LFS242 - Cloud Native Logging with Fluentd

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

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

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

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

Objectives

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

0. Prepare the lab system

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

Install Docker with the quick installation script:

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

...

ubuntu@labsys:~$
```

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

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

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

#!/bin/sh

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

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

ubuntu@labsys:~$ ./fluentd-setup

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

```
fluentd 1.14.6

ubuntu@labsys:~$
```

1. Preparing a MongoDB database

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

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

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

ubuntu@labsys:~$
```

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

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

...

13e1440044db5a40b21eb68f70447e133aca0843556357fe75f607382ab8531f

ubuntu@labsys:~$
```

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

Use `docker logs` to see the MongoDB information:

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

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

ubuntu@labsys:~$
```

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

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

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

172.17.0.2

ubuntu@labsys:~$
```

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

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

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

ubuntu@labsys:~$
```

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

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

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

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

ubuntu@labsys:~$
```

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

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

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

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

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

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

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

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

MongoDB shell version v4.2.21
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("1d59768f-3af3-49ea-ad46-230c831cc9b5") }
MongoDB server version: 4.2.21
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

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

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

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

>
```

Using the terminal output, answer the following questions:

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

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

Create a database for Fluentd:

```
> use fluentd

switched to db fluentd
```

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

Create a collection called "lab3":

```
> db.createCollection("lab3")

{ "ok" : 1 }
```

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

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

```
ubuntu@labsys:~$ sudo docker logs mongodb --tail 3

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

ubuntu@labsys:~$
```

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

MongoDB is now ready to accept events from Fluentd!

2. Prepare Fluentd for interaction with MongoDB

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

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

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

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

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

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

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

  Examples:
    gem install rake
    gem list --local
    gem build package.gemspec
    gem help install

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

  Further information:
    http://guides.rubygems.org

ubuntu@labsys:~$
```

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

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

*** LOCAL GEMS ***

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

...

ubuntu@labsys:~$
```

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

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

The list option can also be used to search its configured remote sources for plugins, which by default will include Rubygems.org.

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

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

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

ubuntu@labsys:~$
```

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

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

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

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

ubuntu@labsys:~$
```

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

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

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

*** REMOTE GEMS ***

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

ubuntu@labsys:~$
```

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

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

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

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

ubuntu@labsys:~$
```

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

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

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

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

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

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

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

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

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

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

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

ubuntu@labsys:~$
```

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

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

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

ubuntu@labsys:~$
```

All of the gem files are placed into directories here.

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

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

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

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

ubuntu@labsys:~$
```

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

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

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

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

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

ubuntu@labsys:~$
```

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

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

Prepare a new Fluentd configuration file, mongo.conf

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

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

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

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

```
</match>

ubuntu@labsys:~$
```

The intent of this configuration is:

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

2b. Start Fluentd using the mongo.config

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

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

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

2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: parsing config file is succeeded
path="mongo.conf"
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: gem 'fluent-plugin-mongo' version '1.5.0'
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: gem 'fluentd' version '1.14.6'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered output plugin 'mongo'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-20 15:47:15 +0000 [debug]: fluent/log.rb:309:debug: Setup mongo configuration: mode = normal
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'regexp'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'multiline'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered input plugin 'tail'
2022-06-20 15:47:15 +0000 [trace]: fluent/log.rb:287:trace: registered parser plugin 'none'
2022-06-20 15:47:15 +0000 [debug]: fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: using configuration file: <ROOT>
  <source>
    @type tail
    path "/tmp/mongotrack"
    pos_file "/tmp/mongotrack.pos"
    tag "mongo.lab3"
    <parse>
      @type "none"
      unmatched_lines
    </parse>
  </source>
  <match mongo.**>
    @type mongo
    host "172.17.0.2"
    port 27017
    database "fluentd"
    collection "lab3"
    buffer_chunk_limit 8m
    <inject>
      time_key "time"
    </inject>
  </match>
</ROOT>
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: starting fluentd-1.14.6 pid=44180
ruby="2.7.0"
2022-06-20 15:47:15 +0000 [info]: fluent/log.rb:330:info: spawn command to main:  cmdline=
["/usr/bin/ruby2.7", "-Eascii-8bit:ascii-8bit", "/usr/local/bin/fluentd", "--config", "mongo.conf",
"-vv", "--under-supervisor"]
2022-06-20 15:47:16 +0000 [info]: fluent/log.rb:330:info: adding match pattern="mongo.**"
type="mongo"
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered output plugin 'mongo'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered metrics plugin 'local'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered buffer plugin 'memory'
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: Setup mongo configuration: mode =
normal
2022-06-20 15:47:16 +0000 [info]: fluent/log.rb:330:info: adding source type="tail"
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'regexp'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'multiline'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered input plugin 'tail'
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: registered parser plugin 'none'
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: No fluent logger for internal event
2022-06-20 15:47:16 +0000 [info]: #0 fluent/log.rb:330:info: starting fluentd worker pid=44185
ppid=44180 worker=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: buffer started instance=1860
stage_size=0 queue_size=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: flush_thread actually running
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: tailing paths: target =  | existing =
2022-06-20 15:47:16 +0000 [info]: #0 fluent/log.rb:330:info: fluentd worker is now running worker=0
2022-06-20 15:47:16 +0000 [debug]: #0 fluent/log.rb:309:debug: enqueue_thread actually running
2022-06-20 15:47:16 +0000 [trace]: #0 fluent/log.rb:287:trace: enqueueing all chunks in buffer
instance=1860
```

- What plugins were loaded by the above configuration?

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

3. Test the MongoDB forwarding settings

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

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

```
ubuntu@labsys:~$ cd ~/lab3

ubuntu@labsys:~/lab3$ touch /tmp/mongotrack

ubuntu@labsys:~/lab3$ echo "First message" >> /tmp/mongotrack

ubuntu@labsys:~/lab3$
```

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

```
...

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

```
...
```

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

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

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

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

ubuntu@labsys:~/lab3$
```

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

```
...

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

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

...
```

- What did Fluentd do before it wrote to MongoDB?

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

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

...

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

ubuntu@labsys:~/lab3$
```

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

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

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

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

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

...

> use fluentd

switched to db fluentd

> db.lab3.find()

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

>
```

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

4. Application Logging to MongoDB

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

4a. Preparing MongoDB for the new application

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

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

```
> use nginx

switched to db nginx

> db.createCollection("access")

{ "ok" : 1 }

> db.createCollection("error")

{ "ok" : 1 }
```

Confirm that the collections have been created:

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

{
        "cursor" : {
                "id" : NumberLong(0),
                "ns" : "nginx.$cmd.listCollections",
                "firstBatch" : [
                        {
                                "name" : "error",
                                "type" : "collection",
                                "options" : {

                                },
                                "info" : {
                                        "readOnly" : false,
                                        "uuid" : UUID("449214ee-f744-4a5b-a17c-e93a00ab675f")
                                },
                                "idIndex" : {
                                        "v" : 2,
                                        "key" : {
                                                "_id" : 1
                                        },
                                        "name" : "_id_",
                                        "ns" : "nginx.error"
                                }
                        },
                        {
                                "name" : "access",
                                "type" : "collection",
                                "options" : {

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

>
```

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

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

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

```
> db.error.find()

>

> db.access.find()

>
```

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

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

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

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

```
> quit()

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

exit

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

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

...

46f1675745e7b7749c7800e8cedc887fdae24a791dfb109f6865fa5c21f8b1b6
```

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

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

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

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

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

ubuntu@labsys:~/lab3$
```

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

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

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

172.17.0.3

ubuntu@labsys:~/lab3$ curl 172.17.0.3

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

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

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

ubuntu@labsys:~/lab3$
```

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

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

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

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

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

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

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

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

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

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

ubuntu@labsys:~/lab3$
```

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

For the `<source>` directives:

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

For the `<match>` directives:

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

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

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

```
^C

...

2022-06-20 15:54:59 +0000 [info]: fluent/log.rb:330:info: Worker 0 finished with status 0

ubuntu@labsys:~/lab3$
```

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

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

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

...

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

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

4d. Interact with NGINX to test log event forwarding

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

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

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

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

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

ubuntu@labsys:~/lab3$
```

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

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

```
ubuntu@labsys:~/lab3$ pkill -SIGUSR1 fluentd

ubuntu@labsys:~/lab3$
```

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

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

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

> use nginx

switched to db nginx

> db.access.find()

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

>
```

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

```
> quit()
```

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

exit

ubuntu@labsys:~/lab3$
```

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

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

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

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

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

...

ubuntu@labsys:~/lab3$
```

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

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

ubuntu@labsys:~/lab3$
```

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

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

> use nginx

> db.access.find()

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

> db.error.find()

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

>
```

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

Cleanup

Use CTRL C to terminate any running instances of Fluentd:

```
^C

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

ubuntu@labsys:~/lab3$
```

Now tear down any running Docker containers:

```
> quit()

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

mongodb
nginx

ubuntu@labsys:~/lab3$
```

Congratulations, you have completed the lab!