
2022-06-20 18:30:35 +0000 [info]: #0 adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:848"
2022-06-20 18:30:35 +0000 [info]: #0 adding forwarding server 'aggregator2' host="172.17.0.3"
port=25500 weight=60 plugin_id="object:848"
2022-06-20 18:30:35 +0000 [info]: adding source type="forward"
2022-06-20 18:30:35 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 18:30:35 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 18:30:36 +0000 [info]: #0 shutting down output plugin type=:forward
plugin_id="object:730"
2022-06-20 18:30:36 +0000 [info]: #0 restart fluentd worker worker=0
2022-06-20 18:30:36 +0000 [warn]: #0 define <match fluent.*> to capture fluentd logs in top level
is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 18:30:36 +0000 [info]: #0 listening port port=32767 bind="0.0.0.0"
```

- What weight was assigned to aggregator2?

Now, stop aggregator1 again using **docker stop** :

```
ubuntu@labsys:~/lab7$ sudo docker container stop aggregator1

aggregator1
```

```
ubuntu@lab7$ sudo docker logs aggregator1 --tail 5

2022-06-20 18:31:06.940383345 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
stopping worker=0"}
2022-06-20 18:31:06 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 18:31:06 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 18:31:06 +0000 [info]: #0 shutting down output plugin type=:stdout plugin_id="object:730"
2022-06-20 18:31:07 +0000 [info]: Worker 0 finished with status 0

ubuntu@lab7$
```

In your working terminal, send an event over to the forwarder, ensuring to flush the buffers after:

```
ubuntu@lab7$ echo '{"json":"message"}' | fluent-cat aggregate.me --port 32767

ubuntu@lab7$ pkill -sigusr1 fluentd

pkill: killing pid 119973 failed: Operation not permitted

ubuntu@lab7$
```

Eventually, the forwarder will realize that aggregator1 is no longer accessible:

```
2022-06-20 18:31:29 +0000 [info]: #0 force flushing buffered events
2022-06-20 18:31:29 +0000 [info]: #0 flushing all buffer forcedly
2022-06-20 18:31:45 +0000 [warn]: #0 detached forwarding server 'aggregator1' host="172.17.0.2"
port=24500 phi=17.037374590761466 phi_threshold=16
2022-06-20 18:31:45 +0000 [warn]: #0 using standby node 172.17.0.3:25500 weight=60
```

Depending on when the event was sent, it may try to flush its events, only to find that it cannot connect to it! Luckily, the standby node was picked up by the forwarder and the event was sent there instead.

Check the events of the backup aggregator, aggregator2:

```
ubuntu@lab7$ sudo docker logs aggregator2 --tail 5

2022-06-20 18:29:36 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 18:29:36.529998794 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-20 18:29:36.530408174 +0000 fluent.info: {"port":25500,"bind":"0.0.0.0","message":"listening
port port=25500 bind=\"0.0.0.0\""}
2022-06-20 18:29:36.530880378 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-20 18:31:26.230841588 +0000 aggregate.me: {"json":"message"}

ubuntu@lab7$
```

The backup aggregator was able to pick up the events and receive them, with very little downtime in between. It will continue to do so until aggregator1 is restored.

Use `docker container start -a` to recover aggregator1:

```
ubuntu@lab7$ sudo docker container start aggregator1

ubuntu@lab7$ sudo docker logs aggregator1 --tail 5
```

```

2022-06-20 18:33:20 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-20 18:33:20 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-20 18:33:20.868116085 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-20 18:33:20.868543627 +0000 fluent.info: {"port":24500,"bind":"0.0.0.0","message":"listening
port port=24500 bind=\"0.0.0.0\""}
2022-06-20 18:33:20.869045569 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

ubuntu@labsys:~/lab7$

```

Soon after, the forwarder will be able to recover its connection with aggregator1:

```

...

2022-06-20 18:33:31 +0000 [warn]: #0 recovered forwarding server 'aggregator1' host="172.17.0.2"
port=24500

```

You have now learned how to perform a basic setup for a multi-instance Fluentd deployment, and how to ensure high availability with failover capabilities.

To recap, much of the high availability solution revolves around the **forward** input and output plugins, and their ability to communicate. As long as they are able to communicate, Fluentd instances will use a combination of repeated retries and event buffering to preserve events that come in before or after a Fluentd instance loss. Understanding how to best configure those plugins is central to a solid Fluentd deployment.

Try tweaking this solution for better performance with the following tasks:

- Remove the **standby true** from the aggregator2 **<server>** subdirective on the forwarder, and send events to it.
 - Does the behavior change?
 - If it does, does the **weight** parameter have any effect?

Cleanup

To prepare for any subsequent labs, please shut down any existing instances using the following commands:

Tear down any running Docker containers:

```

ubuntu@labsys:~/lab7$ sudo docker container rm $(sudo docker container stop aggregator1 aggregator2)

aggregator1
aggregator2

ubuntu@labsys:~/lab7$ cd

ubuntu@labsys:~$

```

Use **CTRL C** to terminate any locally running instances of Fluentd, like the forwarder:

```

^C

2022-06-20 18:36:54 +0000 [info]: Received graceful stop
2022-06-20 18:36:55 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 18:36:55 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 18:36:55 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:870"
2022-06-20 18:36:55 +0000 [info]: #0 shutting down output plugin type=:forward

```

```
plugin_id="object:848"  
2022-06-20 18:36:58 +0000 [info]: Worker 0 finished with status 0  
  
ubuntu@labsys:~/lab7/forwarder$ cd  
  
ubuntu@labsys:~$
```

It may take some time for the log forwarder to shut down, if it has not detected that both of the log aggregators it is connected to have gone down.

Congratulations, you have completed the lab!

LFS242 - Cloud Native Logging with Fluentd

Lab 7 – Using Multiple Fluentd Instances, Configuring High Availability and Testing Failover in Fluentd

- Use **Ctrl C** on the forwarder first, send to an event to it, then restore the forwarder. What happened? What behavior was observed? Was the observed behavior similar or different to what happened when a log aggregator fails?
 - The forwarder did not receive the message since the connection was refused. This is different from the log aggregator, which was able to receive the event after it came back because the forwarder kept trying to send it the buffered chunk. A raw event coming to a downed forwarder cannot do anything.

```
^C

2022-06-20 18:23:41 +0000 [info]: Received graceful stop
2022-06-20 18:23:42 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 18:23:42 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 18:23:42 +0000 [info]: #0 shutting down input plugin type=:forward plugin_id="object:758"
2022-06-20 18:23:43 +0000 [info]: #0 shutting down output plugin type=:forward plugin_id="object:730"
2022-06-20 18:23:43 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab7/forwarder$ echo '{"json":"message"}' | fluent-cat aggregate.me --port 32767

connect failed: Connection refused - connect(2) for "127.0.0.1" port 32767
connect failed: Connection refused - connect(2) for "127.0.0.1" port 32767

^C

Traceback (most recent call last):
  15: from /usr/local/bin/fluent-cat:23:in `<main>'
  14: from /usr/local/bin/fluent-cat:23:in `load'
  13: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/bin/fluent-cat:5: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/cat.rb:330:in `<top (required)>'
   9: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:189:in `write'
   8: from /usr/lib/ruby/2.7.0/monitor.rb:202:in `mon_synchronize'
   7: from /usr/lib/ruby/2.7.0/monitor.rb:202:in `synchronize'
   6: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:190:in `block in write'
   5: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:239:in `write_impl'
   4: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:259:in `get_socket'
   3: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:277:in `try_connect'
   2: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:277:in `try_connect'
   1: from /var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:276:in `try_connect'
/var/lib/gems/2.7.0/gems/fluentd-1.14.6/lib/fluent/command/cat.rb:276:in `sleep': Interrupt

ubuntu@labsys:~/lab7/forwarder$ fluentd -c forwarder.conf

2022-06-20 18:24:18 +0000 [info]: parsing config file is succeeded path="forwarder.conf"
2022-06-20 18:24:18 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 18:24:18 +0000 [info]: adding forwarding server 'aggregator1' host="172.17.0.2"
```

```

port=24500 weight=60 plugin_id="object:730"
2022-06-20 18:24:18 +0000 [warn]: define <match fluent.**> to capture fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 18:24:18 +0000 [info]: using configuration file: <ROOT>
  <system>
    process_name "forwarder"
  </system>
  <source>
    @type forward
    port 32767
  </source>
  <match>
    @type forward
    <server>
      name "aggregator1"
      host "172.17.0.2"
      port 24500
    </server>
  </match>
</ROOT>
2022-06-20 18:24:18 +0000 [info]: starting fluentd-1.14.6 pid=51991 ruby="2.7.0"
2022-06-20 18:24:18 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c", "forwarder.conf", "--under-supervisor"]
2022-06-20 18:24:18 +0000 [info]: adding match pattern="*" type="forward"
2022-06-20 18:24:18 +0000 [info]: #0 adding forwarding server 'aggregator1' host="172.17.0.2"
port=24500 weight=60 plugin_id="object:730"
2022-06-20 18:24:18 +0000 [info]: adding source type="forward"
2022-06-20 18:24:18 +0000 [warn]: #0 define <match fluent.**> to capture fluentd logs in top level is deprecated. Use <label @FLUENT_LOG> instead
2022-06-20 18:24:18 +0000 [info]: #0 starting fluentd worker pid=51996 ppid=51991 worker=0
2022-06-20 18:24:18 +0000 [info]: #0 listening port port=32767 bind="0.0.0.0"
2022-06-20 18:24:18 +0000 [info]: #0 fluentd worker is now running worker=0

```

- Send an event to the forwarder, **then** use **Ctrl C** to shut it down before it sends events to the log aggregator. Restore the forwarder after a minute or so. Did the log aggregator ever receive the event?
 - Yes, it did. The forwarder was able to receive the event then store it in its buffer before it went down.
- What weight was assigned to aggregator2?
 - 60
- Remove the **standby true** from the aggregator2 **<server>** subdirective on the forwarder, and send events to it. Does the behavior change? If it does, does the **weight** parameter have any effect?
 - The Fluentd deployment becomes a load balanced deployment, and the weight ratio will now influence how events are distributed between the two Fluentd aggregators.

LFS242 - Cloud Native Logging with Fluentd

Lab 8 – Monitoring Fluentd

Fluentd is typically a critical component in visibility solutions used to monitor applications and appliances. A Fluentd failure can eliminate the ability to monitor applications which rely on Fluentd for log forwarding. Monitoring Fluentd itself is therefore important so that application log forwarding outages do not occur silently.

This lab will cover four key metrics that a user needs to track when monitoring Fluentd:

- Fluentd process health to ensure that Fluentd itself is healthy
- Network connectivity to ensure that Fluentd can reach and can be reached by its source and destination systems
- Fluentd resource usage to ensure that Fluentd's processes have enough resources to perform their configured job
- Message throughput, measured in either bytes or messages per unit of time, to ensure Fluentd is actually receiving, processing and sending events

The first three of these metrics can be explored through host-level inspections, using tools such as `top`, `df`, and `ps`. The fourth requires the use of an external solution, like Prometheus, to collect Fluentd level metrics.

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

Objectives

- Learn where the key metrics of monitoring a unified logging layer are found in a Fluentd deployment
- Collect application metrics from Fluentd using Prometheus
- Explore the use of the REST API as an alternative to using Prometheus

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:~$ ./fluentd-setup

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

...

Done installing documentation for fluentd after 2 seconds
1 gem installed

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6
```



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

This lab will also be using Docker to run Prometheus instances, 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. Exploring key metrics

For the key metrics described above, it is important to understand where they fit in for any Fluentd deployment. The following sections will provide insight on what needs to be monitored and what host OS level tools (in this case, Ubuntu Linux) can be used to monitor those key metrics so that generic or custom monitoring solutions can be used to watch Fluentd.

1a. Process health

The first key metric to watch is whether the Fluentd process itself is alive and running. Fluentd launches at least two processes per instance: one supervisor instance, which receives traffic, and one worker instance, which executes the data processing pipeline. Naturally, without a Fluentd process functioning, none of the other metrics matter since Fluentd isn't working in the first place!

Run a Fluentd instance with the default setup:

```
ubuntu@labsys:~$ fluentd --setup ~
Installed /home/ubuntu/fluent.conf.
ubuntu@labsys:~$ fluentd -c ~/fluent.conf
...
2021-07-22 22:14:44 +0000 [info]: #0 fluentd worker is now running worker=0
```

Check the number of Fluentd and Ruby processes with `ps`, then `grep "fluentd\|ruby"` to retrieve the statuses for Ruby and Fluentd:

N.B. If you do not have a Fluentd instance, run `fluentd --setup ~` and run `fluentd -c ~/fluent.conf`

```
ubuntu@labsys:~$ ps -e | grep "fluentd\|ruby"

 52734 pts/0    00:00:00 fluentd
 52739 pts/0    00:00:00 ruby2.7

ubuntu@labsys:~$ ps -ef | grep "fluentd\|ruby"

ubuntu      52734   44064   4 20:14 pts/0    00:00:00 /usr/bin/ruby2.7 /usr/local/bin/fluentd -c
/home/ubuntu/fluent.conf
ubuntu      52739   52734   3 20:14 pts/0    00:00:00 /usr/bin/ruby2.7 -Eascii-8bit:ascii-8bit
/usr/local/bin/fluentd -c /home/ubuntu/fluent.conf --under-supervisor
```

```
ubuntu      52753   44150   0 20:14 pts/1    00:00:00 grep --color=auto fluentd\|ruby
ubuntu@labsys:~$
```

The **fluentd** process is the one that was launched by the user. It represents the supervisor instance that presents all configured sockets (such as forward, http, etc.) to all external sources and distributes events among workers. The ruby process, pid 4824, is a worker process - these processes execute all directives and pipelines inside a configuration file. The worker process is denoted with the **--under-supervisor** flag.

Shut down any Fluentd processes you have running before proceeding:

```
...
^C
2022-06-20 20:14:51 +0000 [info]: Received graceful stop
2022-06-20 20:14:51 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 20:14:51 +0000 [info]: #0 shutting down fluentd worker worker=0
...
2022-06-20 20:15:04 +0000 [info]: Worker 0 finished with status 0
ubuntu@labsys:~$
```

1b. Network Connectivity

Typical Fluentd deployments rely on the network to connect to both source and destination systems. If Fluentd is inaccessible to the network, it will be unable to gather events to process and distribute.

The `in_forward` and `in_http` input plugins open a TCP and HTTP port on a Fluentd process, presenting a built-in means to communicate with Fluentd. Configuring a Fluentd instance with either of these plugins allows external processes to check the availability of these ports.

In the case of the forward plugin, a forwarder Fluentd instance connecting to that port will continually send heartbeats to ensure that the receiving Fluentd instance is still alive. If a heartbeat fails after a specified amount of time, that forwarder will disconnect from the receiving instance. This process was explored in Lab 7.

A more universal monitoring approach is achieved by exposing a network endpoint that other applications can send requests to. For this, the `in_http` plugin is the solution. One case where this approach works would be with Kubernetes, which can utilize HTTP endpoints in liveness probes to continually check if a Fluentd process is still healthy.

Configure Fluentd with an HTTP source:

```
ubuntu@labsys:~/$ mkdir -p ~/lab8/monitored && cd $_
ubuntu@labsys:~/lab8/monitored$ nano monitored-fluentd.conf && cat $_

# Monitoring sources
<source>
  @type http
  port 32474
  @label HEARTBEAT
</source>

# Events coming in from monitoring sources will be thrown out
<label HEARTBEAT>
  <match>
    @type null
```

```
</match>
</label>

ubuntu@labsys:~/lab8/monitored$
```

Since this particular HTTP source is strictly meant for heartbeat traffic, all events produced by it will be discarded. This will ensure that Fluentd does not expend any other resources on its host to process a simple heartbeat. A label was used in this configuration in case other heartbeat sources need to be added.

Start Fluentd with that configuration:

```
ubuntu@labsys:~/lab8/monitored$ fluentd -c ~/lab8/monitored/monitored-fluentd.conf

2022-06-20 20:16:21 +0000 [info]: parsing config file is succeeded
path="/home/ubuntu/lab8/monitored/monitored-fluentd.conf"
2022-06-20 20:16:21 +0000 [info]: gem 'fluentd' version '1.14.6'
2022-06-20 20:16:21 +0000 [warn]: LoadError
2022-06-20 20:16:21 +0000 [info]: using configuration file: <ROOT>
  <source>
    @type http
    port 32474
    @label HEARTBEAT
  </source>
  <label HEARTBEAT>
    <match>
      @type null
    </match>
  </label>
</ROOT>
2022-06-20 20:16:21 +0000 [info]: starting fluentd-1.14.6 pid=52770 ruby="2.7.0"
2022-06-20 20:16:21 +0000 [info]: spawn command to main: cmdline=["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c", "/home/ubuntu/lab8/monitored/monitored-fluentd.conf", "--under-supervisor"]
2022-06-20 20:16:22 +0000 [info]: adding match in HEARTBEAT pattern="*" type="null"
2022-06-20 20:16:22 +0000 [info]: adding source type="http"
2022-06-20 20:16:22 +0000 [warn]: #0 LoadError
2022-06-20 20:16:22 +0000 [info]: #0 starting fluentd worker pid=52775 ppid=52770 worker=0
2022-06-20 20:16:22 +0000 [info]: #0 fluentd worker is now running worker=0
```

In another terminal, send a `curl` request to retrieve a status of Fluentd at that port, with the `-i` flag to include the protocol header for both the request and response:

```
ubuntu@labsys:~$ cd ~/lab8

ubuntu@labsys:~/lab8$ curl -i -X POST -d "json={"heartbeat":"ping"}" http://localhost:32474/ping

HTTP/1.1 200 OK
Content-Type: text/plain
Connection: Keep-Alive
Content-Length: 0

ubuntu@labsys:~/lab8$
```

By continually sending curl (or other HTTP) requests to this endpoint, any monitoring solution can be configured to keep track of whether Fluentd can be reached. The response "200 OK" from Fluentd signals that the instance can be reached on that port; this can serve as a general indicator for network connectivity to that host and instance.

1c. Host Resource usage

Fluentd requires host resources to perform its tasks:

- CPU to perform general calculations and processing
- Memory to load directive-provided configurations (and also as a buffering destination, if configured to do so)
- Local storage to maintain output plugin buffers (by default) and to store collected events in files if configured to do so

`top` can be used to measure total CPU and memory usage for the Fluentd supervisor and worker processes:

```
ubuntu@lab8sys:~/lab8$ top -p `pgrep -d "," fluentd\|ruby`

top - 20:19:38 up 5:39, 3 users, load average: 0.00, 0.00, 0.00
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, 397.6 free, 337.8 used, 3130.5 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 3235.7 avail Mem

   PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 52770 ubuntu    20   0 304844 45916 9060 S   0.0   1.2   0:00.92 fluentd
 52775 ubuntu    20   0 169644 45236 8952 S   0.0   1.1   0:00.73 ruby2.7

q

ubuntu@lab8sys:~/lab8$
```

Remember that a Fluentd instance consists of both a supervisor process run by the `Fluentd` command and a worker spawned by Ruby. In order to see Fluentd's total CPU and memory footprint, all supervisor and worker processes should be monitored. To retrieve the pids for each of those processes, use `pgrep` to search for both `fluentd` and `ruby` processes. Provide their pids to `top` for monitoring.

Disk I/O usage of Fluentd directories can be tracked using `iotop`. This can be useful to check how Fluentd is interacting with its buffer, as many output plugins utilize a buffer to store chunks of events before submitting them for final storage. It can also indicate that Fluentd is properly reading (tailing) and writing to files, if it is configured to do so.

Install `iotop`:

```
ubuntu@lab8sys:~/lab8$ sudo apt install iotop -y

...

Unpacking iotop (0.6-24-g733f3f8-1) ...
Setting up iotop (0.6-24-g733f3f8-1) ...
Processing triggers for man-db (2.9.1-1) ...

ubuntu@lab8sys:~/lab8$
```

Now use `iotop` to track the rate of disk interaction (read and write) for the Fluentd supervisor and worker. `sudo` is required:

```
ubuntu@lab8sys:~/lab8$ sudo iotop -p `pgrep fluentd -n` -p `pgrep ruby -n`

Total DISK READ:      0.00 B/s | Total DISK WRITE:      0.00 B/s
Current DISK READ:    0.00 B/s | Current DISK WRITE:    0.00 B/s
   TID  PRIO  USER    DISK READ  DISK WRITE  SWAPIN     IO>     COMMAND
   52770 be/4  ubuntu    0.00 B/s   0.00 B/s   ?unavailable? ruby2.7 /usr/local/bin/fluentd -c
/home/ubuntu/lab8/monitored/monitored-fluentd.conf
   52775 be/4  ubuntu    0.00 B/s   0.00 B/s   ?unavailable? ruby2.7 -Eascii-8bit:ascii-8bit
```

```
/usr/local/bin/fluentd -c /home/ubuntu/lab8/monitored/monitored-fluentd.conf --under-supervisor  
q  
ubuntu@labsys:~/lab8$
```

iotop requires separate flags per pid, so be sure to include a **-p** flag for both **fluentd** and **ruby** processes.

This Fluentd instance is idle at the moment, and is not configured with any buffered plugins that would cause Fluentd to use additional memory.

2. Monitoring Fluentd event throughput with Prometheus

Ensuring that Fluentd is alive, can be reached by the network, and has the resources to perform its processing duties does not provide a full picture. To truly understand if Fluentd is doing its job, it is important to measure its throughput: the total rate at which events are ingested and distributed.

Prometheus is an open-source, time-series metrics database that is widely used to provide metrics aggregation. It is often paired with Grafana to provide visual metrics evaluation. It is also a graduated CNCF project like Fluentd and it is the officially recommended monitoring solution for Fluentd.

Fluentd has a set of plugins that can be used to output its message throughput to metrics endpoints that can be collected by Prometheus.

2a. Installing Prometheus

Prometheus will be run in a container for this lab, but it first needs to be configured to listen for Fluentd traffic on a certain port. The fluentd-plugin-prometheus repo provides a Prometheus configuration that already listens for Fluentd traffic, so it will be used as the configuration for the Prometheus container.

Retrieve the prometheus.yaml containing a Fluentd configuration from the fluent-plugin-prometheus repo:

```
ubuntu@labsys:~/lab8$ wget https://raw.githubusercontent.com/fluent/fluent-plugin-  
prometheus/master/misc/prometheus.yaml  
  
--2022-06-20 20:22:12-- https://raw.githubusercontent.com/fluent/fluent-plugin-  
prometheus/master/misc/prometheus.yaml  
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133,  
185.199.108.133, ...  
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443...  
connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 310 [text/plain]  
Saving to: 'prometheus.yaml'  
  
prometheus.yaml                               100%[=====>]          310  --.-KB/s  
in 0s  
  
2022-06-20 20:22:12 (14.8 MB/s) - 'prometheus.yaml' saved [310/310]  
  
ubuntu@labsys:~/lab8$
```

N.B. If the file cannot be reached, refer to the example below.

Check the contents of the configuration:

```

ubuntu@lab8$ cat prometheus.yaml

# A job to scrape an endpoint of Fluentd running on localhost.
scrape_configs:
- job_name: 'prometheus'
  scrape_interval: 5s
  static_configs:
    - targets:
      - 'localhost:9090'
- job_name: 'fluentd'
  scrape_interval: 5s
  static_configs:
    - targets:
      - 'localhost:24231'
  metrics_path: /metrics

ubuntu@lab8$

```

Of note for Fluentd is the last line entry. It will tell Prometheus to scrape, or retrieve metrics from, a `/metrics` endpoint open at port 24231.

Run Prometheus in a container, binding the container ports 9090 and 24231 to the host equivalents and using the `prometheus.yaml` from the `fluent-plugin-prometheus` repo:

```

ubuntu@lab8$ sudo docker run -d --name prometheus --network host \
-v ~/lab8/prometheus.yaml:/etc/prometheus/prometheus.yml prom/prometheus:v2.36.2

...

45c8afb10d3c8840e7de9f470a5b8a0904fade6d0b9b36051ca5ce4b145e0d6


ubuntu@lab8$



```

The resulting Prometheus container is running with the following options:


- `-d` to daemonize the container, making it run in the background
- `--network host` to ensure the container uses the host's network, to allow it to resolve against localhost and all other ports with minimal configuration
- `-v ~/prometheus.yaml:/etc/prometheus/prometheus.yml` tells the Prometheus container to use the configuration listening for Fluentd traffic.

After launching a Prometheus container, navigate to the Prometheus GUI by going to the container host's IP address (local or otherwise) and accessing port 9090.


 Prometheus Alerts Graph Status ▾ Help



☐ Use local time ☐ Enable query history ☒ Enable autocomplete ☒ Enable highlighting ☒ Enable linter



Expression (press Shift+Enter for newlines)

 Execute

Table

Graph

<

Evaluation time

>

No data queried yet

Remove Panel

Add Panel

Prometheus is up and running. Now it's time to configure Fluentd to send metrics to Prometheus.

2b. Configure Fluentd to send metrics to Prometheus

There are a set of plugins that need to be configured for Fluentd to successfully report metrics to Prometheus. Fluentd does not install the Prometheus plugins by default.

Install the fluent-prometheus-plugin Ruby gem using `fluent-gem`:

```
ubuntu@lab8$ sudo fluent-gem install fluent-plugin-prometheus -N -v 2.0.3

Fetching fluent-plugin-prometheus-2.0.3.gem
Fetching prometheus-client-4.0.0.gem
Successfully installed prometheus-client-4.0.0
Successfully installed fluent-plugin-prometheus-2.0.3
2 gems installed

ubuntu@lab8$
```

N.B. To install a specific version of a gem, pass the `-v` flag followed the a version number. This lab has been tested with version 1.4.0 and 1.7.0.

With the Prometheus plugins installed, configured Fluentd to use them:

```
ubuntu@lab8$ nano prometheus-fluentd.conf && cat $_

<source>
  @type prometheus
</source>
<source>
  @type prometheus_output_monitor
</source>
<source>
  @type forward
</source>
<filter mod8.*>
  @type prometheus
```

```

<metric>
  name fluentd_retrieved_status_codes_in
  type histogram
  desc The total number of status code-bearing requests ingested
  key message
</metric>
</filter>
<match mod8.*>
  @type copy
  <store>
    @type file
    path /tmp/fluentd.monitored.log
  </store>
  <store>
    @type prometheus
    <metric>
      name fluentd_retrieved_status_codes_out
      type histogram
      desc The total number of status code-bearing requests delivered
      key message
    </metric>
  </store>
</match>

ubuntu@lab8sys:~/lab8$

```

The prometheus-fluentd.conf above uses four of the six plugins included in the Prometheus set:

- The `prometheus` input plugin, the first plugin configured above, exposes the `/metrics` endpoint for Fluentd. This is the endpoint that Prometheus will reach out to when it needs to retrieve, or scrape, metrics from Fluentd. This is required for all Fluentd-Prometheus integrations, as Prometheus will be unable to read Fluentd metrics otherwise.
- The `prometheus_output_monitor` exposes a variety of metrics for output plugins, such as buffer queue lengths and bytes read, the retry count, an error count, and more. This is similar to the more stable `prometheus_monitor plugin`, which does the same thing but only exposes three metrics emitted by buffered output plugins.
- The pipeline has been instrumented to count the flow of messages that bear a code, which will be counted as they progress through the pipeline. The filter `prometheus` plugin will count messages as they are ingested, and the output Prometheus plugin will count messages as they are sent out of the pipeline. The `<match>` directive in this configuration is using the `out_copy` plugin to simultaneously output events to both the Prometheus monitoring counter and a file.

There are two additional plugins that were not used: the `prometheus_monitor` and the `prometheus_tail_monitor` plugin, which provides metrics for the `in_tail` plugin.

Terminate the previous monitored Fluentd instance:

```

^C

2022-06-20 20:27:12 +0000 [info]: Received graceful stop
2022-06-20 20:27:13 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-20 20:27:13 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-20 20:27:13 +0000 [info]: #0 shutting down input plugin type=http plugin_id="object:758"
2022-06-20 20:27:13 +0000 [info]: #0 shutting down output plugin type=null plugin_id="object:730"
2022-06-20 20:27:14 +0000 [info]: Worker 0 finished with status 0

ubuntu@lab8sys:~/lab8/monitored$

```

Run Fluentd using that configuration:


```
ubuntu@lab8sys:~/lab8/monitored$ fluentd -c ~/lab8/prometheus-fluentd.conf -v
```

```
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="/home/ubuntu/lab8/prometheus-fluentd.conf"
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-formatter_pretty_json'
version '1.0.0'
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-prometheus' version
'2.0.3'
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: gem 'fluentd' version '1.14.6'
2022-06-20 20:27:25 +0000 [debug]: fluent/log.rb:309:debug: adding store type="file"
2022-06-20 20:27:25 +0000 [debug]: fluent/log.rb:309:debug: adding store type="prometheus"
2022-06-20 20:27:25 +0000 [debug]: fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: using configuration file: <ROOT>
  <source>
    @type prometheus
  </source>
  <source>
    @type prometheus_output_monitor
  </source>
  <source>
    @type forward
  </source>
  <filter mod8.*>
    @type prometheus
    <metric>
      name fluentd_retrieved_status_codes_in
      type histogram
      desc The total number of status code-bearing requests ingested
      key message
    </metric>
  </filter>
  <match mod8.*>
    @type copy
    <store>
      @type "file"
      path "/tmp/fluentd.monitored.log"
      <buffer time>
        path "/tmp/fluentd.monitored.log"
      </buffer>
    </store>
    <store>
      @type "prometheus"
      <metric>
        name fluentd_retrieved_status_codes_out
        type histogram
        desc The total number of status code-bearing requests delivered
        key message
      </metric>
    </store>
  </match>
</ROOT>
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: starting fluentd-1.14.6 pid=53141
ruby="2.7.0"
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: spawn command to main: cmdline=
["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "-c",
"/home/ubuntu/lab8/prometheus-fluentd.conf", "-v", "--under-supervisor"]
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: adding filter pattern="mod8.*"
type="prometheus"
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: adding match pattern="mod8.*" type="copy"
2022-06-20 20:27:25 +0000 [debug]: #0 fluent/log.rb:309:debug: adding store type="file"
2022-06-20 20:27:25 +0000 [debug]: #0 fluent/log.rb:309:debug: adding store type="prometheus"
```

```

2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: adding source type="prometheus"
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: adding source
type="prometheus_output_monitor"
2022-06-20 20:27:25 +0000 [info]: fluent/log.rb:330:info: adding source type="forward"
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 20:27:26 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=53146
ppid=53141 worker=0
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1900
stage_size=0 queue_size=0
2022-06-20 20:27:26 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24224
bind="0.0.0.0"
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: listening prometheus http server on
http://0.0.0.0:24231//metrics for worker0
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 20:27:26 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 20:27:26 +0000 [debug]: #0 fluent/log.rb:309:debug: Start webrick HTTP server listening

```

With Fluentd up and running, check if the metrics endpoint is exposed. This endpoint is only exposed by configuring the Prometheus input plugin, which opens the metrics endpoint on port 24231 by default. Fluentd will report that it is listening for Prometheus at DEBUG verbosity (running Fluentd with the `-v` flag).

Send a curl request to the metrics endpoint:

```

ubuntu@lab8:~/lab8$ curl http://localhost:24231/metrics

# TYPE fluentd_retrieved_status_codes_in histogram
# HELP fluentd_retrieved_status_codes_in The total number of status code-bearing requests ingested
fluentd_retrieved_status_codes_in_bucket{le="0.005"} 0.0

...

ubuntu@lab8:~/lab8$

```

These are all of the available metrics that can be scraped by Prometheus from Fluentd. You should be able to see the `fluentd_retrieved_status_codes_out` and `fluentd_retrieved_status_codes_in` metrics configured in the Prometheus filter and output plugins. The metrics prefixed by `fluentd_output_status_` were provided by the `prometheus_output_monitor` plugin.

Now check if Prometheus itself has scraped these metrics. Open Prometheus in a browser session, or if you already have one, hit refresh. You should now see more metrics available for selection under the metric explorer:

complete

lighting ☒ Enable linter



Metrics Explorer

Search

fluentd_output_status_buffer_available_space_ratio





fluentd_output_status_buffer_queue_length

In another terminal, submit several events to Fluentd, forcing a flush by sending a `sigusr1`:



```
ubuntu@labsys:~/lab8$ echo '{"message":'$RANDOM'}' | fluent-cat mod8.prometheus
ubuntu@labsys:~/lab8$ echo '{"message":'$RANDOM'}' | fluent-cat mod8.prometheus
...
ubuntu@labsys:~/lab8$ echo '{"message":'$RANDOM'}' | fluent-cat mod8.prometheus
ubuntu@labsys:~/lab8$ pkill -sigusr1 fluentd
ubuntu@labsys:~/lab8$
```

Each message will bear a random number, representing an integer code being sent to Fluentd.

Select the `fluentd_retrieved_status_codes_in_count` metric from the dropdown and click `execute`. Depending on which one you choose, there may not be any data available; some may not be present at all until they receive a value:

 Prometheus Alerts Graph Status ▾ Help   

☐ Use local time ☐ Enable query history ☒ Enable autocomplete ☒ Enable highlighting ☒ Enable linter

 fluentd_retrieved_status_codes_in_count 

Execute

Table **Graph** Load time: 87ms Resolution: 14s Result series: 1

< Evaluation time >

fluentd_retrieved_status_codes_in_count(instance="localhost:24231", job="fluentd") 5

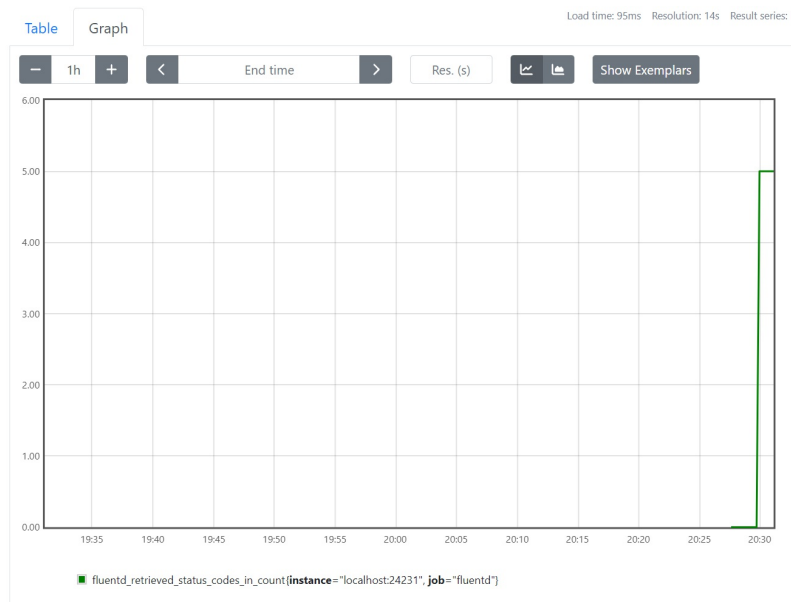
Curl the metrics endpoint again, this time grepping for a specific metric, like the custom message counters configured earlier, `fluentd_retrieved_status_codes_`:

```
ubuntu@labsys:~/lab8$ curl -s http://localhost:24231/metrics | grep
fluentd_retrieved_status_codes_in_count
fluentd_retrieved_status_codes_in_count 5.0
```

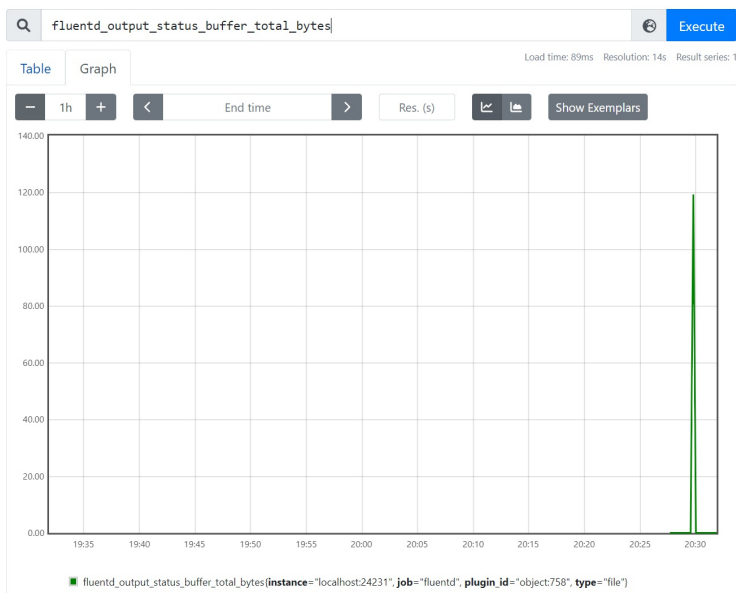
```
ubuntu@labsys:~/lab8$
```

The events received and output have been counted, which shows with `fluentd_retrieved_status_codes_out_count` metric.

Click on "Graph" to see a visual representation of the `fluentd_retrieved_status_codes_in_count` metric:



Since the file buffered plugin is used, it's worthwhile to check if the buffer size grew at some point as well by looking at the `fluentd_output_status_buffer_total_bytes` metric:



You have successfully configured Prometheus to scrape throughput metrics (among others) from Fluentd! Continue sending events to Fluentd and seeing how your metrics increment in Prometheus.

3. Other methods of monitoring Fluentd

Prometheus is not the only way to monitor Fluentd. The `in_monitor_agent` is a built-in plugin that exposes some Fluentd metrics to a REST API.

Append the directives below the `# REST API metrics` comment of code to the end of the `prometheus-fluentd.conf` file:

```

ubuntu@lab8sys:~/lab8$ nano prometheus-fluentd.conf && cat $_

<source>
  @type prometheus
</source>
<source>
  @type prometheus_output_monitor
</source>
<source>
  @type forward
</source>
<filter mod8.*>
  @type prometheus
  <metric>
    name fluentd_retrieved_status_codes_in
    type histogram
    desc The total number of status code-bearing requests ingested
    key message
  </metric>
</filter>
<match mod8.*>
  @type copy
  <store>
    @type file
    path /tmp/fluentd.monitored.log
  </store>
  <store>
    @type prometheus
    <metric>
      name fluentd_retrieved_status_codes_out
      type histogram
      desc The total number of status code-bearing requests delivered
      key message
    </metric>
  </store>
</match>

# REST API metrics

<source>
  @type monitor_agent
  bind 0.0.0.0
  port 32767
  include_config true
  # Emit metrics as Fluentd events
  tag mod8.monitoring
  emit_interval 30
  @label REST
</source>

<label REST>
  <match >
    @type file
    path /tmp/lab8/rest.monitor
  </match>
</label>

ubuntu@lab8sys:~/lab8$

```

This will open an endpoint on the REST API on port 32767, allowing metrics to be reported to the rest API on a per-plugin basis.

Reconfigure Fluentd with a **SIGUSR2** :

```
ubuntu@lab8:~/lab8$ kill -SIGUSR2 fluentd

ubuntu@lab8:~/lab8$
```

The Fluentd terminal should report its reconfiguration, adding the `monitor_agent` plugin to the end:

```
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=11680
stage_size=0 queue_size=0
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=11580
stage_size=0 queue_size=0
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: listening monitoring http server on
http://0.0.0.0:32767/api/plugins for worker0
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: Start webrick HTTP server listening
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: tag parameter is specified. Emit
plugins info to 'mod8.monitoring'
2022-06-20 20:32:45 +0000 [info]: #0 fluent/log.rb:330:info: listening port port=24224
bind="0.0.0.0"
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: listening prometheus http server on
http://0.0.0.0:24231/metrics for worker0
2022-06-20 20:32:45 +0000 [debug]: #0 fluent/log.rb:309:debug: Start webrick HTTP server listening
```

Fluentd indicates that it has loaded the REST api metrics with `adding source type="monitor_agent"` . At DEBUG verbosity, it will also provide the port and endpoint URL for a worker's metrics. Any events emitted by that `<source>` directive using the `monitor_agent` plugin will be sent to a file at `/tmp/lab8`

With Fluentd reconfigured, send a curl request to the endpoint:

```
ubuntu@lab8:~/lab8$ curl -s http://localhost:32767/api/plugins.json

{"plugins":[{"plugin_id":"object:2ddc","plugin_category":"input","type":"prometheus","config":
{"@type":"prometheus"},"output_plugin":false,"retry_count":null,"emit_records":0,"emit_size":0},
{"plugin_id":"object:2df0","plugin_category":"input","type":"prometheus_output_monitor","config":
{"@type":"prometheus_output_monitor"},"output_plugin":false,"retry_count":null,"emit_records":0,"emi
t_size":0},{"plugin_id":"object:2e04","plugin_category":"input","type":"forward","config":
{"@type":"forward"},"output_plugin":false,"retry_count":null,"emit_records":0,"emit_size":0},
{"plugin_id":"object:2e18","plugin_category":"input","type":"monitor_agent","config":
{"@type":"monitor_agent","bind":"0.0.0.0","port":"32767","include_config":"true","tag":"mod8.monitor
ing","emit_interval":"30","@label":"REST"},"output_plugin":false,"retry_count":null,"emit_records":0
,"emit_size":0},{"plugin_id":"object:2d78","plugin_category":"output","type":"copy","config":
{"@type":"copy"},"output_plugin":false,"retry_count":0},
{"plugin_id":"object:2d8c","plugin_category":"output","type":"file","config":
{"@type":"file","path":"/tmp/fluentd.monitored.log"},"output_plugin":true,"buffer_queue_length":0,"b
uffer_timekeys":
[],"buffer_total_queued_size":0,"retry_count":0,"emit_records":0,"emit_size":0,"emit_count":0,"write
_count":0,"rollback_count":0,"slow_flush_count":0,"flush_time_count":0,"buffer_stage_length":0,"buff
er_stage_byte_size":0,"buffer_queue_byte_size":0,"buffer_available_buffer_space_ratios":100.0,"retry
":{"plugin_id":"object:2dc8","plugin_category":"output","type":"prometheus","config":
{"@type":"prometheus"},"output_plugin":true,"retry_count":0,"emit_records":0,"emit_size":0,"emit_cou
nt":0,"write_count":0,"rollback_count":0,"slow_flush_count":0,"flush_time_count":0,"retry":{"}}
```

```
{
  "plugin_id": "object:2d64",
  "plugin_category": "filter",
  "type": "prometheus",
  "config": {
    "@type": "prometheus",
    "output_plugin": false,
    "retry_count": null,
    "emit_records": 0,
    "emit_size": 0
  },
  "plugin_id": "object:2d28",
  "plugin_category": "output",
  "type": "file",
  "config": {
    "@type": "file",
    "path": "/tmp/lab8/rest.monitor",
    "output_plugin": true,
    "buffer_queue_length": 0,
    "buffer_timekeys": [],
    "buffer_total_queued_size": 0,
    "retry_count": 0,
    "emit_records": 0,
    "emit_size": 0,
    "emit_count": 0,
    "write_count": 0,
    "rollback_count": 0,
    "slow_flush_count": 0,
    "flush_time_count": 0,
    "buffer_stage_length": 0,
    "buffer_stage_byte_size": 0,
    "buffer_queue_byte_size": 0,
    "buffer_available_buffer_space_ratios": 100.0,
    "retry": {}
  }
}
```

ubuntu@lab8sys:~/lab8\$

Although Fluentd is reporting metrics, but it's not necessarily consumable in this form.

To remedy this, install `jq` on the VM to pipe the above output and format the resulting JSON into a more human readable format:

```
ubuntu@lab8sys:~/lab8$ sudo apt install jq -y

...

Setting up jq (1.6-1ubuntu0.20.04.1) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...

ubuntu@lab8sys:~/lab8$
```

Now, try the curl again, except pipe the output into JQ:

```
ubuntu@lab8sys:~/lab8$ curl -s http://localhost:32767/api/plugins.json | jq .

{
  "plugins": [
    {
      "plugin_id": "object:2ddc",
      "plugin_category": "input",
      "type": "prometheus",
      "config": {
        "@type": "prometheus"
      },
      "output_plugin": false,
      "retry_count": null,
      "emit_records": 0,
      "emit_size": 0
    },
    ...
  ]
}
```

ubuntu@lab8sys:~/lab8\$

Much better. In this view, you should be able to see the following:

- Each plugin is separated into its own JSON map, and is identified by a `plugin_id`. This is currently using an object ID, which can be changed by providing an `@id` parameter to each directive in a configuration file.
- Each plugin entry provides the category and names the type of output, as well as the configuration. The configurations are included in each entry because the `include_config` parameter was set to true.
- Lastly, metrics are provided as key-value pairs in JSON. The available metrics being reported have some overlap with what's being reported by the `prometheus_output_monitor` plugin and cannot be changed. Overlapping monitors include the

`buffer_queue_length` and `buffer_total_queued_size` , to name a few.

At this point, you have successfully configured a Fluentd instance that's reporting metrics to both Prometheus and the REST API.

Cleanup

To prepare for any subsequent labs, please shut down any existing Fluentd instances and containers using the following commands:

Tear down any running Docker containers:

```
ubuntu@lab8$ sudo docker container rm $(sudo docker stop prometheus)
prometheus
ubuntu@lab8$
```

Send a `pkill -f` to terminate any locally running instances of Fluentd, or use `CTRL C` in any terminals running Fluentd:

```
ubuntu@lab8$ pkill -f fluentd
ubuntu@lab8$ cd
ubuntu@lab8$
```

Congratulations, you have completed the lab!

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

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

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

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

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

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

ubuntu@lab9:~/lab9$

```

After the correction, try to run Fluentd. Use the `-o` flag to redirect Fluentd's standard output to a file `/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@lab9:~/lab9$ fluentd -c error1.conf -o /tmp/lab9/error1.out

^C

ubuntu@lab9:~/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@lab9:~/lab9$

```

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

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

```

ubuntu@lab9:~$ 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@lab9sys:~$
```

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

```
ubuntu@lab9sys:~/lab9$ fluentd -c error2.conf -o /tmp/lab9/error2.out

ubuntu@lab9sys:~/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@lab9sys:~/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@lab9sys:~/lab9$ fluentd -c error2.conf -v -o /tmp/lab9/error2.out

ubuntu@lab9sys:~/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@lab9sys:~/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@lab9sys:~/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@lab9sys:~/lab9$

```

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

```

ubuntu@lab9sys:~/lab9$ fluentd -c error2.conf -v -o /tmp/lab9/error2.out

^C

ubuntu@lab9sys:~/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 e 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@lab9:~/lab9$
```

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

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

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

```
ubuntu@lab9:~/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@lab9sys:~$ cd ~/lab9

ubuntu@lab9sys:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000

ubuntu@lab9sys:~/lab9$ pkill -sigusr1 fluentd

ubuntu@lab9sys:~$

```

And check the file:

```

ubuntu@lab9sys:~/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@lab9sys:~/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 forcibly
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@lab9sys:~/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@lab9sys:~/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@lab9:~/lab9$
```

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

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

ubuntu@lab9:~/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@lab9sys:~/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@lab9sys:~/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@lab9sys:~/lab9$
```

Now rerun Fluentd again:

```
ubuntu@lab9sys:~/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@lab9sys:~/lab9$
```

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

```
ubuntu@lab9sys:~/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@lab9sys:~/lab9$
```

With those changes in place rerun Fluentd again:

```
ubuntu@lab9sys:~/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":"dequeuing a chunk
instance=1880"}
```

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

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

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

Rerun Fluentd after making those changes:

```
ubuntu@lab9:~/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@lab9:~/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@lab9:~/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@lab9:~/lab9$ echo '{"lfs242":"event"}' | fluent-cat mod9.lab -p 24000  
  
ubuntu@lab9:~/lab9$ pkill -sigusr1 fluentd  
  
ubuntu@lab9:~/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@lab9:~/lab9$ tail -5 /tmp/lab9/wordpress-log.202206202129_0.log  
  
2022-06-20T21:29:23+00:00      mod9.lab      {"lfs242":"event"}  
  
ubuntu@lab9:~/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@lab9:~/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@lab9sys:~/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@lab9:~/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@lab9:~/lab9$ fluentd -c tuneme.conf -o /tmp/lab9/tuningconf.out &
```

```
[1] 54245
```

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

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

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

```
ubuntu@lab9:~/lab9$ # press enter
```

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

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

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

```
ubuntu@lab9sys:~/lab9$ RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR=0.1 \  
fluentd -c tuneme.conf -o /tmp/lab9/tuningconf.out &
```

```
[1] 54268
```

```
ubuntu@lab9sys:~/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@lab9sys:~/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@lab9sys:~/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@lab9sys:~/lab9$ pkill -f fluentd
```

```
ubuntu@lab9sys:~/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@lab9sys:~/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@lab9$ fluentd -c multiworker.conf -o /tmp/lab9/multiworker.out -vv &

[1] 54312

ubuntu@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@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@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@lab9$
```

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

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

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

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

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

ubuntu@lab9:~/lab9$
```

Edit the `multiworker.conf`:

```
ubuntu@lab9:~/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@lab9:~/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 5 starting 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 `kill 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@lab9:~/lab9$ kill fluentd

ubuntu@lab9:~/lab9$ # press enter

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

ubuntu@lab9:~/lab9$ cd

ubuntu@lab9:~$
```

Congratulations, you have completed the Lab.

LFS242 - Cloud Native Logging with Fluentd

Lab 10 – Fluent Bit and Fluentd

Fluent Bit is a lightweight alternative to Fluentd from the original developers of Fluentd, Arm Treasure Data. It shares many functions with Fluentd, but focuses on high performance with a very small resource footprint - ideal for resource limited environments like embedded systems, IoT devices, and containers. As a part of the Fluentd project ecosystem, it can act as a lighter-weight forwarder to its larger cousin Fluentd, allowing both programs to take advantage of the speed and leanness of Fluent Bit and the flexibility provided by Fluentd.

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

- Understand the similarities and differences between Fluent Bit and Fluentd
- Configure Fluent Bit to work together with Fluentd
- See an example of a Fluent Bit and Fluentd working as part of a fully functional log collection and analysis stack

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

ubuntu@labsys:~$ cat fluentd-setup

#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

ubuntu@labsys:~$ fluentd --version

fluentd 1.14.6

ubuntu@labsys:~$
```

This lab will also be using Docker to run instances of Fluentd, Elasticsearch, and Kibana, 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. Installing Fluent Bit

Fluent Bit is coded entirely in C and this lack of external dependencies on Ruby allows Fluent Bit to have a smaller footprint when compared to Fluentd. Fluent Bit is available as a package through many package managers, including Ubuntu's **apt**. The installation is very simple and can be done with a convenient installation script which installs the latest version:

```
ubuntu@ip-172-31-10-56:~$ mkdir ~/fluent-bit && cd $_

ubuntu@ip-172-31-10-56:~/fluent-bit$ curl https://raw.githubusercontent.com/fluent/fluent-bit/master/install.sh | sh

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 3105  100 3105    0     0  32343      0 --:--:-- --:--:-- --:--:-- 32343
=====
Fluent Bit Installation Script
=====
This script requires superuser access to install packages.
You will be prompted for your password by sudo.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 3175  100 3175    0     0  28603      0 --:--:-- --:--:-- --:--:-- 28603
deb [signed-by=/usr/share/keyrings/fluentbit-keyring.gpg] https://packages.fluentbit.io/ubuntu/focal
focal main
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:5 https://download.docker.com/linux/ubuntu focal InRelease
Hit:6 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
Get:7 https://packages.fluentbit.io/ubuntu/focal focal InRelease [10.9 kB]
Get:8 https://packages.fluentbit.io/ubuntu/focal focal/main amd64 Packages [14.2 kB]
Fetched 361 kB in 1s (456 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpq5
The following NEW packages will be installed:
  fluent-bit libpq5
0 upgraded, 2 newly installed, 0 to remove and 62 not upgraded.
Need to get 20.2 MB of archives.
After this operation, 59.2 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 libpq5 amd64 12.11-0ubuntu0.20.04.1 [117 kB]
Get:2 https://packages.fluentbit.io/ubuntu/focal focal/main amd64 fluent-bit amd64 1.9.4 [20.1 MB]
Fetched 20.2 MB in 1s (19.2 MB/s)
Selecting previously unselected package libpq5:amd64.
(Reading database ... 90784 files and directories currently installed.)
Preparing to unpack .../libpq5_12.11-0ubuntu0.20.04.1_amd64.deb ...
Unpacking libpq5:amd64 (12.11-0ubuntu0.20.04.1) ...
Selecting previously unselected package fluent-bit.
Preparing to unpack .../fluent-bit_1.9.4_amd64.deb ...
Unpacking fluent-bit (1.9.4) ...
Setting up libpq5:amd64 (12.11-0ubuntu0.20.04.1) ...
Setting up fluent-bit (1.9.4) ...
Processing triggers for libc-bin (2.31-0ubuntu9.7) ...

Installation completed. Happy Logging!
```

Fluent Bit should now be installed.

Verify it by running `fluent-bit --version`:

```
ubuntu@labsys:~/fluent-bit$ sudo ln -s /opt/fluent-bit/bin/fluent-bit /usr/bin/fluent-bit

ubuntu@labsys:~/fluent-bit$ fluent-bit --version

Fluent Bit v1.9.4
Git commit:

ubuntu@labsys:~/fluent-bit$
```

2. Running and Configuring Fluent Bit

Fluent Bit shares functional concepts with Fluentd and can interoperate with Fluentd, but unlike Fluentd it is written completely in C, so there are some key differences that need to be explored before using Fluent Bit.

2a. Running Fluent Bit imperatively

Fluent Bit can be run with a certain configuration imperatively from the command line. This is recommended for testing configurations before committing them to more permanent configuration files. One advantage of running Fluent Bit this way is the that it eschews the need for storage space on the Fluent Bit host since no configuration file is required.

Return to your home directory, and dump the `--help` output for Fluent Bit:

```
ubuntu@labsys:~/fluent-bit/build$ mkdir ~/lab10 && cd $_

ubuntu@labsys:~/lab10$ fluent-bit --help

Usage: fluent-bit [OPTION]

Available Options
  -b --storage_path=PATH  specify a storage buffering path
  -c --config=FILE        specify an optional configuration file
  -d, --daemon            run Fluent Bit in background mode
  -D, --dry-run           dry run
  -f, --flush=SECONDS     flush timeout in seconds (default: 1)
  -C, --custom=CUSTOM     enable a custom plugin
  -i, --input=INPUT       set an input
  -F --filter=FILTER      set a filter
  -m, --match=MATCH       set plugin match, same as '-p match=abc'
  -o, --output=OUTPUT     set an output
  -p, --prop="A=B"        set plugin configuration property
  -R, --parser=FILE       specify a parser configuration file
  -e, --plugin=FILE       load an external plugin (shared lib)
  -l, --log_file=FILE     write log info to a file
  -t, --tag=TAG           set plugin tag, same as '-p tag=abc'
  -T, --sp-task=SQL       define a stream processor task
  -v, --verbose           increase logging verbosity (default: info)
  -vv                    trace mode (available)
  -w, --workdir           set the working directory
  -H, --http              enable monitoring HTTP server
  -P, --port              set HTTP server TCP port (default: 2020)
  -s, --coro_stack_size  set coroutines stack size in bytes (default: 24576)
  -q, --quiet             quiet mode
  -S, --sosreport         support report for Enterprise customers
  -V, --version           show version number
```

```

-h, --help          print this help

Inputs
  cpu                CPU Usage
  mem                Memory Usage
  thermal            Thermal
...

ubuntu@labsys:~/lab10$

```

The selection of plugins available represent plugins that are built into the Fluent Bit binary when it was installed. Plugins can be included or excluded at build time, meaning a Fluent Bit instance can be built from the ground up to be as lightweight as possible.

The `-i`, `-o`, and `-F` flags are of note here. These flags allow a user to specify plugins that a Fluent Bit instance can be launched with.

Run a Fluent Bit instance that will send CPU usage metrics to STDOUT:

```

ubuntu@labsys:~/lab10$ fluent-bit -i cpu -o stdout

Fluent Bit v1.9.4
* Copyright (C) 2015-2022 The Fluent Bit Authors
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2022/06/14 22:37:07] [ info] [fluent bit] version=1.9.4, commit=, pid=205042
[2022/06/14 22:37:07] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
[2022/06/14 22:37:07] [ info] [cmetrics] version=0.3.1
[2022/06/14 22:37:07] [ info] [sp] stream processor started
[2022/06/14 22:37:07] [ info] [output:stdout:stdout.0] worker #0 started
[0] cpu.0: [1655246227.851492750, {"cpu_p"=>0.000000, "user_p"=>0.000000, "system_p"=>0.000000,
"cpu0.p_cpu"=>0.000000, "cpu0.p_user"=>0.000000, "cpu0.p_system"=>0.000000, "cpu1.p_cpu"=>0.000000,
"cpu1.p_user"=>0.000000, "cpu1.p_system"=>0.000000}]
[0] cpu.0: [1655246228.851460098, {"cpu_p"=>0.000000, "user_p"=>0.000000, "system_p"=>0.000000,
"cpu0.p_cpu"=>0.000000, "cpu0.p_user"=>0.000000, "cpu0.p_system"=>0.000000, "cpu1.p_cpu"=>1.000000,
"cpu1.p_user"=>1.000000, "cpu1.p_system"=>0.000000}]

...

```

The emitted events share the same basic structure as a Fluentd instance's events:

- The event receives a tag, `cpu.0`
- A timestamp, in unix time, is attached to the event
- A record section enclosed in `{}` that contains the actual event data. In this case, it is output to STDOUT in msgpack.

Terminate the instance with `Ctrl C`.

```

...

^C

[2022/06/14 22:37:14] [engine] caught signal (SIGINT)
[2022/06/14 22:37:14] [ info] [input] pausing cpu.0
[2022/06/14 22:37:14] [ warn] [engine] service will shutdown in max 5 seconds
[0] cpu.0: [1655246233.851476002, {"cpu_p"=>0.000000, "user_p"=>0.000000, "system_p"=>0.000000,
"cpu0.p_cpu"=>0.000000, "cpu0.p_user"=>0.000000, "cpu0.p_system"=>0.000000, "cpu1.p_cpu"=>0.000000,
"cpu1.p_user"=>0.000000, "cpu1.p_system"=>0.000000}]

```

```
[2022/06/14 22:37:14] [ info] [engine] service has stopped (0 pending tasks)
[2022/06/14 22:37:14] [ info] [output:stdout:stdout.0] thread worker #0 stopping...
[2022/06/14 22:37:14] [ info] [output:stdout:stdout.0] thread worker #0 stopped
```

```
ubuntu@labsys:~/lab10$
```

Imperatively run configurations can configure plugins using the `-p` flag following the `-i`, `-o` or `-F` flags. The argument for the `-p` flag is the `key=value` for a plugin's setting.

Run Fluent Bit again with the forward plugin listening on port 31955 and outputting to STDOUT:

```
ubuntu@labsys:~/lab10$ fluent-bit -i forward -p port=31955 -o stdout -p format=json_lines
```

```
Fluent Bit v1.9.4
```

```
* Copyright (C) 2015-2022 The Fluent Bit Authors
```

```
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
```

```
* https://fluentbit.io
```

```
[2022/06/14 22:37:58] [ info] [fluent bit] version=1.9.4, commit=, pid=205047
```

```
[2022/06/14 22:37:58] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
```

```
[2022/06/14 22:37:58] [ info] [cmetrics] version=0.3.1
```

```
[2022/06/14 22:37:58] [ info] [input:forward:forward.0] listening on 0.0.0.0:31955
```

```
[2022/06/14 22:37:58] [ info] [sp] stream processor started
```

```
[2022/06/14 22:37:58] [ info] [output:stdout:stdout.0] worker #0 started
```

- `-i forward` specifies that the forward plugin should be used to listen for Fluent protocol traffic
- `-p port=31955` immediately follows `-i forward` flag, telling the forward plugin to listen on port 31955
- `-o stdout` will submit events to STDOUT
- `-p format=json_lines` configures the stdout plugin to output events in JSON format rather than msgpack

In another terminal, send an event to Fluent Bit using `fluent-cat` :

```
ubuntu@labsys:~$ cd ~/lab10/
```

```
ubuntu@labsys:~/lab10$ echo '{"lfs242":"Hello from Fluent Bit"}' | fluent-cat mod10.lab -p 31955
```

```
ubuntu@labsys:~/lab10$
```

In the Fluent Bit terminal, the message is received:

```
{"date":1655246294.582881,"lfs242":"Hello from Fluent Bit"}
```

With the `json_lines` configuration for the STDOUT plugin, Fluent Bit outputs a JSON map containing the date timestamp and the message.

When you're done, use `CTRL C` in the Fluent Bit terminal to shut down the instance:

```
^C
```

```
[2022/06/14 22:38:57] [engine] caught signal (SIGINT)
```

```
[2022/06/14 22:38:57] [ info] [input] pausing forward.0
```

```
[2022/06/14 22:38:57] [ warn] [engine] service will shutdown in max 5 seconds
```

```
[2022/06/14 22:38:57] [ info] [engine] service has stopped (0 pending tasks)
```

```
[2022/06/14 22:38:57] [ info] [output:stdout:stdout.0] thread worker #0 stopping...
```

```
[2022/06/14 22:38:57] [ info] [output:stdout:stdout.0] thread worker #0 stopped
```

```
ubuntu@labsys:~/lab10$
```

Try running the following imperative Fluent Bit configurations:

- Send memory usage information to a file, `mod10.mem.txt`
- Tail a file, `log.txt`, and send output as `msgpack` to `stdout`

You should now know how to run Fluent Bit imperatively from the command line with a variety of configurations.

2b. Creating a Fluent Bit configuration file

Fluent Bit's configuration format differs greatly from Fluentd, though the overall principal is the same: inputs, filters and outputs are configured under their own sections. Each section declares a single plugin that dictates how an event is received, processed and delivered to a configured destination. Plugins are configured with a set of parameters, known as entries, that make up a majority of a section's syntax. Each entry is a key-value pair that influences how a plugin ultimately behaves.

Create a `fluent-bit.conf` file:

```
ubuntu@labsys:~/lab10$ nano fluent-bit.conf && cat $_

[INPUT]
name cpu

[OUTPUT]
name file
path ~/lab10/output.txt

ubuntu@labsys:~/lab10$
```

- `[INPUT]` is the equivalent of a Fluentd `<source>` directive and is the section that declares an event source
- `name cpu` is the plugin declaration, equivalent to the `@type` parameter in Fluentd
- `[OUTPUT]` is the functional equivalent to the `<match>` directive and declares an event destination. It will write to a file using the `out_file` plugin.
- `path ~/lab10/output.txt` entry tells the `out_file` plugin to write to `output.txt` in the working directory

There are no subsections that need to be configured. To further configure a plugin, only additional entries need to be added.

Try to run it:

```
ubuntu@labsys:~/lab10$ fluent-bit -c fluent-bit.conf

Fluent Bit v1.9.4
* Copyright (C) 2015-2022 The Fluent Bit Authors
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2022/06/14 22:42:12] [error] [config] indentation level is too low
[2022/06/14 22:42:12] [error] [config] error in fluent-bit.conf:6: invalid indentation level
[2022/06/14 22:42:12] [error] [config] section 'input' is missing the 'name' property
[2022/06/14 22:42:12] [error] configuration file contains errors, aborting.

ubuntu@labsys:~/lab10$
```

Fluent Bit enforces a strict indentation schema in its configuration files. The configuration file assembled earlier did not indent the plugin configuration entries. By default, Fluent Bit expects plugin entries to be indented with 4 spaces.

Revise the `fluent-bit.conf` file, adding four spaces to each entry under both `[INPUT]` and `[OUTPUT]` sections:

```
ubuntu@labsys:~/lab10$ nano fluent-bit.conf && cat $_

[INPUT]
    name cpu

[OUTPUT]
    name file
    path ~/lab10/cpu-output

ubuntu@labsys:~/lab10$
```

Give this one a try:

```
ubuntu@labsys:~/lab10$ fluent-bit -c fluent-bit.conf

[INPUT]
    name cpu

[OUTPUT]
    name file
    path ~/lab10/output.txt
ubuntu@labsys:~/lab10$ fluent-bit -c fluent-bit.conf
Fluent Bit v1.9.4
* Copyright (C) 2015-2022 The Fluent Bit Authors
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2022/06/14 22:42:51] [ info] [fluent bit] version=1.9.4, commit=, pid=205081
[2022/06/14 22:42:51] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
[2022/06/14 22:42:51] [ info] [cmetrics] version=0.3.1
[2022/06/14 22:42:51] [ info] [sp] stream processor started
[2022/06/14 22:42:51] [ info] [output:file:file.0] worker #0 started
[2022/06/14 22:42:52] [error] [plugins/out_file/file.c:475 errno=2] No such file or directory
[2022/06/14 22:42:52] [error] [output:file:file.0] error opening: ~/lab10/output.txt/cpu.0

^C

[2022/06/14 22:42:53] [engine] caught signal (SIGINT)
[2022/06/14 22:42:53] [ info] [input] pausing cpu.0
[2022/06/14 22:42:53] [ warn] [engine] service will shutdown in max 5 seconds
[2022/06/14 22:42:53] [error] [plugins/out_file/file.c:475 errno=2] No such file or directory
[2022/06/14 22:42:53] [error] [output:file:file.0] error opening: ~/lab10/output.txt/cpu.0
[2022/06/14 22:42:53] [ info] [engine] service has stopped (0 pending tasks)
[2022/06/14 22:42:53] [ info] [output:file:file.0] thread worker #0 stopping...
[2022/06/14 22:42:53] [ info] [output:file:file.0] thread worker #0 stopped

ubuntu@labsys:~/lab10$
```

It failed, with the `out_file` plugin unable to create the destination. This is because, like `Fluentd`, `Fluent Bit` configurations do not support UNIX environment variables (like `$HOME`) or their aliases (Like `~`) being directly in the configuration file. In order to refer to the home directory, a full path must be used.

Change the home directory reference to a full path:

```
ubuntu@labsys:~/lab10$ nano fluent-bit.conf && cat $_
```



```
[INPUT]
  name cpu

[OUTPUT]
  name file
  path /home/ubuntu/lab10/cpu-output

ubuntu@labsys:~/lab10$
```

Try to run it now:

```
ubuntu@labsys:~/lab10$ mkdir ~/lab10/cpu-output

ubuntu@labsys:~/lab10$ fluent-bit -c fluent-bit.conf

...

[2022/06/14 22:43:41] [ info] [fluent bit] version=1.9.4, commit=, pid=205095
[2022/06/14 22:43:41] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
[2022/06/14 22:43:41] [ info] [cmetrics] version=0.3.1
[2022/06/14 22:43:41] [ info] [sp] stream processor started
[2022/06/14 22:43:41] [ info] [output:file:file.0] worker #0 started
```

Now it's running, and no errors are being reported.

In another terminal session, check the contents of the `~/lab10/cpu-output/` directory:

```
ubuntu@labsys:~/lab10$ ls -l ~/lab10/cpu-output/

total 4
-rw-rw-r-- 1 ubuntu ubuntu 4026 Jun 14 22:44 cpu.0

ubuntu@labsys:~/lab10$
```

output.txt has been created. Check the contents:

```
ubuntu@labsys:~/lab10$ tail -5 cpu-output/cpu.0

cpu.0: [1655246661.851526735,
{"cpu_p":0.0,"user_p":0.0,"system_p":0.0,"cpu0.p_cpu":0.0,"cpu0.p_user":0.0,"cpu0.p_system":0.0,"cpu
1.p_cpu":0.0,"cpu1.p_user":0.0,"cpu1.p_system":0.0}]
cpu.0: [1655246662.851512133,
{"cpu_p":0.0,"user_p":0.0,"system_p":0.0,"cpu0.p_cpu":0.0,"cpu0.p_user":0.0,"cpu0.p_system":0.0,"cpu
1.p_cpu":0.0,"cpu1.p_user":0.0,"cpu1.p_system":0.0}]
cpu.0: [1655246663.851535888,
{"cpu_p":0.0,"user_p":0.0,"system_p":0.0,"cpu0.p_cpu":0.0,"cpu0.p_user":0.0,"cpu0.p_system":0.0,"cpu
1.p_cpu":0.0,"cpu1.p_user":0.0,"cpu1.p_system":0.0}]
cpu.0: [1655246664.851534962,
{"cpu_p":0.0,"user_p":0.0,"system_p":0.0,"cpu0.p_cpu":0.0,"cpu0.p_user":0.0,"cpu0.p_system":0.0,"cpu
1.p_cpu":0.0,"cpu1.p_user":0.0,"cpu1.p_system":0.0}]
cpu.0: [1655246665.851524646,
{"cpu_p":0.0,"user_p":0.0,"system_p":0.0,"cpu0.p_cpu":0.0,"cpu0.p_user":0.0,"cpu0.p_system":0.0,"cpu
1.p_cpu":0.0,"cpu1.p_user":0.0,"cpu1.p_system":0.0}]

ubuntu@labsys:~/lab10$
```

Excellent! You should now know how to create a basic Fluent Bit configuration as well as imperatively run an instance of Fluent Bit. Remember that:

- The sections are named differently from Fluentd directives
- There are strict spacing and indentation requirements
- Unix Environment Variables (those prefixed with \$) cannot be written directly into configuration files

Terminate the Fluent Bit instance with `CTRL C` or `kill -f fluent-bit`:

```
^C

[2022/06/14 22:44:42] [engine] caught signal (SIGINT)
[2022/06/14 22:44:42] [ info] [input] pausing cpu.0
[2022/06/14 22:44:42] [ warn] [engine] service will shutdown in max 5 seconds
[2022/06/14 22:44:42] [ info] [engine] service has stopped (0 pending tasks)
[2022/06/14 22:44:42] [ info] [output:file:file.0] thread worker #0 stopping...
[2022/06/14 22:44:42] [ info] [output:file:file.0] thread worker #0 stopped

ubuntu@labsys:~/lab10$
```

3. Fluentd and Fluent Bit comparison

Fluent Bit aims to be much lighter weight than Fluentd, in both configuration complexity (as shown above) and resource footprint (which will be shown in the coming steps).

3a. Setting up Fluentd and Fluent Bit to work together:

Fluent Bit can work together with Fluentd in a multi-instance deployment, taking the role of an log forwarder. In this step, you will set up a multi-instance deployment with a Fluent Bit forwarder, Fluentd log forwarder and Fluentd log aggregator.

Create a simple Fluentd log aggregator configuration file:

```
ubuntu@labsys:~/lab10$ nano fd-aggregator.conf && cat $_

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

<match>
  @type stdout
</match>

ubuntu@labsys:~/lab10$
```

Run the log aggregator in a Docker container, mounting the `fd-aggregator.conf` above and running it with the `-o` flag to write the log aggregator's output to an easily accessible file:

```
ubuntu@labsys:~/lab10$ sudo docker run -d --network host --name aggregator1 \
-v $HOME/lab10/fd-aggregator.conf:/fluentd/etc/fd-aggregator.conf \
-e FLUENTD_CONF=fd-aggregator.conf \
fluent/fluentd:v1.14-1 fluentd -c /fluentd/etc/fd-aggregator.conf

...

40a279037688e5551328eb15088c98d8cebc1f219ed90eca2e1406cdee83b6a9
```

```

ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 5

2022-06-14 23:06:55 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-14 23:06:55 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-14 23:06:55.191329748 +0000 fluent.info: {"pid":16,"ppid":7,"worker":0,"message":"starting
fluentd worker pid=16 ppid=7 worker=0"}
2022-06-14 23:06:55.191725212 +0000 fluent.info: {"port":24500,"bind":"0.0.0.0","message":"listening
port port=24500 bind=\"0.0.0.0\""}
2022-06-14 23:06:55.192198669 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}

ubuntu@labsys:~/lab10$

```

There is now a Fluentd log aggregator instance waiting to receive events from log forwarders.

Next, prepare a Fluentd log forwarder configuration:

```

ubuntu@labsys:~/lab10$ touch app.log

ubuntu@labsys:~/lab10$ nano fd-forwarder.conf && cat $_

<source>
  @type tail
  path /home/ubuntu/lab10/app.log
  pos_file /tmp/app.log.pos
  tag fd.forward
  <parse>
    @type none
  </parse>
</source>

<match>
  @type forward
  <server>
    name aggregator1
    host 127.0.0.1
    port 24500
  </server>
</match>

ubuntu@labsys:~/lab10$

```

Since the log aggregator container was run with the `--network host`, it can be accessed via the local IP: 127.0.0.1.

Then run the Fluentd log forwarder:

```

ubuntu@labsys:~/lab10$ fluentd -c fd-forwarder.conf

...

2022-06-14 23:07:31 +0000 [info]: #0 starting fluentd worker pid=205262 ppid=205257 worker=0
2022-06-14 23:07:31 +0000 [info]: #0 following tail of /home/ubuntu/lab10/app.log
2022-06-14 23:07:31 +0000 [info]: #0 fluentd worker is now running worker=0

```

When running the log forwarder, make sure that it is able to find the app.log file, which was created before the log forwarder configuration was written.

In a new terminal window, create a log forwarder instance of Fluent Bit with a configuration file that's tailing the same file as the Fluentd log forwarder:

```
ubuntu@labsys:~$ cd lab10

ubuntu@labsys:~/lab10$ nano fb-forwarder.conf && cat $_

[INPUT]
  name tail
  path /home/ubuntu/lab10/app.log

[OUTPUT]
  name forward
  host 127.0.0.1
  port 24500

ubuntu@labsys:~/lab10$
```

Notice how much simpler the Fluent Bit log forwarder file is compared to the Fluentd log forwarder's file:

- There is no `<parse>` -equivalent entry in the Fluent Bit configuration
- The tail input plugin for Fluent Bit does not record its last read position to a file

Run Fluent Bit, using the `-v` flag to get higher verbosity:

```
ubuntu@labsys:~/lab10$ fluent-bit -c fb-forwarder.conf -v

Fluent Bit v1.9.4
* Copyright (C) 2015-2022 The Fluent Bit Authors
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2022/06/14 23:08:02] [ info] Configuration:
[2022/06/14 23:08:02] [ info] flush time      | 1.000000 seconds
[2022/06/14 23:08:02] [ info] grace        | 5 seconds
[2022/06/14 23:08:02] [ info] daemon       | 0
[2022/06/14 23:08:02] [ info] _____
[2022/06/14 23:08:02] [ info] inputs:
[2022/06/14 23:08:02] [ info]   tail
[2022/06/14 23:08:02] [ info] _____
[2022/06/14 23:08:02] [ info] filters:
[2022/06/14 23:08:02] [ info] _____
[2022/06/14 23:08:02] [ info] outputs:
[2022/06/14 23:08:02] [ info]   forward.0
[2022/06/14 23:08:02] [ info] _____
[2022/06/14 23:08:02] [ info] collectors:
[2022/06/14 23:08:02] [ info] [fluent bit] version=1.9.4, commit=, pid=205270
[2022/06/14 23:08:02] [debug] [engine] coroutine stack size: 24576 bytes (24.0K)
[2022/06/14 23:08:02] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
[2022/06/14 23:08:02] [ info] [cmetrics] version=0.3.1
[2022/06/14 23:08:02] [debug] [tail:tail.0] created event channels: read=21 write=22
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] flb_tail_fs_inotify_init() initializing inotify
tail input
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] inotify watch fd=27
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] scanning path /home/ubuntu/lab10/app.log
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] inode=1039240 with offset=0 appended as
/home/ubuntu/lab10/app.log
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] scan_glob add(): /home/ubuntu/lab10/app.log, inode
1039240
```

```
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] 1 new files found on path
'/home/ubuntu/lab10/app.log'
[2022/06/14 23:08:02] [debug] [forward:forward.0] created event channels: read=29 write=30
[2022/06/14 23:08:02] [ info] [output:forward:forward.0] worker #0 started
[2022/06/14 23:08:02] [debug] [router] default match rule tail.0:forward.0
[2022/06/14 23:08:02] [ info] [output:forward:forward.0] worker #1 started
[2022/06/14 23:08:02] [ info] [sp] stream processor started
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] inode=1039240 file=/home/ubuntu/lab10/app.log
promote to TAIL_EVENT
[2022/06/14 23:08:02] [ info] [input:tail:tail.0] inotify_fs_add(): inode=1039240 watch_fd=1
name=/home/ubuntu/lab10/app.log
[2022/06/14 23:08:02] [debug] [input:tail:tail.0] [static files] processed 0b, done
```

In a new or free terminal session, use `ps` to see all running instances of Fluentd and Fluent Bit:

```
ubuntu@labsys:~/lab10$ ps -ef | grep fluent

systemd+ 205222 205197 0 23:06 ?        00:00:00 tini -- /bin/entrypoint.sh fluentd -c
/fluentd/etc/fd-aggregator.conf
systemd+ 205234 205222 0 23:06 ?        00:00:00 /usr/bin/ruby /usr/bin/fluentd -c
/fluentd/etc/fd-aggregator.conf --plugin /fluentd/plugins
systemd+ 205243 205234 0 23:06 ?        00:00:00 /usr/bin/ruby -Eascii-8bit:ascii-8bit
/usr/bin/fluentd -c /fluentd/etc/fd-aggregator.conf --plugin /fluentd/plugins --under-supervisor
ubuntu 205257 204978 1 23:07 pts/0    00:00:01 /usr/bin/ruby2.7 /usr/local/bin/fluentd -c fd-
forwarder.conf
ubuntu 205262 205257 0 23:07 pts/0    00:00:00 /usr/bin/ruby2.7 -Eascii-8bit:ascii-8bit
/usr/local/bin/fluentd -c fd-forwarder.conf --under-supervisor
ubuntu 205270 201448 0 23:08 pts/1    00:00:00 fluent-bit -c fb-forwarder.conf -v
ubuntu 205399 205373 0 23:08 pts/2    00:00:00 grep --color=auto fluent

ubuntu@labsys:~/lab10$
```

Note the following:

- PID `205234` and `205243` are the log aggregator instance supervisor and worker processes
- The Fluentd forwarder processes are running on PIDs `205257` and `205262`
- Fluent-Bit is running on pid `205270`

Now use `top` to compare their resource footprints. Use `pgrep` to retrieve the pids for all processes run with `fluent` and `ruby`:

```
ubuntu@labsys:~/lab10$ top -p `pgrep -d "," fluent\|ruby`

top - 23:09:28 up 6:31, 3 users, load average: 0.04, 0.04, 0.01
Tasks: 5 total, 0 running, 5 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.2 sy, 0.0 ni, 99.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.3 st
MiB Mem : 3865.9 total, 2011.0 free, 417.8 used, 1437.0 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 3187.0 avail Mem

   PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 205243 systemd+  20   0 108548  39676  6596 S   0.3   1.0   0:00.77 ruby
 205270 ubuntu    20   0  87008  12860 10760 S   0.3   0.3   0:00.02 fluent-bit
 205234 systemd+  20   0 105712  38820  6620 S   0.0   1.0   0:00.80 fluentd
 205257 ubuntu    20   0 305320  47064  9624 S   0.0   1.2   0:01.32 fluentd
 205262 ubuntu    20   0 440412  49564  9260 S   0.0   1.3   0:00.78 ruby2.7

q

ubuntu@labsys:~/lab10$
```

Compared to the Fluentd log forwarder, Fluent Bit is consuming far fewer resources than either of its Fluentd cousins. It's easy to see that in a resource constrained environment, Fluent Bit would be the way to go.

3b. Forwarding Events to Fluentd with Fluent Bit

Now it's time to compare the resulting events forwarded from the same file between Fluentd and Fluent Bit.

In a free terminal, send an event to the app.log file that both log forwarders are tailing. Be sure to force the Fluentd log forwarder to flush its buffers.

```
ubuntu@labsys:~/lab10$ echo "This is an event" >> ~/lab10/app.log

ubuntu@labsys:~/lab10$ pkill -sigusr1 fluentd

pkill: killing pid 29564 failed: Operation not permitted

ubuntu@labsys:~/lab10$
```

In the Fluent Bit terminal, it should have recorded the event being captured:

```
...

[2022/06/14 23:09:59] [debug] [out flush] cb_destroy coro_id=0
[2022/06/14 23:09:59] [debug] [task] destroy task=0x7fab59a3d460 (task_id=0)
[2022/06/14 23:10:01] [debug] [input:tail:tail.0] scanning path /home/ubuntu/lab10/app.log
[2022/06/14 23:10:01] [debug] [input:tail:tail.0] scan_blog add(): dismissed:
/home/ubuntu/lab10/app.log, inode 1039240
[2022/06/14 23:10:01] [debug] [input:tail:tail.0] 0 new files found on path
'/home/ubuntu/lab10/app.log'
[2022/06/14 23:10:29] [debug] [upstream] drop keepalive connection #-1 to 127.0.0.1:24500 (keepalive
idle timeout)
```

Check the log aggregator output to see if it received events from its log forwarders:

```
ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 5

2022-06-14 23:07:31.002443014 +0000 fluent.info: {"worker":0,"message":"fluentd worker is now
running worker=0"}
2022-06-14 23:09:58.921970203 +0000 tail.0: {"log":"This is an event"}
2022-06-14 23:09:58.922182768 +0000 fd.forward: {"message":"This is an event"}
2022-06-14 23:10:01.937402218 +0000 fluent.info: {"message":"force flushing buffered events"}
2022-06-14 23:10:01.937904760 +0000 fluent.info: {"message":"flushing all buffer forcedly"}

ubuntu@labsys:~/lab10$
```

The log aggregator received an event from both Fluentd (as seen from the `fd.forward` tagged event) and Fluent Bit (which sent the event under `tail.0`). Fluent Bit is now successfully forwarding events to Fluentd!

4. Complete Logging Pipeline with Apache, Fluent Bit, Fluentd, Elasticsearch and Kibana

Fluentd is as part of a full log collection and analysis stack with Elasticsearch, an open source search engine based on Lucene, and Kibana, an open source data visualization frontend for Elasticsearch. This combination is called an EFK stack. In this final step, you will set up a full EFK stack, with Fluent Bit acting as a Fluentd forwarder.

4a. Apache to Fluent Bit and Fluentd

To start setting up this EFK stack, the Apache Webserver will be used to feed Fluent Bit, Fluentd, and Elasticsearch with event data.

Install the Apache webserver:

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

...

ubuntu@labsys:~$
```

Relaunch the Fluent Bit instance from the previous step so it tails the Apache access log and forwards to Fluentd:

```
...
^C

[2022/06/14 23:11:49] [engine] caught signal (SIGINT)
[2022/06/14 23:11:49] [ info] [input] pausing tail.0
[2022/06/14 23:11:49] [ warn] [engine] service will shutdown in max 5 seconds
[2022/06/14 23:11:49] [ info] [engine] service has stopped (0 pending tasks)
[2022/06/14 23:11:49] [debug] [input:tail:tail.0] inode=1039240 removing file name
/home/ubuntu/lab10/app.log
[2022/06/14 23:11:49] [ info] [input:tail:tail.0] inotify_fs_remove(): inode=1039240 watch_fd=1
[2022/06/14 23:11:49] [ info] [output:forward:forward.0] thread worker #0 stopping...
[2022/06/14 23:11:49] [ info] [output:forward:forward.0] thread worker #0 stopped
[2022/06/14 23:11:49] [ info] [output:forward:forward.0] thread worker #1 stopping...
[2022/06/14 23:11:49] [ info] [output:forward:forward.0] thread worker #1 stopped

ubuntu@labsys:~/lab10$
```

```
ubuntu@labsys:~/lab10$ fluent-bit -i tail -p path=/var/log/apache2/access.log -o
forward://127.0.0.1:24500 -v
```

```
* Copyright (C) 2015-2022 The Fluent Bit Authors
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io
```

```
[2022/06/14 23:11:59] [ info] Configuration:
[2022/06/14 23:11:59] [ info]   flush time      | 1.000000 seconds
[2022/06/14 23:11:59] [ info]   grace        | 5 seconds
[2022/06/14 23:11:59] [ info]   daemon       | 0
[2022/06/14 23:11:59] [ info]   _____
[2022/06/14 23:11:59] [ info]   inputs:
[2022/06/14 23:11:59] [ info]     tail
[2022/06/14 23:11:59] [ info]   _____
[2022/06/14 23:11:59] [ info]   filters:
[2022/06/14 23:11:59] [ info]   _____
[2022/06/14 23:11:59] [ info]   outputs:
[2022/06/14 23:11:59] [ info]     forward.0
[2022/06/14 23:11:59] [ info]   _____
[2022/06/14 23:11:59] [ info]   collectors:
[2022/06/14 23:11:59] [ info] [fluent bit] version=1.9.4, commit=, pid=206431
[2022/06/14 23:11:59] [debug] [engine] coroutine stack size: 24576 bytes (24.0K)
[2022/06/14 23:11:59] [ info] [storage] version=1.2.0, type=memory-only, sync=normal,
checksum=disabled, max_chunks_up=128
[2022/06/14 23:11:59] [ info] [cmetrics] version=0.3.1
[2022/06/14 23:11:59] [debug] [tail:tail.0] created event channels: read=21 write=22
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] flb_tail_fs_inotify_init() initializing inotify
```

```

tail input
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] inotify watch fd=27
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] scanning path /var/log/apache2/access.log
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] inode=1802060 with offset=0 appended as
/var/log/apache2/access.log
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] scan_glob add(): /var/log/apache2/access.log,
inode 1802060
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] 1 new files found on path
'/var/log/apache2/access.log'
[2022/06/14 23:11:59] [debug] [forward:forward.0] created event channels: read=29 write=30
[2022/06/14 23:11:59] [debug] [router] default match rule tail.0:forward.0
[2022/06/14 23:11:59] [ info] [sp] stream processor started
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] inode=1802060 file=/var/log/apache2/access.log
promote to TAIL_EVENT
[2022/06/14 23:11:59] [ info] [input:tail:tail.0] inotify_fs_add(): inode=1802060 watch_fd=1
name=/var/log/apache2/access.log
[2022/06/14 23:11:59] [debug] [input:tail:tail.0] [static files] processed 0b, done
[2022/06/14 23:11:59] [ info] [output:forward:forward.0] worker #0 started
[2022/06/14 23:11:59] [ info] [output:forward:forward.0] worker #1 started

```

In a free terminal, modify the Fluentd log aggregator configuration that parses the incoming log message with a filter using the Apache2 parser:

```

ubuntu@labsys:~$ nano ~/lab10/fd-aggregator.conf && cat $_

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

<filter tail.0>
  @type parser
  key_name log
  <parse>
    @type apache2
  </parse>
</filter>

<match tail.0>
  @type stdout
</match>

ubuntu@labsys:~/lab10$

```

As observed from the previous step, Fluent Bit is forwarding events under the `tail.0` tag, so configure both the `<filter>` and `<match>` directives to capture events tagged `tail.0`.

Restart the log aggregator Fluentd instance to reload the configuration and check the log aggregator's output to ensure it restarted correctly:

```

ubuntu@labsys:~/lab10$ sudo docker restart aggregator1

aggregator1

ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 5

2022-06-14 23:13:08 +0000 [info]: adding match pattern="tail.0" type="stdout"
2022-06-14 23:13:08 +0000 [info]: adding source type="forward"

```



```
2022-06-14 23:13:08 +0000 [info]: #0 starting fluentd worker pid=16 ppid=7 worker=0
2022-06-14 23:13:08 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-14 23:13:08 +0000 [info]: #0 fluentd worker is now running worker=0

ubuntu@labsys:~/lab10$
```

The log aggregator is now capturing all events tagged `tail.0` with both a `<filter>` and `<match>` directive. Your Fluent Bit and Fluentd stack is now ready to receive events.

Send a `curl` request to localhost to write an event to Apache:

```
ubuntu@labsys:~$ curl localhost

...

ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 1

2022-06-14 23:13:22.000000000 +0000 tail.0:
{"host":"127.0.0.1","user":null,"method":"GET","path":"/","code":200,"size":11173,"referer":null,"agent":"curl/7.68.0"}

ubuntu@labsys:~/lab10$
```

Nice! Fluent Bit successfully submitted an Apache access event to Fluentd.

So far, the pipeline consists of a Fluent Bit log forwarder tailing the Apache access log, which forwards to a Fluentd log aggregator that performs the parsing. Fluent Bit is submitting those events to Fluentd under the `tail.0` tag.

Now that Fluentd is able to receive events from Fluent Bit, it's time to establish a more meaningful (and common) destination for Fluentd events: Elasticsearch.

Run an Elasticsearch and Kibana container, using `--network host` to ensure those containers can communicate with the Fluentd log aggregator.

N.B. Due to the complexity of setup introduced by Elasticsearch 8, this lab will run Elasticsearch 7. If you want more details on setting up a Fluentd-powered log pipeline with Elasticsearch 8, see lab 4 of this course.

```
ubuntu@labsys:~$ sudo docker run -d --name elasticsearch -p 9200:9200 -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:7.17.4

...

22afa36eaa1235ffcf85f096b5f1869db383a3a7a682fe2d3d38832ef5537c33

ubuntu@labsys:~$ sudo docker run -d --name kibana --net host -e
ELASTICSEARCH_HOSTS=http://localhost:9200 docker.elastic.co/kibana/kibana:7.17.4

...

0c842526eff763d713e6ffc3b1b4455d69dec4f74d19422db8fd06a6129b3201

ubuntu@labsys:~$
```

Now it's time to prepare the Fluentd log aggregator to submit events to Elasticsearch.

Gem installations of Fluentd do not install the Elasticsearch plugin by default, so you will need to install the Fluentd Elasticsearch Plugin in the log aggregator container.

To do this cleanly, create a Dockerfile that will install the Elasticsearch plugin:

```
ubuntu@labsys:~/lab10$ nano Dockerfile && cat $_  
  
FROM fluent/fluentd:v1.14-1  
USER root  
RUN ["fluent-gem", "install", "elasticsearch", "-N", "-v", "7.17.1"]  
RUN ["fluent-gem", "install", "elasticsearch-transport", "-N", "-v", "7.17.1"]  
RUN ["fluent-gem", "install", "fluent-plugin-elasticsearch", "-N", "-v", "5.2.2"]  
USER fluent  
  
ubuntu@labsys:~/lab10$
```

Now build the container, tagging it `lfs242/fluent` with the tag `elasticsearch` :

```
ubuntu@labsys:~/lab10$ sudo docker build -t lfs242/fluentd:elasticsearch .  
  
Sending build context to Docker daemon 19.46kB  
Step 1/6 : FROM fluent/fluentd:v1.14-1  
--> 396489e40ea6  
Step 2/6 : USER root  
--> Using cache  
--> a03e2f8b9ada  
Step 3/6 : RUN ["fluent-gem", "install", "elasticsearch", "-N", "-v", "7.17.1"]  
--> Running in 8d2963b49921  
Successfully installed multi_json-1.15.0  
Successfully installed faraday-em_http-1.0.0  
Successfully installed faraday-em_synchrony-1.0.0  
Successfully installed faraday-excon-1.1.0  
Successfully installed faraday-httpclient-1.0.1  
Successfully installed multipart-post-2.2.3  
Successfully installed faraday-multipart-1.0.4  
Successfully installed faraday-net_http-1.0.1  
Successfully installed faraday-net_http_persistent-1.2.0  
Successfully installed faraday-patron-1.0.0  
Successfully installed faraday-rack-1.0.0  
Successfully installed faraday-retry-1.0.3  
Successfully installed ruby2_keywords-0.0.5  
Successfully installed faraday-1.10.0  
Successfully installed elasticsearch-transport-7.17.1  
Successfully installed elasticsearch-api-7.17.1  
Successfully installed elasticsearch-7.17.1  
17 gems installed  
Removing intermediate container 8d2963b49921  
--> f8b9aae39822  
Step 4/6 : RUN ["fluent-gem", "install", "elasticsearch-transport", "-N", "-v", "7.17.1"]  
--> Running in e62b7e0ca648  
Successfully installed elasticsearch-transport-7.17.1  
1 gem installed  
Removing intermediate container e62b7e0ca648  
--> 3a6b3c45169c  
Step 5/6 : RUN ["fluent-gem", "install", "fluent-plugin-elasticsearch", "-N", "-v", "5.2.2"]  
--> Running in af1d2b2845ba  
Successfully installed excon-0.92.3  
Successfully installed fluent-plugin-elasticsearch-5.2.2  
2 gems installed  
Removing intermediate container af1d2b2845ba  
--> f9341a8fd0da  
Step 6/6 : USER fluent
```

```
---> Running in d31767fc6803
Removing intermediate container d31767fc6803
---> 7e51a29a099f
Successfully built 7e51a29a099f
Successfully tagged lfs242/fluentd:elasticsearch

ubuntu@labsys:~/lab10$
```

Now configure the log aggregator to output events to Elasticsearch:

```
ubuntu@labsys:~$ nano ~/lab10/fd-aggregator.conf && cat $_

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

<filter tail.0>
  @type parser
  key_name log
  <parse>
    @type apache2
  </parse>
</filter>

<match tail.0>
  @type elasticsearch
  host localhost
  port 9200
  logstash_format true
  include_timestamp true
</match>

ubuntu@labsys:~$
```

Restart the log aggregator container to reload the configuration:

```
ubuntu@labsys:~/lab10$ sudo docker container rm $(sudo docker container stop aggregator1)

aggregator1

ubuntu@labsys:~/lab10$ sudo docker run -d --network host --name aggregator1 -v $HOME/lab10/fd-
aggregator.conf:/fluentd/etc/fd-aggregator.conf -e FLUENTD_CONF=fd-aggregator.conf
lfs242/fluentd:elasticsearch -c /fluentd/etc/fd-aggregator.conf

4789ca4b0d038d1252c0c6765f10b68987d8ecb68cd32bfe019ae1bc466f5853

ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 5

2022-06-14 23:36:20 +0000 [warn]: #0 Detected ES 7.x: `_doc` will be used as the document `_type`.
2022-06-14 23:36:20 +0000 [info]: adding source type="forward"
2022-06-14 23:36:20 +0000 [info]: #0 starting fluentd worker pid=16 ppid=7 worker=0
2022-06-14 23:36:20 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-14 23:36:20 +0000 [info]: #0 fluentd worker is now running worker=0

ubuntu@labsys:~/lab10$
```

Curl Apache to generate another event:

```
ubuntu@labsys:~$ curl localhost
```

```
...
```

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

Flush the Fluentd Buffer to send the event to Elasticsearch:

```
ubuntu@labsys:~/lab10$ sudo docker kill --signal=SIGUSR1 aggregator1
```

```
aggregator1
```

```
ubuntu@labsys:~/lab10$ sudo docker logs aggregator1 --tail 5
```

```
2022-06-14 23:36:20 +0000 [info]: #0 starting fluentd worker pid=16 ppid=7 worker=0
2022-06-14 23:36:20 +0000 [info]: #0 listening port port=24500 bind="0.0.0.0"
2022-06-14 23:36:20 +0000 [info]: #0 fluentd worker is now running worker=0
2022-06-14 23:36:41 +0000 [info]: #0 force flushing buffered events
2022-06-14 23:36:41 +0000 [info]: #0 flushing all buffer forcedly
```

```
ubuntu@labsys:~/lab10$
```

Now check the Elasticsearch indices to see if any data has been written:

```
ubuntu@labsys:~$ curl 'localhost:9200/_cat/indices?v'
```

health	status	index	uuid	pri	rep	docs.count	docs.deleted
green	open	.geoip_databases	8a3cCTHoTyqFSw2T0ak-2Q	1	0	41	0
38.9mb		38.9mb					
green	open	.apm-custom-link	qlaRKbhdQpuNxtB3euRbcQ	1	0	0	0
226b		226b					
green	open	.apm-agent-configuration	LbGiQD_9TQOasmU7WYMAIw	1	0	0	0
226b		226b					
yellow	open	logstash-2022.06.14	ORiTznsQR3eNmXLgGM_cPA	1	1	1	0
6.3kb		6.3kb					
green	open	.kibana_task_manager_7.17.4_001	KLvhu06mTv-1MC7u5t12Rg	1	0	17	183
156.3kb		156.3kb					
green	open	.kibana_7.17.4_001	nOfw9d6wRJiTYHh0Rr7lMw	1	0	11	0
2.3mb		2.3mb					

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

Fluentd has been configured to submit its events to Elasticsearch like Logstash (another log processing solution also made by Elastic), so it is forwarding events to Elasticsearch to a **logstash-** prefixed index.

Search the **logstash-** index for data. Elasticsearch should return as many results as you have sent to Apache so far:

```
ubuntu@labsys:~$ curl -s localhost:9200/logstash-*/_search?pretty
```

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  }
}
```

```

},
"hits" : {
  "total" : {
    "value" : 1,
    "relation" : "eq"
  },
  "max_score" : 1.0,
  "hits" : [
    {
      "_index" : "logstash-2022.06.14",
      "_type" : "_doc",
      "_id" : "MReTZIEB0ikbxoe_kbLg",
      "_score" : 1.0,
      "_source" : {
        "host" : "127.0.0.1",
        "user" : null,
        "method" : "GET",
        "path" : "/",
        "code" : 200,
        "size" : 11173,
        "referer" : null,
        "agent" : "curl/7.68.0",
        "@timestamp" : "2022-06-14T23:36:38.000000000+00:00"
      }
    }
  ]
}
}
}

ubuntu@labsys:~$

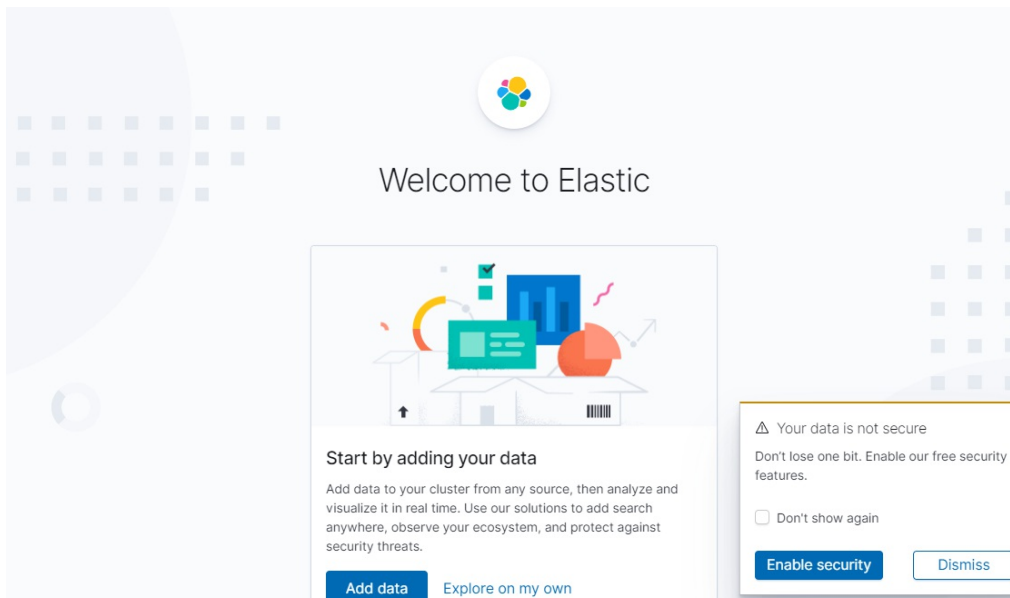
```

An Apache event has been sent into Elasticsearch by Fluent Bit and Fluentd!

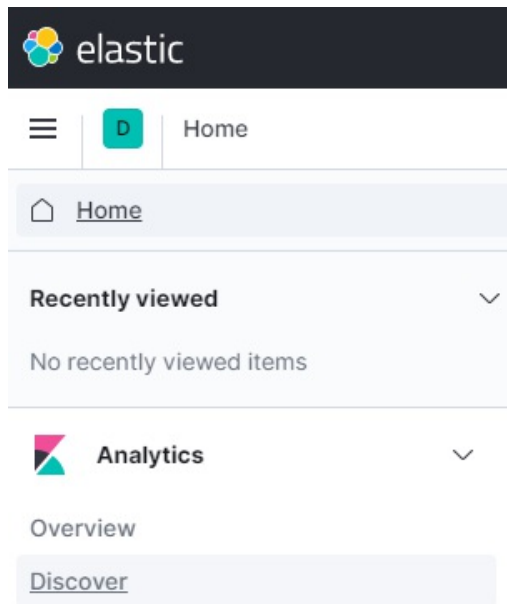
Now that Apache is sending logs to Elasticsearch with Fluentd, you can use Kibana to perform data analysis. One of the easiest ways to do this is by using Kibana's data visualizations to quickly discern patterns over time. In order to use data from events in visualizations, an index pattern needs to be created to tell Kibana which Elasticsearch indices it needs to pull data from.

Using a browser, navigate to your Kibana instance at your VM's IP at port **5601**.

When the welcome screen comes up choose "Explore on my own" and dismiss the security window.



In the Side Menu, click "Discover"



In the "Create index pattern" section, enter `logstash*` in the "Step 1 of 2: Define index pattern" input field.

Create index pattern

Name

Use an asterisk (*) to match multiple characters. Spaces and the characters `/`, `?`, `"`, `<`, `>`, `|` are not allowed.

Timestamp field

Select a timestamp field for use with the global time filter.

[Show advanced settings](#)

✓ Your index pattern matches 1 source.

logstash-2022.06.14

Index

Rows per page: 10

You should see the "Your index pattern matches 1 source." message and the Films index selected.

Choose the "@timestamp" option from the "Timestamp field" dropdown.

Create index pattern

Name

Use an asterisk (*) to match multiple characters. Spaces and the characters `/`, `?`, `"`, `<`, `>`, `|` are not allowed.

Timestamp field

Select a timestamp field for use with the global time filter.

[Show advanced settings](#)

✓ Your index pattern matches 1 source.

logstash-2022.06.14

Index

Rows per page: 10













Now click "Create index pattern".

This should bring you to a dialog that allows you to explore the Logstash (Fluentd) index structure.

Time field: '@timestamp'

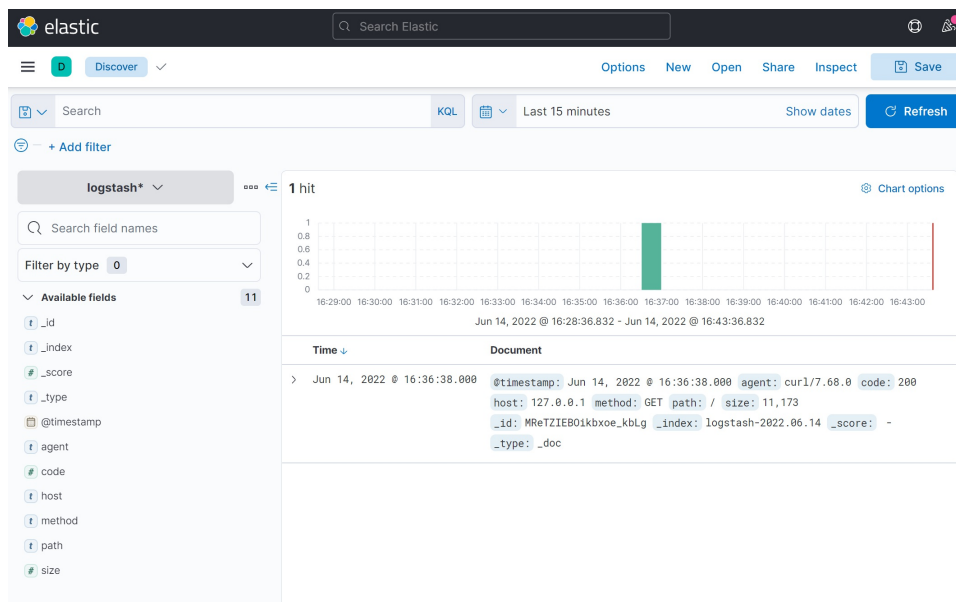
View and edit fields in **logstash***. Field attributes, such as type and searchability, are based on [field mappings](#) in Elasticsearch.

Fields (16) **Scripted fields (0)** **Field filters (0)**

<input type="text" value="Search"/> All field types Add field					
Name ↑	Type	Format	Searchable	Aggregatable	Excluded
@timestamp 	date				
_id	_id				
_index	_index				
_score					
_source	_source				

The main reason why Fluentd was configured to utilize the logstash format when sending events to Elasticsearch is that if it had been configured more simply without the logstash format, then there would be no time filters available to sort data with.

If you go to Discover again, you should now be able to see the logstash index. If not, adjust the time interval. That amount of events depends on how many `curl` requests you've sent to Apache:



If you would like to create your own visualizations, go to **Visualize Library** and select **"Create a visualization"**:

Discover

Discover

Home

Recently viewed

No recently viewed items

Analytics

Overview

Discover

Dashboard

Canvas

Maps

Machine Learning

Visualize Library

Visualize Library

Building a dashboard? Create content directly from the [Dashboard application](#) using a new integrated workflow.



Create your first visualization

You can create different visualizations based on your data.

Create new visualization

Pick "Lens":

New visualization



Lens

Create visualizations with our drag and drop editor. Switch between visualization types at any time. *Recommended for most users.*



Maps

Create and style maps with multiple layers and indices.



TSVB

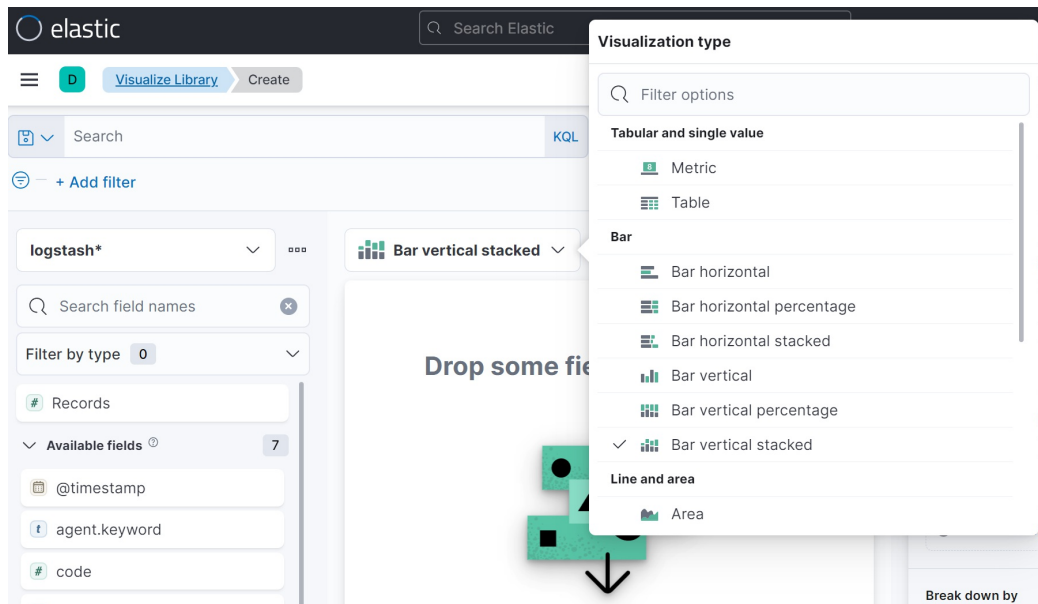
Perform advanced analysis of your time series data.



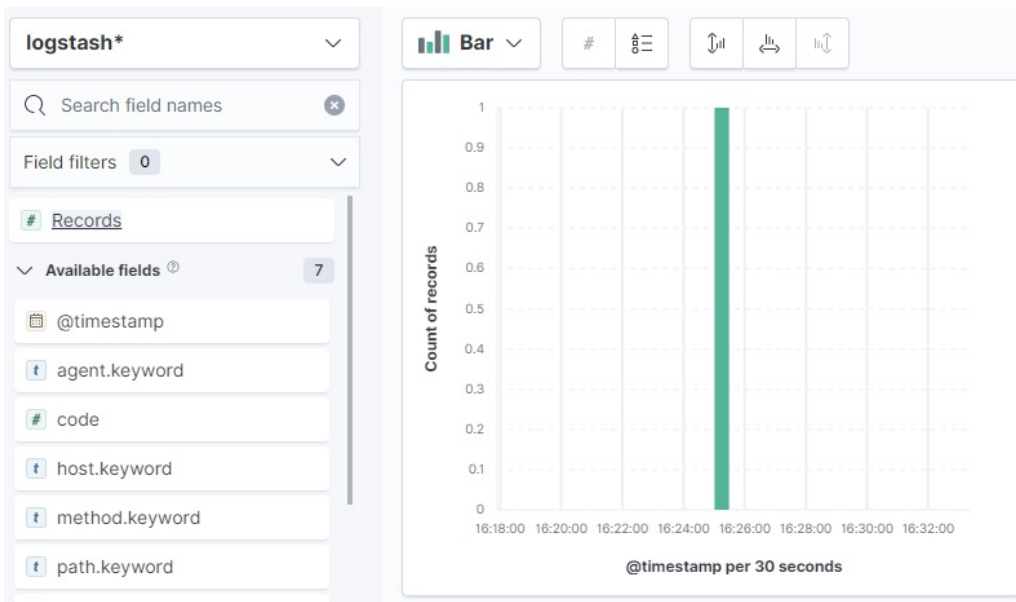
Custom visualization

Use Vega to create new types of visualizations. *Requires knowledge of Vega syntax.*

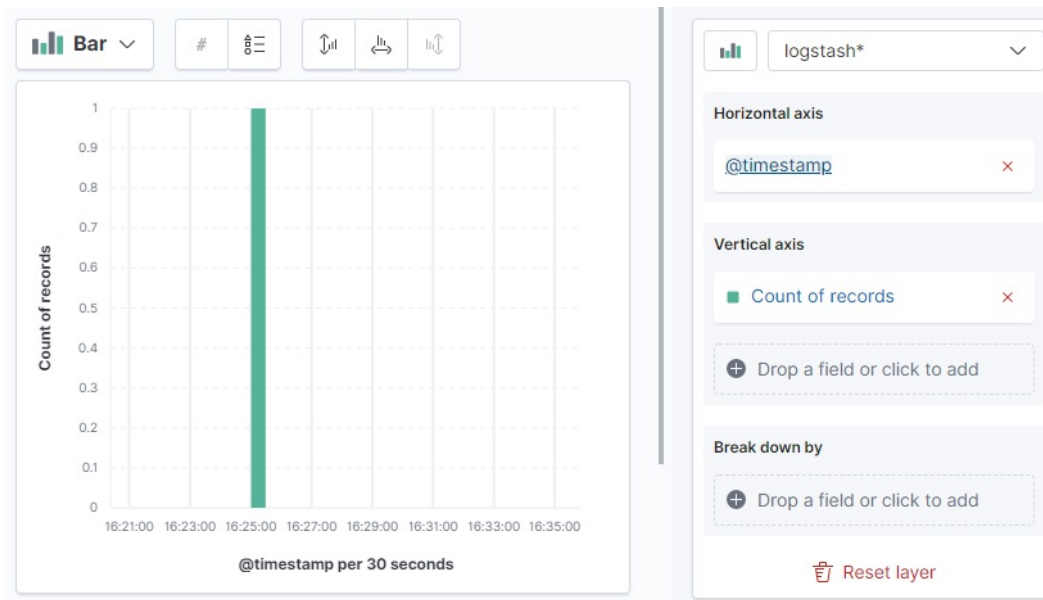
Select "Bar vertical stacked":



For the source, make sure **logstash*** is selected and drag the **Records** field to the center:



Under On the right side pane, select `@timestamp` under **Horizontal axis**:



Check "Customize time interval" and set the minimum interval to `1` and pick `minutes` from the dropdown.

← Horizontal axis configuration

Select a function

Date histogram

Filters

Intervals

Top values

Select a field

@timestamp

How it works

☒ Customize time interval

Minimum interval

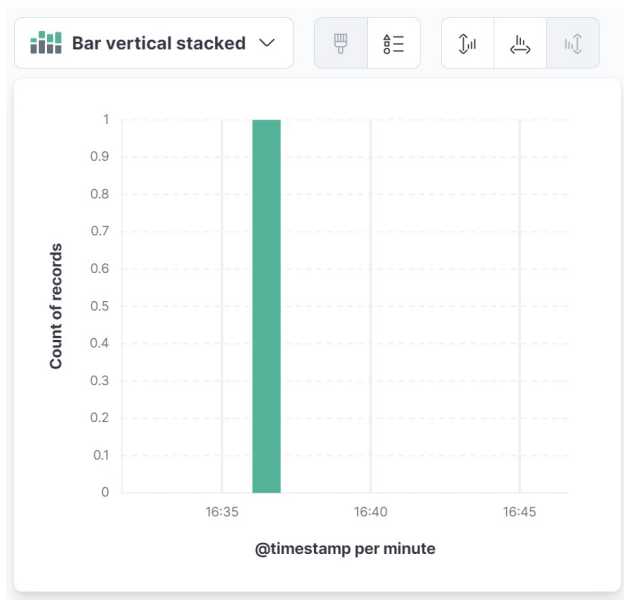
1

minutes

Display name

@timestamp

Click **X Close** to dismiss the Horizontal Axis settings pane. Your graph will update accordingly:



Submitting additional curl requests to your Apache webserver will cause this graph to grow.

Click Save and name your visualization so you can access it later.

Excellent! With this lab, you have now successfully established a basic Elasticsearch, Fluentd (fed by Fluent Bit!) and Kibana logging stack.

5. Clean up

When you have finished exploring, follow these steps to tear down all containers, Fluent Bit and Fluentd instances.

Use **pkill** to shut down all Fluentd and Fluent Bit instances:

```
ubuntu@labsys:~$ pkill -f fluent
ubuntu@labsys:~$
```

Fluent Bit should shut down:

```
...
[2022/06/14 23:47:36] [engine] caught signal (SIGTERM)
[2022/06/14 23:47:36] [ info] [input] pausing tail.0
[2022/06/14 23:47:36] [ warn] [engine] service will shutdown in max 5 seconds
[2022/06/14 23:47:36] [ info] [engine] service has stopped (0 pending tasks)
[2022/06/14 23:47:36] [debug] [input:tail:tail.0] inode=1802060 removing file name
/var/log/apache2/access.log
[2022/06/14 23:47:36] [ info] [input:tail:tail.0] inotify_fs_remove(): inode=1802060 watch_fd=1
[2022/06/14 23:47:36] [ info] [output:forward:forward.0] thread worker #0 stopping...
[2022/06/14 23:47:36] [ info] [output:forward:forward.0] thread worker #0 stopped
[2022/06/14 23:47:36] [ info] [output:forward:forward.0] thread worker #1 stopping...
[2022/06/14 23:47:36] [ info] [output:forward:forward.0] thread worker #1 stopped

ubuntu@labsys:~/lab10$
```

The Fluentd forwarder too:

```
...
2022-06-14 23:47:36 +0000 [info]: Received graceful stop
2022-06-14 23:47:36 +0000 [info]: #0 fluentd worker is now stopping worker=0
2022-06-14 23:47:36 +0000 [info]: #0 shutting down fluentd worker worker=0
2022-06-14 23:47:36 +0000 [info]: #0 shutting down input plugin type=:tail plugin_id="object:758"
2022-06-14 23:47:37 +0000 [info]: #0 shutting down output plugin type=:forward
plugin_id="object:730"
2022-06-14 23:47:38 +0000 [info]: Worker 0 finished with status 0

ubuntu@labsys:~/lab10$
```

Finally, use `docker container rm` to shut down all containers:

```
ubuntu@labsys:~$ sudo docker container rm $(sudo docker container stop aggregator1 elasticsearch kibana)

aggregator1
elasticsearch
kibana

ubuntu@labsys:~$
```

Congratulations, you have completed the Lab.

LFS242 - Cloud Native Logging with Fluentd

Lab 10 – Fluent Bit and Fluentd

- Send memory usage information to a file, mod10.mem.txt

```
ubuntu@labsys:~/lab10$ fluent-bit -i mem -o file -p path=/home/ubuntu/lab10/mod10.mem.txt
```

Fluent Bit v1.0.6

Copyright (C) Treasure Data

```
[2019/05/16 20:41:11] [ info] [storage] initializing...
[2019/05/16 20:41:11] [ info] [storage] in-memory
[2019/05/16 20:41:11] [ info] [storage] normal synchronization mode, checksum disabled
[2019/05/16 20:41:11] [ info] [engine] started (pid=35155)
^C
```

```
[engine] caught signal (SIGINT)
[2019/05/16 20:41:13] [ info] [input] pausing mem.0
```

```
ubuntu@labsys:~/lab10$ tail -3 mod10.mem.txt
mem.0: [1558064443.000471, {"Mem.total":4028944, "Mem.used":2420388, "Mem.free":1608556,
"Swap.total":4194300, "Swap.used":4316, "Swap.free":4189984}]
mem.0: [1558064444.000240, {"Mem.total":4028944, "Mem.used":2420140, "Mem.free":1608804,
"Swap.total":4194300, "Swap.used":4316, "Swap.free":4189984}]
mem.0: [1558064445.000237, {"Mem.total":4028944, "Mem.used":2420016, "Mem.free":1608928,
"Swap.total":4194300, "Swap.used":4316, "Swap.free":4189984}]
ubuntu@labsys:~/lab10$
```

- Tail a file, log.txt, and send output as msgpack to stdout

```
ubuntu@labsys:~/lab10$ echo "Hello LFS242 Student" >> log.txt
```

```
ubuntu@labsys:~/lab10$ fluent-bit -i tail -p path=log.txt -o stdout -p format=msgpack
```

Fluent Bit v1.0.6

Copyright (C) Treasure Data

```
[2019/05/16 20:43:29] [ info] [storage] initializing...
[2019/05/16 20:43:29] [ info] [storage] in-memory
[2019/05/16 20:43:29] [ info] [storage] normal synchronization mode, checksum disabled
[2019/05/16 20:43:29] [ info] [engine] started (pid=35162)
[0] tail.0: [1558064609.323909472, {"log"=>"Hello LFS242 Student"}]
```

```
^C
[engine] caught signal (SIGINT)
[2019/05/16 20:43:36] [ info] [input] pausing tail.0
```

```
ubuntu@labsys:~/lab10$
```