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.