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@labsys:~/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@labsys:~/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@labsys:~/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@labsys:~/lab8$
```

Now use `iotop` to track the rate of disk interaction (read and write) for the Fluentd supervisor and worker. `sudo` is required:

```
ubuntu@labsys:~/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@labsys:~/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@labsys:~/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.

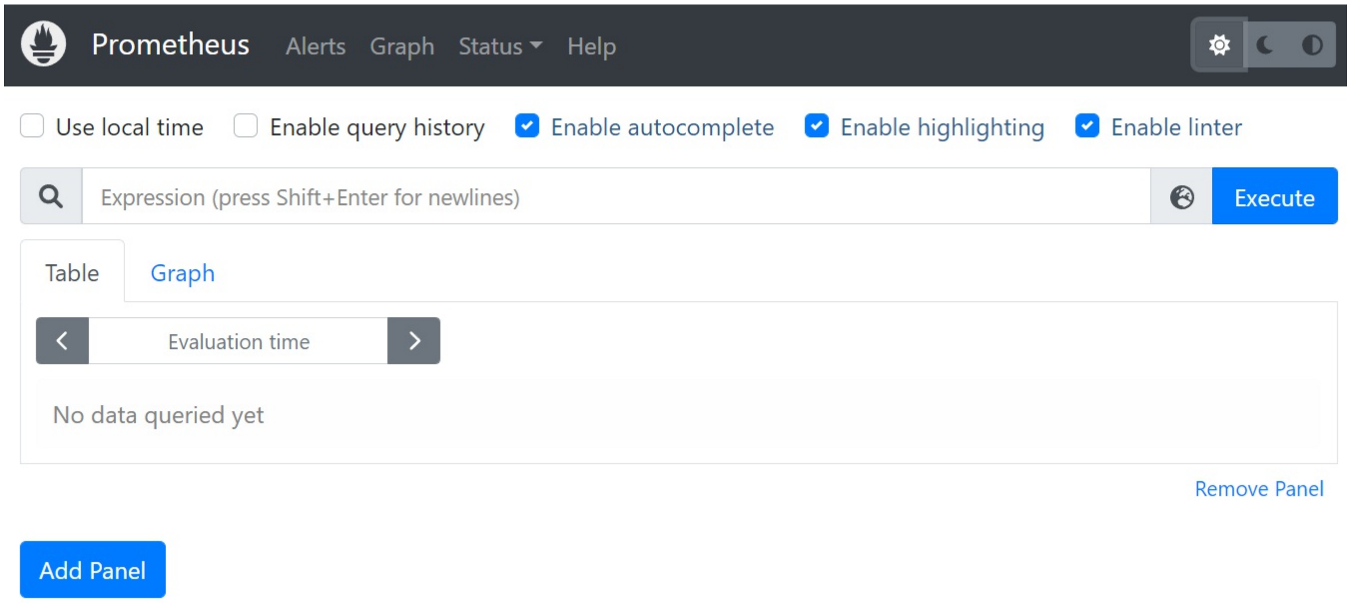
Run Prometheus in a container, binding the container ports 9090 and 24231 to the host equivalents and using the `prometheus.yaml` from the `fluent-plugin-prometheus` repo:

```
ubuntu@labsys:~/lab8$ sudo docker run -d --name prometheus --network host \
-v ~/lab8/prometheus.yaml:/etc/prometheus/prometheus.yml prom/prometheus:v2.36.2

...

45c8afba10d3c8840e7de9f470a5b8a0904fade6d0b9b36051ca5ce4b145e0d6

ubuntu@labsys:~/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 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@labsys:~/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@labsys:~/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@labsys:~/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@labsys:~/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@labsys:~/lab8/monitored$
```

Run Fluentd using that configuration:

```
ubuntu@labsys:~/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
```

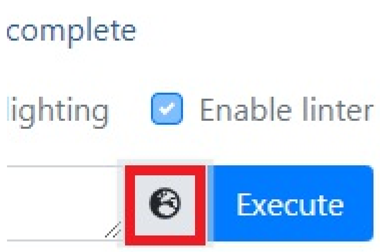
With Fluentd up and running, check if the metrics endpoint is exposed. This endpoint is only exposed by configuring the Prometheus input plugin, which opens the metrics endpoint on port 24231 by default. Fluentd will report that it is listening for Prometheus at DEBUG verbosity (running Fluentd with the `-v` flag).

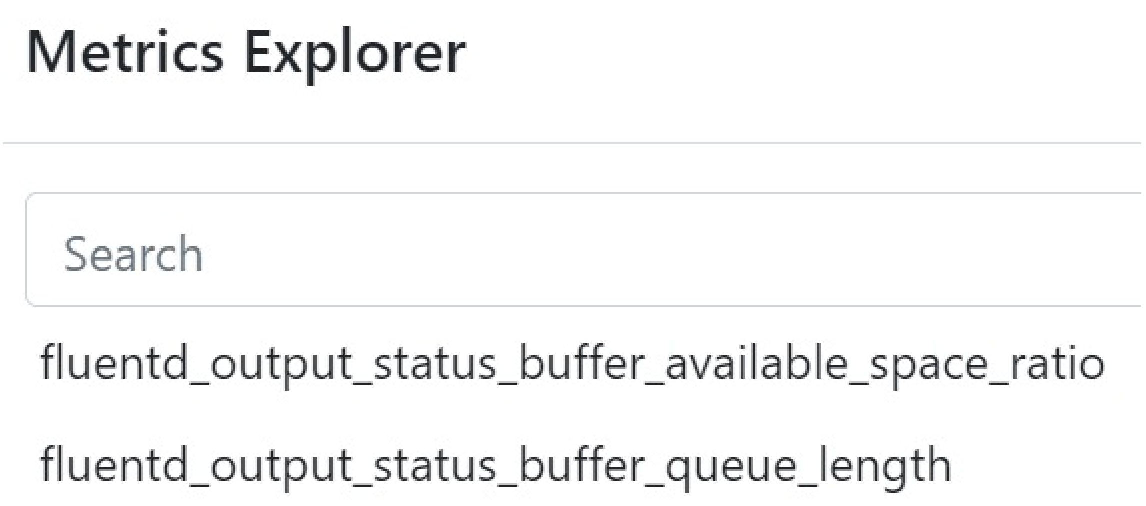
Send a curl request to the metrics endpoint:

```
ubuntu@labsys:~/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@labsys:~/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:





fluentd_output_status_buffer_available_space_ratio

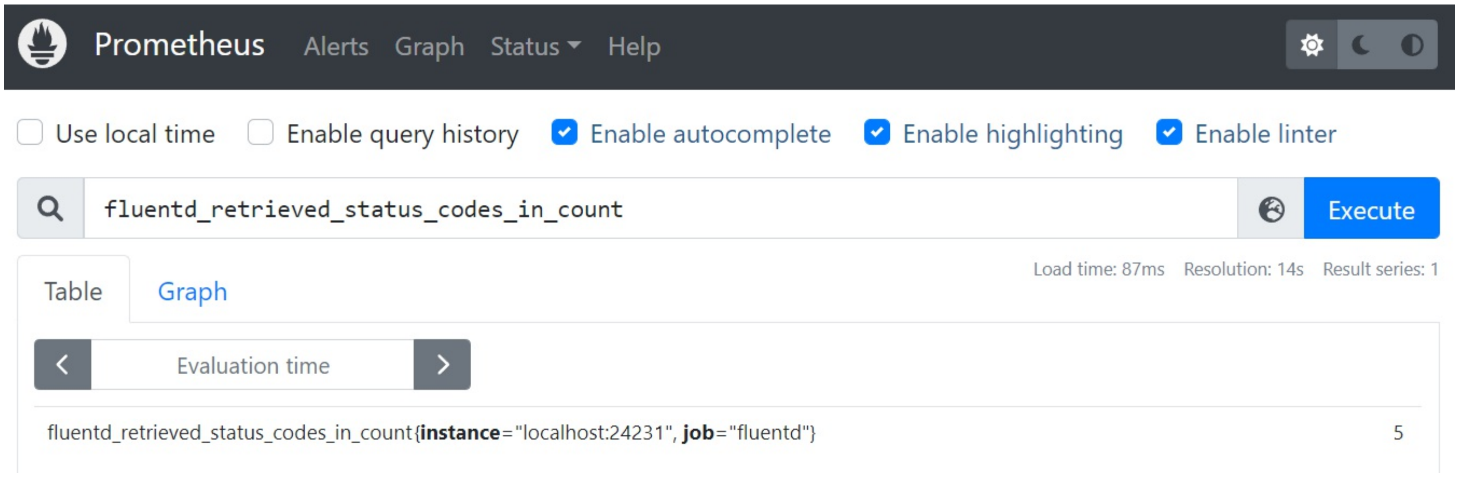
fluentd_output_status_buffer_queue_length

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:



fluentd_retrieved_status_codes_in_count

Load time: 87ms Resolution: 14s Result series: 1

fluentd_retrieved_status_codes_in_count{**instance**="localhost:24231", **job**="fluentd"} 5

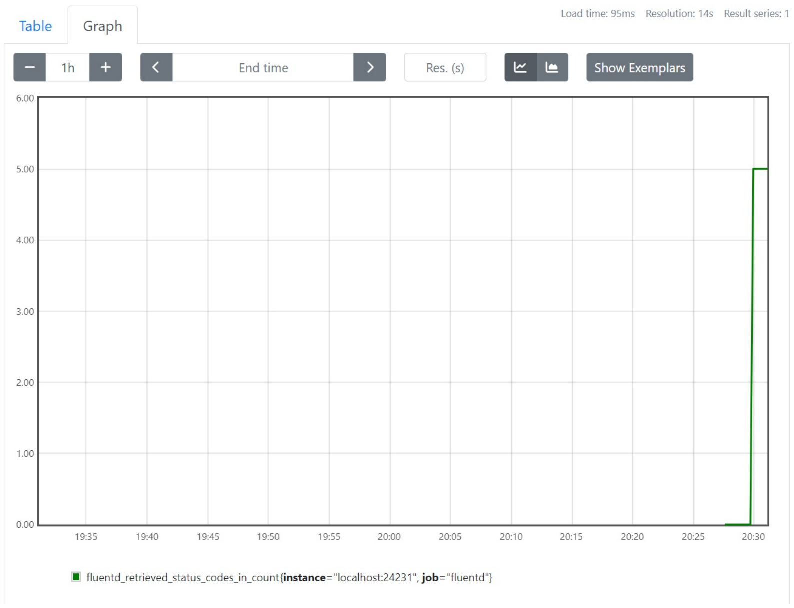
Curl the metrics endpoint again, this time grepping for a specific metric, like the custom message counters configured earlier, `fluentd_retrieved_status_codes_`:

```
ubuntu@labsys:~/lab8$ curl -s http://localhost:24231/metrics | grep
fluentd_retrieved_status_codes_in_count

fluentd_retrieved_status_codes_in_count 5.0
```

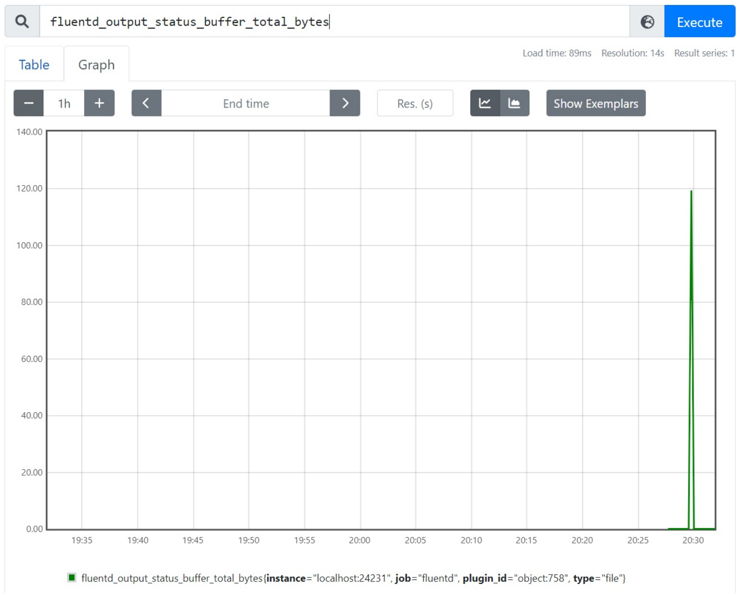
```
ubuntu@labsys:~/lab8$
```

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@labsys:~/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@labsys:~/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@labsys:~/lab8$ pkill -SIGUSR2 fluentd

ubuntu@labsys:~/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@labsys:~/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,"buffe
r_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
":{}}]}

ubuntu@labsys:~/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@labsys:~/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@labsys:~/lab8$
```

Now, try the curl again, except pipe the output into JQ:

```
ubuntu@labsys:~/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@labsys:~/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@labsys:~/lab8$ sudo docker container rm $(sudo docker stop prometheus)

prometheus

ubuntu@labsys:~/lab8$
```

Send a `pkill -f` to terminate any locally running instances of Fluentd, or use `CTRL C` in any terminals running Fluentd:

```
ubuntu@labsys:~/lab8$ pkill -f fluentd

ubuntu@labsys:~/lab8$ cd

ubuntu@labsys:~$
```

Congratulations, you have completed the lab!