데이터 엔지니어링 파일 수집

파일 데이터 수집 도구 플루언트디

Park Suhyuk


psyoblade


psyoblade

Fluentd 1.0
ubuntu 18.04 LTS

NCSOFT®

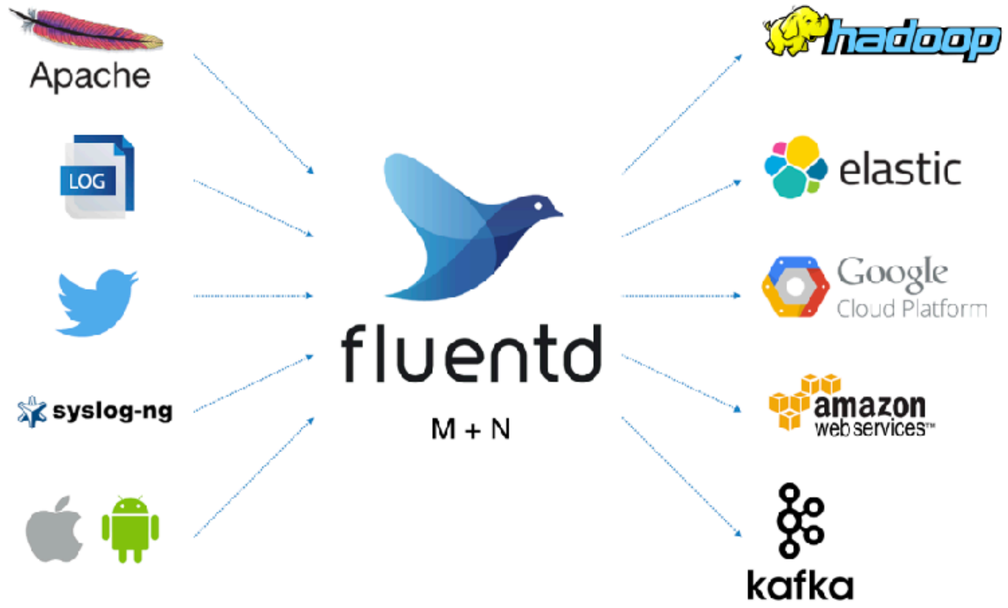
목차

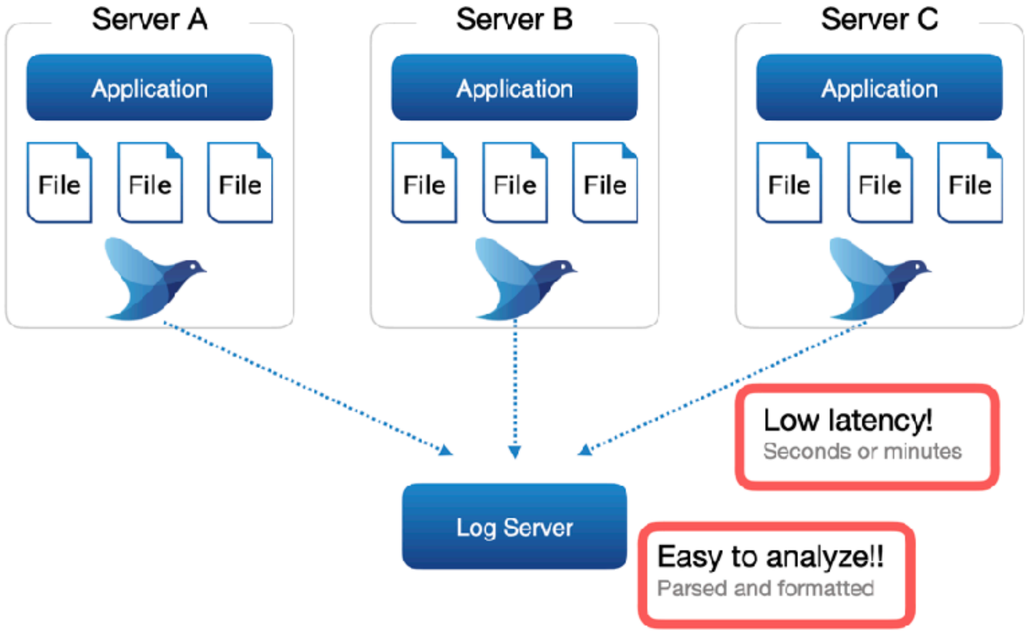
플루언트디 개요 - 특징 및 기능

플루언트디 - 개요

What is 'Fluentd'?

플루언트디는 TreasureData 창립자인 Sadayuki 가 2011년 경에 오픈소스로 고안, 현재는 로그 및 데이터 수집 통합 도구로써 급격히 성장하고 있으며, 현재는 Arm Treasure Data 의 Masahiro 씨가 주요 관리자입니다. 통합된 로그 수집 프레임워크를 제공하며, 데이터 포맷은 Json 기반으로 다양한 로그 유형을 통합이 가능하고 , 대부분의 기능을 plugin 형태로 구성할 수 있으며 외부 시스템이나 인프라에 의존적이지 않다는 장점을 가지고 있습니다. C + Ruby 기반의 엔진으로 초 당 1.3만 데이터 처리에 약 3~40mb 정도의 메모리 소모만으로 가능하며, in-memory or file buffer 지원과 failover 를 통한 ha 구성이 가능한 로그 전송에 촛점을 맞춘 데이터 수집 도구입니다.





Fluentd vs. td-agent

Fluentd 는 커뮤니티 버전의 오픈소스이며 td-agent 는 Treasure Data 에서 운영관리하는 배포판입니다

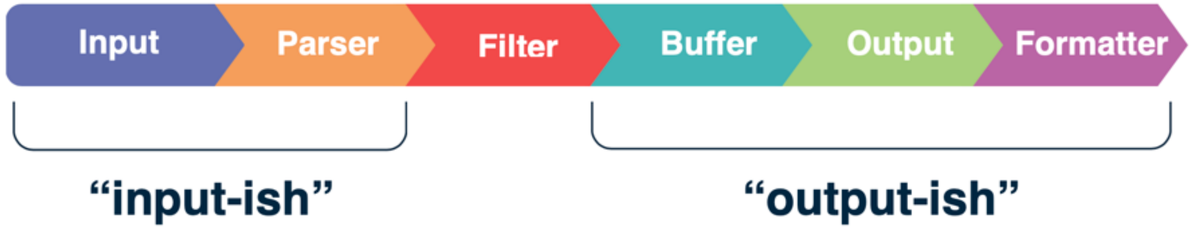
	Fluentd	**td-agent**
QA/Support	Community-driven	QAed by Treasure Data, Inc maintainers Support available for Treasure Data customers
Installation	Ruby gems	rpm / deb / dmg packages.
Configuration	Self service	Preconfigured with a couple of recommended settings, and sending data to Treasure Data by default (can be modified)
Adding 3rd party plugins	$ fluent-gem install fluent-plugin-xx	$ /usr/sbin/td-agent-gem install fluent-plugin-xx
/etc/init.d/ scripts	No (the user needs to write shell script to set it up)	Yes (shipped with .deb and .rpm)
Chef recipe	No	Yes
Memory allocator	OS default	Optimized (jemalloc)

https://www.fluentd.org/faqs

플루언트디 개요 – 아키텍처

플루언트디 – 아키텍처

Fluentd System Architecture

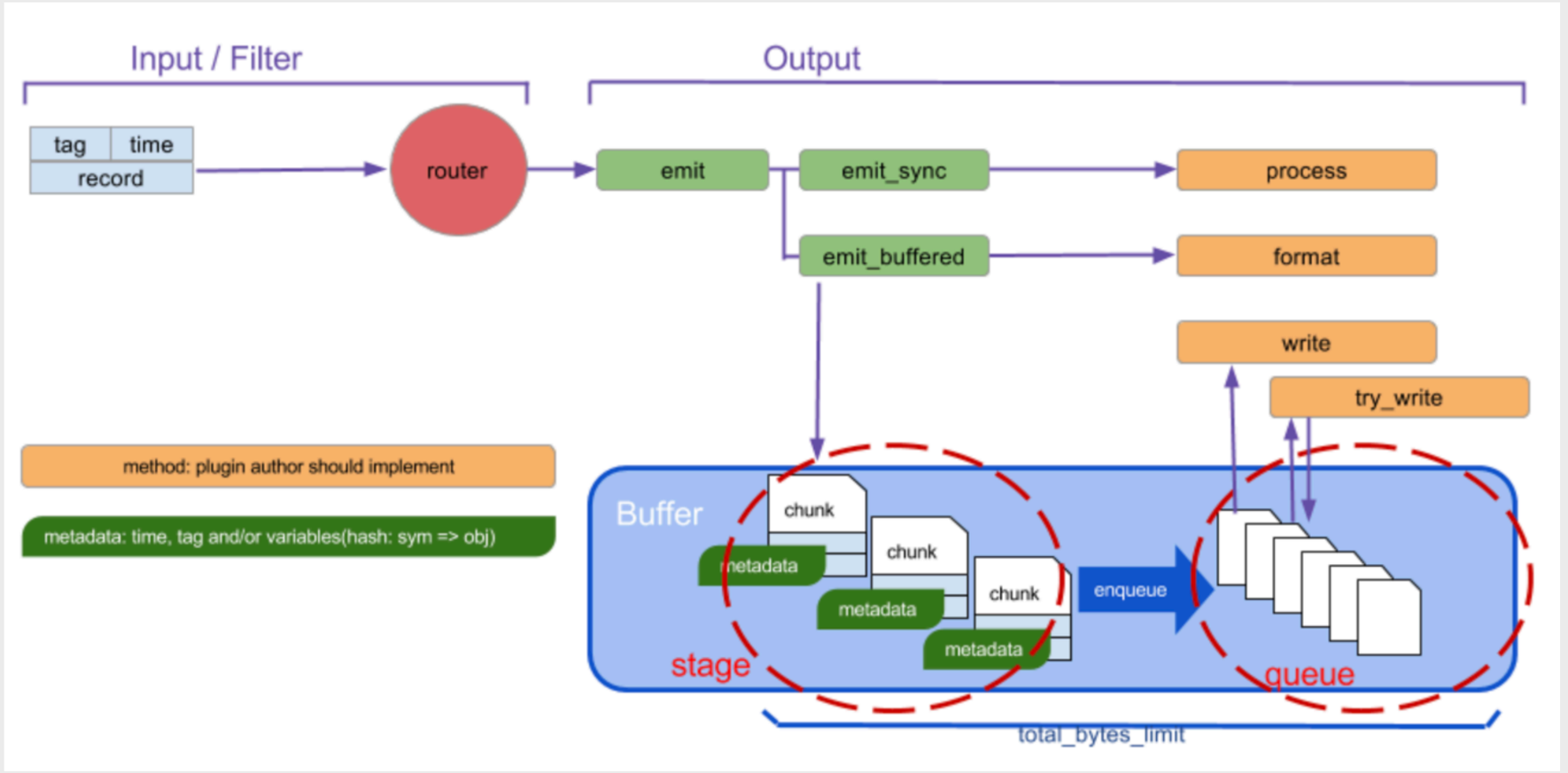


Core

- Buffer & Retry
- Error Handling
- Event routing
- Parallelism

Plugins

- Read / receive data
- Parse data
- Filter / enrich data
- Buffer data
- Format data
- Write / send data



플루언트디 – 이벤트 자료 구조

Event Structure & Simplified Data Pipeline

Time

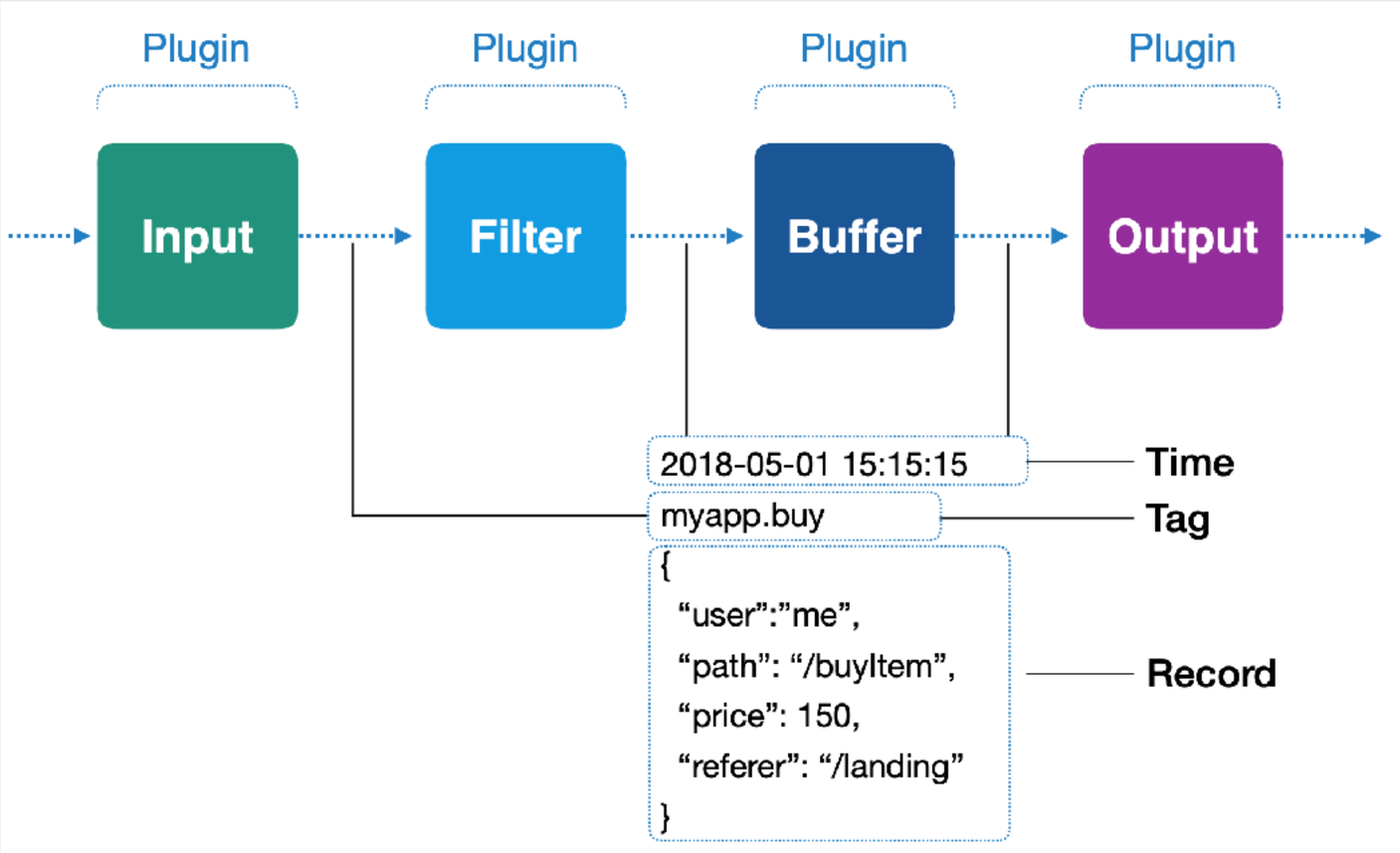
- Nano-second unit
- from logs (event-time)

Tag

- for event routing
- identify data source

Record

```
{ -- json object
   "str_field" : "hello",
   "num_field" : 100,
   "bool_field" : true
   "array_field" : [ "item1", "item2" ]
 }
```



환경 구성 및 설치 – 설치 전

Setup NTP

초기 장비 입수 시에 설정되지 않은 경우도 있으며, 장비 간에 시간 차이에 따라 지표에 영향이 있을 수 있으므로 반드시 사전에 확인이 필요합니다. 주로 chrony, ntpd 등의 NTP daemon 을 활용하고, AWS-hosted NTP server 도 참고하면 좋겠습니다.

환경 구성 및 설치 – ulimit

Maximum Number of File Descriptor

초기 설치시에 systemctl 과 함께 설정해 두지 않으면 서버 재시작이 필요하므로 반드시 장비 인수 시점에 설정확인이 필요합니다. 터미널 및 세션 수준에서의 설정은 /etc/security/limits.conf 파일을 변경해야 하고, 시스템으로 기동되는 설정은 LimitNOFILE 값을 굳이 설정해 주어야만 합니다. 이는 systemctl 을 통한 데몬 프로세스의 기동이 systemd 를 통해 수행되므로 시스템 기동 시에 설정받은 값을 세션 혹은 유저 수준에서 변경이 불가합니다.

```
$ ulimit -n
65535


$ cat /etc/security/limits.conf
root soft nofile 65536
root hard nofile 65536
* soft nofile 65536
* hard nofile 65536
```

```
[Unit]
...
[Service]
User=psyoblade
...
LimitNOFILE=65536
LimitNPROC=65536
...
[Install]
...
```

https://fredrikaverpil.github.io/2016/04/27/systemd-and-resource-limits/

환경 구성 및 설치 – network

Optimize the Network Kernel Parameters

"How Netflix Tunes EC2 Instances for Performance" 에서 소개된 내용으로 워크로드가 심한 환경에서의 Fluentd 의 경우 네트워크 설정까지 신경써 주는 편이 좋다고 합니다. (sysctl -p 로 적용)

```
$ cat /etc/sysctl.conf
net.core.somaxconn = 1024
net.core.netdev_max_backlog = 5000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_wmem = 4096 12582912 16777216
net.ipv4.tcp_rmem = 4096 12582912 16777216
net.ipv4.tcp_max_syn_backlog = 8096
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
```

https://docs.fluentd.org/installation/before-install

환경 구성 및 설치 – 설치

환경 구성 및 설치 – install

Install from Apt Repository

해당 설치 사이트에 추가적인 버전의 설치 가이드가 존재하며 개별 장비에 설치 시에 참고하시면 됩니다.

본 강의에서는 Docker 를 통한 방식으로 기동될 예정입니다.

```
// RPM, Deb or DMG
$ sudo vi /etc/td-agent/td-agent.conf

// Gem
$ sudo fluentd --setup /etc/fluent
$ sudo vi /etc/fluent/fluent.conf

// Docker
docker run -ti -v `pwd`/etc:/fluentd/etc fluentd
export FLUENT_CONF=/etc/td-agent/td-agent.conf
```

https://docs.fluentd.org/v/0.12/articles/install-by-deb

환경 구성 및 설치 – launch

Launch Daemon

/etc/init.d/td-agent 스크립트를 통해 서비스를 재시작 혹은 종료할 수 있습니다

```
$ sudo /etc/init.d/td-agent restart
$ sudo /etc/init.d/td-agent status
td-agent (pid  21678) is running...

$ sudo /etc/init.d/td-agent start
$ sudo /etc/init.d/td-agent stop
$ sudo /etc/init.d/td-agent restart
$ sudo /etc/init.d/td-agent status

# location of configuration file : /etc/td-agent/td-agent.conf
```

https://docs.fluentd.org/v/0.12/articles/install-by-deb

환경 구성 및 설치 – dockerhub.com

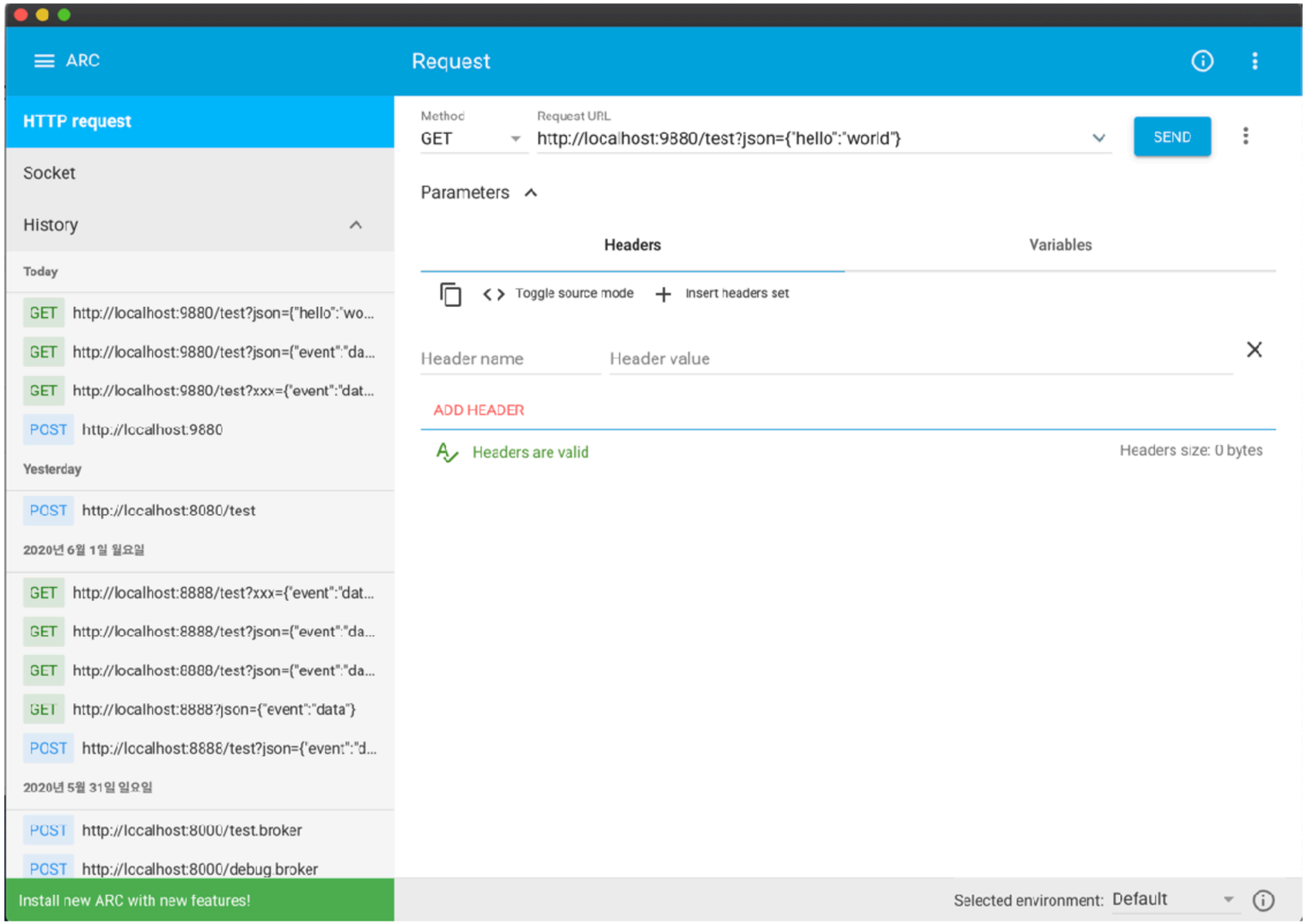
psyoblade/data-engineer-intermediate-day2-fluentd

```
ARG BASE_CONTAINER=fluent/fluentd:v1.11-debian-1
FROM $BASE_CONTAINER
LABEL maintainer="park.suhyuk@gmail.com"
USER root

RUN buildDeps="sudo make gcc g++ libc-dev" \
    && apt-get update \
    && gem install fluent-plugin-route \
    ....  // 기본 패키지에 포함되지 않은 elasticsearch, route, kafka, mongo, webhdfs, multiprocess 를 다 포함한 이미지 입니다.
    && rm -rf /tmp/* /var/tmp/* /usr/lib/ruby/gems/*/cache/*.gem

COPY fluent.conf /fluentd/etc/
COPY entrypoint.sh /bin/
EXPOSE 9880
```

https://hub.docker.com/repository/docker/psyoblade/data-engineer-intermediate-day2-fluentd

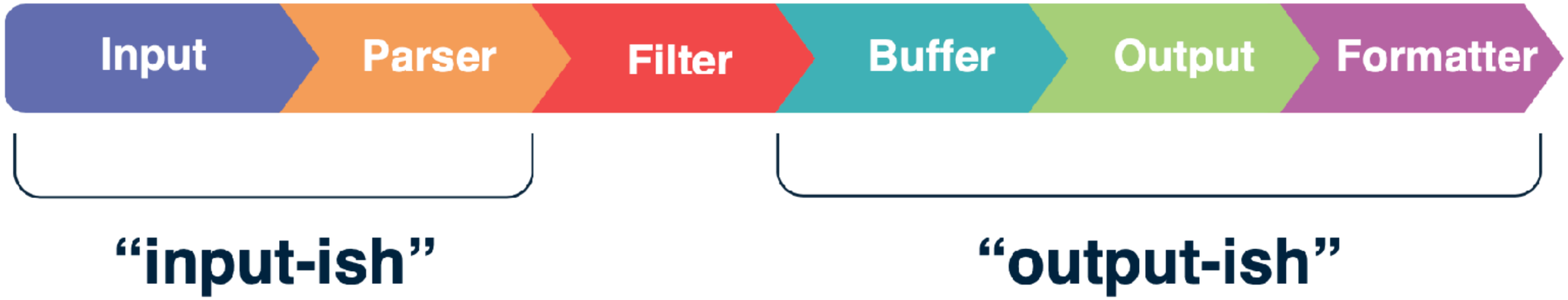
환경 구성 및 설치 – 웹 클라이언트

Advanced REST Client



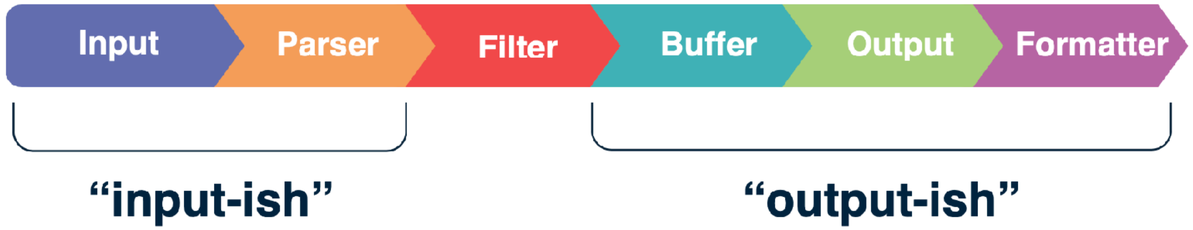
데이터 파이프라인 : input ~ output

Fluentd Data Pipelines



플루언트디 플러그인 – Input

Input Plugins



HTTP 와 로컬 시스템 로그를 TAILING 하는 예제

```ruby
<source>
  @type http
</source>

<source>
  @type tail
  path /fluentd/source/system.log
  pos_file /fluentd/source/system.log.pos
  tag syslog
  <parse>
    @type syslog
  </parse>
</source>
```

Ruby

Input

File tail (in_tail)
Syslog (in_syslog)
HTTP (in_http)
RDBMS (in_sql)

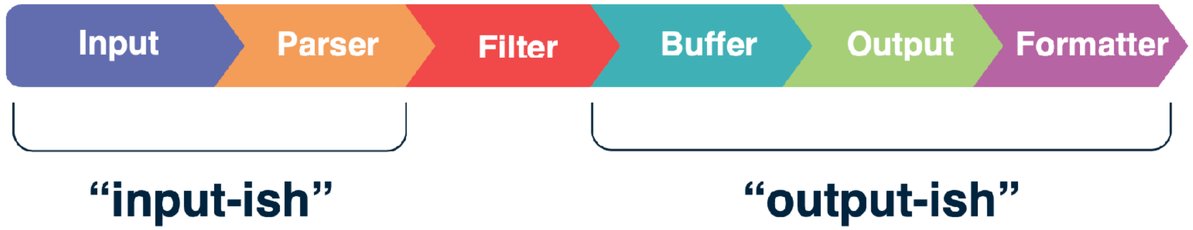
✓ Receive logs

✓ Or pull logs from data sources

✓ non-blocking

플루언트디 플러그인 – Parse

Parse Plugins



logtime 값에 localtime 으로 저장되는 문자열 시간이 수신되는 경우

```
<source>
    @type http
    port 8080
    <parse>
        @type json
        time_type float
        time_key logtime
        types column1:integer,column2:string,logtime:time:unixtime
        localtime true
        keep_time_key true
    </parse>
</source>
```

Parser

JSON
Regexp
Apache/Nginx
Syslog
CSV/TSV
etc.

✓ Parse into JSON

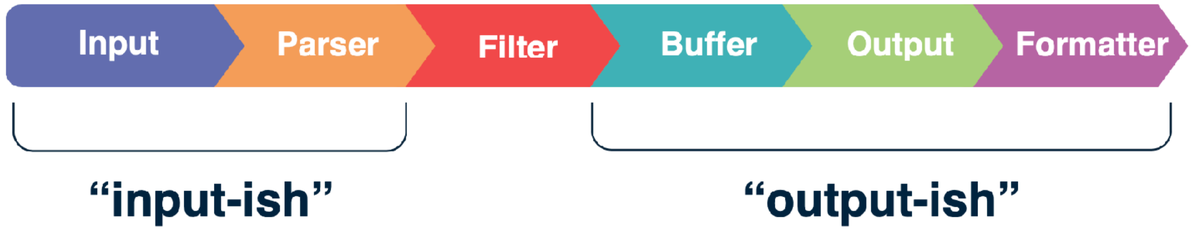
✓ Common formats out of the box

✓ Some inputs plugin depends on

Parser plugin

플루언트디 플러그인 – Filter

Filter Plugins



* 태그 test.cycle 인 경우 login 으로 시작하는 로그를 filter out 합니다

```
<filter test.cycle>
  @type grep
  <exclude>
    key action
    pattern ^login$
  </exclude>
</filter>
```

Filter

grep
record_transformer
suppress

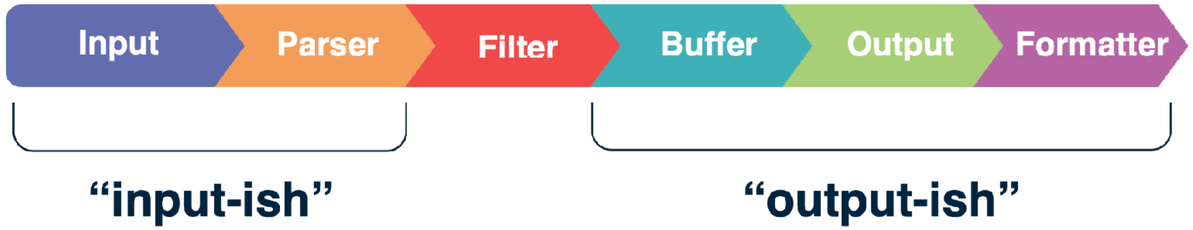
- ✓ Filter / Mutate record
- ✓ Record level and Stream level
- ✓ v0.12 and above

* 태그 myapp.access 인 경우 루비 API 를 통해 현재 호스트 명을 추가할 수 있습니다

```
<filter myapp.access>
  @type record_transformer
  <record>
    host_param "#{Socket.gethostname}"
  </record>
</filter>
```

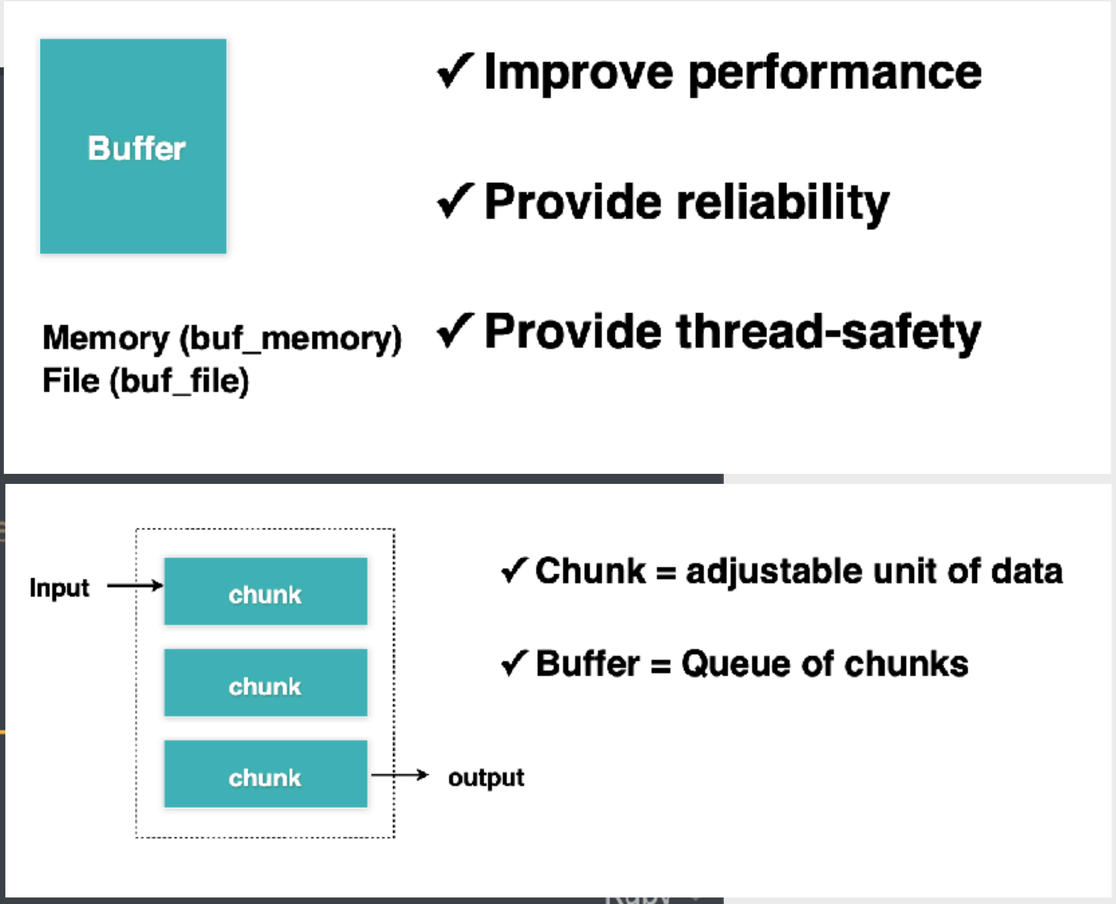
플루언트디 플러그인 – Buffer

Buffer Plugins

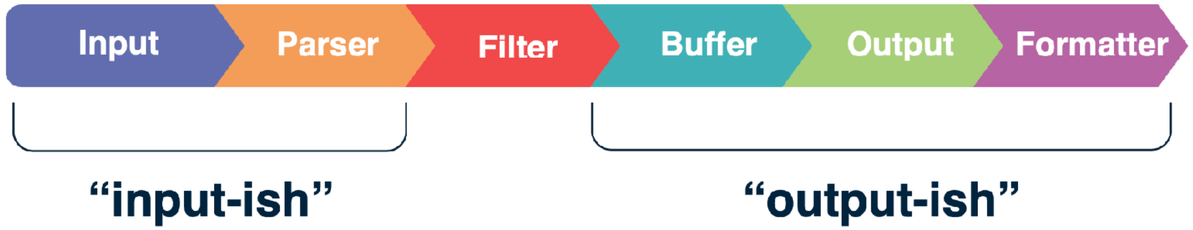


* 별도로 청크 키를 지정하지 않으면 하나의 청크에 저장합니다.

```
<match tag.**>
  # ...
  <buffer>
    # ...
  </buffer>
</match>

# No chunk keys: All events will be appended into the

11:59:30 web.access {"key1":"yay","key2":100}  --|
                                                 |
12:00:01 web.access {"key1":"foo","key2":200}  --|---
                                                 |
12:00:25 ssh.login  {"key1":"yay","key2":100}  --|
```



플루언트디 플러그인 - Buffer



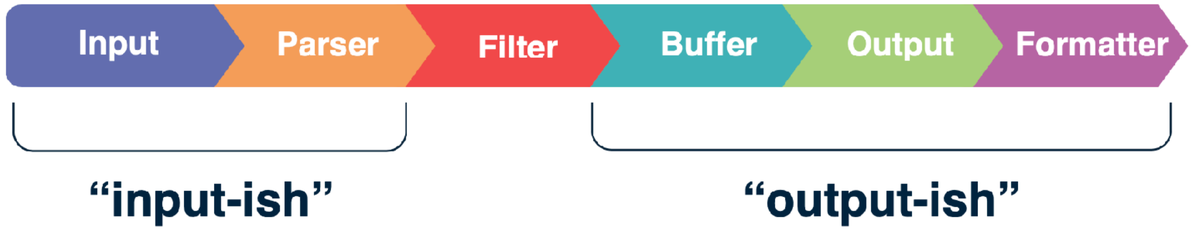
Flush-time w/ timekey

* 키 지정에 따른 플러시 타임

```
# 타임키와 지연시간에 따른 플러시 타임은 아래와 같습니다.
timekey: 3600
 ---------------------------------------------------------
 time range for chunk | timekey_wait | actual flush time
  12:00:00 - 12:59:59 |           0s |          13:00:00
  12:00:00 - 12:59:59 |      60s (1m) |          13:01:00
  12:00:00 - 12:59:59 |    600s (10m) |          13:10:00
```

플루언트디 플러그인 - Buffer

Partitioned by tag, time

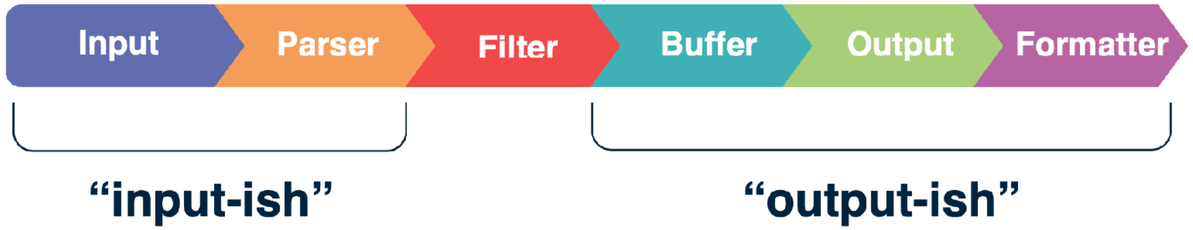


* 일반적으로 많이 적용되는 방식이며 시간과 도메인 값을 기준으로 파티셔닝이 가능합니다

```ruby
# <buffer tag,time>

11:58:01 ssh.login  {"key1":"yay","key2":100}  ------> CHUNK_A

11:59:13 web.access {"key1":"yay","key2":100}  --|
                                                |---> CHUNK_B
11:59:30 web.access {"key1":"yay","key2":100}  --|

12:00:01 web.access {"key1":"foo","key2":200}  ------> CHUNK_C

12:00:25 ssh.login  {"key1":"yay","key2":100}  ------> CHUNK_D
```

플루언트디 플러그인 - Format

Formatter Plugins



* 표준 출력으로 입력받은 데이터를 JSON 형식으로 출력합니다.

```
<match test>
    @type stdout
    <format>
        @type json
        time_format %Y-%m-%d %H:%M:%S.%L
        timezone +09:00
    </format>
</match>

# record example - https://www.unixtimestamp.com/index.php
{ "column1":"1", "column2":"hello-world", "logtime": 1593379470 }
```

```
fluentd    | 2020-08-08 14:26:37 +0000 [info]: #0 starting fluentd worker pid=16 ppid=6 worker=0
fluentd    | 2020-08-08 14:26:37 +0000 [info]: #0 fluentd worker is now running worker=0
fluentd    | {"column1":1,"column2":"hello-world","logtime":1593379470,"filtered_logtime":"2020-06-28 21:24:30"}
```

Formatter

JSON
CSV/TSV
“single value”
msgpack

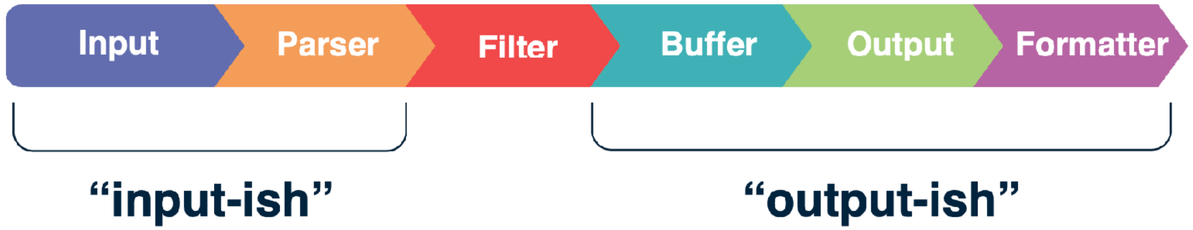
✓ Format output

✓ Convert json into other format

✓ Some plugins depends on

Formatter plugins

Output Plugins




```
# chunk_key: tag
# ${tag} will be replaced with actual tag string
<match log.*>
  @type file
  path  /data/${tag}/access.log  #=> "/data/log.map/access.log"
  <buffer tag>
    # ...
  </buffer>
</match>
```

```
# chunk_key: tag and time
# %Y, %m, %d, %H, %M, %S: strptime placeholder are available when "time" chunk key specified
<match log.*>
  @type file
  path  /data/${tag[1]}/access.%Y-%m-%d.%H%M.log #=> "/data/map/access.2017-02-28.20:48.log"
  <buffer tag,time>
    timekey 1m
  </buffer>
</match>
```

Ruby

Output

File (out_file)
Amazon S3 (out_s3)
Kafka (out_kafka)

✓Write to external systems

✓Buffered & Non-buffered

✓300+ plugins

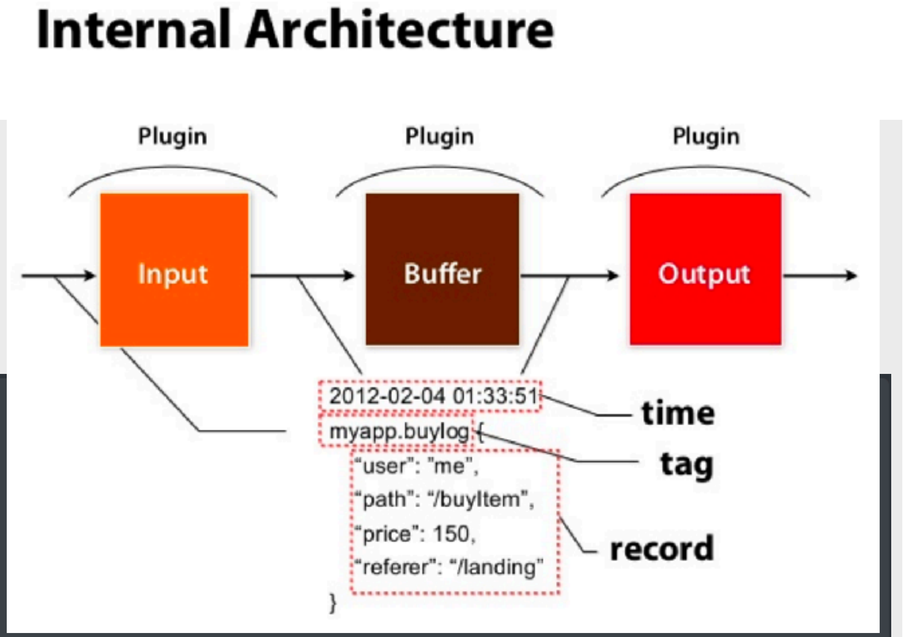
코어 지시자 – core directive

코어 지시자 – 입력

source

입력 데이터 소스를 결정하는 지시자

* 이벤트 데이터는 Time, Tag, Record 3가지 파트로 구성됩니다
  - time : 이벤트 발생 시간 (나노 초 정밀도)
  - tag : 이벤트가 어디로부터 오는 지에 대한 정보
    (이 태그 값에 따라 로그의 필터, 라우팅과 같은 플러그인을 선택합니다)
  - record : 실제 로그 컨텐츠 (JSON 포맷)
* http 는 HTTP endpoint 로, forward 는 TCP endpoint 로 동작합니다
* @type 지시자는 필수이며 plugin 을 지정합니다
* tag 값을 데이터베이스, 테이블 혹은 킷 값으로 정해서 출력 위치를 결정하는 역할로 주로 사용되며 lower alpha-numeric 값을 사용하는 것을 추천합니다
  → ^[a-z0-9_]+$




```
<source>
  @type http
</source>

<source>
  @type tail
  path /fluentd/source/system.log
  pos_file /fluentd/source/system.log.pos
  tag syslog
  <parse>
    @type syslog
  </parse>
</source>
```

Ruby

코어 지시자 – 출력

match

매칭 되는 tag 를 가진 이벤트를 찾아서 다른 시스템에 이벤트를 출력하는 지시자로 사용되며 "output plugins" 라고 부릅니다

* <match ${tag}> ... </match> 와 같이 매칭되는 tag 를 @type 에 출력하게 됩니다

* 아래의 경우 2개의 데이터 소스 가운데 tag 가 myapp.access 이기만 하면 해당 out_file 플러그인을 통해 저장하게 됩니다.

```
<source> # Receive events from 24224/tcp
  @type forward
  port 24224
</source>
<source> # http://this.host:9880/myapp.access?json={"event":"data"}
  @type http
  port 9880
</source>
<match myapp.access> # Match events tagged with "myapp.access" and store them to /var/log/fluent/access.%Y-%m-%d
  @type file
  path /var/log/fluent/access
</match>
```

코어 지시자 – 필터

filter

이벤트 처리 파이프라인의 방향을 결정합니다. 작성 순서대로 필터를 적용할 수 있고 추출하는 grep 과 컬럼 추가를 위한 record_transformer 필터가 존재하며, 필요에 따라 hostname 을 추가할 수 있는 루비 API 를 이용한 함수도 존재합니다

```
<source> # http://this.host:9880/myapp.access?json={"event":"data"}
  @type http
  port 9880
</source>
<filter myapp.access>
  @type record_transformer
  <record>
    host_param "#{Socket.gethostname}" # 루비 API 를 통해 호스트 이름을 이벤트 메시지에 추가합니다
  </record>
</filter>
<match myapp.access>
  @type file
  path /var/log/fluent/access
</match>
```

코어 지시자 – 필터

filter w/ exclude

정규식을 이용하여 완전일치 logout 에 대해서는 제외하고 매칭하게 합니다

```
$> cat etc/fluent.conf
<source ... />
<filter test>
      <exclude>
            @type grep
            key action
            pattern ^logout$
      </exclude>
</filter>
<match ... />

$> curl -i -X POST -d 'json={"action":"login","user":2}' http://localhost:8888/test
$> curl -i -X POST -d 'json={"action":"logout","user":2}' http://localhost:8888/test
```

system

시스템 전체에 적용이 가능한 환경 변수를 말하며 로그 레벨 혹은 프로세스 이름 등을 변경할 수 있으며, 플러글인 디버깅 시에 log_level 설정이 가능하며 전역적으로 혹은 플러그인 수준에서 로그 레벨 변경이 가능합니다.

```
<system>
  log_level error  # equals to -qq option
  without_source  # equals to --without-source option
  emit_error_log_interval  SECONDS  # equals to --emit-error-log-interval SECONDS
  suppress_repeated_stacktrace  true  # equals to --suppress-repeated-stacktrace [VALUE]
  suppress_config_dump
  process_name fluentd1  # 프로세스 이름 변경은 system directive 를 통해서만 가능합니다.
</system>

% ps aux | grep fluentd1
foo     45673   0.4  0.2  2523252  38620 s001  S+    7:04AM   0:00.44 worker:fluentd1
foo     45647   0.0  0.1  2481260  23700 s001  S+    7:04AM   0:00.40 supervisor:fluentd1
```

코어 지시자 – 라벨

label

내부 라우팅을 위해 출력 혹은 필터 등을 묶을 때에 사용되며, 태그 한가지로 모든 파이프라인을 구성하기는 어렵기 때문에 기본 적용과 그룹화 된 임의의 파이프라인을 지정할 수 있습니다. 기본적으로 모든 플루언트디 동작은 위에서 아래로 순차적으로 적용되나, 라벨이 지정된 경우는 예외입니다. (like GOTO)

```
# 1/3 of fluent.conf

<source>
  @type forward
</source>
<source>
  @type tail
  @label @SYSTEM
</source>
```

```
# 2/3 of fluent.conf

<filter access.**>
  @type record_transformer
  <record>
    # ...
  </record>
</filter>

<match **>
  @type elasticsearch
  # ...
</match>
```

```
# 3/3 of fluent.conf

<label @SYSTEM>
  <filter var.log.middleware.**>
    @type grep
    # ...
  </filter>
  <match **>
    @type s3
    # ...
  </match>
</label>
```

코어 지시자 – 와일드카드 매치

wildcard match

```
# match single tag
<match log.*>
  ...
</match>

# log.* → log.info 와 같이 하나의 태그를 매칭
# log 혹은 log.info.detail 는 매칭하지 않습니다
```

```
# match multi tags
<match log.**>
  ...
</match>

# 0 ~ N 개의 태그 파트를 모두 매칭합니다
# log, log.info, log.info.detail 모두 매칭 됩니다
```

```
# selective tags
<match log.{warn, error}>
  ...
</match>

# 지정된 하위 태그만 매칭 합니다.
# log.warn, log.error 는 되지만 log.info 는 안 됩니다
```

```
# match embedded ruby expression
<match "app.#{ENV['FLUENTD_TAG']}">
  @type stdout
</match>

# FLUENT_TAG 값이 debug 라면 app.debug 태그에 매치됩니다
```

```
# match space delimited tags
<match info warn>
<match info.** warn.**>


# 공백으로 구분된 하나 이상의 태그도 매칭 됩니다
```

코어 지시자 – 순서 및 데이터 유형

order of match & data types

기본적으로 Config 파일에 노출된 순서대로 적용됩니다. (위 → 아래)

```
# ** matches all tags. Bad :(
<match **>
  @type blackhole_plugin
</match>

<match myapp.access>
  @type file
  path /var/log/fluent/access
</match>


# 즉 위의 myapp.access 는 영원히 도달하지 않습니다
```

string : 인용 부호가 없는 경우, 인용부호 (' 혹은 ")에 의해 감싼 경우 모두 문자열로 인식 됩니다
integer : 정수
float : 실수
size : <INTEGER> k, K → kilobytes 와 같이 m, g, t 값은 mega, giga, terabytes 를 말합니다
time : <INTEGER>s → seconds 와 같이 m, h, d 는 minutes, hours, days 를 말합니다
array : JSON array 타입 표현식이며 ["key1", "key2"] 와 key1, key 는 같은 표현입니다
hash : { "key1" : "value1", "key2" : "value2" } 는 key1:value1, key2:value2 와 같습니다.

코어 지시자 – 라우팅 예제 (1/2)

Examples of event routing

```
# 예제 1. 입력 -> 필터 -> 출력
<source>
  @type forward
</source>

<filter app.**>
  @type record_transformer
  <record>
    hostname "#{Socket.gethostname}"
  </record>
</filter>

<match app.**>
  @type file
  # ...
</match>
```

```
# 예제 2. 다중 입력 -> 필터 -> 출력
<source>
  @type forward
</source>
<source>
  @type tail
  tag system.logs
</source>
<filter app.**>
  @type record_transformer
  <record>
    hostname "#{Socket.gethostname}"
  </record>
</filter>
<match {app.**,system.logs}>
  @type file
</match>
```

```
# 예제 3. 입력 -> 필터 -> 출력 + 라벨
<source>
  @type forward
</source>
<source>
  @type http
  @label @METRICS
</source>
<filter app.**>
  @type record_transformer
</filter>
<label @METRICS>
  <match **>
    @type elasticsearch
 </match>
</label>
```

코어 지시자 – 라우팅 예제 (2/2)

Examples of event routing

```
# 예제 4. 태그를 이용한 라우팅
<match worker.**>
  @type route
  remove_tag_prefix worker
  add_tag_prefix metrics.event
  <route **>
    copy # For fall-through.
    # Without copy, routing is stopped here.
  </route>
  <route **>
    copy
    @label @BACKUP
  </route>
</match>

CONTINUE ...
```

```
... FROM

<match metrics.event.**>
  @type stdout
</match>

<label @BACKUP>
  <match metrics.event.**>
    @type file
    path /var/log/fluent/backup
  </match>
</label>
```

```
# 예제 5. 데이터 값을 이용한 라우팅
<source>
  @type forward
</source>
<match app.**>
  @type rewrite_tag_filter
  <rule>
    key message
    pattern ^\[(\w+)\]
    tag $1.${tag}
  </rule>
</match>
<match alert.app.**>   @type mail
</match>
<match *.app.**>
  @type file
</match>
```

코어 지시자 – 임포트

@include

너무 파일이 크거나 공통 파라미터 등을 참조하는 경우 임포트 명령어인 @include 를 통해 추가 할 수 있습니다

```
# absolute path
@include /path/to/config.conf

# if using a relative path, the directive will use
# the dirname of this config file to expand the
path
@include extra.conf

# glob match pattern
@include config.d/*.conf

# http
@include http://example.com/fluent.conf
```

```
# config file


<match pattern>
  @type forward
  # ...
  <buffer>
    @type file
    path /path/to/buffer/forward
    @include /path/to/out_buf_params.conf
  </buffer>
</match>
```

```
# /path/to/out_buf_params.conf


flush_interval    5s
total_limit_size  100m
chunk_limit_size  1m
```

코어 지시자 – 공통 파라미터

@type, @id, @log_level

```
# 플러그인 유형 설정 (필수)

<source>
  @type my_plugin_type
</source>

<filter>
  @type my_filter
</filter>
```

```
# 플러그인의 고유 구분자

<match>
  @type file
  @id   service_www_accesslog
  path  /path/to/my/access.log
</match>


# root_dir, workers 등을 전역적으로 사용하기 위해서는 반드시 명시되어야 하므로, 주로 {type}_{plugin} 형식으로 사용합니다
```

```
# 전역 혹은 플러그인 수준에서 로그 레벨을 지정

<system>
  log_level info
</system>

<source>
 @log_level debug
  # show debug log only for this plugin
</source>
```

코어 지시자 – 데이터 해석 (1/2)

parse

<source>, <match> 혹은 <filter> 섹션에 추가될 수 있으며 원본 데이터를 읽는 방법을 정의하는데, 대개 source 섹션에 지정하며 가장 중요한 이벤트 타임을 결정하는 섹션이며, apache2, regexp, csv, json 등이 있습니다. 대부분의 설정은 기본 값이 있어 그대로 사용할 수 있습니다.

```
# 플러그인 유형 설정 (필수)

<parse>
    @type json
    time_type string
    time_format %d/%b/%Y:%H:%M:%S %z
</parse>

# record example
{"key":"value","time":"28/Feb/2013:12:00:00 +0900"}
```

```
# types : 데이터 유형을 명시합니다
* string : to_s 메소드를 통해 변환합니다
* bool : 문자열 "true", "yes" 및 "1" 값을 true 로 외에는 false 로 인식합니다
* integer : int 는 동작하지 않으며, to_i 메소드를 통해 변환합니다
* float : to_f 메소드를 통해 변환하며 문자열 "7.45" 을 7.45 로 변환합니다
* time : 시간을 나타내는 문자열을 아래와 같이 인식합니다
  - date:time:%d/%b/%Y:%H:%M:%S %z    # time_format 에 의해 출력된 문자열
  - date:time:unixtime                # Epoch 이후의 초
  - date:time:float                   # Epoch 이후의 초 + Nano 초

# 시간 포맷 레퍼런스
  - https://docs.ruby-lang.org/en/2.4.0/Time.html#method-i-strftime
  - https://docs.ruby-lang.org/en/2.4.0/Time.html#method-c-strptime
```

코어 지시자 – 데이터 해석 (2/2)

parse

```
# 입력 시간을 localtime 으로 변환하여 저장해야 할 필요가 있습니다.
{ "column1":"1", "column2":"hello-world", "logtime": 1593379470 }
+ "filtered_logtime":"2020-06-28 21:24:30"
```

```
# 데이터 유형 설정 및 입력
<source>
  @type http
  port 8080
  <parse>
    @type json
    time_type float          # 입력 시간의 데이터 유형
    time_key logtime         # 이벤트 타임의 기준을 정하는 가장 중요한 필드
    types column1:integer,column2:string,logtime:time:unixtime
    localtime true           # 시스템 로컬 타임으로 변환
    keep_time_key true       # 시간 필드를 메시지에 유지
  </parse>
</source>
```

```
# 입력 데이터 time 값을 record_transformer 을 통해 포맷팅 합니다
<filter test>
  @type record_transformer
  enable_ruby
  <record>
    filtered_logtime ${Time.at(time).strftime('%Y-%m-%d %H:%M:%S')}
  </record>
</filter>
<match test>
  @type stdout
  <format>
    @type json                                  # 출력 데이터 포맷
    time_format %Y-%m-%d %H:%M:%S  # 출력 문자열 포맷
    timezone +09:00                             # 타임존을 설정
  </format>
</match>
```

https://docs.fluentd.org/configuration/parse-section

코어 지시자 – 버퍼를 통한 출력 (1/4)

buffer

반드시 <match> 섹션 내에 존재해야 하며 버퍼링을 지원하는 플러그인에 버퍼링을 지정, <buffer> 섹션은 한 번만 등장 하며, 서드파티 플러그인도 존재하지만 코어 엔진에서 제공하는 버퍼는 file 과 memory (default) 가 있으며 file 이 안정적입니다. 버퍼링을 통해 결과 파일 저장 시에 디렉토리 혹은 파일 별 파티셔닝을 가능하게 하며 이를 청크라고 표현합니다

```
# 아무런 옵션을 지정하지 않은 버퍼 (심플)

<match tag.**>
  # ...
  <buffer>
    # ...
  </buffer>
</match>
```

```
# 청크 키를 지정하지 않은 경우 하나의 청크에 모두 저장됩니다

11:59:30 web.access {"key1":"yay","key2":100}   --|
                                                  |
12:00:01 web.access {"key1":"foo","key2":200}  --|---> CHUNK_A
                                                  |
12:00:25 ssh.login  {"key1":"yay","key2":100}     --|
```

코어 지시자 – 버퍼를 통한 출력 (2/4)

buffer by tag or column

```
# 매치로 부터 전달된 태그 값을 이용하여 임의의 태그 기준으로 청킹
<match tag.**>
 <buffer tag>
  # ...
 </buffer>
</match>
```

```
# 태그 값을 기준으로 청크 저장이 가능합니다

11:59:30 web.access {"key1":"yay","key2":100}   --|
                                                  |---> CHUNK_A
12:00:01 web.access {"key1":"foo","key2":200}  --|

12:00:25 ssh.login  {"key1":"yay","key2":100}   --------> CHUNK_B
```

```
# 임의의 컬럼 값을 기준으로 청킹
<match tag.**>
  # ...
  <buffer key1>
    # ...
  </buffer>
</match>
```

```
# 내가 선택한 컬럼을 기준으로 청킹도 가능합니다

11:59:30 web.access {"key1":"yay","key2":100}   --|---> CHUNK_A
                                                  |
12:00:01 web.access {"key1":"foo","key2":200} -) | (--> CHUNK_B
                                                  |
12:00:25 ssh.login  {"key1":"yay","key2":100}     --|
```

코어 지시자 – 버퍼를 통한 출력 (3/4)

buffer by time

```
# 시간 값을 이용하여 청킹
<match tag.**>
 <buffer time>
   timekey      1h          # 시간 단위로 파일 저장
   timekey_wait 5m          # 5분 대기 후 플러시 (지연 로그 수신을 위함)
  </buffer>
</match>
```

```
# 입력된 시간을 기준으로 1시간 단위로 청킹 됩니다

11:59:30 web.access {"key1":"yay","key2":100}  ------> CHUNK_A

12:00:01 web.access {"key1":"foo","key2":200}  --|
                                                 |---> CHUNK_B
12:00:25 ssh.login  {"key1":"yay","key2":100}     --|
```

```
# 타임키와 지연시간에 따른 플러시 타임은 아래와 같습니다.
timekey: 3600
 ------------------------------------------------------
 time range for chunk | timekey_wait | actual flush time
  12:00:00 - 12:59:59 |           0s |          13:00:00
  12:00:00 - 12:59:59 |      60s (1m) |          13:01:00
  12:00:00 - 12:59:59 |    600s (10m) |          13:10:00
```

코어 지시자 – 버퍼를 통한 출력 (4/4)

buffer by composite columns

```
# 시간 값을 이용하여 청킹
<match tag.**>
 <buffer tag,time>
   timekey      1h          # 시간 단위로 파일 저장
   timekey_wait 5m          # 5분 대기 후 플러시 (지연 로그 수신을 위함)
  </buffer>
</match>

# 태그의 2번째 값을 아래와 같이 액세스 할 수 있습니다. zero 베이스
<match log.*>
  @type file
  path  /data/${tag[1]}/access.%Y-%m-%d.%H%M.log
  <buffer tag,time>
    timekey 1m
  </buffer>
</match>
```

```
# 입력된 시간을 기준으로 1시간 단위로 청킹 됩니다

# <buffer tag,time>

11:58:01 ssh.login  {"key1":"yay","key2":100}      ------> CHUNK_A

11:59:13 web.access {"key1":"yay","key2":100}   --|
                                                    |---> CHUNK_B
11:59:30 web.access {"key1":"yay","key2":100}  --|

12:00:01 web.access {"key1":"foo","key2":200} -----> CHUNK_C

12:00:25 ssh.login  {"key1":"yay","key2":100}   ------> CHUNK_D
```

플러그인 - plugin

코어 플러그인 - 파일 입력

in_tail

기본 제공도는 텍스트 파일을 테일링 하면서 계속 읽는 방식이며 tail -F 명령어와 유사합니다.

```
<source>
    @type tail
    path /fluentd/source/*.log
    exclude_path /fluentd/source/system.log
    refresh_interval 5
    limit_recently_modified 1d
    pos_file /fluentd/source/access.log.pos
    read_from_head true
    rotate_wait 5
    <parse>
        @type apache2
    </parse>
    open_on_every_update true
    emit_unmatched_lines true
</source>
```

path : 읽어야 할 파일의 경로
exclude_path : 와일드카드 문자를 통한 조회 시에 제외할 경로
refresh_interval (default: 60s) : 얼마나 자주 파일 변경을 체크할 것인지
limit_recently_modified (default: nil) : 지정된 기간 내에 변경된 파일만 수집
pos_file : 마지막으로 읽었던 위치에 대한 파일 별 체크 포인트
read_from_head (default: false) : 초기 로딩 시에 대상 파일들의 처음부터 읽음
rotate_wait (default: 5s) : 로테이팅이 일어나도 이전 파일의 플러시를 대기하는 시간
parse : format 은 deprecated 되어 parse 옵션을 사용합니다
open_on_every_update : 매번 파일을 열고 닫음, logrotate 가 자주 일어나는 경우
emit_unmatched_lines (default: false) : { "unmatched_line" : incoming line } 출력

https://docs.fluentd.org/input/tail

코어 플러그인 - 파일 출력

out_file

out_file 플러그인은 TimeSliced Output plugin 으로 이벤트들을 파일로 저장합니다. 기본 값은 일별 대략 00:10 시간 기준으로 파일들을 생성하는데 이는 결국 일별 자정이 넘어서야 파일을 생성한다는 말이므로, 즉시 파일이 생성되지 않습니다

```
<match weblog.info>
    @type file
    @log_level info
    path /fluentd/target/${tag}/%Y%m%d/access.%Y%m%d.%H
    <format>
        @type json
    </format>
    add_path_suffix true
    path_suffix .log
    <buffer>
        ...
    </buffer>
</match>
```

@log_level : fatal, error, warn, info, debug, trace 수준으로 로그를 출력
path : 써야 할 파일의 경로
append (default: false) : append 가 안되면 에 file.20200809.log_0 형식으로 저장
<format> : format 옵션은 deprecated 되어 <format> 을 사용합니다.
add_path_suffix (default: true) : 경로를 포함하여 저장합니다
path_suffix (default: ".log") : 확장자를 지정합니다.

- time 기준으로 포맷 설정은 이전 parse 페이지를 참고하세요
- buffer 출력의 상세한 설정은 다음 buffer 페이지를 참고하세요

https://docs.fluentd.org/output/file

코어 플러그인 - 버퍼 출력

out_buffer

버퍼에 사용되는 Placeholder 적용을 위해서는 매개변수로 키가 전달되어야 합니다. 그리고 매개변수 키 예제 (tag 와 time 은 킷 값이 아니라 예약어 필드입니다)

```
<buffer tag, time, key1>
    @type file
    @log_level info
    add_path_suffix true
    path_suffix .log
    path /fluentd/target/${tag}/%Y%m%d/access.%Y%m%d.%H
    <buffer time, tag>
        timekey 1h
        timekey_use_utc false
        timekey_wait 10s
        timekey_zone +0900
        flush_mode immediate
        flush_thread_count 8
 </buffer>
```

```
# Time parameters
timekey (required) : 지정한 시간을 기준으로 청크 파일이 생성됩니다
timekey_wait (default: 600) : 지정 시간 대기 후에 청크 파일을 플러시 합니다
timekey_use_utc (default: false) : 시간 포맷 지정에 대해 localtime 사용 유무
timekey_timezone (default: local timezone) : 타임존 (+0900 or Asia/Seoul)

# Buffering parameters
chunk_limit_size (default: 8mb) : 청크의 최대의 크기
total_limit_size (default: 512mb-memory, 64gb-file) : 버퍼 인스턴스 최대 메모리
  (메모리 한계를 넘어서 종료 시에는 저장하지 않은 정보는 모두 사라집니다)
compress (default: text) : 압축 옵션이며 text, gzip 을 선택 가능합니다
```

https://docs.fluentd.org/configuration/buffer-section

코어 플러그인 – 버퍼 출력

out_buffer

Flushing parameters
flush_at_shutdown (default: false) : 종료 시에 플러시 여부를 결정합니다
flush_mode (default: default) : default, lazy, interval, immediate
flush_interval (default: 60s) : interval mode 에서 플러시 주기
flush_thread_count (default: 1) : 동시에 수행 가능한 flush/write 청크 수
flush_thread_interval (default: 1.0) : 다음 플러시 까지의 sleep 시간
flush_thread_burst_interval (default: 1.0) : 대기 플러시 청크시 sleep 시간
overflow_action (default: throw_exception) : 버퍼가 꽉 찼을 때의 동작
  throw_exception : 예외를 던지고, 로그에 오류를 출력
  block : 버퍼에 이벤트를 출력하는 입력 플러그인의 처리를 막습니다
  drop_oldest_chunk : 새로운 청크를 받기 위해 오래된 청크를 삭제합니다.

Retries parameters
retry_timeout (default: 72h) : 플러시에 실패한 청크 재시도 최대 타임아웃
retry_max_times (default: none) : 최대 재시도 횟수
retry_secondary_threshold (default: 0.8) : 세컨더리로 전환하는 실패 비율
retry_type (default: exponential_backoff) : 재시도 전략
  exponential_backoff : 실패 시에 지수적으로 sleep 시간을 늘리는 전략
  periodic : 정해진 주기에 따라 대기
retry_wait (default: 1s) : 재시도 사이의 sleep 시간
retry_exponential_backoff_base (default: 2) : e_b 의 재시도 sleep 시간

https://docs.fluentd.org/configuration/buffer-section

Q&A