The Benefits of Logging and a Look into Fluentd

Benjamin Moeller
EECS 465
University of Kansas

5/1/2022

1 Introduction

With all of the different types of data that can come from a variety of sources, it is advantageous to have a unified logging layer to sort and export all these different data sources. On the market, some of the tools that can accomplish this tend to be on the expensive side, but there are also open source tools that work to compete with these tools. One such tool is Fluentd, which is a logging layer comparable to Logstash, another popular open-source logging tool. Both of these tools can be connected to Kibana and ElasticSearch, making them powerful logging tools when used correctly. The rest of the paper is structured as follows. The next section gives the motivation as to why logging tools are important. Section 3 goes into more detail on how Fluentd works under the hood. Section 4 gives a comparison between the two popular logging layer tools, Logstash and Fluentd. Section 5 will give a demo on how to utilize Fluentd in different logging scenarios, and then Section 6 will conclude the paper.

2 Motivation

Logging tools are important due to the structure and organization they can provide to data on a day to day basis. For example, if a company has multiple machines running applications and other programs, it would be nice to have the log data of all the machines centralized in a single logging server. Tools like Fluentd can do this, as they provide features for listening and filtering logs as they are created (more on how this works in the next section). There are a variety of different logging tools out that have different functionalities and purposes. The two tools that will be discussed in this paper, Fluentd and Logstash, are known for their ability to be unified logging layers that can listen for and send data to a variety of destinations. Organization can be key when it comes to security and other subjects, as there are some things that can only be seen when data has been brought together into one place and compared. Fluentd and other logging tools can help accomplish this, which is why I believe that they are extremely important to both utilize and understand.

3 How Fluentd Works

3.1 Introduction

At its most basic level, Fluentd is a unified logging layer that can take input from many different data sources and route them to different destinations depending on what the user desires. Below is a chart from the Fluentd website that shows how different data sources can be mapped.

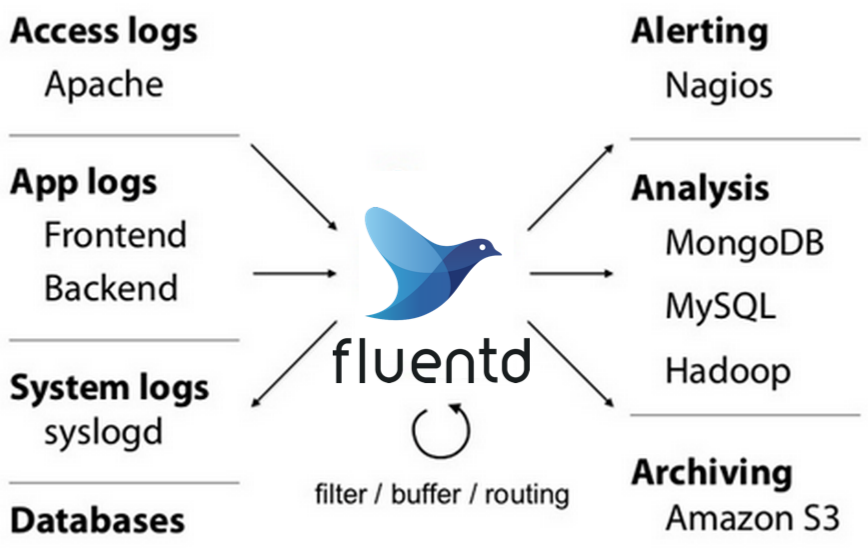


Figure 1: Fluentd Flow Diagram [2]

With this you could route a single data source to multiple different outputs, or you could even map several different data sources to a single output depending on the use case. This is done through a configuration file that has the following directives inside (See Figure 3 below for an example configuration file).

3.2 The Source Directive

The source directive is the part of the configuration file that "determines the input of the [data] sources"[1]. So within the source directive, you can list the type of source you want to receive from, which can be anything ranging from a file, a TCP (transmission control protocol) packet, a UDP (user datagram protocol) packet, or most importantly, a forward packet from another Fluentd instance. With the last source mentioned in particular, this allows multiple machine to link if they are each running a Fluentd instance that can communicate with each other.

3.3 The Match Directive

The match directive, opposite of the source directive, is a directive that "determines the output destinations"[1] of the corresponding input sources. Some possible output locations Fluentd can send data to are a file, stdout (standard out), an HTTP (hypertext transfer protocol) address, and more. In order to forward data to other services such as ElasticSearch, Kafka, MongoDB, and others it is necessary to do some extra installation for those output plugins. The installation for those plugin services listed never was too strenuous though, with the steps listed only taking one to two command line instructions to install all the dependencies.

3.4 The Filter Directive

The filter directive is an unique directive that "determines the event processing pipelines"[1] of each matching source directive. What this means is that it can be used to alter a data stream as it goes through Fluentd. For example, a piece of data could come into Fluentd via a source directive, go through a filter directive to add to the contents of that data stream, and then finally go to through a match directive and get forwarded to the output destination of choice.

3.5 Putting everything together

With all three of these directives put together, it is possible to achieve data organization and forwarding in the following way.

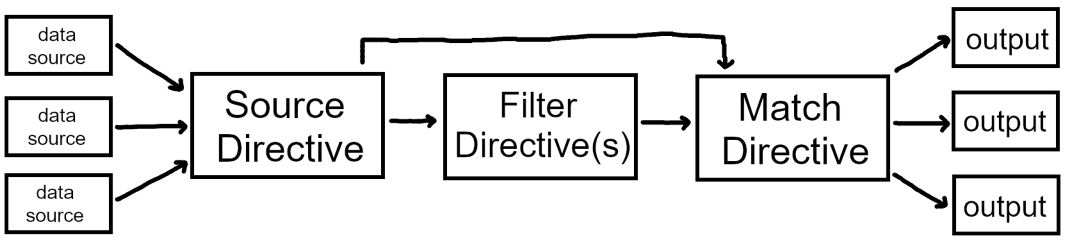


Figure 2: How Data Flows Through Directives

Data starts by entering into the source section, and then has two options. It can either go straight to a match directive, or it can go through a series of one or more filter directives. This way, data that has been identified by a source directive can have the option of going through filters or not depending on the tag that a filter plugin is given. This gives users a lot of customizability for how they want their data to be altered.

4 Fluentd and Logstash

4.1 Introduction

Fluentd and Logstash are both widely used open-source logging tools, and are compared due the breadth of similar features that they provide. They are both data collection tools that work to route data to their correct sources, and they are both used inside of the ELK Stack (ElasticSearch, Logstash, and Kibana) and EFK Stack (ElasticSearch, Fluentd, and Kibana) respectively. Due to these highly effective means of organizing and storing data, both Fluentd and Logstash have made large contributions within the logging industry. Although they do relatively similar jobs, there are some key advantages that both have over the other.

4.2 Advantages of Fluentd

In general, "it is known that Logstash consumes more memory than Fluentd"[3]. Due to this Fluentd is a better option to deploy with if a machine is computationally limited. For even smaller machine or embedded systems, there is even a separate, lightweight version of Fluentd called Fluent-Bit that "is implemented primarily in C"[3]. According to its website, FluentBit is also highly recommended for use in containerized environments, such as Docker[5]. Whether using Fluentd or FluentBit, they are both considered to be the right choice when using containerized environments even though Logstash also has a lightweight version called Elastic Beats. This is because "if your use case goes beyond mere data transport, to also require data pulling and aggregation, then you'd need both Logstash and Elastic Beats"[3]. Due to this additional overhead that Fluentd doesn't require, Fluentd is considered the better option for users with limited hardware resources.

4.3 Advantages of Logstash

While Fluentd has the advantage when it comes to lightweight and containerized environments, Logstash excels in other areas than Fluentd does. Because Logstash "was built with ElasticSearch and Kibana in mind"[4], it is more convenient to use when combining these tools together. In general, the "vast plugin ecosystem"[4] that Logstash offers is more convenient than what Fluentd has to offer. While Fluentd has a lot good default options for outputs to match to, in order to connect to services like ElasticSearch there is a bit more installation and setup that is needed. As a result, Logstash becomes the more appealing option if a centralized plugin ecosystem is preferred to Fluentd's "decentralized yet vast plugin ecosystem hosted in individual repositories"[3].

4.4 Conclusion

All in all, both Fluentd and Logstash are both amazing options when it comes to data collection and organization. While Fluentd holds the advantage for smaller, more lightweight systems, Logstash definitely pulls ahead when it comes to ease of use with ElasticSearch and other plugins within its ecosystem.

5 Fluentd demo

5.1 Demo Dependencies

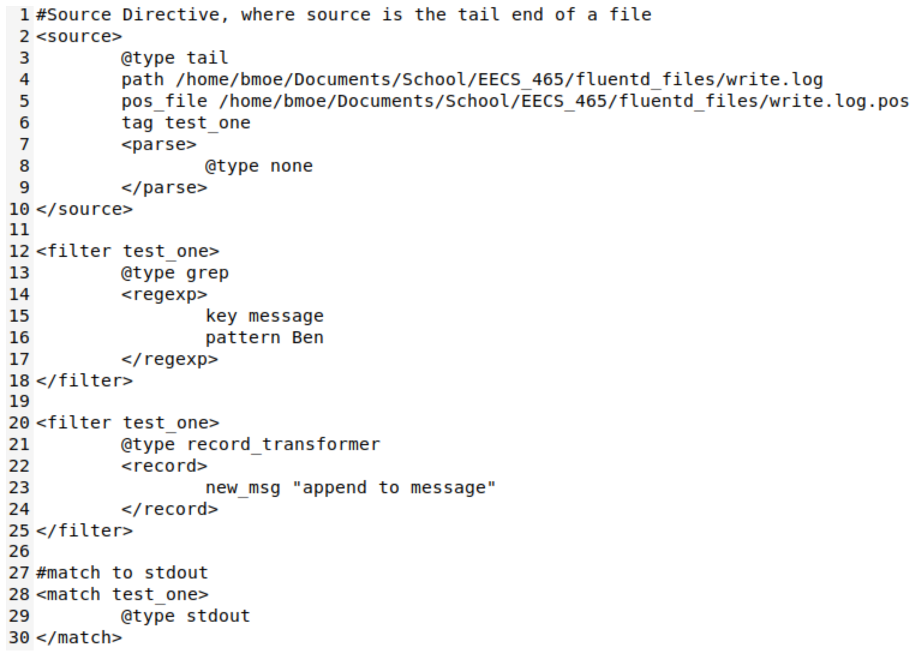
This demo will be performed using the Fluentd Logging platform. With that, there will also be two machines in use. One machine that will be running on Ubuntu Linux version 20.04.1, and another machine that is running Kali Linux version 5.10.28. The demo will also use some python scripts to help with generating data for Fluentd, so python3.8 is also installed on both machines.

5.2 The Objective of the Demo

The main goal of this demo is to test two questions. One, how well is Fluentd able to sort through and edit certain patterns in data using the filter directive. Two, is it possible to connect two instances of Fluentd on two separate machines? These questions will be analyzed in the following sections.

5.3 Filtering with Fluentd

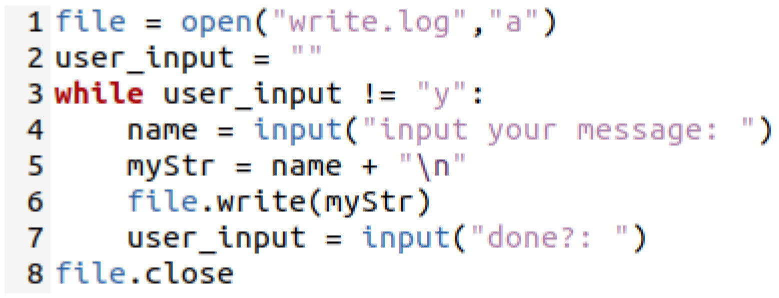
This test will work in the following way. Fluentd will be started up as a background process with the following configuration file.



```
#Source Directive, where source is the tail end of a file
<source>
        @type tail
        path /home/bmoe/Documents/School/EECS_465/fluentd_files/write.log
        pos_file /home/bmoe/Documents/School/EECS_465/fluentd_files/write.log.pos
        tag test_one
        <parse>
                @type none
        </parse>
</source>

<filter test_one>
        @type grep
        <regexp>
                key message
                pattern Ben
        </regexp>
</filter>

<filter test_one>
        @type record_transformer
        <record>
                new_msg "append to message"
        </record>
</filter>

#match to stdout
<match test_one>
        @type stdout
</match>
```

Figure 3: Test 1 configuration file

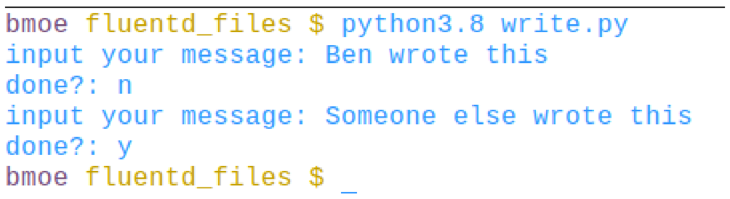
This configuration file has a source directive that points the the tail of a file named “write.log”. Fluentd also has a position file that tracks the current end of the file, and when a new line is added it will update Fluentd. That data will then run through the filter directive, and if it finds the name “Ben” in the message, it will append the string “append to message” to the data. All of this will be printed out to stdout using the match directive at the bottom, if and only if it makes it through the filter. Also in order to write to the file, the following python script is used.



```python
file = open("write.log","a")
user_input = ""
while user_input != "y":
    name = input("input your message: ")
    myStr = name + "\n"
    file.write(myStr)
    user_input = input("done?: ")
file.close
```

Figure 4: Test 1 python file

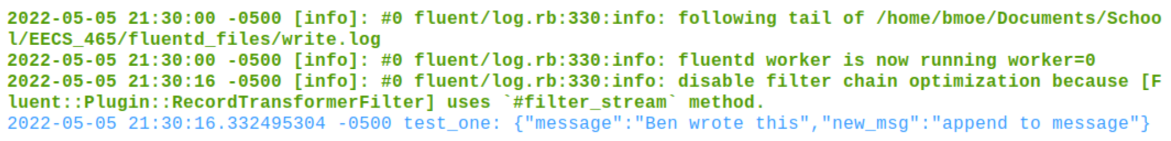
To start up Fluentd, all that is required is to type "Fluentd -c ./fluent/fluent.conf -vv &", where the "fluent.conf" file is the configuration file shown above. After Fluentd is up and running, I ran the python file above and wrote the following lines to file.



```
bmoe fluentd_files $ python3.8 write.py
input your message: Ben wrote this
done?: n
input your message: Someone else wrote this
done?: y
bmoe fluentd_files $ _
```

Figure 5: Test 1 python input

With this, we should expect only one message to pass through, which is the message that contained "Ben" inside of it. The following was the output in Fluentd.



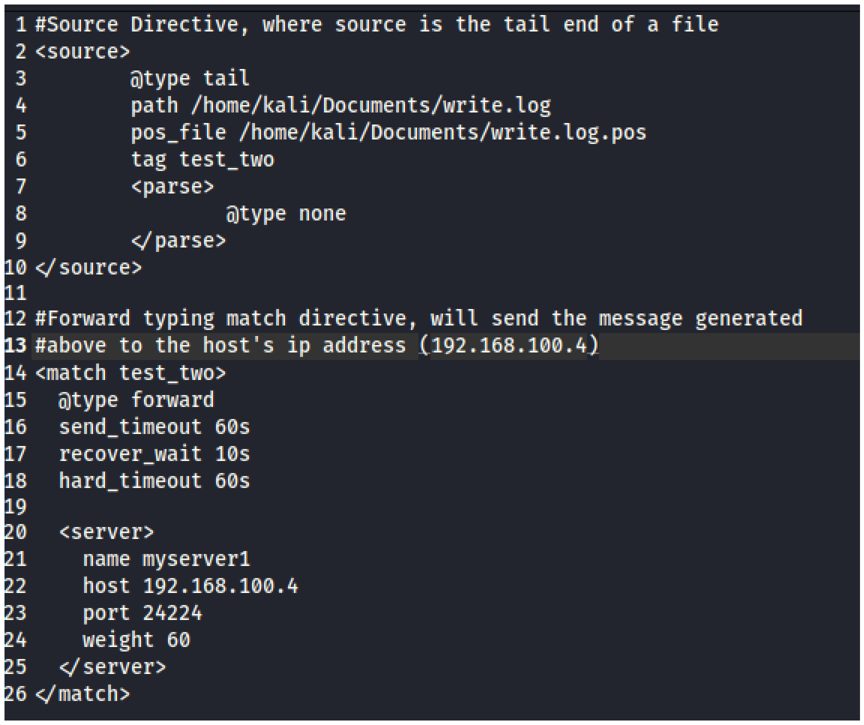
```
2022-05-05 21:30:00 -0500 [info]: #0 fluent/log.rb:330:info: following tail of /home/bmoe/Documents/Schoo
l/EECS_465/fluentd_files/write.log
2022-05-05 21:30:00 -0500 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-05-05 21:30:16 -0500 [info]: #0 fluent/log.rb:330:info: disable filter chain optimization because [F
luent::Plugin::RecordTransformerFilter] uses `#filter_stream` method.
2022-05-05 21:30:16.332495304 -0500 test_one: {"message":"Ben wrote this","new_msg":"append to message"}
```

Figure 6: Test 1 Fluentd output

Since the message with "Ben" inside of it was the only one that passed though, and the new message had a new string appended to it, this part of the demo was a success.

5.4 Connecting two machines with Fluentd

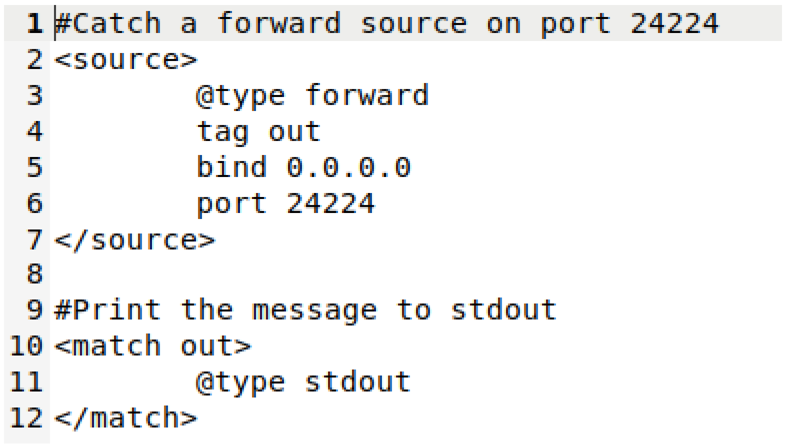
For this test, two different virtual machines will be used as mentioned earlier, one Ubuntu virtual machine and one Kali Linux virtual machine. In this scenario, the goal is to send a message from the Kali machine to the Ubuntu machine. This will be done by using the forward typing match directive, which will basically take a message as input, and forward it to another instance of Fluentd by connecting the two machines by their IP Addresses. Here is the following configuration file that was used for the Kali machine.



```
#Source Directive, where source is the tail end of a file
<source>
        @type tail
        path /home/kali/Documents/write.log
        pos_file /home/kali/Documents/write.log.pos
        tag test_two
        <parse>
                @type none
        </parse>
</source>

#Forward typing match directive, will send the message generated
#above to the host's ip address (192.168.100.4)
<match test_two>
  @type forward
  send_timeout 60s
  recover_wait 10s
  hard_timeout 60s

  <server>
    name myserver1
    host 192.168.100.4
    port 24224
    weight 60
  </server>
</match>
```

Figure 7: Test 2 Kali Machine Fluentd configuration file

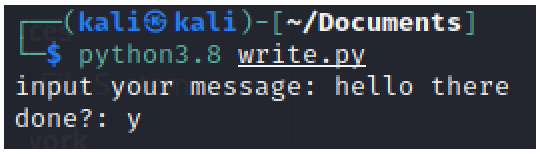
In this configuration file, the source directive will be reading from the end of a file, using the same python file setup used in the previous test (see Figure 4). So once Fluentd notices that there is a new line in the file, it will use the match directive with the forward type to send the message to the Ubuntu machine with IP Address “192.268.100.4” at port 24224. From there it is up to the Ubuntu virtual machine running its own instance of Fluentd to catch the message with the following configuration file.



```
#Catch a forward source on port 24224
<source>
        @type forward
        tag out
        bind 0.0.0.0
        port 24224
</source>

#Print the message to stdout
<match out>
        @type stdout
</match>
```

Figure 8: Test 2 Ubuntu Machine Fluentd configuration file

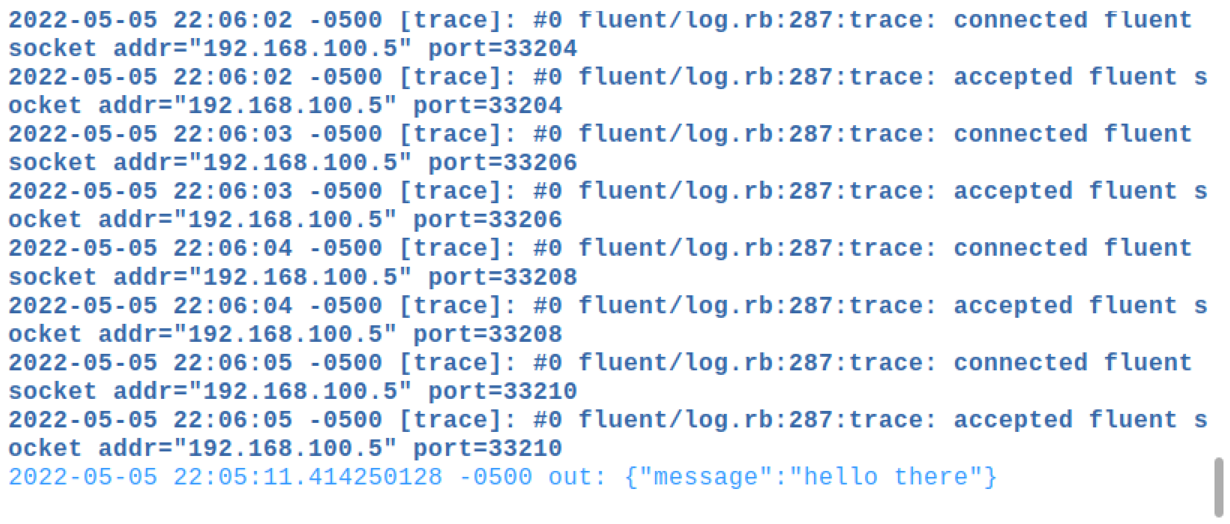
The source directive in this configuration file will listen for any data that arrive at port 24224. Once it does, it will then send the message that it received to stdout for printing. So to start the test, I started running Fluentd on both machines as background processes. Then on the Kali machine I ran the previously shown python file (Figure 4) with the following message.



```
┌──(kali㉿kali)-[~/Documents]
└─$ python3.8 write.py
input your message: hello there
done?: y
```

Figure 9: Test 2 Kali message sent

Then on the Ubuntu machine, the following message was printed to the terminal.



```
2022-05-05 22:06:02 -0500 [trace]: #0 fluent/log.rb:287:trace: connected fluent
socket addr="192.168.100.5" port=33204
2022-05-05 22:06:02 -0500 [trace]: #0 fluent/log.rb:287:trace: accepted fluent s
ocket addr="192.168.100.5" port=33204
2022-05-05 22:06:03 -0500 [trace]: #0 fluent/log.rb:287:trace: connected fluent
socket addr="192.168.100.5" port=33206
2022-05-05 22:06:03 -0500 [trace]: #0 fluent/log.rb:287:trace: accepted fluent s
ocket addr="192.168.100.5" port=33206
2022-05-05 22:06:04 -0500 [trace]: #0 fluent/log.rb:287:trace: connected fluent
socket addr="192.168.100.5" port=33208
2022-05-05 22:06:04 -0500 [trace]: #0 fluent/log.rb:287:trace: accepted fluent s
ocket addr="192.168.100.5" port=33208
2022-05-05 22:06:05 -0500 [trace]: #0 fluent/log.rb:287:trace: connected fluent
socket addr="192.168.100.5" port=33210
2022-05-05 22:06:05 -0500 [trace]: #0 fluent/log.rb:287:trace: accepted fluent s
ocket addr="192.168.100.5" port=33210
2022-05-05 22:05:11.414250128 -0500 out: {"message":"hello there"}
```

Figure 10: Test 2 Fluentd output on Ubuntu machine

Through this, it shows that it is possible to connect multiple machines running different instances of Fluentd. In an enterprise scale, instead of printing to stdout, this data could then be sent off to files, or even to a program like ElasticSearch. This is all made possible through the organization Fluentd can provide.

6 Conclusion

In conclusion, logging tools are extremely useful when it comes to collecting and organizing the data we create on a day to day basis. Without these tools, it would be difficult to collect and send data to a centralized location where the data can be analyzed. Whether using a tool like Fluentd that excels on lightweight or containerized environments, or Logstash that works well with ElasticSearch, both options will increase the efficiency of data collection dramatically.

References

[1] Fluentd 1.0 Documentation. (2021). Fluentd. https://docs.Fluentd.org/

[2] Fluentd Architecture Overview. (2021). What is Fluentd?. https://www.Fluentd.org/architecture

[3] Platform9. (2020, February 11). Kubernetes Logging: Comparing Fluentd vs. Logstash. https://platform9.com/blog/kubernetes-logging-comparing-Fluentd-vs-Logstash/

[4] Chelat, Ajit. (2021, April 9). Fluentd vs. Logstash: The Ultimate Log Agent Battle. https://www.logiq.ai/Fluentd-vs-Logstash-the-ultimate-log-battle/

[5] Docker docs. (2021). Fluentd logging driver. https://docs.docker.com/config/containers/logging/Fluentd/