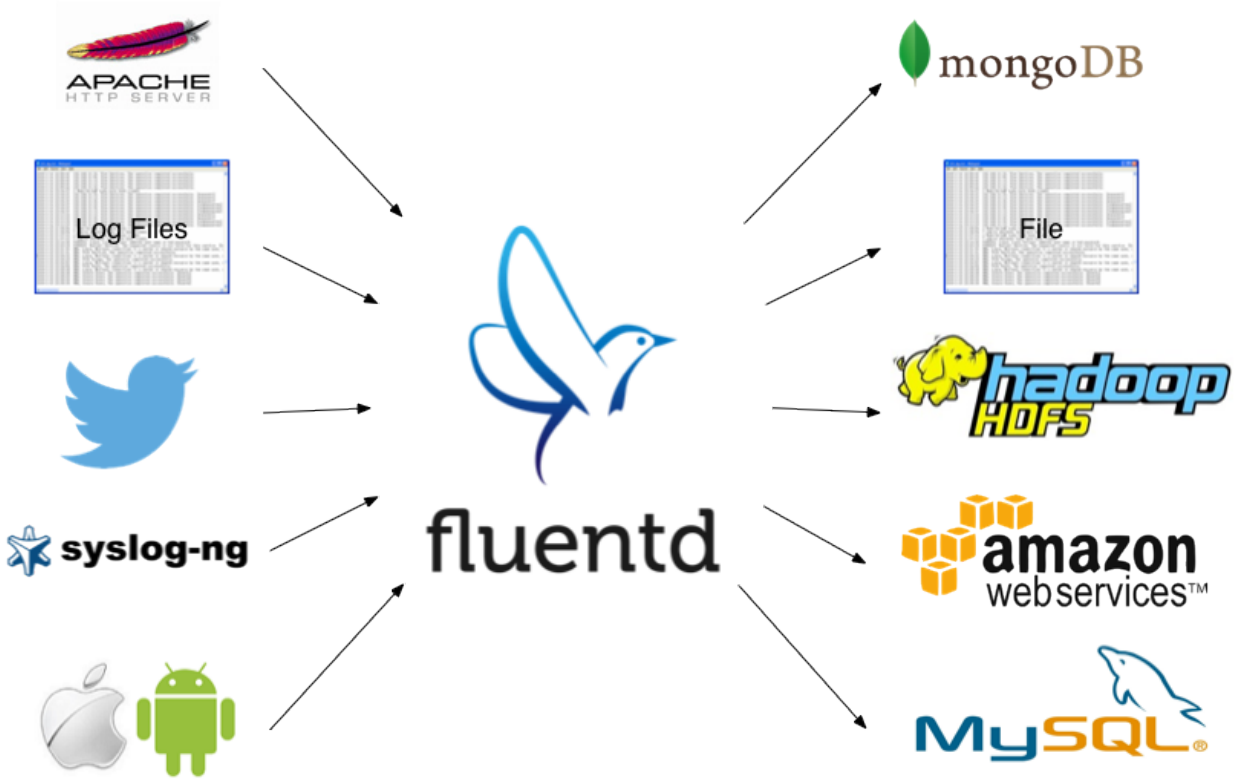
Fluentd User Manual

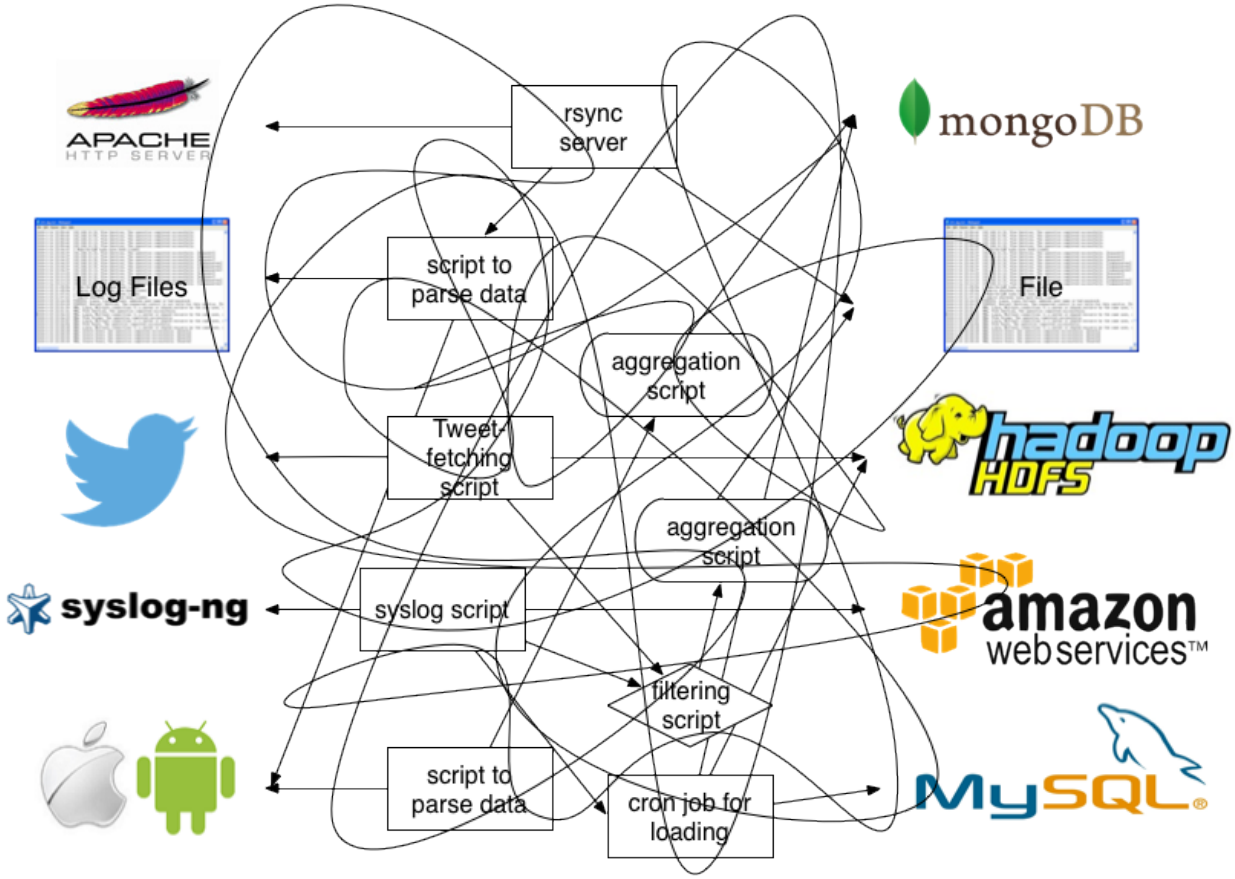
fluentd

Contents

Overview of Fluentd

Fluentd is a fully free and open-source log management tool that **simplifies your data collection and storage pipeline**. It eliminates the need to maintain a set of ad-hoc scripts.


APACHE
HTTP SERVER
Log Files
syslog-ng
mongoDB
File
hadoop
HDFS
amazon
web services™
MySQL
fluentd


APACHE
HTTP SERVER
Log Files
syslog-ng
mongoDB
File
hadoop
HDFS
amazon
web services™
MySQL
rsync
server
script to
parse data
aggregation
script
Tweet-
fetching
script
aggregation
script
syslog script
filtering
script
script to
parse data
cron job for
loading

After Fluentd

Before

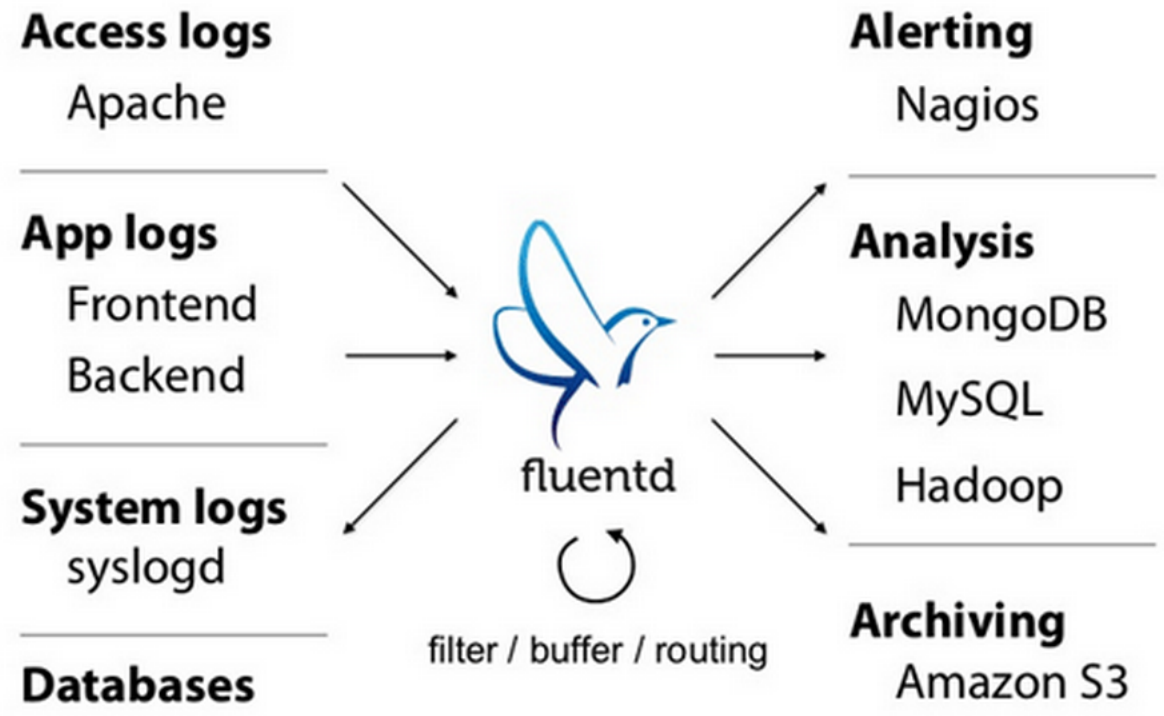
This presentation provides an overview of Fluentd, an introduction to its unique features, and a comparison with other prominent logging solutions.

Now that you have a better understanding of Fluentd, let's give it a spin!

Get Started

Quickstart Guide

Let's get started with **Fluentd**! **Fluentd** is a fully free and fully open-source log collector that instantly enables you to have a '**Log Everything**' architecture with 125+ types of systems.



Fluentd treats logs as JSON, a popular machine-readable format. It is written primarily in C with a thin-Ruby wrapper that gives users flexibility.

Fluentd's performance has been proven in the field: its largest user currently collects logs from **5000+ servers**, 5 TB of daily data, handling 50,000 msgs/sec at peak time.

Step1: Installing Fluentd

Please follow the installation/quickstart guides below that matches your environment.

- Install Fluentd by Homebrew (Mac OS X)
- Install Fluentd by RPM package (Redhat Linux)
- Install Fluentd by Deb package (Ubuntu/Debian Linux)
- Install Fluentd by Ruby Gem

- Install Fluentd by Chef
- Install Fluentd from source

NOTE: Fluentd is currently not supported on Windows. Please see the FAQ for details.

Step2: Use Cases

The articles shown below cover the typical use cases of Fluentd. Please refer to the article(s) that suits your needs.

- Use Cases
  - Data Search like Splunk
  - Data Filtering and Alerting
  - Data Analytics with Treasure Data
  - Data Collection to MongoDB
  - Data Collection to HDFS
  - Data Archiving to Amazon S3
  - Windows Event Collection
- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Happy Users :)
  - Users

Step3: Learn More

The articles shown below will provide detailed information for you to learn more about Fluentd.

- Architecture Overview
- Plugin Overview
  - Input Plugins
  - Output Plugins
  - Buffer Plugins
- High Availability Configuration
- FAQ

Before Installing Fluentd

You MUST set up your environment according to the steps below before installing Fluentd. Failing to do so will be the cause of many unnecessary problems.

Set Up NTP

It's HIGHLY recommended that you set up *ntpd* on the node to prevent invalid timestamps in your logs.

Increase Max # of File Descriptors

Please increase the maximum number of file descriptors. You can check the current number using the `ulimit -n` command.

```
$ ulimit -n
65535
```

If your console shows `1024`, it is insufficient. Please add following lines to your `/etc/security/limits.conf` file and reboot your machine.

```
root soft nofile 65536
root hard nofile 65536
* soft nofile 65536
* hard nofile 65536
```

Optimize Network Kernel Parameters

For high load environments consisting of many Fluentd instances, please add these parameters to your `/etc/sysctl.conf` file. Please either type `sysctl -w` or reboot your node to have the changes take effect. If your environment doesn't have a problem with TCP_WAIT, then these changes are not needed.

```
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240    65535
```

Installing Fluentd Using rpm Package

This article explains how to install the td-agent rpm package, the stable Fluentd distribution package maintained by Treasure Data, Inc.

What is td-agent?

Fluentd is written in Ruby for flexibility, with performance sensitive parts written in C. However, casual users may have difficulty installing and operating a Ruby daemon.

That's why Treasure Data, Inc is providing **the stable distribution of Fluentd**, called `td-agent`. The differences between Fluentd and td-agent can be found here.

Step0: Before Installation

Please follow the Preinstallation Guide to configure your OS properly. This will prevent many unnecessary problems.

Step1: Install from rpm Repository

CentOS and RHEL 5.0+ are currently supported.

Executing install-redhat.sh will automatically install td-agent on your machine. This shell script registers a new rpm repository at `/etc/yum.repos.d/td.repo` and installs the `td-agent` rpm package.

```
$ curl -L http://toolbelt.treasuredata.com/sh/install-redhat.sh | sh
```

NOTE: It's HIGHLY recommended that you set up ntpd on the node to prevent invalid timestamps in your logs.

Step2: Launch Daemon

The `/etc/init.d/td-agent` script is provided to start, stop, or restart the agent.

```
$ /etc/init.d/td-agent start
Starting td-agent: [  OK  ]
$ /etc/init.d/td-agent status
td-agent (pid  21678) is running...
```

The following commands are supported:

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

Please make sure **your configuration file** is located at `/etc/td-agent/td-agent.conf`.

Step3: Post Sample Logs via HTTP

By default, `/etc/td-agent/td-agent.conf` is configured to take logs from HTTP and route them to stdout (`/var/log/td-agent/td-agent.log`). You can post sample log records using the curl command.

```
$ curl -X POST -d 'json={"json":"message"}' http://localhost:8888/debug.test
```

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3

  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Please refer to the resources below for further steps.

- ChangeLog of td-agent
- Chef Cookbook

Installing Fluentd Using deb Package

This article explains how to install the td-agent deb package, the stable Fluentd distribution package maintained by Treasure Data, Inc.

NOTE: "Ubuntu 12.04 LTS / Precise" and "Ubuntu 10.04 LTS / Lucid" are currently supported. If you are interested in Debian, please contact us on the mailing list. If enough people express interest, we may consider supporting it.

What is td-agent?

Fluentd is written in Ruby for flexibility, with performance sensitive parts written in C. However, casual users may have difficulty installing and operating a Ruby daemon.

That's why Treasure Data, Inc is providing **the stable distribution of Fluentd**, called `td-agent`. The differences between Fluentd and td-agent can be found here.

Step0: Before Installation

Please follow the Preinstallation Guide to configure your OS properly. This will prevent many unnecessary problems.

GPG key

The deb package is signed with the Treasure Data GPG key, which can be found here. We recommend importing the GPG key to apt.

```
$ wget -O - http://packages.treasure-data.com/debian/RPM-GPG-KEY-td-agent | sudo apt-key add -
```

Step1 (Ubuntu): Install from Apt Repository

"Ubuntu 12.04 LTS / Precise" and "Ubuntu 10.04 LTS / Lucid" are currently supported.

Ubuntu Precise Executing install-ubuntu-precise.sh will automatically install td-agent on your machine. This shell script registers a new apt repository at `/etc/apt/sources.list.d/treasure-data.list` and installs the `td-agent` deb package.

```
$ curl -L http://toolbelt.treasuredata.com/sh/install-ubuntu-precise.sh | sh
```

Ubuntu Lucid Executing install-ubuntu-lucid.sh will automatically install td-agent on your machine. This shell script registers a new apt repository at `/etc/apt/sources.list.d/treasure-data.list` and installs the `td-agent` deb package.

```
$ curl -L http://toolbelt.treasuredata.com/sh/install-ubuntu-lucid.sh | sh
```

NOTE: It's HIGHLY recommended that you set up ntpd on the node to prevent invalid timestamps in your logs.

Step2: Launch Daemon

The `/etc/init.d/td-agent` script is provided to start, stop, or restart the agent.

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

The following commands are supported:

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

Please make sure **your configuration file** is located at `/etc/td-agent/td-agent.conf`.

Step3: Post Sample Logs via HTTP

By default, `/etc/td-agent/td-agent.conf` is configured to take logs from HTTP and route them to stdout (`/var/log/td-agent/td-agent.log`). You can post sample log records using the curl command.

```
$ curl -X POST -d 'json={"json":"message"}' http://localhost:8888/debug.test
```

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3
  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Please refer to the resources below for further steps.

- ChangeLog of td-agent
- Chef Cookbook

Installing Fluentd Using Ruby Gem

This article explains how to install Fluentd using Ruby gem.

Step0: Before Installation

Please follow the Preinstallation Guide to configure your OS properly. This will prevent many unnecessary problems.

Step1: Install Ruby interpreter

Please install Ruby >= 1.9.2 on your local environment.

Step2: Install Fluentd gem

Fetch and install the `fluentd` Ruby gem using the `gem` command. The official ruby gem page is here.

```
$ gem install fluentd --no-ri --no-rdoc
```

Step3: Run

Run the following commands to confirm that Fluentd was installed successfully:

```
$ fluentd --setup ./fluent
$ fluentd -c ./fluent/fluent.conf -vv &
$ echo '{"json":"message"}' | fluent-cat debug.test
```

The last command sends Fluentd a message '{"json":"message"}' with a "debug.test" tag. If the installation was successful, Fluentd will output the following message:

```
2011-07-10 16:49:50 +0900 debug.test: {"json":"message"}
```

NOTE: It's HIGHLY recommended that you set up ntpd on the node to prevent invalid timestamps in your logs.

NOTE: For large deployments, you must use jemalloc to avoid memory fragmentation. This is already included in the rpm and deb packages.

NOTE: The Fluentd gem doesn't come with /etc/init.d/ scripts. You should use process management tools such as daemontools, runit, supervisord, or upstart.

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File

- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3
  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Installing Fluentd Using Chef

This article explains how to install Fluentd using Chef.

Step0: Before Installation

Please follow the Preinstallation Guide to configure your OS properly. This will prevent many unnecessary problems.

Step1: Import Recipe

The chef recipe to install td-agent can be found here. Please import the recipe, add it to run_list, and upload it to the Chef Server.

Step2: Run chef-client

Please run chef-client to install td-agent across your machines.

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3
  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Installing Fluentd using Homebrew (MacOS X)

This article explains how to install td-agent, the stable Fluentd distribution package maintained by Treasure Data, Inc, on MacOS X.

What is td-agent?

Fluentd is written in Ruby for flexibility, with performance sensitive parts written in C. However, casual users may have difficulty installing and operating a Ruby daemon.

That's why Treasure Data, Inc is providing **the stable distribution of Fluentd**, called `td-agent`. The differences between Fluentd and td-agent can be found here.

For MacOS X, we're using the Homebrew packaging manager to distribute td-agent.

NOTE: If you are using RVM, please disable it. RVM will cause an error where fluentd cannot be loaded from the td-agent formula.

Step1: Install Homebrew

If you haven't installed Homebrew on your Mac yet, please install it using the command below.

```
$ ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"
$ which brew
/usr/local/bin/brew
```

Step2: Install td-agent

Once you have the `brew` command, please use `brew install` to fetch the td-agent formula and install td-agent.

```
$ brew install "http://toolbelt.treasuredata.com/brew/td-agent.rb"
```

You can launch `td-agent` in a foreground process.

```
$ td-agent
2013-04-19 16:55:03 -0700 [info]: starting fluentd-0.10.33
2013-04-19 16:55:03 -0700 [info]: reading config file path="/usr/local/etc/td-agent/td-agent.conf"
```

Alternatively, you can manage the `td-agent` daemon process with `launchctl`. This is not necessary, but it's convenient if you want to continuously run td-agent in the background. Let's register `td-agent` as a built-in daemon and launch it using the `launchctl` command.

```
# Register
$ ln -sfv /usr/local/opt/td-agent/homebrew.mxcl.td-agent.plist ~/Library/LaunchAgents/.

# Start
$ launchctl load ~/Library/LaunchAgents/homebrew.mxcl.td-agent.plist
$ ps aux | grep 'td-agent'
kzk 66680 0.0 0.0 2432768   616 s002 R+ 4:49PM 0:00.00 grep td-agent
kzk 66673 0.0 0.1 2476340 20604  ?? S  4:49PM 0:00.10 /usr/local/Cellar/td-agent/1.1.12/bin/ruby /usr/local
kzk 66672 0.0 0.1 2454904 14624  ?? S  4:49PM 0:00.12 /usr/local/Cellar/td-agent/1.1.12/bin/ruby /usr/local

# Stop
$ launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.td-agent.plist
```

Please make sure **your configuration file** is located at `/usr/local/etc/td-agent/td-agent.conf`. Your plugin directory is at `/usr/local/etc/td-agent/plugin`.

Step3: Post Sample Logs via HTTP

By default, `/usr/local/etc/td-agent/td-agent.conf` is configured to take logs from HTTP and route them to stdout (`/usr/local/var/log/td-agent/td-agent.log`). You can post sample log records using the curl command.

```
$ curl -X POST -d 'json={"json":"message"}' http://localhost:8888/debug.test
$ tail -n 1 /usr/local/var/log/td-agent/td-agent.log
2013-04-19 16:51:47 -0700 debug.test: {"json":"message"}
```

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3
  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Please refer to the resources below for further steps.

- ChangeLog of td-agent
- Chef Cookbook

Installing Fluentd from Source

This article explains how to install Fluentd from source code (git repository). This is useful for developers.

Step1: Install Ruby interpreter

Please install Ruby >= 1.9.2 on your local environment.

Step2: Fetch Source Code

Fetch the source code from github. The official repository is located here.

```
$ git clone https://github.com/fluent/fluentd.git
$ cd fluentd
```

Step3: Build and Install

Build the package with `rake` and install it with `gem`.

```
$ rake build
Successfully built RubyGem
Name: fluentd
Version: xxx
File: fluentd-xxx.gem
$ gem install pkg/fluentd-xxx.gem
```

Step4: Run

Run the following commands to to confirm that Fluentd was installed successfully:

```
$ fluentd --setup ./fluent
$ fluentd -c ./fluent/fluent.conf -vv &
$ echo '{"json":"message"}' | fluent-cat debug.test
```

The last command sends Fluentd a message '{"json":"message"}' with a "debug.test" tag. If the installation was successful, Fluentd will output the following message:

```
2011-07-10 16:49:50 +0900 debug.test: {"json":"message"}
```

NOTE: It's HIGHLY recommended that you set up ntpd on the node to prevent invalid timestamps in your logs.

NOTE: For large deployments, you must use jemalloc to avoid memory fragmentation. This is already included in the rpm and deb packages.

Next Steps

You're now ready to collect your real logs using Fluentd. Please see the following tutorials to learn how to collect your data from various data sources.

- Basic Configuration
  - Config File
- Application Logs
  - Ruby, Java, Python, PHP, Perl, Node.js, Scala
- Examples
  - Store Apache Log into Amazon S3
  - Store Apache Log into MongoDB
  - Data Collection into HDFS

Install Fluentd (td-agent) on Heroku

This article describes how to install Fluentd (td-agent) on Heroku.

Create Your App

Heroku doesn't allow users to install separate processes within a single dyno. You will thus need to setup Fluentd as a separate Heroku application. This will become you central log aggregation server.

Treasure Data provides a boilerplate repository to get you started. Follow the steps below to create Fluentd (td-agent) as a Heroku application.

```
# Clone
$ git clone git://github.com/treasure-data/heroku-td-agent.git
$ cd heroku-td-agent
$ rm -fR .git
$ git init
$ git add .
$ git commit -m 'initial commit'

# Create the app & deploy
$ heroku create --stack cedar
$ git push heroku master

# Modify your conf
$ vi td-agent.conf
$ git commit -a -m 'update config file'

# Deploy
$ git push heroku master
```

Test

Let's confirm that the log aggregation server has been set up correctly. Please send a GET request to the log server, http://td-agent-on-heroku.herokuapp.com, as shown below. This will log the event { "json": "message" } with a debug.sample tag. Note how the JSON-formatted data is passed as a query parameter value.

```
$ curl "http://td-agent-on-heroku.herokuapp.com/debug.sample?json=%7B%22json%22%3A%22message%22%7D"
```

In general, the URL format is

```
http://{YOUR LOG SERVER DOMAIN}/td.{DB_NAME}.{TABLE_NAME}?json={JSON_FORMATTED_DATA}
```

The output will be available on STDOUT.

```
$ heroku logs --tail
```

Installing Fluentd (td-agent) on Elastic Beanstalk

A boilerplate application to install td-agent on your AWS Elastic Beanstalk application is provided in this repository. Please follow the instructions in the README file.

- treasure-data/elastic-beanstalk-td-agent

Fluentd Users

These are the some of Fluentd users.

NOTE: Are you using Fluentd? If so, please share your testimonial with us to help expand the community! :) Please send me a message with your testimonial, name, title, and company's logo!

Backplane, Inc.

When I first started at Backplane, I wanted to get us to start logging everything and aggregating it in JSON, and I wanted that system up and running fast. Fluentd got me there in less than a day. I was skeptical of reliability at first, but it takes so little time to setup, there's no reason not to try it.

It hasn't crashed once yet, works exactly as I need it to thanks to the huge plugin library, and likely saved us a ton of time. As an added bonus: the user list for Fluentd was also way more responsive when I ran into initial setup questions than those of any competitors. If you want to start logging your data fast, Fluentd is a great way to go. - Eli Finkelshteyn, Lead Data Scientist

ContextLogic, Inc.

At wish.com, we started using Fluentd from early on to collect user event data. Its reliability and ease of use from application code (Python for us) make it easy for us to keep track of the key metrics. - Danny Zhang, co-founder

CyberAgent, Inc.

CyberAgent (TYO:4751) is a Tokyo based company that specializes in internet business. We are expanding our businesses in the field of Social Network Services and Web advertising.

Though collecting and analyzing log data is quite important in our services, we had no choice but to conduct in conventional methods such as using Hadoop to analyze log data or rsync and script.

That is why we began to use Fluentd. Fluentd made it possible to collect existing log with significantly lower resources, while it is not affecting the analyzing process. We introduced this system successfully and safely.

Fluentd helps us collect and analyze an app log and midleware log efficiently. And as a result, it improves our daily operation and services. - Tomomitsu Tsuda, General Manager

DeNA Co., Ltd.

DeNA Co., Ltd. (pronounced “D-N-A”) is a global Internet company (TSE:2432) focused on social games and e-commerce. DeNA operates the Mobage platform, which is one of the most successful gaming service in Japan with more than 49 million registered players as of March, 2013.

We use Fluentd for log monitoring, visualization of metrics, and realtime log collection. It can deal with the log of our large-scaled servers. We are very happy with its reliability, and flexibility provided by its plugin mechanism. - Naotoshi Seo, Infrastructure Engineer

Drecom Co., Ltd.

Drecom (TYO:3793), based in Tokyo, is planning and developing many types of services which are focusing on word “communication” such as: mobile contents, internet advertising, etc. In our business, Log collection and analysis is essential in order to continue to provide services of the highest satisfaction to the user.

Fluentd is easy to install, reliable, and provides support with out interfering with our regular tasks. If anything, we are paying attention to not forget the presence of the silent daemon. It can not only provide many plugins, but also have and can develop add plugins with necessary functions.

We are pleased to fluent work with the Fluentd daemon.

GREE, Inc.

Data Science is at the core of GREE (TYO:3632)’s business, and it starts with logging user events and Advertisement related data. We currently deploy Fluentd for a number of our games and ad services to federate user event data, and use that data to inform our product decisions.

So far, we’ve been pleased with Fluentd’s robustness and performance: it has held up strongly against our massive data, allowing us to focus on data analytics instead of logging. - Masaki Fujimoto, CTO

LINE Corporation.

LINE Corp., based in Tokyo, is web services company. Log collection and analysis is one the most important missions in our day-to-day operations. As such we have spent quite a long time to search for the right tool that can process and collect data from various sources with great flexibility and high throughput – and we came up with Fluentd.

We utilize Fluentd to collect a very large amount of logs. The logs are written into Hadoop HDFS clusters, and are also used to analyze various service statuses in realtime. We also use many plugins from rubygems.org to further enhance this mechanism. - Satoshi Tagomori

Livesense, Inc.

Livesense Inc. (TYO:6054) engages in the internet business. We operates internet media in the fields of HR (jobs), real estate and new businesses. We were founded in 2006 and is headquartered in Tokyo, Japan.

Since we adopted Fluentd, our services have acquired to integrate collecting various structured events and log data. It is not only for collecting logs, it could preprocess logs with many swiss-army knife plugins to make it easy to analyze them. Also we could easily to create input/output plugins as needed. Fluentd has been worked in our business to help day-by-day data driven development for our Growth Hacker. Therefore, I felt Fluentd is revolutionary log collection tool I have ever seen. - Kentaro Yoshida

NAMCO BANDAI Studios Inc.

NAMCO BANDAI Studios Inc. is a platform-agnostic development studio that mainly develops console games, arcade games, online games, and mobile games. Besides games, NBSI also produces music, movie, and other entertainment content.

Log file analysis is key to examining user trends for our game services. We keep numerous logs such as access logs, error logs, and action logs. In order to efficiently analyze the information, we needed a solution flexible enough to aggregate the data from all the various logs.

The Fluentd concept fits our needs perfectly. For a large-scale service with hundreds of thousands of users, Fluentd's scalability has proven to be valuable. - Yasuhiro Ishimaru

Nintendo, Inc.

We use Fluentd to collect massive data logs for our platforms. Having developed a system based on Fluentd, we are now effectively monitoring and analyzing our services in real-time. We are very much satisfied with its flexibility, especially how easy it is to use in tandem with other systems.

PPLive, Inc.

PPTV.com is one of the largest Chinese online TV provider, which has 35 million active users. PPTV clients can be installed on multi-platforms (Windows, Mac, Android, iOS, WP, etc).

We (PPTV site operation team) tried to find a solution to do realtime log-collection and analyse based on the massive logfiles. Finally, we found that Fluentd is the perfect solution to do real-time log collection. Log can be JSON-structured to MongoDB cluster.

Realtime monitoring/reporting/charting can be generated from MongoDB data, which is helpful to our operation work. Fluentd is a great tool! - Xinyi Zhou, Site Operation Team

SlideShare, Inc.

SlideShare uses Fluentd to stream logs and events in the cloud. We were looking for a versatile yet lightweight logger in Ruby, and Fluentd fit the bill perfectly. - Sylvain Kalache, Operations Engineer

Uken Games

Uken Games is a Canadian cross-platform gaming company. Our users generate a lot of logs, and Fluentd made it really easy for us to manage them. It also allowed us to painlessly try and compare new solutions for log analysis, simply by modifying configs and zero-downtime restarting the client. Deploying Fluentd was a breeze with Chef. - Pitr Vernigorov

Viki, Inc.

At Viki, we use Fluentd deployed on Heroku to collect analytics events from various apps which then go into Treasure Data and in-house Hadoop Cluster for analysis. Fluentd integrates seamlessly with Heroku, and we've been impressed with its versatility and reliability. - Abhishek Parolkar, BigData Infrastructure Manager

FAQ

Fluentd Core

Fluentd is written in Ruby. Is it slow?

The most performance sensitive parts of Fluentd are written in C. The Ruby code acts as a wrapper that provides flexibility to the overall solution. In particular, the networking layer and object serialization layer are written in C (See cool.io and MessagePack).

Since Fluentd is not written entirely in C or C++, it may be slow in its Ruby parts. But by giving up a little bit of speed, we have gained many plugins from the Ruby community :). Fluentd's performance has been put to the test at many large services; in fact, a regular PC box can handle 18,000 messages/second with a single process.

If this number is insufficient for your application, please consider other solutions which are written entirely in C-family or Java. However, please note that you will lose some flexibility as a tradeoff.

Does Fluentd run on Windows?

Unfortunately, no. There are two issues.

1. Fluentd depends on a Ruby gem called cool.io, which doesn't work on Windows
2. Fluentd uses *nix dependent code

We have recently resolved issue 1 by becoming the Cool.io maintainer ourselves. Cool.io v1.2 now supports the Windows environment. We're now working on resolving issue 2 on Windows branch. See github: Windows branch

How can I collect logs from a Windows Machine?

You can use nxlog with the syslog protocol. Please see the Collecting Log Data from Windows article for details.

Does Fluentd have UI or storage?

No, Fluentd doesn't have either UI or storage. However, Kibana integration may be useful for you.

What is Fluentd's 'tag'?

Each Fluentd event log consists of the three components:

- tag
- time
- message

A tag must be specified for every log sent to Fluentd. The tag is used to route the message within Fluentd. The 'match' section in the configuration file specifies the tags that are routed to each destination.

Let's take a look at the example below. This configuration tails /var/log/combine.log and generates Fluentd logs with tag 'test-tag.'

```
<source>
  type tail
  format /^(?<wholemsg>.*)$/
  path /var/log/combined.log
  tag test-tag
</source>
```

If you add the following lines to the configuration file, the logs are routed to stdout.

```
<match test-tag>
  type stdout
</match>
```

On the other hand, adding the following lines to the configuration file will not route the logs anywhere, since the match section doesn't match the tag.

```
<match some-other-tag>
  type stdout
</match>
```

In essence, you can control your data process flow by using tags.

NOTE: The match section specifies the regexp used to look for matching tags. If a matching tag is found in a log, then the config inside the 'match' section is used (i.e. the log is routed according to the config inside).

How can I estimate Fluentd's resource usage?

Fluentd consumes more resources as more logs are thrown at it. We have found that the CPU becomes a major bottleneck for heavily loaded Fluentd systems. To combat this bottleneck, Fluentd can leverage multiple CPU cores through its multi-process mode. A regular PC box is able to handle around 18,000 msgs/second.

How is Fluentd's performance?

The most performance sensitive parts of Fluentd are carefully written in C, with Ruby code acting as a flexible wrapper. In addition, Fluentd can leverage multiple CPU cores through its multi-process plugin.

In our benchmark experiments, a regular PC box was able to deliver 18,000 messages/second. If you need to handle a higher load, please consider using multiple Fluentd servers. If your application needs 100x this performance, please consider using other solutions or creating a custom solution.

Treasure Agent(td-agnt)

What are the differences between td-agent and Fluentd?

`td-agent` is the stable distribution package of Fluentd which includes its own Ruby installation. The differences are as follows:

```
<th>
    <td>fluentd</td>
    <td>td-agent</td>
```

```
</th>
<tr>
    <td>Installation</td>
    <td>gem install fluentd</td>
    <td><a
```

href="http://docs.fluentd.org/articles/install-by-rpm">.rpm/.deb packages for Linux. Homebrew for OSX

Configuration

generic

preconfigured to send data to Treasure Data (can be modified)

Adding 3rd party plugins

fluent-gem (ex: fluent-gem install fluent-plugin-td)

fluent-gem and manual setup (but it ships with several plugins pre-loaded)

init.d script (for production deployment)

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)

QA/Support

Community-driven

QA by Treasure Data/Support for Treasure Data's paid customers

Please also see this section before installing plugins.

Should I use td-agent or the Fluentd gem?

td-agent stresses stability over new features. If you wish to control Fluentd features and updates on your own, using the Fluentd gem is recommended. If you are using Fluentd for the first time or are using it in a large scale environment, using td-agent is recommended. A new version of td-agent is released every 2 or 3 months.

Fluentd compared to other projects

What's the difference between Logstash and Fluentd?

Fluentd's built-in buffering mechanism offers both simplicity and robustness simultaneously. Check out the blog post below for more details.

- Fluentd vs Logstash

What's the difference between Scribe and Fluentd?

First, Fluentd is actively maintained while Scribe is not. Scribe is no longer used by Facebook, and is thus no longer maintained. Facebook has rewritten Scribe in Java, and now calls it Calligraphus.

Second, Scribe is more performant than Fluentd because it's written in C++; but as a tradeoff, it is difficult to extend compared to Fluentd.

What's the difference between Kafka and Fluentd?

Pull v.s. Push. Kafka is almost like a queue. Kafka needs another application to "pull" logged events to store them. Meanwhile, Fluentd supports output plugins which "push" logged events to a storage destination.

Fluentd's push-based architecture also takes care of "buffering". When file transfers to your backend storage fails, Fluentd automatically retries the transfer of the buffered file. When using Kafka, you need to handle the retry mechanism yourself.

What's the difference between Flume and Fluentd?

`Java` v.s. `C + Ruby`. If you intend to use the Hadoop-family of products especially CDH (Cloudera Distribution for Hadoop) as your backend, Flume may provide better support or compatibility than Fluentd.

Meanwhile, Fluentd is backend-storage agnostic, has less memory footprint, and is easy to use and extend.

What's the difference between Splunk and Fluentd?

Splunk is a log collection + search engine for unstrucuted, text-based logs. It has a nice UI and indexing engine for searching for terms within text log files.

Fluentd is a log collector daemon for semi-structured JSON-based logs. It deals with machine-readable logs, whereas Splunk handles text-based logs. Also, Fluentd is simply a mechanism for receiving, buffering, and forwarding data to another destination. It does not have search engine functionality but you can use ElasticSearch + Kibana as a backend to search the logs.

Operations

I have a weird timestamp value, what happened?

The timestamps of Fluentd and its logger libraries depend on your system's clock. It's highly recommended that you set up NTP on your nodes so that your clocks remain synced with the correct clocks.

I installed td-agent and want to add custom plugins. How do I do it?

Please use `fluent-gem` as shown below.

```
$ /usr/lib/fluent/ruby/bin/fluent-gem install <plugin name>
```

For example, issue the following command if you are adding `fluent-plugin-twitter`.

```
$ /usr/lib/fluent/ruby/bin/fluent-gem install fluent-plugin-twitter
```

(If you can't find fluent-gem in the above directory, try looking in `/usr/lib64/fluent/ruby/bin/fluent-gem`)

Now you might be wondering, "Why do I need to specify the full path?" The reason is that td-agent does not modify any host environment variable, including `PATH`. If you want to make all td-agent/fluentd related programs available without writing "/usr/lib/..." every time, you can add

```
export PATH=$PATH:/usr/lib/fluent/ruby/bin/
```

to your `~/.bash_profile`.

If you would like to find out more about plugin mangement, please take a look at the Plugin Management article.

How can I match (send) an event to multiple outputs?

You can use the `copy` output plugin to send the same event to multiple output destinations.

Plugin Development

How do I develop a custom plugin?

Please refer to the Plugin Development Guide.

Data Import from Ruby Applications

The 'fluent-logger-ruby' library is used to post records from Ruby applications to Fluentd.

This article explains how to use the fluent-logger-ruby library.

Prerequisites

- Basic knowledge of Ruby
- Basic knowledge of Fluentd
- Ruby 1.8 or later

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-ruby

First, add the 'fluent-logger' gem to your Gemfile.

```
gem 'fluent-logger', "~> 0.4.3"
```

Next, please initialize and post the records as shown below.

```
require 'fluent-logger'
Fluent::Logger::FluentLogger.open(nil, :host=>'localhost', :port=>24224)
Fluent::Logger.post("fluentd.test.follow", {"from"=>"userA", "to"=>"userB"})
```

Executing the script will send the logs to Fluentd.

```
$ ruby test.rb
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from Python Applications

The 'fluent-logger-python', library is used to post records from Python applications to Fluentd.

This article explains how to use the fluent-logger-python library.

Prerequisites

- Basic knowledge of Python
- Basic knowledge of Fluentd
- Python 2.6 or higher

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-python

First, install the fluent-logger library via pip.

```
$ pip install fluent-logger
```

Next, initialize and post the records as shown below.

```
# test.py
from fluent import sender
from fluent import event
sender.setup('fluentd.test', host='localhost', port=24224)
event.Event('follow', {
  'from': 'userA',
  'to':   'userB'
})
```

Executing the script will send the logs to Fluentd.

```
$ python test.py
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from PHP Applications

The 'fluent-logger-php' library is used to post records from PHP applications to Fluentd.

This article explains how to use the fluent-logger-php library.

Prerequisites

- Basic knowledge of PHP
- Basic knowledge of Fluentd
- PHP 5.3 or higher

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
# Unix Domain Socket Input
<source>
  type unix
  path /var/run/td-agent/td-agent.sock
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-php

To use fluent-logger-php, copy the library into your project directory.

```
$ git clone https://github.com/fluent/fluent-logger-php.git
$ cp -r src/Fluent <path/to/your_project>
```

Next, initialize and post the records as shown below.

```
<?php
require_once __DIR__.'/src/Fluent/Autoloader.php';
use Fluent\Logger\FluentLogger;
Fluent\Autoloader::register();
$logger = new FluentLogger("unix:///var/run/td-agent/td-agent.sock");
$logger->post("fluentd.test.follow", array("from"=>"userA", "to"=>"userB"));
```

Executing the script will send the logs to Fluentd.

```
$ php test.php
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from Perl Applications

The 'Fluent::Logger' library is used to post records from Perl applications to Fluentd.

This article explains how to use the Fluent::Logger library.

Prerequisites

- Basic knowledge of Perl
- Basic knowledge of Fluentd
- Perl 5.10 or higher

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using Fluent::Logger

First, install the *Fluent::Logger* library via CPAN.

```
$ cpan
cpan[1]> install Fluent::Logger
```

Next, initialize and post the records as shown below.

```
# test.pl
use Fluent::Logger;
my $logger = Fluent::Logger->new(
    host => '127.0.0.1',
    port => 24224,
    tag_prefix => 'fluentd.test',
);
$logger->post("follow", { "entry1" => "value1", "entry2" => 2 });
```

Executing the script will send the logs to Fluentd.

```
$ perl test.pl
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from Node.js Applications

The 'fluent-logger-node' library is used to post records from Node.js applications to Fluentd.

This article explains how to use the fluent-logger-node library.

Prerequisites

- Basic knowledge of Node.js and NPM
- Basic knowledge of Fluentd
- Node.js 0.6 or higher

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-node

Obtaining the Most Recent Version

The most recent version of fluent-logger-node can be found here.

A Sample Application

A sample Express app using fluent-logger-node is shown below.

package.json

```
{
  "name": "node-example",
  "version": "0.0.1",
  "dependencies": {
    "express": "2.5.9",
    "fluent-logger": "0.1.0"
  }
}
```

Now use *npm* to install your dependencies locally:

```
$ npm install
fluent-logger@0.1.0 ./node_modules/fluent-logger
express@2.5.9 ./node_modules/express
|-- qs@0.4.2
|-- mime@1.2.4
|-- mkdirp@0.3.0
|-- connect@1.8.6 (formidable@1.0.9)
```

web.js This is the simplest web app.

```
var express = require('express');
var app = express.createServer(express.logger());

var logger = require('fluent-logger');
logger.configure('fluentd.test', {host: 'localhost', port: 24224});

app.get('/', function(request, response) {
  logger.emit('follow', {from: 'userA', to: 'userB'});
  response.send('Hello World!');
});
var port = process.env.PORT || 3000;
app.listen(port, function() {
  console.log("Listening on " + port);
});
```

Execute the app and go to `http://localhost:3000/` in your browser. This will send the logs to Fluentd.

```
$ node web.js
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from Java Applications

The 'fluent-logger-java' library is used to post records from Java applications to Fluentd.

This article explains how to use the fluent-logger-java library.

Prerequisites

- Basic knowledge of Java
- Basic knowledge of Fluentd
- Java 6 or higher

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package

- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-java

First, please add the following lines to pom.xml. The logger's revision information can be found in CHANGES.txt.

```
<dependencies>
  ...
  <dependency>
    <groupId>org.fluentd</groupId>
    <artifactId>fluent-logger</artifactId>
    <version>${logger.version}</version>
  </dependency>
  ...
</dependencies>
```

Next, please insert the following lines into your application. Further information regarding the API can be found here.

```
import java.util.HashMap;
import java.util.Map;
import org.fluentd.logger.FluentLogger;

public class Main {
    private static FluentLogger LOG = FluentLogger.getLogger("fluentd.test");

    public void doApplicationLogic() {
        // ...
        Map<String, String> data = new HashMap<String, String>();
        data.put("from", "userA");
        data.put("to", "userB");
```

```
        LOG.log("follow", data);
        // ...
    }
}
```

Executing the script will send the logs to Fluentd.

```
$ java -jar test.jar
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Data Import from Scala Applications

The 'fluent-logger-scala' library is used to post records from Scala applications to Fluentd.

This article explains how to use the fluent-logger-scala library.

Prerequisites

- Basic knowledge of Scala and sbt
- Basic knowledge of Fluentd
- Scala 2.9.0 or 2.9.1
- sbt 0.12.0 or later

Installing Fluentd

Please refer to the following documents to install fluentd.

- Install Fluentd with rpm Package
- Install Fluentd with deb Package
- Install Fluentd with Ruby Gem
- Install Fluentd from source

Modifying the Config File

Next, please configure Fluentd to use the forward Input plugin as its data source.

```
<source>
  type forward
  port 24224
</source>
<match fluentd.test.**>
  type stdout
</match>
```

Please restart your agent once these lines are in place.

```
# for rpm/deb only
$ sudo /etc/init.d/td-agent restart
```

Using fluent-logger-scala

First, please add the following lines to build.sbt. The logger's revision information can be found in the ChangeLog.

```
resolvers += "Apache Maven Central Repository" at "http://repo.maven.apache.org/maven2/"

libraryDependencies += "org.fluentd" %% "fluent-logger-scala" % "0.3.0"
```

or

```
resolvers += "Sonatype Repository" at "http://oss.sonatype.org/content/repositories/releases"

libraryDependencies += "org.fluentd" %% "fluent-logger-scala" % "0.3.0"
```

Next, please insert the following lines into your application. Further information regarding the API can be found here.

```
import org.fluentd.logger.scala.FluentLoggerFactory
import scala.collection.mutable.HashMap

object Sample {
  val LOG = FluentLoggerFactory.getLogger("fluentd.test")

  def main(args: Array[String]): Unit = {

    ...
    val data = new HashMap[String, String](#);
    data.put("from", "userA");
    data.put("to", "userB");
    LOG.log("follow", data);
    ...
  }

}
```

Executing the script will send the logs to Fluentd.

```
$ sbt
> run
```

The logs should be output to `/var/log/td-agent/td-agent.log` or stdout of the Fluentd process via the stdout Output plugin.

Production Deployments

Output Plugins

Various output plugins are available for writing records to other destinations:

- Examples
- Store Apache Logs into Amazon S3
- Store Apache Logs into MongoDB
- Data Collection into HDFS
- List of Plugin References
- Output to Another Fluentd
- Output to MongoDB or MongoDB ReplicaSet
- Output to Hadoop
- Output to File
- etc...

High-Availablability Configurations of Fluentd

For high-traffic websites (more than 5 application nodes), we recommend using a high availability configuration of td-agent. This will improve data transfer reliability and query performance.

- High-Availability Configurations of Fluentd

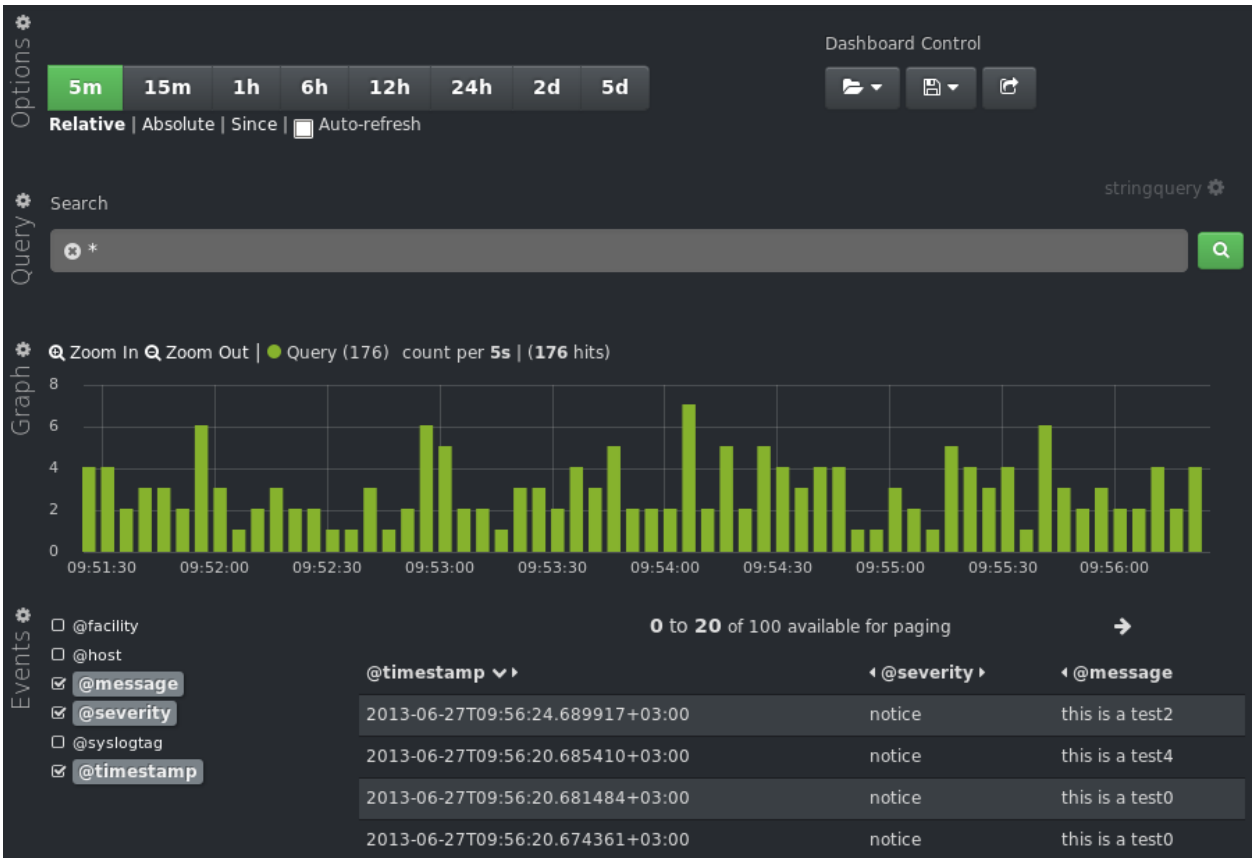
Monitoring

Monitoring Fluentd itself is also important. The article below describes general monitoring methods for td-agent.

- Monitoring Fluentd

Free Alternative to Splunk Using Fluentd

Splunk is a great tool for searching logs, but its high cost makes it prohibitive for many teams. In this article, we present a free and open source alternative to Splunk by combining three open source projects: Elasticsearch, Kibana, and Fluentd.



Click to See the Demo

Elasticsearch is an open source search engine known for its ease of use. Kibana is an open source Web UI that makes Elasticsearch user friendly for marketers, engineers and data scientists alike.

By combining these three tools (Fluentd + Elasticsearch + Kibana) we get a scalable, flexible, easy to use log search engine with a great Web UI that provides an open-source Splunk alternative, all for free!

In this guide, we will go over installation, setup, and basic use of this combined log search solution. The contents of this article were tested on Mac OS X Mountain Lion. **If you're not familiar with Fluentd**, please learn more about Fluentd first.

Learn More

Prerequisites

Java for Elasticsearch

Please confirm that your Java version is 6 or higher.

```
$ java -version
java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06-451-11M4406)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01-451, mixed mode)
```

Now that we've checked for prerequisites, we're now ready to install and set up the three open source tools.

Set Up Elasticsearch

To install Elasticsearch, please download and extract the Elasticsearch package as shown below.

```
$ curl -O https://download.elasticsearch.org/elasticsearch/elasticsearch/elasticsearch-0.90.0.RC2.tar.gz
$ tar zxvf elasticsearch-0.90.0.RC2.tar.gz
$ cd elasticsearch-0.90.0.RC2/
```

Once installation is complete, start Elasticsearch.

```
$ ./bin/elasticsearch -f
```

Setup Kibana

To install Kibana, download it via the official webpage and extract it. Kibana is a HTML / CSS / JavaScript application.

```
$ curl -O https://download.elasticsearch.org/kibana/kibana/kibana-3.0.0milestone5.tar.gz
$ tar zxvf kibana-3.0.0milestone5.tar.gz
$ cd kibana-3.0.0milestone5/
```

Once installation is complete, start Kibana and open `index.html`. You can modify Kibana's configuration via `config.js`.

```
$ open index.html
```

Setup Fluentd (td-agent)

In this guide We'll install td-agent, the stable release of Fluentd. Please refer to the guides below for detailed installation steps.

- Debian Package
- RPM Package
- Ruby gem

Next, we'll install the Elasticsearch plugin for Fluentd: fluent-plugin-elasticsearch.

```
$ /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-elasticsearch
```

We'll configure td-agent (Fluentd) to interface properly with Elasticsearch. Please modify `/etc/td-agent/td-agent.conf` as shown below:

```
<source>
  type syslog
  port 42185
  tag syslog
</source>

<source>
  type forward
</source>

<match syslog.**>
  type elasticsearch
  logstash_format true
  flush_interval 10s # for testing
</match>
```

fluent-plugin-elasticsearch comes with a logstash_format option that allows Kibana to search stored event logs in Elasticsearch.

Once everything has been set up and configured, we'll start td-agent.

```
$ sudo /etc/init.d/td-agent start
```

Setup rsyslogd

In our final step, we'll forward the logs from your rsyslogd to Fluentd. Please add the following line to your `/etc/rsyslog.conf`, and restart rsyslog. This will forward your local syslog to Fluentd, and Fluentd in turn will forward the logs to Elasticsearch.

```
*.* @127.0.0.1:42185
```

Please restart the rsyslog service once the modification is complete.

```
$ sudo /etc/init.d/rsyslog restart
```

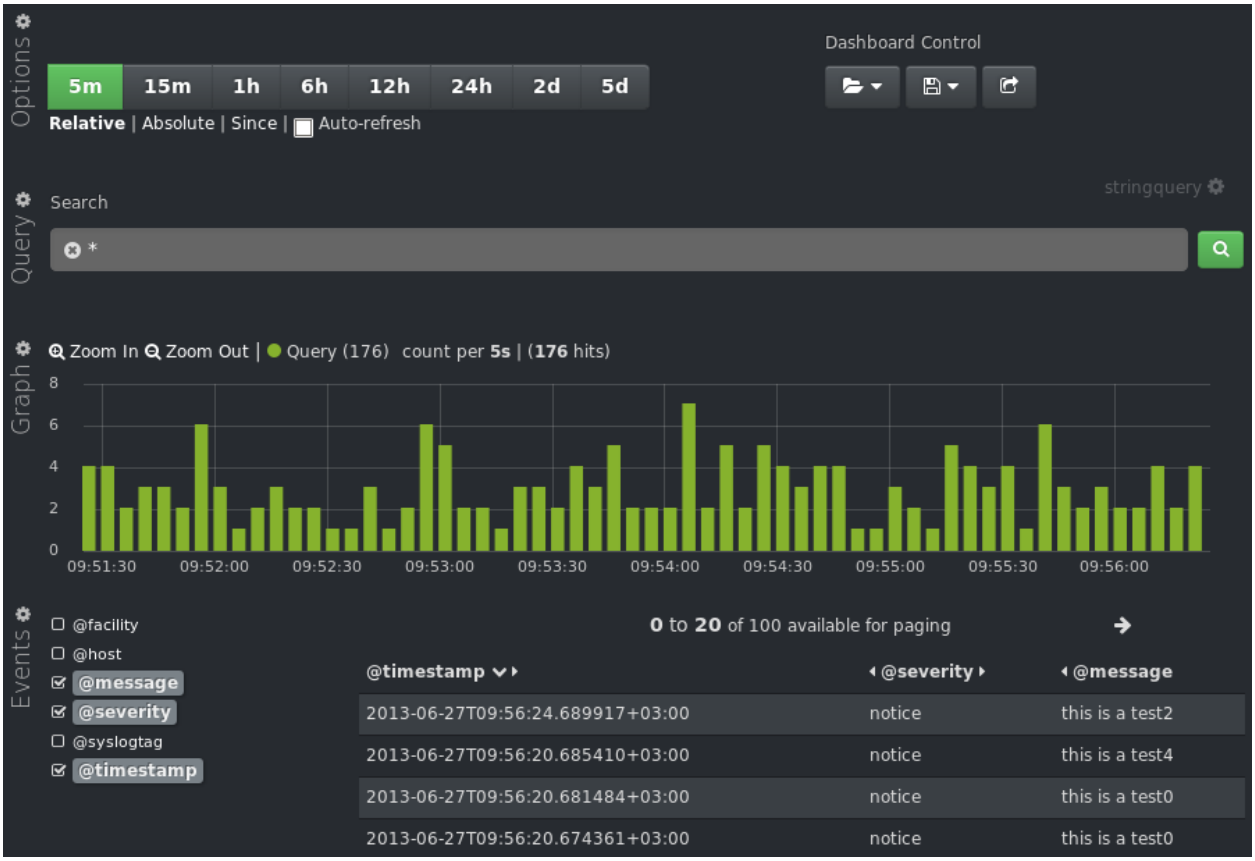
Store and Search Event Logs

Once Fluentd receives some event logs from rsyslog and has flushed them to Elasticsearch, you can search the stored logs using Kibana by accessing http://127.0.0.1:5601/ in your browser.

To manually send logs to Elasticsearch, please use the `logger` command.

```
$ logger -t test foobar
```

When debugging your td-agent configuration, using out_copy + out_stdout will be useful. All the logs including errors can be found at `/etc/td-agent/td-agent.log`.




```
<match syslog.**>
  type copy
  <store>
    # for debug (see /var/log/td-agent.log)
    type stdout
  </store>
  <store>
    type elasticsearch
    logstash_format true
    flush_interval 10s # for testing
  </store>
</match>
```

Demo Environment

Please access the Kibana Demo Environment from the link below.

- Kibana Demo Environment

Conclusion

This article introduced the combination of Fluentd and Kibana (with Elasticsearch) which achieves a free alternative to Splunk: storing and searching machine logs. The examples provided in this article have not been tuned.

If you will be using these components in production, you may want to modify some of the configurations (e.g. JVM, Elasticsearch, Fluentd buffer, etc.) according to your needs.

Learn More

- Fluentd Architecture
- Fluentd Get Started

How To Filter Or Modify Data Inside Fluentd (Apache as an Example)

In this article, we introduce several common data manipulation challenges faced by our users (such as filtering and modifying data) and explain how to solve each task using one or more Fluentd plugins.

Scenario: Filtering Data by the Value of a Field

Let's suppose our Fluentd instances are collecting data from Apache web server logs via in_tail. Our goal is to filter out all the 200 requests.

Solution: Use fluent-plugin-grep

fluent-plugin-grep is a plugin that can "grep" data according to the different fields within Fluentd events.

If our events looks like

```
{
    "code": 200,
    "url": "http://yourdomain.com/page.html",
    "size": 2344,
    "referer": "http://www.treasuredata.com"
    ...
}
```

then we can filter out all the requests with status code 200 as follows:

```
...
<match apache.**>
    type grep
    input_key code
    exclude ^200$
    add_tag_prefix filtered
</match>
```

By using the `add_tag_prefix` option, we can prepend a tag in front of filtered events so that they can be matched to a subsequent section. For example, we can send all logs with non-200 status codes to Treasure Data, as shown below:

```
...
<match apache.**>
    type grep
    input_key code
    exclude ^200$
    add_tag_prefix filtered
</match>
<match filtered.apache.**>
    type td_lot
    apikey XXXXX
    ...
</match>
```

`fluent-plugin-grep` can filter based on multiple fields as well. The config below keeps all requests with status code 4xx that are NOT referred from yourdomain.com (a real world use case: figuring out how many dead links there are in the wild by filtering out internal links)

```
...
<match apache.**>
    type grep
    regexp1 code ^4\d\d$
    exclude1 referer ^https?://yourdomain.com
    add_tag_prefix external_dead_links
</match>
...
```

Scenario: Adding a New Field (such as hostname)

When collecting data, we often need to add a new field or change an existing field in our log data. For example, many Fluentd users need to add the hostname of their servers to the Apache web server log data in order to compute the number of requests handled by each server (i.e., store them in MongoDB/HDFS and run GROUP-BYs).

Solution: Use fluent-plugin-record-modifier

fluent-plugin-record-modifier can add a new field to each data record.

If our events looks like

```
{"code":200, "url":"http://yourdomain.com", "size":1232}
```

then we can add a new field with the hostname information as follows:

```
<match foo.bar>
    type record_modifier
    gen_host ${hostname}
    tag with_hostname
</match>
...
<match with_hostname>
    ...
</match>
```

The modified events now look like

```
{"gen_host": "our_server", code":200, "url":"http://yourdomain.com", "size":1232}
```

NOTE: The `${hostname}` placeholder is powered by fluent-mixin-config-placeholder. It inlines the host name of the server that the Fluentd instance is running on (in this example, our server's name is "our_server").

Splunk-like Grep-and-Alert-Email System Using Fluentd

Splunk is a great tool for searching logs. One of its key features is the ability to "grep" logs and send alert emails when certain conditions are met.

In this little "how to" article, we will show you how to build a similar system using Fluentd. More specifically, we will create a system that sends an alert email when it detects a 5xx HTTP status code in an Apache access log.

By the way, Splunk happens to be quite expensive. If you're interested in a free alternative, check out our article here.

Installing the Needed Plugins

Install Fluend if you haven't yet.

Please install `fluent-plugin-grepcounter` by running:

```
$ gem install fluent-plugin-grepcounter
```

Next, please install `fluent-plugin-mail` by running:

```
$ gem install fluent-plugin-mail
```

Configuration

Configuration File: Soup to Nuts

Here is an example configuration file. It's a bit long, but each part is well-commented, so don't be afraid.

```
<source>
  type http #This is for testing
  port 8888
</source>

<source>
  type tail
  format apache2
  path /var/log/apache2/access.log #This is the location of your Apache log
  tag apache.access
</source>

<match apache.access>
  type grepcounter
```

```
  count_interval 3 #Time window to grep and count the # of events
  input_key code #We look at the (http status) "code" field
  regexp ^5\d\d$ #This regexp matches 5xx status codes
  threshold 1 #The # of events to trigger emitting an output
  add_tag_prefix error_5xx #The output event's tag will be error_5xx.apache.access
</match>

<match error_5xx.apache.access>
  # The event that comes here looks like
  #{
  #  "count":1,
  #  "input_tag":"error_5xx.apache.access",
  #  "input_tag_last":"access",
  #  "message":[500]
  #}

  type copy #Copying events, one to send to stdout, another for email alerts

  <store>
    type stdout
  </store>

  <store>
    type mail
    host smtp.gmail.com #This is for Gmail and Google Apps. Any SMTP server should work
    port 587 #This is the port for smtp.gmail.com
    user kiyoto@treasure-data.com #I work here! Use YOUR EMAIL.
    password XXXXXX #I can't tell you this! Use YOUR PASSWORD!
    enable_starttls_auto true
    from YOUR_SENDER_EMAIL_HERE
    to YOUR_RECIPIENT_EMAIL_HERE
    subject [URGENT] APACHE 5XX ERROR
    message Total 5xx error count: %s\n\nPlease check your Apache webserver ASAP
    message_out_keys count #The value of 'count' will be substituted into %s above.
  </store>
</match>
```

Save the above into your own configuration file (**We assume it's called `test.conf` for the rest of this page**). Make sure your SMTP is configured correctly (otherwise, you will get a warning when you run the program).

What the Configuration File Does

The config above does three things:

1. Sets up Fluentd to tail an Apache log file (located at `/var/log/apache2/access.log`).
2. Every 3 seconds, it counts the number of events whose "code" field is 5xx. If the number is at least 1 (because of `threshold 1`), emit an event with the tag `error_5xx.apache.access`. All of this is done by `fluent-plugin-grepcounter`.
3. Sends an email to dev@treasure-data.com (and also outputs to STDOUT for debugging & testing) for each event with the tag `error_5xx.apache.access`.

We can do all this **without writing a single line of code or paying a dime!**

Testing

Just run

```
$ fluentd -c test.conf
```

to start Fluentd.

To trigger the alert email, you can either manually append a 5xx error log line to your Apache log or visit (on the same server)

```
http://localhost:8888/apache/access?json={"code":"500"}
```

(This uses the in_http plugin). You should be receiving an alert email with the subject line "[URGENT] APACHE 5XX ERROR" in your inbox right about now!

What's Next?

Admittedly, this is a contrived example. In reality, you would set the threshold higher. Also, you might be interested in tracking 4xx pages as well. In addition to Apache logs, Fluentd can handle Nginx logs, syslogs, or any single- or multi-lined logs.

You can learn more about Fluentd and its plugins by

- exploring other plugins
- browsing recipes
- asking questions on the mailing list
- signing up for our newsletters

Cloud Big Data Analytics with Treasure Data

This article explains how to use Fluentd's Treasure Data Output plugin to aggregate semi-structured logs into Treasure Data (TD), which offers Cloud Data Service.

Background

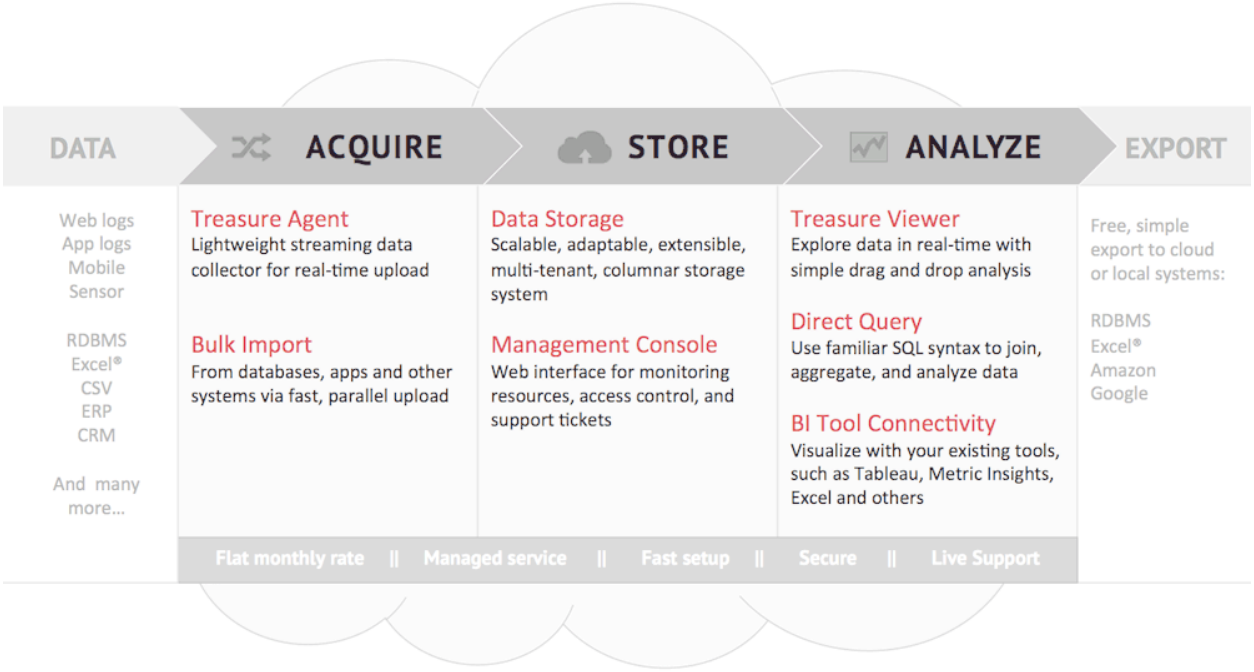
Fluentd is an advanced open-source log collector originally developed at Treasure Data, Inc. Fluentd is specifically designed to solve the big-data log collection problem.

Treasure Data provides Cloud Data Service, which Fluentd users can use to easily store and analyze data on the cloud. Fluentd is designed to flexibly connect with many systems via plugins, but Treasure Data should be your top choice if you don't want to spend engineering resources maintaining your backend infrastructure.

This article will show you how to use Fluentd to receive data from HTTP and stream it into TD.

Architecture

The figure below shows the high-level architecture.



Install

For simplicity, this article will describe how to set up an one-node configuration. Please install the following software on the same node.

- Fluentd
- TD Output Plugin

The TD Output plugin is included in Fluentd's deb/rpm package (`td-agent`) by default. If you want to use Ruby Gems to install the plugin, please use `gem install fluent-plugin-td`.

- Debian Package
- RPM Package
- Ruby gem

Signup

Next, please sign up to TD and get your apikey using the `td apikey:show` command.

```
$ td account -f
Enter your Treasure Data credentials.
Email: your.email@gmail.com
Password (typing will be hidden):
$ td apikey:show
kdfasklj218dsakfdas0983120
```

Fluentd Configuration

Let's start configuring Fluentd. If you used the deb/rpm package, Fluentd's config file is located at /etc/td-agent/td-agent.conf. Otherwise, it is located at /etc/fluentd/fluentd.conf.

HTTP Input

For the input source, we will set up Fluentd to accept records from HTTP. The Fluentd configuration file should look like this:

```
<source>
  type http
  port 8888
</source>
```

Treasure Data Output

The output destination will be Treasure Data. The output configuration should look like this:

```
# Treasure Data output
<match td.*.*>
  type tdlog
  apikey YOUR_API_KEY_IS_HERE
  auto_create_table
  buffer_type file
  buffer_path /var/log/td-agent/buffer/td
  use_ssl true
</match>
```

The match section specifies the regexp used to look for matching tags. If a matching tag is found in a log, then the config inside `<match>...</match>` is used (i.e. the log is routed according to the config inside).

Test

To test the configuration, just post the JSON to Fluentd. Sending a USR1 signal flushes Fluentd's buffer into TD.

```
$ curl -X POST -d 'json={"action":"login","user":2}' \
  http://localhost:8888/td.testdb.www_access
$ kill -USR1 `cat /var/run/td-agent/td-agent.pid`
```

Next, please use the `td tables` command. If the count is not zero, the data was imported successfully.

```
$ td tables
+----------+------------+------+-------+--------+
| Database | Table      | Type | Count | Schema |
+----------+------------+------+-------+--------+
| testdb   | www_access | log  |     1 |        |
+----------+------------+------+-------+--------+
```

You can now issues queries against the imported data.

```
$ td query -w -d testdb \
  "SELECT COUNT(1) AS cnt FROM www_access"
queued...
started at 2012-04-10T23:44:41Z
2012-04-10 23:43:12,692 Stage-1 map = 0%,  reduce = 0%
2012-04-10 23:43:18,766 Stage-1 map = 100%,  reduce = 0%
2012-04-10 23:43:32,973 Stage-1 map = 100%,  reduce = 100%
Status     : success
Result     :
+-----+
| cnt |
+-----+
|   1 |
+-----+
```

NOTE: It's not advisable to send sensitive user information to the cloud. To assist with this need, out_tdlog comes with some anonymization systems. Please see the Treasure Data plugin article for details.

Conclusion

Fluentd + Treasure Data gives you a data collection and analysis system in days, not months. Treasure Data is a useful solution if you don't want to spend engineering resources maintaining the backend storage and analytics infrastructure.

Learn More

- Fluentd Architecture
- Fluentd Get Started
- Treasure Data: Cloud Data Service
- Treasure Data: Documentation

Store Apache Logs into Amazon S3

This article explains how to use Fluentd's Amazon S3 Output plugin (out_s3) to aggregate semi-structured logs in real-time.

Background

Fluentd is an advanced open-source log collector originally developed at Treasure Data, Inc. One of the main objectives of log aggregation is data archiving. Amazon S3, the cloud object storage provided by Amazon, is a popular solution for data archiving.

This article will show you how to use Fluentd to import Apache logs into Amazon S3.

Mechanism

Fluentd does 3 things:

1. It continuously "tails" the access log file.
2. It parses the incoming log entries into meaningful fields (such as `ip`, `path`, etc.) and buffers them.
3. It writes the buffered data to Amazon S3 periodically.

Install

For simplicity, this article will describe how to set up an one-node configuration. Please install the following software on the same node.

- Fluentd
- Amazon S3 Output Plugin
- Your Amazon Web Services Account
- Apache (with the Combined Log Format)

The Amazon S3 Output plugin is included in the latest version of Fluentd's deb/rpm package. If you want to use Ruby Gems to install the plugin, please use `gem install fluent-plugin-s3`.

- Debian Package
- RPM Package
- Ruby gem

Configuration

Let's start configuring Fluentd. If you used the deb/rpm package, Fluentd's config file is located at /etc/td-agent/td-agent.conf. Otherwise, it is located at /etc/fluentd/fluentd.conf.

Tail Input

For the input source, we will set up Fluentd to track the recent Apache logs (typically found at /var/log/apache2/access_log) The Fluentd configuration file should look like this:

```
<source>
  type tail
  format apache2
  path /var/log/apache2/access_log
  pos_file /var/log/td-agent/apache2.access_log.pos
  tag s3.apache.access
</source>
```

NOTE: Please make sure that your Apache outputs are in the default 'combined' format. `format apache2` cannot parse custom log formats. Please see the in_tail article for more information.

Let's go through the configuration line by line.

1. `type tail`: The tail Input plugin continuously tracks the log file. This handy plugin is included in Fluentd's core.

2. `format apache2`: Uses Fluentd's built-in Apache log parser.
3. `path /var/log/apache2/access_log`: The location of the Apache log. This may be different for your particular system.
4. `tag s3.apache.access`: `s3.apache.access` is used as the tag to route the messages within Fluentd.

That's it! You should now be able to output a JSON-formatted data stream for Fluentd to process.

Amazon S3 Output

The output destination will be Amazon S3. The output configuration should look like this:

```
<match s3.*.*>
  type s3

  aws_key_id YOUR_AWS_KEY_ID
  aws_sec_key YOUR_AWS_SECRET/KEY
  s3_bucket YOUR_S3_BUCKET_NAME
  path logs/
  buffer_path /var/log/td-agent/s3

  time_slice_format %Y%m%d%H
  time_slice_wait 10m
  utc

  buffer_chunk_limit 256m
</match>
```

The match section specifies the regexp used to look for matching tags. If a matching tag is found in a log, then the config inside `<match>...</match>` is used (i.e. the log is routed according to the config inside). In this example, the `s3.apache.access` tag (generated by `tail`) is always used.

Test

To test the configuration, just ping the Apache server. This example uses the `ab` (Apache Bench) program.

```
$ ab -n 100 -c 10 http://localhost/
```

Then, log into your AWS Console and look at your bucket.

WARNING: By default, files are created on an hourly basis (around xx:10). This means that when you first import records using the plugin, no file is created immediately. The file will be created when the `time_slice_format` condition has been met. To change the output frequency, please modify the `time_slice_format` value. To write files every minute, please use `%Y%m%d%H%M` for the `time_slice_format`.

Conclusion

Fluentd + Amazon S3 makes real-time log archiving simple.

Learn More

- Fluentd Architecture
- Fluentd Get Started
- Amazon S3 Output plugin

Store Apache Logs into MongoDB

This article explains how to use Fluentd's MongoDB Output plugin (out_mongo) to aggregate semi-structured logs in real-time.

Background

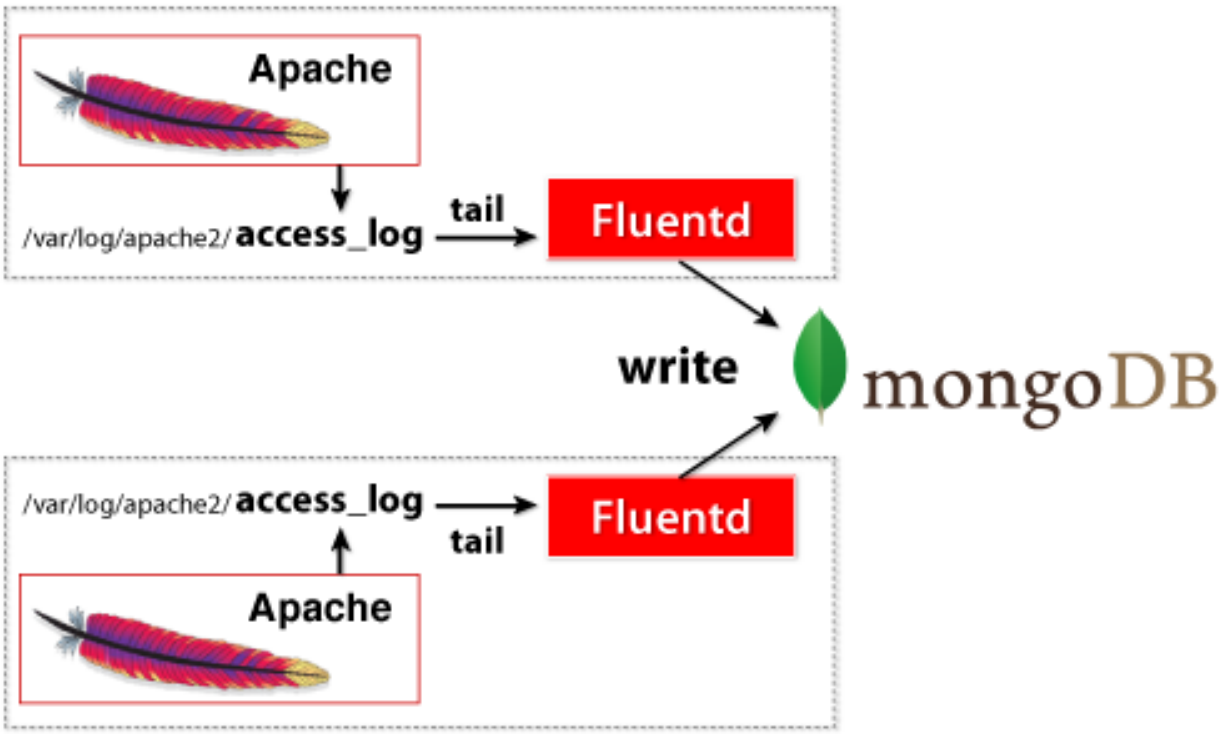
Fluentd is an advanced open-source log collector originally developed at Treasure Data, Inc. Because Fluentd handles logs as semi-structured data streams, the ideal database should have strong support for semi-structured data. There are several candidates that meet this criterion, but we believe MongoDB is the market leader.

MongoDB is an open-source, document-oriented database developed at 10gen, Inc. It is schema-free and uses a JSON-like format to manage semi-structured data.

This article will show you how to use Fluentd to import Apache logs into MongoDB.

Mechanism

The figure below shows how things will work.



Fluentd does 3 things:

1. It continuously "tails" the access log file.
2. It parses the incoming log entries into meaningful fields (such as `ip`, `path`, etc.) and buffers them.
3. It writes the buffered data to MongoDB periodically.

Install

For simplicity, this article will describe how to set up an one-node configuration. Please install the following software on the same node.

- Fluentd
- MongoDB Output Plugin
- MongoDB
- Apache (with the Combined Log Format)

The MongoDB Output plugin is included in the latest version of Fluentd's deb/rpm package. If you want to use Ruby Gems to install the plugin, please use `gem install fluent-plugin-mongo`.

- Debian Package
- RPM Package
- Ruby gem

For MongoDB, please refer to the following downloads page.

- MongoDB Downloads

Configuration

Let's start configuring Fluentd. If you used the deb/rpm package, Fluentd's config file is located at /etc/td-agent/td-agent.conf. Otherwise, it is located at /etc/fluentd/fluentd.conf.

Tail Input

For the input source, we will set up Fluentd to track the recent Apache logs (typically found at /var/log/apache2/access_log) The Fluentd configuration file should look like this:

```
<source>
  type tail
  format apache2
  path /var/log/apache2/access_log
  pos_file /var/log/td-agent/apache2.access_log.pos
  tag mongo.apache.access
</source>
```

NOTE: Please make sure that your Apache outputs are in the default 'combined' format. `format apache2` cannot parse custom log formats. Please see the in_tail article for more information.

Let's go through the configuration line by line.

1. `type tail`: The tail Input plugin continuously tracks the log file. This handy plugin is included in Fluentd's core.

2. `format apache2`: Uses Fluentd's built-in Apache log parser.
3. `path /var/log/apache2/access_log`: The location of the Apache log. This may be different for your particular system.
4. `tag mongo.apache.access`: `mongo.apache.access` is used as the tag to route the messages within Fluentd.

That's it! You should now be able to output a JSON-formatted data stream for Fluentd to process.

MongoDB Output

The output destination will be MongoDB. The output configuration should look like this:

```
<match mongo.*.*>
  # plugin type
  type mongo

  # mongodb db + collection
  database apache
  collection access

  # mongodb host + port
  host localhost
  port 27017

  # interval
  flush_interval 10s
</match>
```

The match section specifies the regexp used to look for matching tags. If a matching tag is found in a log, then the config inside `<match>...</match>` is used (i.e. the log is routed according to the config inside). In this example, the `mongo.apache.access` tag (generated by `tail`) is always used.

The `**` in `match.**` matches zero or more period-delimited tag parts (e.g. match/match.a/match.a.b).

flush_interval specifies how often the data is written to MongoDB. The other options specify MongoDB's host, port, db, and collection.

NOTE: For additional configuration parameters, please see the MongoDB Output plugin article. If you are using ReplicaSet, please see the MongoDB ReplicaSet Output plugin article.

Test

To test the configuration, just ping the Apache server. This example uses the `ab` (Apache Bench) program.

```
$ ab -n 100 -c 10 http://localhost/
```

Then, access MongoDB and see the stored data.

```
$ mongo
> use apache
> db["access"].findOne();
{ "_id" : ObjectId("4ed1ed3a340765ce73000001"), "host" : "127.0.0.1", "user" : "-", "method" : "GET", "path"
{ "_id" : ObjectId("4ed1ed3a340765ce73000002"), "host" : "127.0.0.1", "user" : "-", "method" : "GET", "path"
{ "_id" : ObjectId("4ed1ed3a340765ce73000003"), "host" : "127.0.0.1", "user" : "-", "method" : "GET", "path"
```

Conclusion

Fluentd + MongoDB makes real-time log collection simple, easy, and robust.

Learn More

- Fluentd Architecture
- Fluentd Get Started
- MongoDB Output Plugin
- MongoDB ReplicaSet Output Plugin

Fluentd + HDFS: Instant Big Data Collection

This article explains how to use Fluentd's WebHDFS Output plugin to aggregate semi-structured logs into Hadoop HDFS.

Background

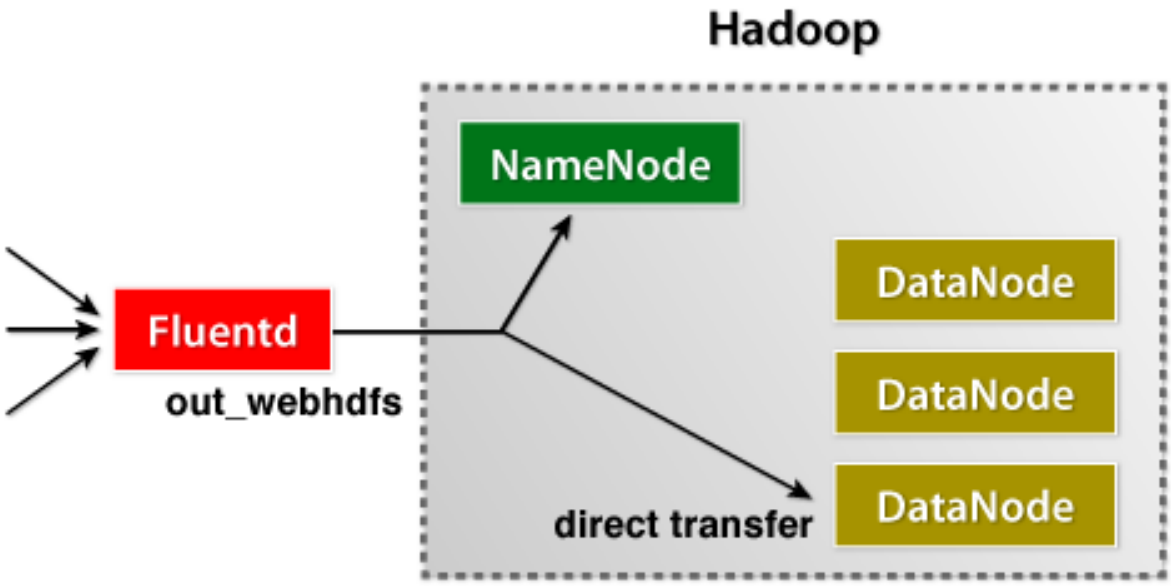
Fluentd is an advanced open-source log collector originally developed at Treasure Data, Inc. Fluentd is specifically designed to solve the big-data log collection problem. A lot of users are using Fluentd with MongoDB, and have found that it doesn't scale well for now.

HDFS (Hadoop) is a natural alternative for storing and processing a huge amount of data, but it didn't have an accessible API other than its Java library until recently. From Apache 1.0.0, CDH3u5, or CDH4 onwards, HDFS supports an HTTP interface called WebHDFS.

This article will show you how to use Fluentd to receive data from HTTP and stream it into HDFS.

Architecture

The figure below shows the high-level architecture.



Install

For simplicity, this article will describe how to set up an one-node configuration. Please install the following software on the same node.

- Fluentd
- WebHDFS Output Plugin (out_webhdfs)
- HDFS (Apache 1.0.0, CDH3u5 or CDH4 onwards)

The WebHDFS Output plugin is included in the latest version of Fluentd's deb/rpm package (v1.1.10 or later). If you want to use Ruby Gems to install the plugin, please use `gem install fluent-plugin-webhdfs`.

- Debian Package
- RPM Package
- For CDH, please refer to the downloads page (CDH3u5 and CDH4 onwards)
- Ruby gem

Fluentd Configuration

Let's start configuring Fluentd. If you used the deb/rpm package, Fluentd's config file is located at /etc/td-agent/td-agent.conf. Otherwise, it is located at /etc/fluentd/fluentd.conf.

HTTP Input

For the input source, we will set up Fluentd to accept records from HTTP. The Fluentd configuration file should look like this:

```
<source>
  type http
  port 8888
</source>
```

WebHDFS Output

The output destination will be WebHDFS. The output configuration should look like this:

```
<match hdfs.*.*>
  type webhdfs
  host namenode.your.cluster.local
  port 50070
  path /log/%Y%m%d_%H/access.log.${hostname}
  flush_interval 10s
</match>
```

The match section specifies the regexp used to look for matching tags. If a matching tag is found in a log, then the config inside `<match>...</match>` is used (i.e. the log is routed according to the config inside).

flush_interval specifies how often the data is written to HDFS. An append operation is used to append the incoming data to the file specified by the **path** parameter.

Placeholders for both time and hostname can be used with the **path** parameter. This prevents multiple Fluentd instances from appending data to the same file, which must be avoided for append operations.

Other options specify HDFS's NameNode host and port.

HDFS Configuration

Append operations are not enabled by default. Please put these configurations into your hdfs-site.xml file and restart the whole cluster.

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>

<property>
  <name>dfs.support.broken.append</name>
  <value>true</value>
</property>
```

Please confirm that the HDFS user has write access to the *path* specified as the WebHDFS output.

Test

To test the configuration, just post the JSON to Fluentd (we use the curl command in this example). Sending a USR1 signal flushes Fluentd's buffer into WebHDFS.

```
$ curl -X POST -d 'json={"action":"login","user":2}' \
  http://localhost:8888/hdfs.access.test
$ kill -USR1 `cat /var/run/td-agent/td-agent.pid`
```

We can then access HDFS to see the stored data.

```
$ sudo -u hdfs hadoop fs -lsr /log/
drwxr-xr-x   - 1 supergroup          0 2012-10-22 09:40 /log/20121022_14/access.log.dev
```

Conclusion

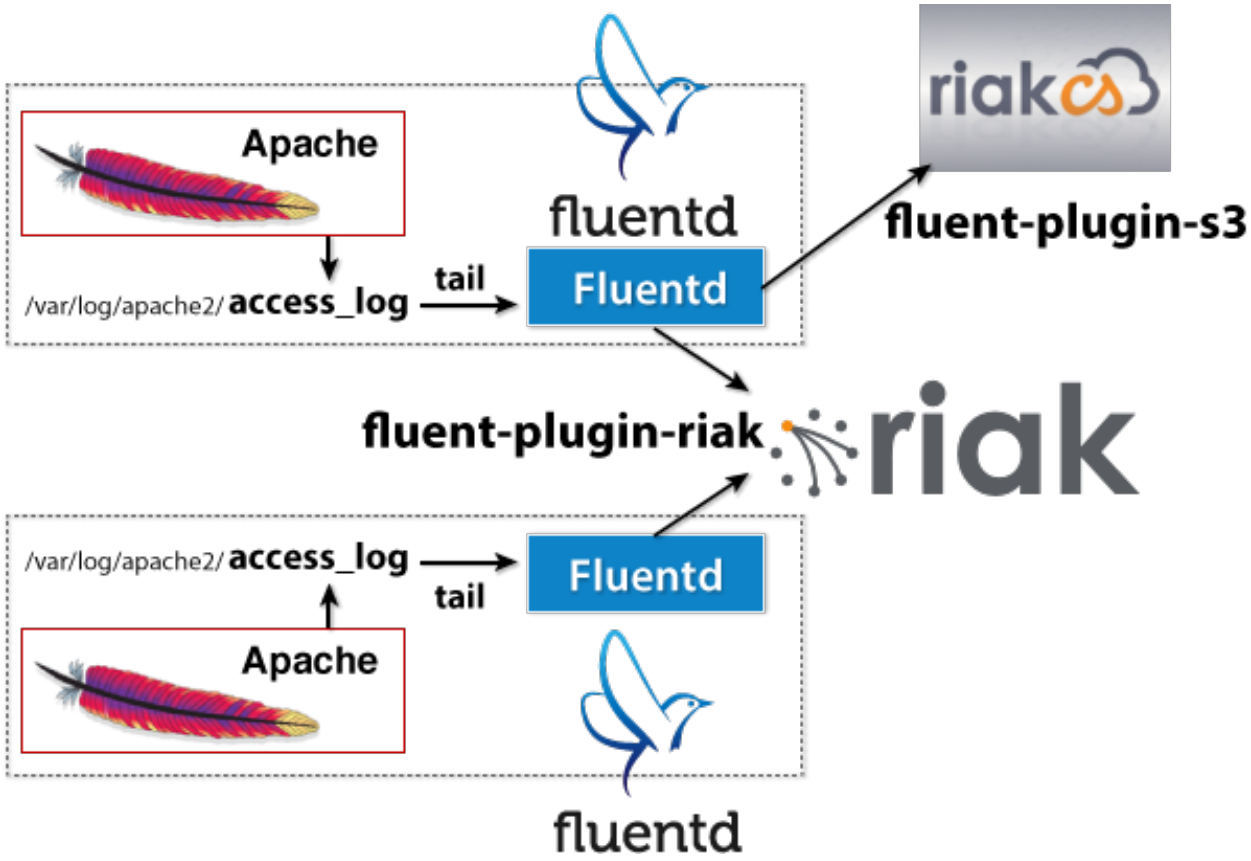
Fluentd + WebHDFS make real-time log collection simple, robust and scalable! [@tagomoris](http://github.com/tagomoris) has already been using this plugin to collect 20,000 msgs/sec, 1.5 TB/day without any major problems for several months now.

Learn More

- Fluentd Architecture
- Fluentd Get Started
- WebHDFS Output Plugin
- Slides: Fluentd and WebHDFS

Store Apache Logs into Riak

This article explains how to use Fluentd's Riak Output plugin (out_riak) to aggregate semi-structured logs in real-time.



Prerequisites

1. An OSX or Linux machine
2. Fluentd is installed (installation guide]
3. Riak is installed
4. An Apache web server log

Installing the Fluentd Riak Output Plugin

The Riak output plugin is used to output data from a Fluentd node to a Riak node.

Rubygems Users

Rubygems users can run the command below to install the plugin:

```
$ gem install fluent-plugin-riak
```

td-agent Users

If you are using td-agent, run `/usr/lib/fluent/ruby/bin/fluent-gem install fluent-plugin-riak` to install the Riak output plugin.

Configuring Fluentd

Create a configuration file called `fluent.conf` and add the following lines:

```
<source>
  type tail
  format apache2
  path /var/log/apache2/access_log
  pos_file /var/log/fluentd/apache2.access_log.pos
  tag riak.apache
</source>

<match riak.**>
  type riak
  buffer_type memory
  flush_interval 5s
  retry_limit 5
  retry_wait 1s
  nodes localhost:8087 # Assumes Riak is running locally on port 8087
</match>
```

The `<source>...</source>` section tells Fluentd to tail an Apache2-formatted log file located at `/var/log/apache2/access_log`. Each line is parsed as an Apache access log event and tagged with the `riak.apache` label.

The `<match riak.**>...</match>` section tells Fluentd to look for events whose tags start with `riak.` and send all matches to a Riak node located at `localhost:8087`. You can send events to multiple nodes by writing `nodes host1 host2 host3` instead.

Testing

Launch Fluentd with the following command:

```
$ fluentd -c fluentd.conf
```

NOTE: Please confirm that you have the file access permissions to (1) read the Apache log file and (2) write to `/var/log/fluentd/apache2.access_log.pos` (sudo-ing might help).

You should now see data coming into your Riak cluster. We can make sure that everything is running smoothly by hitting Riak's HTTP API:

```
$ curl http://localhost:8098/buckets/fluentlog/keys?keys=true
{"keys":["2014-01-23-d30b0698-b9de-4290-b8be-a66555497078", ...]}
$ curl http://localhost:8098/buckets/fluentlog/keys/2014-01-23-d30b0698-b9de-4290-b8be-a66555497078
[
  {
    "tag": "riak.apache",
```

```
    "time": "2004-03-08T01:23:54Z",
    "host": "64.242.88.10",
    "user": null,
    "method": "GET",
    "path": "/twiki/bin/statistics/Main",
    "code": 200,
    "size": 808,
    "referer": null,
    "agent": null
  }
]
```

There it is! (the response JSON is formatted for readability)

Learn More

- Fluentd Architecture
- Fluentd Get Started
- Riak Output Plugin

Collecting Log Data from Windows

In this article, we explain how to get started with collecting data from Windows machines.

As of v10, Fluentd does NOT support Windows. However, there are times when you must collect data streams from Windows machines. For example:

1. **Tailing log files on Windows**: collect and analyze log data from a Windows application.
2. **Collecting Windows Event Logs**: collect event logs from your Windows servers for system analysis, compliance checking, etc.

This setup has been tested on a 64-bit Windows 8 machine.

Prerequisites

1. nxlog, an open source log management tool that runs on Windows.
2. A Linux server (we assume Ubuntu 12 for this article)

Setup

Set up a Linux server with rsyslogd and Fluentd

1. Get hold of a Linux server. In this example, we assume it is Ubuntu.
2. **Make sure it has ports open for UDP. In the following example, we assume port 5140 is open.**
3. Install td-agent. (See here for various ways to install Fluentd/Treasure Agent)
4. Edit td-agent's configuration file located at `/etc/td-agent/td-agent.conf` and add the following lines

```
<source>
  type syslog
  port 5140
  tag windowslog
</source>
<match windowslog.**>
  type stdout
</match>
```

The above code listens to port 5140 (UDP) and outputs the data to stdout (which is piped to `/var/log/td-

5. Start td-agent by running `sudo service td-agent start`

Set up nxlog on Windows

1. Follow this link and download a copy of nxlog onto the Windows machine you want to collect log data from. Open the downloaded installer and follow the instructions. By default, it should be installed in `C:\Program Files (x86)\nxlog`
2. Create an nxlog config file as follows and save it as `nxlog.conf`:

```
define ROOT C:\Program Files (x86)\nxlog
Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log
<Extension syslog>
  Module      xm_syslog
</Extension>
<Extension json>
  Module      xm_json
</Extension>
<Input in>
  Module im_file
  File "<PATH TO THE LOG FILE YOU WANT TO TAIL>"
  SavePos TRUE
  InputType LineBased
</Input>
<Processor t>
  Module pm_transformer
  OutputFormat syslog_bsd
  Exec $Message=(": "+$raw_event);
</Processor>
<Output out>
  Module om_udp
  Host 54.200.236.1 # Your host name here
  Port 5140
</Output>
<Route r>
  Path in => t => out
</Route>
```

This configuration will send each line of the log file (see the File parameter inside <Input in&gt…</Input>) as a syslog message to a remote Fluentd/Treasure Agent instance.

Test

1. Go to nxlog's directory (in Powershell or Command Prompt) and run the following command:

   ```
   \nxlog.exe -f -c  <path to nxlog.conf>
   ```

   The "-f" option runs nxlog in the foreground (this is for testing). If this is for production, you would want to turn it into a Windows Service.

2. Once nxlog is running, add a new line "Windows is awesome" into the tailed file, and save it.

3. Now, go to the Linux server and run

   ```
   $ sudo tail -f /var/log/td-agent/td-agent.log
   ...
   ...
   ...
   2014-01-22 08:26:29 +0000 windowslog.user.notice: {"host":"portly","ident":"","message":"Windows is a
   ```

4. You successfully sent data from a Windows machine to a remote Fluentd instance running on Linux.

Next Step

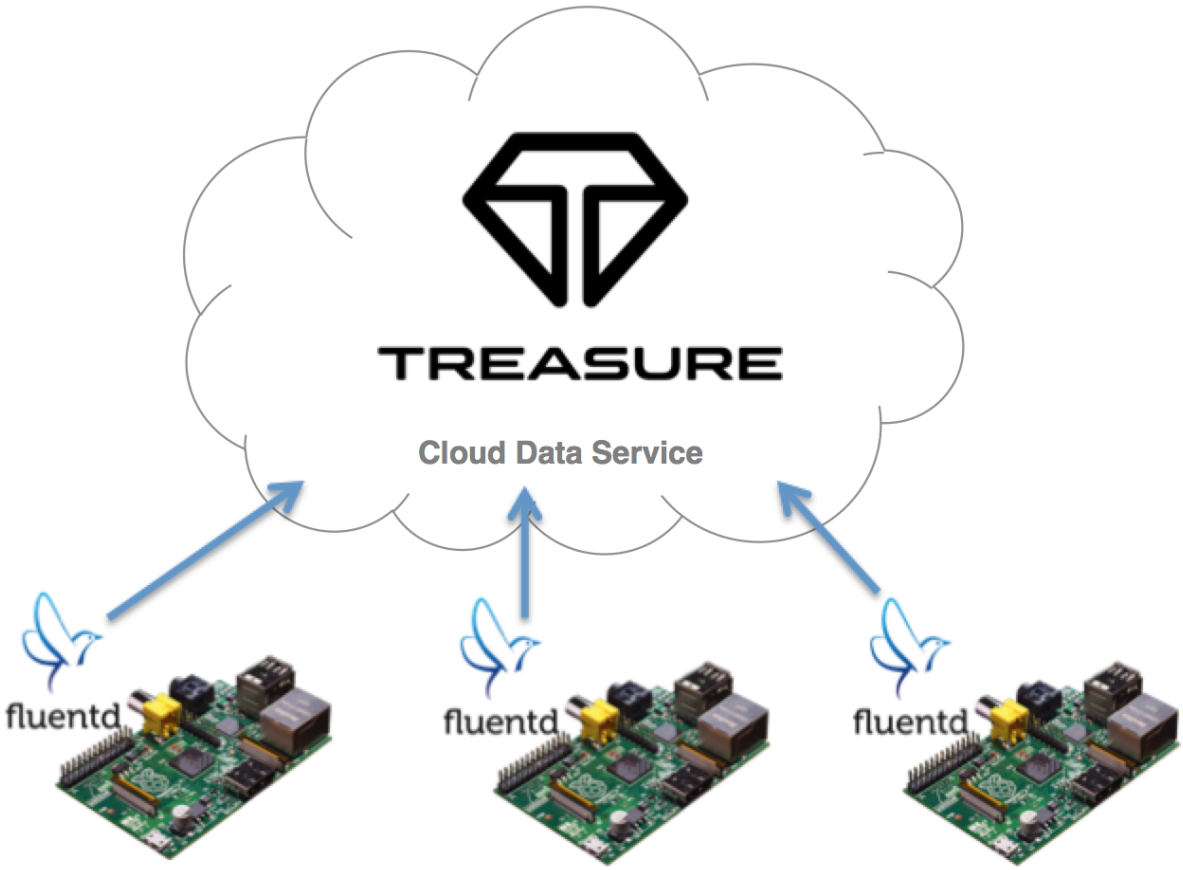
This example showed that we can collect data from a Windows machine and send it to a remote Fluentd instance. However, the data is not terribly useful because each line of data is placed into the "message" field as unstructured text. For production purposes, you would probably want to write a plugin/extend the syslog plugin so that you can parse the "message" field in the event.

Learn More

- Fluentd Architecture
- Fluentd Get Started

Cloud Data Logger by Raspberry Pi

Raspberry Pi is a credit-card-sized single-board computer. Because it is low-cost and easy to equip with various types of sensors, using Raspberry Pi as a cloud data logger is one of its ideal use cases.



This article introduces how to transport sensor data from Raspberry Pi to the cloud, using Fluentd as the data collector. For the cloud side, we'll use the Treasure Data cloud data service as an example, but you can use any cloud service in its place.

Install Raspbian

Raspbian is a free operating system based on Debian, optimized for the Raspberry Pi. Please install Raspbian on your Raspberry Pi by following the instructions in the blog post below:

- Getting Started with Raspberry Pi: Installing Raspbian

Install Fluentd

Next, we'll install Fluentd on Raspbian. Raspbian bundles Ruby 1.9.3 by default, but we need the extra development package to install Fluentd.

```
$ sudo aptitude install ruby-dev
```

We'll now install Fluentd and the necessary plugins.

```
$ sudo gem install fluentd
$ sudo fluent-gem install fluent-plugin-td
```

Configure and Launch Fluentd

Please sign up to Treasure Data from the sign up page. Its free plan lets you store and analyze millions of data points. You can get your account's API key from the users page.

Please prepare the `fluentd.conf` file with the following information, including your API key.

```
<match td.*.*>
  type tdlog
  apikey YOUR_API_KEY_HERE

  auto_create_table
  buffer_type file
  buffer_path /home/pi/fluentd/td
</match>
<source>
  type http
  port 8888
</source>
<source>
  type forward
</source>
```

Finally, please launch Fluentd via your terminal.

```
$ fluentd -c fluent.conf
```

Upload Test

To test the configuration, just post a JSON message to Fluentd via HTTP.

```
$ curl -X POST -d 'json={"sensor1":3123.13,"sensor2":321.3}' \
  http://localhost:8888/td.testdb.raspberrypi
```

NOTE: If you're using Python, you can use Fluentd's python logger library.

Now, access the databases page to confirm that your data has been uploaded to the cloud properly.

- Treasure Data: List of Databases

You can now issue queries against the imported data.

- Treasure Data: New Query

For example, these queries calculate the average sensor1 value and the sum of sensor2 values.

```
SELECT AVG(sensor1) FROM raspberrypi;
SELECT SUM(sensor2) FROM raspberrypi;
```

Conclusion

Raspberry Pi is an ideal platform for prototyping data logger hardware. Fluentd helps Raspberry Pi transfer the collected data to the cloud easily and reliably.

Learn More

- Fluentd Architecture
- Fluentd Get Started

Collecting GlusterFS Logs with Fluentd

This article shows how to use Fluentd to collect GlusterFS logs for analysis (search, analytics, troubleshooting, etc.)

Background

GlusterFS is an open source, distributed file system commercially supported by Red Hat, Inc. Each node in GlusterFS generates its own logs, and it's sometimes convenient to have these logs collected in a central location for analysis (e.g., When one GlusterFS node went down, what was happening on other nodes?).

Fluentd is an open source data collector for high-volume data streams. It's a great fit for monitoring GlusterFS clusters because:

1. Fluentd supports GlusterFS logs as a data source.
2. Fluentd supports various output systems (e.g., Elasticsearch, MongoDB, Treasure Data, etc.) that can help GlusterFS users analyze the logs.

The rest of this article explains how to set up Fluentd with GlusterFS. For this example, we chose Elasticsearch as the backend system.

Setting up Fluentd on GlusterFS Nodes

Step 1: Installing Fluentd

First, we'll install Fluentd using the following command:

```
$ curl -L http://toolbelt.treasuredata.com/sh/install-redhat.sh | sh
```

Next, we'll install the Fluentd plugin for GlusterFS:

```
$ sudo /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-glusterfs
Fetching: fluent-plugin-glusterfs-1.0.0.gem (100%)
Successfully installed fluent-plugin-glusterfs-1.0.0
1 gem installed
Installing ri documentation for fluent-plugin-glusterfs-1.0.0...
Installing RDoc documentation for fluent-plugin-glusterfs-1.0.0...
```

Step 2: Making GlusterFS Log Files Readable by Fluentd

By default, only `root` can read the GlusterFS log files. We'll allow others to read the file.

```
$ ls -alF /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
-rw------- 1 root root 1385  Feb  3 07:21 2014 /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
$ sudo chmod +r /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
$ ls -alF /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
-rw-r--r-- 1 root root 1385  Feb  3 07:21 2014 /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
```

Now, modify Fluentd's configuration file. It is located at `/etc/td-agent/td-agent.conf`.

NOTE: `td-agent` is Fluentd's rpm/deb package maintained by Treasure Data

This is what the configuration file should look like:

```
$ sudo cat /etc/td-agent/td-agent.conf

<source>
  type glusterfs_log
  path /var/log/glusterfs/etc-glusterfs-glusterd.vol.log
  pos_file /var/log/td-agent/etc-glusterfs-glusterd.vol.log.pos
  tag glusterfs_log.glusterd
  format /^(?<message>.*)$/
</source>

<match glusterfs_log.**>
  type forward
  send_timeout 60s
  recover_wait 10s
  heartbeat_interval 1s
  phi_threshold 8
  hard_timeout 60s

  <server>
    name logserver
    host 172.31.10.100
    port 24224
    weight 60
  </server>

  <secondary>
    type file
    path /var/log/td-agent/forward-failed
  </secondary>
</match>
```

NOTE: the … section is for failover (when the aggregator instance at 172.31.10.100:24224 is unreachable).

Finally, start td-agent. Fluentd will started with the updated setup.

```
$ sudo service td-agent start
Starting td-agent:                                         [  OK  ]
```

Step 3: Setting Up the Aggregator Fluentd Server

We'll now set up a separate Fluentd instance to aggregate the logs. Again, the first step is to install Fluentd.

```
$ curl -L http://toolbelt.treasuredata.com/sh/install-redhat.sh | sh
```

We'll set up the node to send data to Elasticsearch, where the logs will be indexed and written to local disk for backup.

First, install the Elasticsearch output plugin as follows:

```
$ sudo /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-glusterfs
```

Then, configure Fluentd as follows:

```
$ sudo cat /etc/td-agent/td-agent.conf
<source>
  type forward
  port 24224
  bind 0.0.0.0
</source>

<match glusterfs_log.glusterd>
  type copy

  #local backup
  <store>
    type file
    path /var/log/td-agent/glusterd
  </store>

  #Elasticsearch
  <store>
    type elasticsearch
    host ELASTICSEARCH_URL_HERE
    port 9200
    index_name glusterfs
    type_name fluentd
    logstash_format true
  </store>
</match>
```

That's it! You should now be able to search and visualize your GlusterFS logs with Kibana.

Acknowledgement

This article is inspired by Daisuke Sasaki's article on Classmethod's website. Thanks Daisuke!

Learn More

- Fluentd Architecture
- Fluentd Get Started
- GlusterFS Input Plugin

Configuration File

This article describes the basic concepts of Fluentd's configuration file.

If you want to know V1 configuration format, please jump to V1 Format section.

Overview

The configuration file allows the user to control the input and output behavior of Fluentd by (1) selecting input and output plugins and (2) specifying the plugin parameters. The file is required for Fluentd to operate properly.

Config File Location

RPM or Deb If you installed Fluentd using the rpm or deb packages, the config file is located at /etc/td-agent/td-agent.conf. `sudo /etc/init.d/td-agent reload` will reload the config file.

```
$ sudo vi /etc/td-agent/td-agent.conf
```

Gem If you installed Fluentd using the Ruby Gem, you can create the configuration file using the following commands. Sending a SIGHUP signal will reload the config file.

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

List of Directives

The configuration file consists of the following directives:

1. **source** directives determine the input sources.
2. **match** directives determine the output destinations.
3. **include** directives include other files.

Let's actually create a configuration file step by step.

(1) "source": where all the data come from

Fluentd's input sources are enabled by selecting and configuring the desired input plugins using **source** directives. Fluentd's standard input plugins include `http` and `forward`. `http` turns fluentd into an HTTP endpoint to accept incoming HTTP messages whereas `forward` turns fluentd into a TCP endpoint to accept TCP packets. Of course, it can be both at the same time (You can add as many sources as you wish)

```
# Receive events from 24224/tcp
# This is used by log forwarding and the fluent-cat command
<source>
  type forward
  port 24224
</source>
```

```
# http://this.host:9880/myapp.access?json={"event":"data"}
<source>
  type http
  port 9880
</source>
```

Each **source** directive must include a `type` parameter. The `type` parameter specifies which input plugin to use.

Interlude: Routing The `source` submits events into the Fluentd's routing engine. An event consists of three entities: **tag**, **time** and **record**. The tag is a string separated by '.'s (e.g. myapp.access), and is used as the directions for Fluentd's internal routing engine. The time field is specified by input plugins, and it must be in the Unix time format. The record is a JSON object.

In the example above, the HTTP input plugin submits the following event::

```
# generated by http://this.host:9880/myapp.access?json={"event":"data"}
tag: myapp.access
time: (current time)
record: {"event":"data"}
```

Didn't find your input source? You can write your own plugin! You can add new input sources by writing your own plugins. For further information regarding Fluentd's input sources, please refer to the Input Plugin Overview article.

(2) "match": Tell fluentd what to do!

The "match" directive looks for events with *match*ing tags and processes them. The most common use of the match directive is to output events to other systems (for this reason, the plugins that correspond to the match directive are called "output plugins"). Fluentd's standard output plugins include `file` and `forward`. Let's add those to our configuration file.

```
# Receive events from 24224/tcp
# This is used by log forwarding and the fluent-cat command
<source>
  type forward
  port 24224
</source>

# http://this.host:9880/myapp.access?json={"event":"data"}
<source>
  type http
  port 9880
</source>

# Match events tagged with "myapp.access" and
# store them to /var/log/fluent/access.%Y-%m-%d
# Of course, you can control how you partition your data
# with the time_slice_format option.
<match myapp.access>
  type file
  path /var/log/fluent/access
</match>
```

Each **match** directive must include a match pattern and a `type` parameter. Only events with a **tag** matching the pattern will be sent to the output destination (in the above example, only the events with the tag "myapp.access" is matched). The `type` parameter specifies the output plugin to use.

Just like input sources, you can add new output destinations by writing your own plugins. For further information regarding Fluentd's output destinations, please refer to the Output Plugin Overview article.

Match Pattern: how you control the event flow inside fluentd

The following match patterns can be used for the `<match>` tag.

- `*` matches a single tag part.
- For example, the pattern `a.*` matches `a.b`, but does not match `a` or `a.b.c`
- `**` matches zero or more tag parts.
- For example, the pattern `a.**` matches `a`, `a.b` and `a.b.c`
- `{X,Y,Z}` matches X, Y, or Z, where X, Y, and Z are match patterns.
- For example, the pattern `{a,b}` matches `a` and `b`, but does not match `c`
- This can be used in combination with the `*` or `**` patterns. Examples include `a.{b,c}.*` and `a.{b,c.**}`
- When multiple patterns are listed inside one `<match>` tag (delimited by one or more whitespaces), it matches any of the listed patterns. For example:
- The patterns `<match a b>` match `a` and `b`.
- The patterns `<match a.** b.*>` match `a`, `a.b`, `a.b.c.` (from the first pattern) and `b.d` (from the second pattern).

Match Order

Fluentd tries to match tags in the order that they appear in the config file. So if you have the following configuration:

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

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

then `myapp.access` is never matched. Wider match patterns should be defined after tight match patterns.

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

```
# Capture all unmatched tags. Good :)
<match **>
  type blackhole_plugin
</match>
```

(3) Re-use your config: the "include" directive

Directives in separate configuration files can be imported using the **include** directive::

```
# Include config files in the ./config.d directory
include config.d/*.conf
```

The **include** directive supports regular file path, glob pattern, and http URL conventions::

```
# 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
```

Supported Data Types for Values

Each Fluentd plugin has a set of parameters. For example, in_tail has parameters such as `rotate_wait` and `pos_file`. Each parameter has a specific type associated with it. They are defined as follows:

NOTE: Each parameter's type should be documented. If not, please let the plugin author know.

- `string` type: the field is parsed as a string. This is the most "generic" type, where each plugin decides how to process the string.
- `integer` type: the field is parsed as an integer.
- `float` type: the field is parsed as a float.
- `size` type: the field is parsed as the number of bytes. There are several notational variations:
  - If the value matches `<INTEGER>k` or `<INTEGER>K`, then the value is the INTEGER number of kilobytes.
  - If the value matches `<INTEGER>m` or `<INTEGER>M`, then the value is the INTEGER number of megabytes.
  - If the value matches `<INTEGER>g` or `<INTEGER>G`, then the value is the INTEGER number of gigabytes.
  - If the value matches `<INTEGER>t` or `<INTEGER>T`, then the value is the INTEGER number of terabytes.
  - Otherwise, the field is parsed as integer, and that integer is the number of bytes.
- `time` type: the field is parsed as a time duration.

  - If the value matches `<INTEGER>s`, then the value is the INTEGER seconds.
  - If the value matches `<INTEGER>m`, then the value is the INTEGER minutes.
  - If the value matches `<INTEGER>h`, then the value is the INTEGER hours.
  - If the value matches `<INTEGER>d`, then the value is the INTEGER days.
  - Otherwise, the field is parsed as float, and that float is the number of seconds. This option is useful for specifying sub-second time durations such as "0.1" (=0.1 second = 100ms).
- `array` type: the field is parsed as a JSON array (since v0.10.46)
- `hash` type: the field is parsed as a JSON object (since v0.10.46)

`array` and `hash` are JSON because almost programming languages and infrastructure tools can generate JSON value easily than unusual format.

V1 Format

You can enable V1 configuration format by passing `--use-v1-config` option to fluentd.

Basic structures are same as old format, e.g. `<source>`, `<match>` and tag matching. This section describes the different points between V1 and old format.

NOTE: V1 means Fluentd v1(Release plan). Fluentd v1 will use this new format by default. So Fluentd provides new format prior to v1 release for smooth and gradual transition.

Multi line support for array and hash values

V1 format allows multi line value for array and hash values.

```
array_param [
  "a", "b"
]
hash_param {
  "k":"v",
  "k1":10
}
```

`"foo"` is interpreted as `foo`, not `"foo"`

" is a quote character of string value. It causes the different behaviour beteen V1 and old format.

```
str_param "foo"
```

- In V1, str_param is `foo`
- In old, str_param is `"foo"`

Allow `#` in string value

`#` is a comment symbol in both V1 and old format. But `#` is allowed in string value in V1 format.

```
str_param foo#bar
```

- In V1, str_param is `foo#bar`
- In old, str_param is `foo`

Embedded Ruby code

You can evaluate the Ruby code with `#{}` in `"` quoted string. This is useful for setting machine information like hostname.

```
host_param "#{Socket.gethostname}" # host_param is actual hostname like `webserver1`.
```

NOTE: config-xxx mixins use "${}", not "#{}". These embedded configurations are two different things.

\ is escape character

\ is interpreted as escape character. You need \ for setting `"`, `\r`, `\n`, `\t`, \ or several characters in parameter.

```
str_param foo\nbar # \n is intereted as actual LF chatacter
```

So if you have \ in your regexp format, you should write `\\` instead of \.

Logging of Fluentd

This article describes Fluentd's logging mechanism.

Fluentd has two log layers: global and per plugin. Different log levels can be set for global logging and plugin level logging.

Log Level

Shown below is the list of supported values, in increasing order of verbosity:

- `fatal`
- `error`
- `warn`
- `info`
- `debug`
- `trace`

The default log level is `info`, and Fluentd outputs `info`, `warn`, `error` and `fatal` logs by default.

Global Logs

Global logging is used by Fluentd core and plugins that don't set their own log levels. The global log level can be adjusted up or down.

Increase Verbosity Level

The `-v` option sets the verbosity to `debug` while the `-vv` option sets the verbosity to `trace`.

```
$ fluentd -v  ... # debug level
$ fluentd -vv ... # trace level
```

These options are useful for debugging purposes.

Decrease Verbosity Level

The `-q` option sets the verbosity to `warn` while the `-qq` option sets the verbosity to `error`.

```
$ fluentd -q  ... # warn level
$ fluentd -qq ... # error level
```

Per Plugin Log (Fluentd v0.10.43 and above)

The `log_level` option sets different levels of logging for each plugin. It can be set in each plugin's configuration file.

For example, in order to debug in_tail but suppress all but fatal log messages for in_http, their respective `log_level` options should be set as follows:

```
<source>
    type tail
    log_level debug
    path /var/log/data.log
    ...
</source>
<source>
    type http
    log_level fatal
</source>
```

If you don't specify the `log_level` parameter, the plugin will use the global log level.

NOTE: Some plugins haven't supported per-plugin logging yet. The logging section of the Plugin Development article explains how to update such plugins to support the new log level system.

Suppress repeated stacktrace

Fluentd can suppress same stacktrace with `--suppress-repeated-stacktrace`. For example, if you pass `--suppress-repeated-stacktrace` to fluentd:

```
2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:154:rescue in emit_stream: emit transaction failed  erro
 2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:140:emit_stream: /Users/repeatedly/devel/fluent/fluer
  [snip]
 2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:140:emit_stream: /Users/repeatedly/devel/fluent/fluer
2013-12-04 15:05:53 +0900 [error]: plugin/in_object_space.rb:113:rescue in on_timer: object space failed to
2013-12-04 15:05:55 +0900 [warn]: fluent/engine.rb:154:rescue in emit_stream: emit transaction failed  erro
 2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:140:emit_stream: /Users/repeatedly/devel/fluent/fluer
  [snip]
```

logs are changed to:

```
2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:154:rescue in emit_stream: emit transaction failed  erro
 2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:140:emit_stream: /Users/repeatedly/devel/fluent/fluer
  [snip]
 2013-12-04 15:05:53 +0900 [warn]: fluent/engine.rb:140:emit_stream: /Users/repeatedly/devel/fluent/fluer
2013-12-04 15:05:53 +0900 [error]: plugin/in_object_space.rb:113:rescue in on_timer: object space failed to
2013-12-04 15:05:55 +0900 [warn]: fluent/engine.rb:154:rescue in emit_stream: emit transaction failed  erro
 2013-12-04 15:05:55 +0900 [warn]: plugin/in_object_space.rb:111:on_timer: suppressed same stacktrace
```

Same stacktrace is replaced with `suppressed same stacktrace` message until other stacktrace is received.

Output to log file

Fluentd outputs logs to `STDOUT` by default. To output to a file instead, please specify the `-o` option.

```
$ fluentd -o /path/to/log_file
```

NOTE: Fluentd doesn't support log rotation yet.

Capture Fluentd logs

Fluentd marks its own logs with the `fluent` tag. You can process Fluentd logs by using `<match fluent.**>`. If you define `<match fluent.**>` in your configuration, then Fluend will send its own logs to this match destination. This is useful for monitoring Fluentd logs.

For example, if you have the following `<match fluent.**>`:

```
# omit other source / match
<match fluent.**>
  type stdout
</match>
```

then Fluentd outputs `fluent.info` logs to stdout like below:

```
2014-02-27 00:00:00 +0900 [info]: shutting down fluentd
2014-02-27 00:00:01 +0900 fluent.info: { "message":"shutting down fluentd"} # by <match fluent.**>
2014-02-27 00:00:01 +0900 [info]: process finished code = 0
```

In production, you can use out_forward to send Fluentd logs to a monitoring server. The monitoring server can then filter and send the logs to your notification system: chat, irc, etc.

Leaf server example:

```
# Add hostname for identifing the server and tag to filter by log level
<match fluent.**>
  type record_modifier
  tag internal.message
  host ${hostname}
  include_tag_key
  tag_key original_tag
</match>

<match internal.message>
  type forward
  <server>
    # Monitoring server parameters
  </server>
</match>
```

Monitoring server example:

```
# Ignore trace, debug and info log
<match internal.message>
  type grep
  regexp1 original_tag fluent.(warn|error|fatal)
  add_tag_prefix filtered
</match>

<match filtered.internal.message>
  # your notification setup. This example uses irc plugin
  type irc
  host irc.domain
  channel notify
  message notice: %s [%s] @%s %s
  out_keys original_tag,time,host,message
</match>
```

If an error occurs, you will get a notification message in your irc `notify` channel.

```
01:01 fluentd: [11:10:24] notice: fluent.warn [2014/02/27 01:00:00] @leaf.server.domain detached forwardir
```

Monitoring Fluentd

This article explains how to monitor the `Fluentd` daemon.

Monitoring Agent

Fluentd has a monitoring agent to retrieve internal metrics in JSON via HTTP. Please add the following lines to your configuration file.

```
<source>
  type monitor_agent
  bind 0.0.0.0
  port 24220
</source>
```

Next, please restart the agent and get the metrics via HTTP.

```
$ curl http://host:24220/api/plugins.json
{"plugins":[{"plugin_id":"object:3fec669d6ac4","type":"forward","output_plugin":false,"config":{"type":
```

Process Monitoring

Two `ruby` processes (parent and child) are executed. Please make sure that these processes are running. The example for `td-agent` is shown below.

```
/usr/lib/fluent/ruby/bin/ruby /usr/sbin/td-agent
  --daemon /var/run/td-agent/td-agent.pid
  --log /var/log/td-agent/td-agent.log
```

For td-agent on Linux, you can check the process statuses with the following command. Two processes should be shown if there are no issues.

```
$ ps w -C ruby -C td-agent --no-heading
32342 ?      Sl    0:00 /usr/lib/fluent/ruby/bin/ruby /usr/sbin/td-agent --daemon /var/run/td-agent/td-ager
32345 ?      Sl    0:01 /usr/lib/fluent/ruby/bin/ruby /usr/sbin/td-agent --daemon /var/run/td-agent/td-ager
```

Port Monitoring

Fluentd opens several ports according to the configuration file. We recommend checking the availability of these ports. The default port settings are shown below:

- TCP 0.0.0.0 8888 (HTTP by default)
- TCP 0.0.0.0 24224 (Forward by default)

Debug Port

A debug port for local communication is recommended for trouble shooting. Please note that the configuration below will be required.

```
<source>
  type debug_agent
  bind 127.0.0.1
  port 24230
</source>
```

You can attach the process using the `fluent-debug` command through dRuby.

Fluentd's Signal Handling

This article explains how `Fluentd` handles UNIX signals.

Process Model

When you launch Fluentd, it creates two processes: supervisor and worker. The supervisor process controls the life cycle of the worker process. Please make sure to send any signals to the supervisor process.

Signals

SIGINT or SIGTERM

Stops the daemon gracefully. Fluentd will try to flush the entire memory buffer at once, but will not retry if the flush fails. Fluentd will not flush the file buffer; the logs are persisted on the disk by default.

SIGUSR1

Forces the buffered logs to be flushed and reopens Fluentd's log. All buffers (including memory and file) will be flushed. Fluentd will try to flush the entire memory buffer at once, but will not retry if the flush fails.

SIGHUP

Reloads the configuration file by gracefully restarting the worker process. Fluentd will try to flush the entire memory buffer at once, but will not retry if the flush fails. Fluentd will not flush the file buffer; the logs are persisted on the disk by default.

Fluentd High Availability Configuration

For high-traffic websites, we recommend using a high availability configuration of `Fluentd`.

Message Delivery Semantics

Fluentd is designed primarily for event-log delivery systems.

In such systems, several delivery guarantees are possible:

- *At most once*: Messages are immediately transferred. If the transfer succeeds, the message is never sent out again. However, many failure scenarios can cause lost messages (ex: no more write capacity)
- *At least once*: Each message is delivered at least once. In failure cases, messages may be delivered twice.
- *Exactly once*: Each message is delivered once and only once. This is what people want.

If the system "can't lose a single event", and must also transfer "*exactly once*", then the system must stop ingesting events when it runs out of write capacity. The proper approach would be to use synchronous logging and return errors when the event cannot be accepted.

That's why *Fluentd guarantees 'At most once' transfers*. In order to collect massive amounts of data without impacting application performance, a data logger must transfer data asynchronously. This improves performance at the cost of potential delivery failures.

However, most failure scenarios are preventable. The following sections describe how to set up Fluentd's topology for high availability.

Network Topology

To configure Fluentd for high availability, we assume that your network consists of '*log forwarders*' and '*log aggregators*'.

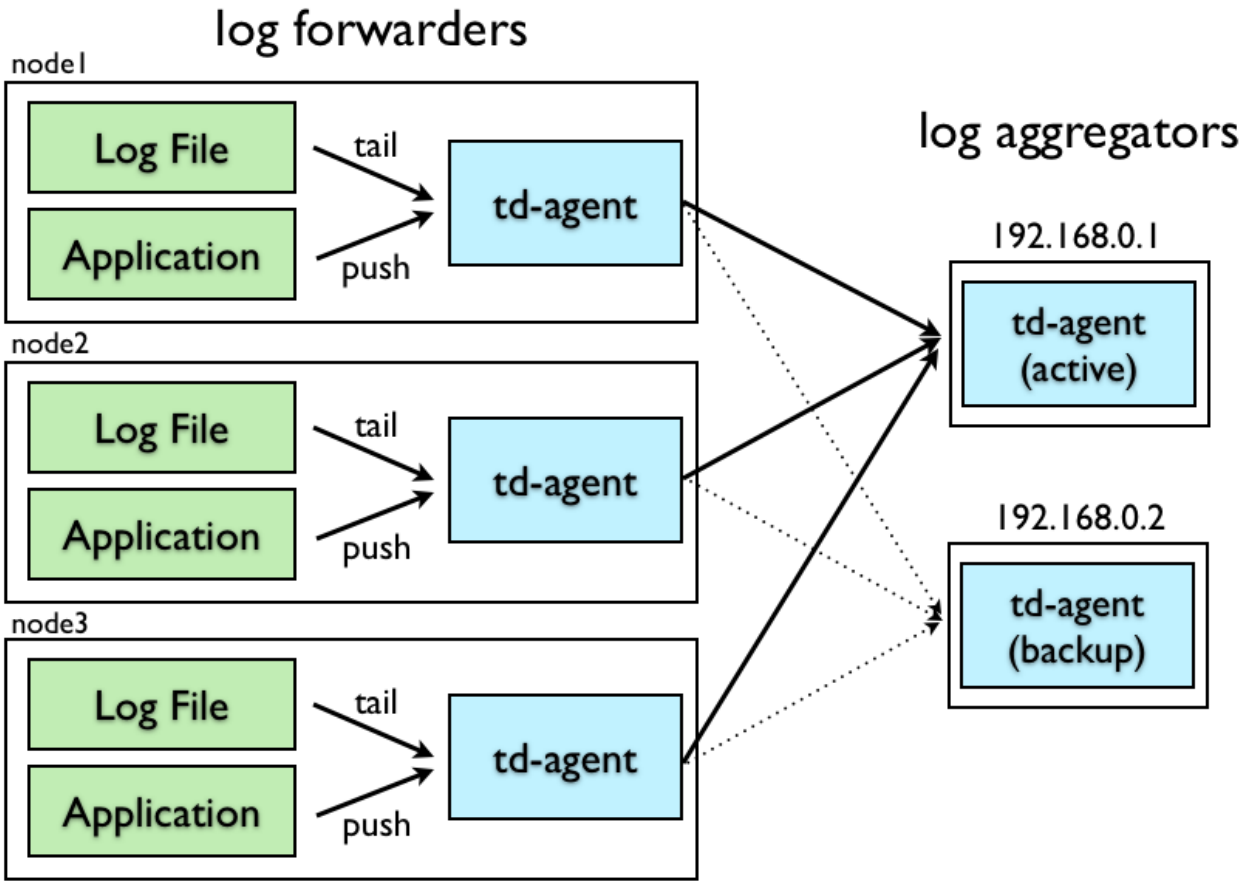
'*log forwarders*' are typically installed on every node to receive local events. Once an event is received, they forward it to the 'log aggregators' through the network.

'*log aggregators*' are daemons that continuously receive events from the log forwarders. They buffer the events and periodically upload the data into the cloud.

Fluentd can act as either a log forwarder or a log aggregator, depending on its configuration. The next sections describes the respective setups. We assume that the active log aggregator has ip '192.168.0.1' and that the backup has ip '192.168.0.2'.

Log Forwarder Configuration

Please add the following lines to your config file for log forwarders. This will configure your log forwarders to transfer logs to log aggregators.


log forwarders
node1
Log File
Application
tail
push
td-agent
node2
Log File
Application
tail
push
td-agent
node3
Log File
Application
tail
push
td-agent
log aggregators
192.168.0.1
td-agent
(active)
192.168.0.2
td-agent
(backup)

```
# TCP input
<source>
  type forward
  port 24224
</source>

# HTTP input
<source>
  type http
  port 8888
</source>

# Log Forwarding
<match mytag.**>
  type forward

  # primary host
  <server>
    host 192.168.0.1
    port 24224
  </server>
  # use secondary host
  <server>
    host 192.168.0.2
    port 24224
    standby
  </server>

  # use longer flush_interval to reduce CPU usage.
  # note that this is a trade-off against latency.
  flush_interval 60s
</match>
```

When the active aggregator (192.168.0.1) dies, the logs will instead be sent to the backup aggregator (192.168.0.2). If both servers die, the logs are buffered on-disk at the corresponding forwarder nodes.

Log Aggregator Configuration

Please add the following lines to the config file for log aggregators. The input source for the log transfer is TCP.

```
# Input
<source>
  type forward
  port 24224
</source>

# Output
<match mytag.**>
  ...
</match>
```

The incoming logs are buffered, then periodically uploaded into the cloud. If upload fails, the logs are stored on the local disk until the retransmission succeeds.

Failure Case Scenarios

Forwarder Failure

When a log forwarder receives events from applications, the events are first written into a disk buffer (specified by buffer_path). After every flush_interval, the buffered data is forwarded to aggregators.

This process is inherently robust against data loss. If a log forwarder's fluentd process dies, the buffered data is properly transferred to its aggregator after it restarts. If the network between forwarders and aggregators breaks, the data transfer is automatically retried.

However, possible message loss scenarios do exist:

- The process dies immediately after receiving the events, but before writing them into the buffer.
- The forwarder's disk is broken, and the file buffer is lost.

Aggregator Failure

When log aggregators receive events from log forwarders, the events are first written into a disk buffer (specified by buffer_path). After every flush_interval, the buffered data is uploaded into the cloud.

This process is inherently robust against data loss. If a log aggregator's fluentd process dies, the data from the log forwarder is properly retransferred after it restarts. If the network between aggregators and the cloud breaks, the data transfer is automatically retried.

However, possible message loss scenarios do exist:

- The process dies immediately after receiving the events, but before writing them into the buffer.
- The aggregator's disk is broken, and the file buffer is lost.

Trouble Shooting

"no nodes are available"

Please make sure that you can communicate with port 24224 using **not only TCP, but also UDP**. These commands will be useful for checking the network configuration.

```
$ telnet host 24224
$ nmap -p 24224 -sU host
```

Please note that there is one known issue where VMware will occasionally lose small UDP packages used for heartbeat.

Failure Scenarios

This article lists various Fluentd failure scenarios. We will assume that you have configured Fluentd for High Availability, so that each app node has its local *forwarders* and all logs are aggregated into multiple *aggregators*.

Apps Cannot Post Records to Forwarder

In the failure scenario, the application sometimes fails to post records to its local Fluentd instance when using logger libraries of various languages. Depending on the maturity of each logger library, some clever mechanisms have been implemented to prevent data loss.

1) Memory Buffering (available for Ruby, Java, Python, Perl) If the destination Fluentd instance dies, certain logger implementations will use extra memory to hold incoming logs. When Fluentd comes back, these loggers will automatically send out the buffered logs to Fluentd again. Once the maximum buffer memory size is reached, most current implementations will write the data onto the disk or throw away the logs.

2) Exponential Backoff (available for Ruby, Java) When trying to resend logs to the local forwarder, some implementations will use exponential backoff to prevent excessive re-connect requests.

Forwarder or Aggregator Fluentd Goes Down

What happens when a Fluentd process dies for some reason? It depends on your buffer configuration.

buf_memory If you're using buf_memory, the buffered data is completely lost. This is a tradeoff for higher performance. Lowering the flush_interval will reduce the probability of losing data, but will increase the number of transfers between forwarders and aggregators.

buf_file If you're using buf_file, the buffered data is stored on the disk. After Fluentd recovers, it will try to send the buffered data to the destination again.

Please note that the data will be lost if the buffer file is broken due to I/O errors. The data will also be lost if the disk is full, since there is nowhere to store the data on disk.

Storage Destination Goes Down

If the storage destination (e.g. Amazon S3, MongoDB, HDFS, etc.) goes down, Fluentd will keep trying to resend the buffered data. The retry logic depends on the plugin implementation.

If you're using buf_memory, aggregators will stop accepting new logs once they reach their buffer limits. If you're using buf_file, aggregators will continue accepting logs until they run out of disk space.

Performance Tuning

Check top command

If Fluentd doesn't perform as well as you had expected, please check the `top` command first. You need to identify which part of your system is the bottleneck (CPU? Memory? Disk I/O? etc).

Multi Process

The CPU is often the bottleneck for Fluentd instances that handle billions of incoming records. To utilize multiple CPU cores, we recommend using the `in_multiprocess` plugin.

- in_multiprocess

Plugin Management

This article explains how to manage Fluentd plugins, including adding 3rd party plugins.

fluent-gem

The `fluent-gem` command is used to install Fluentd plugins. This is a wrapper around the `gem` command.

```
fluent-gem install fluent-plugin-grep
```

NOTE: Ruby doesn't guarantee C extension API compatibility between its major versions. If you update Fluentd's Ruby version, you should re-install the plugins that depend on C extension.

If Using td-agent, Use /usr/lib/fluent/ruby/bin/fluent-gem

If you are using td-agent, please make sure to use td-agent's `fluent-gem` command. Othrewise (e.g. you use the command belonging to system, rvm, etc.), you won't be able to find your "installed" plugins.

Please see this FAQ for more information.

"-p" option

Fluentd's `-p` option is used to add an extra plugin directory to the load path. For example, if you put the `out_foo.rb` plugin into `/path/to/plugin`, you can load the `out_foo.rb` plugin by specifying the `-p` option as shown below.

```
fluentd -p /path/to/plugin
```

You can specify the `-p` option more than once.

Add a Plugin Via /etc/fluent/plugin

Fluentd adds the `/etc/fluent/plugin` directory to its load path by default. Thus, any additional plugins that are placed in `/etc/fluent/plugin` will be loaded automatically.

If Using td-agent, Use /etc/td-agent/plugin

If you are using td-agent, Fluentd uses the `/etc/td-agent/plugin` directory instead of `/etc/fluent/plugin`. Please put your plugins here instead.

"–gemfile" option

A Ruby application manages gem dependencies using Gemfile and Bundler. Fluentd's `--gemfile` option takes the same approach, and is useful for managing plugin versions separated from shared gems.

For example, if you have following Gemfile at /etc/fluent/Gemfile:

```
source 'https://rubygems.org'

gem 'fluentd', '0.10.43'
gem 'fluent-plugin-grep', '0.3.2'
gem 'fluent-plugin-elasticsearch', '0.2.0'
```

You can pass this Gemfile to Fluentd via the `--gemfile` option.

```
fluentd --gemfile /etc/fluent/Gemfile
```

When specifying the `--gemfile` option, Fluentd will try to install the listed gems using Bundler. Fluentd will only load listed gems separated from shared gems, and will also prevent unexpected plugin updates.

In addition, if you update Fluetnd's Ruby version, Bundler will re-install the listed gems for the new Ruby version. This allows you to avoid the C extention API compatibility problem.

Troubleshooting Fluentd

Look at Logs

If things aren't happening as expected, please first look at your logs. For td-agent (rpm/deb), the logs are located at `/var/log/td-agent/td-agent.log`.

Turn on Verbose Logging

You can get more information about the logs if verbose logging is turned on. Please follow the steps below.

rpm

1. edit `/etc/init.d/td-agent`
2. add `-vv` to TD_AGENT_ARGS
3. restart td-agent

```
# at /etc/init.d/td-agent
...
TD_AGENT_ARGS="... -vv"
...
```

deb

1. edit `/etc/init.d/td-agent`
2. add `-vv` to DAEMON_ARGS
3. restart td-agent

```
# at /etc/init.d/td-agent
...
DAEMON_ARGS="... -vv"
...
```

gem

Please add `-vv` to your command line.

```
$ fluentd .. -vv
```

Input Plugin Overview

Fluentd has 3 types of plugins: Input, Output, and Buffer. This article provides an overview of Input plugins.

Overview

Input plugins extend Fluentd to retrieve and pull event logs from external sources. An input plugin typically creates a thread socket and a listen socket. It can also be written to periodically pull data from data sources.

List of Input Plugins

- in_forward
- in_http
- in_tail
- in_exec
- in_syslog
- in_scribe

Other Input Plugins

Please refer to this list of available plugins to find out about other Input plugins.

- Fluentd plugins

forward Input Plugin

The `in_forward` Input plugin listens to a TCP socket to receive the event stream. It also listens to an UDP socket to receive heartbeat messages.

This plugin is mainly used to receive event logs from other Fluentd instances, the fluent-cat command, or client libraries. This is by far the most efficient way to retrieve the records.

Example Configuration

`in_forward` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type forward
  port 24224
  bind 0.0.0.0
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required) The value must be `forward`.

port The port to listen to. Default Value = 24224

bind The bind address to listen to. Default Value = 0.0.0.0 (all addresses)

linger_timeout The timeout time used to set linger option. The default is 0

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Protocol

This plugin accepts both JSON or MessagePack messages and automatically detects which is used. Internally, Fluent uses MessagePack as it is more efficient than JSON.

The time value is a platform specific integer and is based on the output of Ruby's `Time.now.to_i` function. On Linux, BSD and MAC systems, this is the number of seconds since 1970.

Multiple messages may be sent in the same connection.

```
stream:
  message...

message:
  [tag, time, record]
  or
  [tag, [[time,record], [time,record], ...]]

example:
  ["myapp.access", 1308466941, {"a":1}]["myapp.messages", 1308466942, {"b":2}]
  ["myapp.access", [[1308466941, {"a":1}], [1308466942, {"b":2}]]]
```

Secure Forward Input

The `in_secure_forward` input plugin accepts messages via **SSL with authentication** (cf. out_secure_forward).

Example Configurations

This section provides some example configurations for `in_secure_forward`.

Minimalist Configuration

```
<source>
  type secure_forward
  shared_key         secret_string
 self_hostname    server.fqdn.local  # This fqdn is used as CN (Common Name) of certificates
  cert_auto_generate yes                # This parameter MUST be specified
</source>
```

Check username/password from Clients

```
<source>
  type secure_forward
  shared_key         secret_string
  self_hostname      server.fqdn.local
  cert_auto_generate yes
  authentication     yes # Deny clients without valid username/password
  <user>
    username tagomoris
    password foobar012
  </user>
  <user>
    username frsyuki
    password yakiniku
  </user>
</source>
```

Deny Unknown Source IP/hosts

```
<source>
  type secure_forward
  shared_key         secret_string
  self_hostname      server.fqdn.local
  cert_auto_generate     yes
  allow_anonymous_source no  # Allow to accept from nodes of <client>
  <client>
    host 192.168.10.30
    # network address (ex: 192.168.10.0/24) NOT Supported now
  </client>
  <client>
    host your.host.fqdn.local
    # wildcard (ex: *.host.fqdn.local) NOT Supported now
  </client>
</source>
```

You can use the username/password check and client check together:

```
<source>
  type secure_forward
  shared_key         secret_string
  self_hostname      server.fqdn.local
  cert_auto_generate     yes
```

```
  allow_anonymous_source no  # Allow to accept from nodes of <client>
  authentication         yes # Deny clients without valid username/password
  <user>
    username tagomoris
    password foobar012
  </user>
  <user>
    username frsyuki
    password sukiyaki
  </user>
  <user>
    username repeatedly
    password sushi
  </user
  <client>
    host 192.168.10.30      # allow all users to connect from 192.168.10.30
  </client>
  <client>
    host  192.168.10.31
    users tagomoris,frsyuki # deny repeatedly from 192.168.10.31
  </client>
  <client>
    host 192.168.10.32
    shared_key less_secret_string # limited shared_key for 192.168.10.32
    users      repeatedly         # and repatedly only
  </client>
</source>
```

Parameters

type This parameter is required. Its value must be `secure_forward`.

port (integer) The default value is 24284.

bind (sring) The default value is 0.0.0.0.

self_hostname (string) Default value of the auto-generated certificate common name (CN).

shared_key (string) Optional shared key.

allow_keepalive (bool) Accept keepalive connection. The default value is `true`.

allow_anonymous_source (bool) Accept connections from unknown hosts.

authentication (bool) Require password authentication. The default value is `false`.

cert_auto_generate (bool) Auto-generate the CA (see the `generate_*` parameters below). The default value is `false`.

If `cert_auto_generat` is false, `cert_file_path` must be set.

generate_private_key_length (integer) The byte length of the auto-generated private key. The default value is 2048.

generate_cert_country (string) The country of the auto-generated certificate. The default value is "US".

generate_cert_state (string) The state of the auto-generated certificate. The default value is "CA".

generate_cert_locality (string) The locality of the auto-generated certificate. The default value is "Mountain View".

generate_cert_common_name (string) The common name of the auto-generated certificate. The default value is the value of `self_hostname`.

cert_file_path (string) The path to the cert file. If this is not set, `cert_auto_generate` must be set to `true`.

private_key_file (string) The path to the private key file used with the cert file located at `cert_file_path`.

private_key_passphrase (string) The optional passphrase for the private key file found in `private_key_file`.

read_length (size) The number of bytes read per nonblocking read. The default value is 8MB=8*1024*1024 bytes.

read_interval_msec (integer) The interval between the non-blocking reads, in milliseconds. The default value is 50.

socket_interval_msec (integer) The inteval between SSL reconnects in milliseconds. The default value is 200.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

http Input Plugin

The `in_http` Input plugin enables Fluentd to retrieve records from HTTP POST. The URL path becomes the `tag` of the Fluentd event log and the POSTed body element becomes the record itself.

Example Configuration

`in_http` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type http
  port 8888
  bind 0.0.0.0
  body_size_limit 32m
  keepalive_timeout 10s
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Example Usage The example below posts a record using the `curl` command.

```
$ curl -X POST -d 'json={"action":"login","user":2}'
  http://localhost:8888/test.tag.here;
```

Parameters

type (required) The value must be `http`.

port The port to listen to. Default Value = 9880

bind The bind address to listen to. Default Value = 0.0.0.0 (all addresses)

body_size_limit The size limit of the POSTed element. Default Value = 32MB

keepalive_timeout The timeout limit for keeping the connection alive. Default Value = 10 seconds

add_http_headers Add `HTTP_` prefix headers to the record. The default is `false`

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

time query parameter

If you want to pass the event time from your application, please use the `time` query parameter.

```
$ curl -X POST -d 'json={"action":"login","user":2}'
  "http://localhost:8888/test.tag.here?time=1392021185"
```

Unix Domain Socket Input Plugin

The `in_unix` Input plugin enables Fluentd to retrieve records from the Unix Domain Socket. The wire protocol is the same as in_forward, but the transport layer is different.

Example Configuration

`in_unix` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type unix
  path /path/to/socket.sock
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required) The value must be `unix`.

path (required) The path to your Unix Domain Socket.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

tail Input Plugin

The `in_tail` Input plugin allows Fluentd to read events from the tail of text files. Its behavior is similar to the `tail -F` command.

Example Configuration

`in_tail` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type tail
  path /var/log/httpd-access.log
  pos_file /var/log/td-agent/httpd-access.log.pos
  tag apache.access
  format apache2
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

How it Works

- When Fluentd is first configured with `in_tail`, it will start reading from the **tail** of that log, not the beggining.
- Once the log is rotated, Fluentd starts reading the new file from the beggining. It keeps track of the current inode number.
- If `td-agent` restarts, it starts reading from the last position td-agent read before the restart. This position is recorded in the position file specified by the pos_file parameter.

Parameters

type (required) The value must be `tail`.

tag (required) The tag of the event.

`*` can be specified. `*` is replaced by actual file path which is replaced `/` with `.`. If you use following configuration

```
path /path/to/file
tag foo.*
```

in_tail emits the event to 'foo.path.to.file' tag.

path (required) The paths to read. Multiple paths can be specified, separated by ‘,’.

`*` and strftime format can be included to add/remove watch file dynamically. At interval of `refresh_interval`, Fluentd refreshes the list of watch file.

```
path /path/to/%Y/%m/%d/*
```

If the date is 20140401, Fluentd starts to watch the files in /path/to/2014/04/01 directory.

NOTE: You should not use ’*’ with log rotation because it may cause the log duplication. In such case, you should separate in_tail plugin configuration.

refresh_interval The interval of refreshing the list of watch file. Default is 60 seconds.

read_from_head Start to read the logs from the head of file, not bottom. The default is `false`.

format (required) The format of the log. It is the name of a template or regexp surrounded by ‘/’.

The regexp must have at least one named capture (?<NAME>PATTERN). If the regexp has a capture named ‘time’, it is used as the time of the event. You can specify the time format using the time_format parameter.

The following templates are supported:

- regexp

The regexp for the format parameter can be specified. Fluentular is a great website to test your regexp for Fluentd configuration.

- `apache2`

Reads apache’s log file for the following fields: host, user, time, method, path, code, size, referer and agent. This template is analogous to the following configuration:

```
format /^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +(?<path>[^ ]*) +\S*)?"
time_format %d/%b/%Y:%H:%M:%S %z
```

- `nginx`

Reads Nginx’s log file for the following fields: remote, user, time, method, path, code, size, referer and agent. This template is analogous to the following configuration:

```
format /^(?<remote>[^ ]*) (?<host>[^ ]*) (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +(?<path>[^
time_format %d/%b/%Y:%H:%M:%S %z
```

- `syslog`

Reads syslog’s output file (e.g. /var/log/syslog) for the following fields: time, host, ident, and message. This template is analogous to the following configuration:

```
format /^(?<time>[^ ]* [^ ]* [^ ]*) (?<host>[^ ]*) (?<ident>[a-zA-Z0-9_\/\.\-]*)(?:\[(?<pid>[0-9]+)\])?[^\
time_format %b %d %H:%M:%S
```

- `tsv` or `csv`

If you use tsv or csv format, please also specify the `keys` parameter.

```
format tsv
keys key1, key2, key3
time_key key2
```

If you specify the `time_key` parameter, it will be used to identify the timestamp of the record. The timestamp when Fluentd reads the record is used by default.

```
format csv
keys key1, key2, key3
time_key key3
```

- `ltsv`

ltsv (Labeled Tab-Separated Value) is a tab-delimited key-value pair format. You can learn more about it on its webpage.

```
format ltsv
delimiter =               # Optional. ':' is used by default
time_key time_field_name
```

If you specify the `time_key` parameter, it will be used to identify the timestamp of the record. The timestamp when Fluentd reads the record is used by default.

- `json`

One JSON map, per line. This is the most straight forward format :).

```
format json
```

The `time_key` parameter can also be specified.

```
format json
time_key key3
```

- `none`

You can use the `none` format to defer parsing/structuring the data. This will parse the line as-is with the key name "message". For example, if you had a line

```
hello world. I am a line of log!
```

It will be parsed as

```
{"message":"hello world. I am a line of log!"}
```

The key field is “message” by default, but you can specify a different value using the `message_key` parameter as shown below:

```
format none
message_key my_message
```

- `multiline`

Read multiline log with `formatN` and `format_firstline` parameters. `format_firstline` is for detecting start line of multiline log. `formatN`, N’s range is 1..20, is the list of Regexp format for multiline log. Here is Rails log example:

```
format multiline
format_firstline /^Started/
format1 /Started (?<method>[^ ]+) "(?<path>[^"]+)" for (?<host>[^ ]+) at (?<time>[^ ]+ [^ ]+ [^ ]+)\n/
format2 /Processing by (?<controller>[^\u0023]+)\u0023(?<controller_method>[^ ]+) as (?<format>[^ ]+?)\n/
format3 /(  Parameters: (?<parameters>[^ ]+)\n)?/
format4 /  Rendered (?<template>[^ ]+) within (?<layout>.+) \([\d\.]+ms\)\n/
format5 /Completed (?<code>[^ ]+) [^ ]+ in (?<runtime>[\d\.]+)ms \(Views: (?<view_runtime>[\d\.]+)ms \| Act
```

If you have a multiline log

```
Started GET "/users/123/" for 127.0.0.1 at 2013-06-14 12:00:11 +0900
Processing by UsersController#show as HTML
  Parameters: {"user_id"=>"123"}
  Rendered users/show.html.erb within layouts/application (0.3ms)
Completed 200 OK in 4ms (Views: 3.2ms | ActiveRecord: 0.0ms)
```

It will be parsed as

“‘text {“method”:“GET”,“path”:“/users/123/”,“host”:“127.0.0.1”,“controller”:“UsersController”,“controller_method”:“show
”user_id” = >”123”}”, …}

types (optional, v.0.10.42 and up) Although every parsed field has type `string` by default, you can specify other types. This is useful when filtering particular fields numerically or storing data with sensible type information.

The syntax is

```
types <field_name_1>:<type_name_1>,<field_name_2>:<type_name_2>,...
```

e.g.,

```
types user_id:integer,paid:bool,paid_usd_amount:float
```

As demonstrated above, “,” is used to delimit field-type pairs while “:” is used to separate a field name with its intended type.

Unspecified fields are parsed at the default string type.

The list of supported types are shown below:

- string
- bool
- integer (“int” would NOT work!)
- float
- time
- array

For the `time` and `array` types, there is an optional third field after the type name. For the “time” type, you can specify a time format like you would in `time_format`.

For the “array” type, the third field specifies the delimiter (the default is “,”). For example, if a field called “item_ids” contains the value “3,4,5”, `types item_ids:array` parses it as [“3”, “4”, “5”]. Alternatively, if the value is “Adam|Alice|Bob”, `types item_ids:array:|` parses it as [“Adam”, “Alice”, “Bob”].

“‘

pos_file (highly recommended)

This parameter is highly recommended. Fluentd will record the position it last read into this file.

```
pos_file /var/log/td-agent/tmp/access.log.pos
```

time_format The format of the time field. This parameter is required only if the format includes a ‘time’ capture and it cannot be parsed automatically. Please see Time#strftime for additional information.

rotate_wait in_tail actually does a bit more than `tail -F` itself. When rotating a file, some data may still need to be written to the old file as opposed to the new one.

in_tail takes care of this by keeping a reference to the old file (even after it has been rotated) for some time before transitioning completely to the new file. This helps prevent data designated for the old file from getting lost. By default, this time interval is 5 seconds.

The rotate_wait parameter accepts a single integer representing the number of seconds you want this time interval to be.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

exec Input Plugin

The `in_exec` Input plugin executes external programs to receive or pull event logs. It will then read TSV (tab separated values), JSON or MessagePack from the stdout of the program.

You can run a program periodically or permanently. To run periodically, please use the run_interval parameter.

Example Configuration

`in_exec` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type exec
  command cmd arg arg
  keys k1,k2,k3
  tag_key k1
  time_key k2
  time_format %Y-%m-%d %H:%M:%S
  run_interval 10s
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required) The value must be `exec`.

command (required) The command (program) to execute.

format The format used to map the program output to the incoming event.

The following formats are supported:

- tsv (default)
- json
- msgpack

When using the tsv format, please also specify the comma-separated `keys` parameter.

```
keys k1,k2,k3
```

tag (required if tag_key is not specified) tag of the output events.

tag_key The key to use as the event tag instead of the value in the event record. If this parameter is not specified, the `tag` parameter will be used instead.

time_key The key to use as the event time instead of the value in the event record. If this parameter is not specified, the current time will be used instead.

time_format The format of the event time used for the time_key parameter. The default is UNIX time (integer).

run_interval The interval time between periodic program runs.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

syslog Input Plugin

The `in_syslog` Input plugin enables Fluentd to retrieve records via the syslog protocol on UDP.

Example Configuration

`in_syslog` is included in Fluentd's core. No additional installation process is required.

```
<source>
  type syslog
  port 5140
  bind 0.0.0.0
  tag system
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Example Usage The retrieved data is organized as follows. Fluentd's tag is generated by the `tag` parameter (tag prefix), facility level, and priority. The record is parsed by the regexp here.

```
tag = "#{@tag}.#{facility}.#{priority}"

record = {
  "pri": "0",
  "time": 1353436518,
  "host": "host",
  "ident": "ident",
  "pid": "12345",
  "message": "text"
}
```

Parameters

type (required) The value must be `syslog`.

port The port to listen to. Default Value = 5140

bind The bind address to listen to. Default Value = 0.0.0.0 (all addresses)

protocol_type The transport protocol used to receive logs. The default is "udp", but you can select "tcp" as well.

tag (required) The prefix of the tag. The tag itself is generated by the tag prefix, facility level, and priority.

format The format of the log. This option is used to parse non-standard syslog formats using a regexp.

```
<source>
  type syslog
  tag system
  format /YOUR_LOG_FORMAT/
</source>
```

NOTE: Your `format` regexp should not consider the 'priority' prefix of the log. For example, if in_syslog receives the log below:

```
<1>Feb 20 00:00:00 192.168.0.1 fluentd[11111]: [error] hogehoge
```

then the format parser receives the following log:

```
Feb 20 00:00:00 192.168.0.1 fluentd[11111]: [error] hogehoge
```

types (optional, v.0.10.42 and up) Although every parsed field has type `string` by default, you can specify other types. This is useful when filtering particular fields numerically or storing data with sensible type information.

The syntax is

```
types <field_name_1>:<type_name_1>,<field_name_2>:<type_name_2>,...
```

e.g.,

```
types user_id:integer,paid:bool,paid_usd_amount:float
```

As demonstrated above, "," is used to delimit field-type pairs while ":" is used to separate a field name with its intended type.

Unspecified fields are parsed at the default string type.

The list of supported types are shown below:

- string
- bool
- integer ("int" would NOT work!)
- float
- time
- array

For the `time` and `array` types, there is an optional third field after the type name. For the "time" type, you can specify a time format like you would in `time_format`.

For the "array" type, the third field specifies the delimiter (the default is ","). For example, if a field called "item_ids" contains the value "3,4,5", `types item_ids:array` parses it as ["3", "4", "5"]. Alternatively, if the value is "Adam|Alice|Bob", `types item_ids:array:|` parses it as ["Adam", "Alice", "Bob"].

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

scribe Input Plugin

The `in_scribe` Input plugin enables Fluentd to retrieve records through the Scribe protocol. Scribe is another log collector daemon that is open-sourced by Facebook.

Since Scribe hasn't been well maintained recently, this plugin is useful for existing Scribe users who want to use Fluentd with an existing Scribe infrastructure.

Install

`in_scribe` is included in td-agent by default. Fluentd gem users will need to install the fluent-plugin-scribe gem using the following command.

```
$ fluent-gem install fluent-plugin-scribe
```

Example Configuration

```
<source>
  type scribe
  port 1463

  bind 0.0.0.0
  msg_format json
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Example Usage We assume that you're already familiar with the Scribe protocol. This Ruby example code posts logs to `in_scribe`.

Scribe's `category` field becomes the `tag` of the Fluentd event log and Scribe's `message` field becomes the record itself. The `msg_format` parameter specifies the format of the `message` field.

Parameters

type (required) The value must be `scribe`.

port The port to listen to. Default Value = 1463

bind The bind address to listen to. Default Value = 0.0.0.0 (all addresses)

msg_format The message format can be 'text', 'json', or 'url_param' (default: text)

For `json`, Fluentd's record is organized as follows. Scribe's message field must be a valid JSON string representing Hash; otherwise, the record is ignored.

```
tag: $category
record: $message
```

For `text`, Fluentd's record is organized as follows. Scribe's message can be any arbitrary string, but JSON is recommended for ease of use with the subsequent analytics pipeline.

```
tag: $category
record: {'message': $message}
```

For `url_param`, Fluentd's record is organized as follows. Scribe's message field must contain URL parameter style key-value pairs with URL encoding (e.g. key1=val1&key2=val2).

```
tag: $url_param
record: {$key1: $val1, $key2: $val2, ...}
```

is_framed Specifies whether to use Thrift's framed protocol or not (default: true).

server_type Chooses the server architecture. Options are 'simple', 'threaded', 'thread_pool', or 'non-blocking' (default: nonblocking).

add_prefix The prefix string which will always be added to the tag (default: nil).

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Multiprocess Input Plugin

By default, Fluentd only uses a single CPU core on the system. The `in_multiprocess` Input plugin enables Fluentd to use multiple CPU cores by spawning multiple child processes. One Fluentd user is using this plugin to handle 10+ billion records / day.

Install

`in_multiprocess` is NOT included in td-agent by default. td-agent users must install fluent-plugin-multiprocess manually.

```
$ /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-multiprocess
```

Fluentd gem users must install the fluent-plugin-multiprocess gem using the following command.

```
$ fluent-gem install fluent-plugin-multiprocess
```

Example Configuration

```
<source>
  type multiprocess

  <process>
    cmdline -c /etc/fluent/fluentd_child1.conf
    sleep_before_start 1s
    sleep_before_shutdown 5s
  </process>
  <process>
    cmdline -c /etc/fluent/fluentd_child2.conf
    sleep_before_start 1s
    sleep_before_shutdown 5s
  </process>
  <process>
    cmdline -c /etc/fluent/fluentd_child3.conf
    sleep_before_start 1s
    sleep_before_shutdown 5s
  </process>
</source>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required) The value must be `multiprocess`.

graceful_kill_interval The interval to send the signal to gracefully shut down the process (default: 2sec).

graceful_kill_interval_increment The increment time, when graceful shutdown fails (default: 3sec).

graceful_kill_timeout The timeout, to identify the failure of gracefull shutdown (default: 60sec).

process (required) The `process` section sets the command line arguments of a child process. This plugin creates one child process for each section.

cmdline (required) The `cmdline` option is required in a section

sleep_before_start This parameter sets the wait time before starting the process (default: 0sec).

sleep_before_shutdown This parameter sets the wait time before shutting down the process (default: 0sec).

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Other Input Plugins

Please refer to this list of available plugins to find out about other Input plugins.

- Fluentd plugins

Output Plugin Overview

Fluentd has 3 types of plugins: Input, Output, and Buffer. This article provides an overview of Output plugins.

Overview

There are three types of output plugins: Non-Buffered, Buffered, and Time Sliced.

- *Non-Buffered* output plugins do not buffer data and immediately write out results.
- *Buffered* output plugins maintain a queue of chunks (a chunk is a collection of events), and its behavior can be tuned by the "chunk limit" and "queue limit" parameters (See the diagram below).
- *Time Sliced* output plugins are in fact a type of Bufferred plugin, but the chunks are keyed by time (See the diagram below).

The output plugin's buffer behavior (if any) is defined by a separate Buffer plugin. Different buffer plugins can be chosen for each output plugin. Some output plugins are fully customized and do not use buffers.

secondary output

At Buffered output plugin, the user can specify `<secondary>` with any output plugin in `<match>` configuration. If the retry count exceeds the buffer's `retry_limit`, then buffered chunk is output to `<secondary>` output plugin.

`<secondary>` is useful for backup when destination servers are unavilable, e.g. forward, mongo and other plugins. We strongly recommend `out_file` plugin for `<secondary>`.

List of Non-Buffered Output Plugins

- out_copy
- out_null
- out_roundrobin
- out_stdout

List of Buffered Output Plugins

- out_exec_filter
- out_forward
- out_mongo or out_mongo_replset

List of Time Sliced Output Plugins

- out_exec
- out_file
- out_s3
- out_webhdfs

Other Plugins

Please refer to this list of available plugins to find out about other Output plugins.

- others

file Output Plugin

The `out_file` Buffered Output plugin writes events to files. By default, it creates files on a daily basis (around 00:10). This means that when you first import records using the plugin, no file is created immediately. The file will be created when the `time_slice_format` condition has been met. To change the output frequency, please modify the `time_slice_format` value.

Example Configuration

`out_file` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type file
  path /var/log/fluent/myapp
  time_slice_format %Y%m%d
  time_slice_wait 10m
  time_format %Y%m%dT%H%M%S%z
  compress gzip
  utc
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `file`.

path (required)

The Path of the file. The actual path is path + time + ".log". The time portion is determined by the time_slice_format parameter, descried below.

NOTE: Initially, you may see a file which looks like "/path/to/file.20140101.log.b4eea2c8166b147a0". This is an intermediate buffer file ("b4eea2c8166b147a0" identifies the buffer). Once the content of the buffer has been completely flushed, you will see the output file without the trailing identifier.

time_slice_format

The time format used as part of the file name. The following characters are replaced with actual values when the file is created:

- %Y: year including the century (at least 4 digits)
- %m: month of the year (01..12)
- %d: Day of the month (01..31)
- %H: Hour of the day, 24-hour clock (00..23)
- %M: Minute of the hour (00..59)
- %S: Second of the minute (00..60)

The default format is `%Y%m%d`, which creates one file per day. To create a file every hour, use `%Y%m%d%H`.

time_slice_wait

The amount of time Fluentd will wait for old logs to arrive. This is used to account for delays in logs arriving to your Fluentd node. The default wait time is 10 minutes ('10m'), where Fluentd will wait until 10 minutes past the hour for any logs that occured within the past hour.

For example, when splitting files on an hourly basis, a log recorded at 1:59 but arriving at the Fluentd node between 2:00 and 2:10 will be uploaded together with all the other logs from 1:00 to 1:59 in one transaction, avoiding extra overhead. Larger values can be set as needed.

time_format

The format of the time written in files. The default format is ISO-8601.

utc

Uses UTC for path formatting. The default format is localtime.

compress

Compresses flushed files using `gzip`. No compression is performed by default.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

forward Output Plugin

The `out_forward` Buffered Output plugin forwards events to other fluentd nodes. This plugin supports load-balancing and automatic fail-over (a.k.a. active-active backup). For replication, please use the out_copy plugin.

The `out_forward` plugin detects server faults using a " accrual failure detector" algorithm. You can customize the parameters of the algorithm. When a server fault recovers, the plugin makes the server available automatically after a few seconds.

Example Configuration

`out_forward` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type forward
  send_timeout 60s
  recover_wait 10s
  heartbeat_interval 1s
  phi_threshold 8
  hard_timeout 60s

  <server>
    name myserver1
    host 192.168.1.3
    port 24224
    weight 60
  </server>
  <server>
```

```
    name myserver2
    host 192.168.1.4
    port 24224
    weight 60
  </server>
  ...

  <secondary>
    type file
    path /var/log/fluent/forward-failed
  </secondary>
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `forward`.

<server> (at least one is required)

The destination servers. Each server must have following information.

- *name*: The name of the server. This parameter is used in error messages.
- *host* (required): The IP address or host name of the server.
- *port*: The port number of the host. The default is 24224. Note that both TCP packets (event stream) and UDP packets (heartbeat message) are sent to this port.
- *weight*: The load balancing weight. If the weight of one server is 20 and the weight of the other server is 30, events are sent in a 2:3 raito. The default weight is 60.

<secondary> (optional)

The backup destination that is used when all servers are unavailable.

send_timeout

The timeout time when sending event logs. The default is 60 seconds.

recover_wait

The wait time before accepting a server fault recovery. The default is 10 seconds.

heartbeat_type

The transport protocol to use for heartbeats. The default is “udp”, but you can select “tcp” as well.

heartbeat_interval

The interval of the heartbeat packer. The default is 1 second.

phi_failure_detector

Use the "Phi accrual failure detector" to detect server failure. The default value is `true`.

phi_threshold

The threshold parameter used to detect server faults. The default value is 8.

hard_timeout

The hard timeout used to detect server failure. The default value is equal to the send_timeout parameter.

standby

Marks a node as the standby node for an Active-Standby model between Fluentd nodes. When an active node goes down, the standby node is promoted to an active node. The standby node is not used by the `out_forward` plugin until then.

```
<match pattern>
  type forward
  ...

  <server>
    name myserver1
    host 192.168.1.3
    weight 60
  </server>
  <server>  # forward doesn't use myserver2 until myserver1 goes down
    name myserver2
    host 192.168.1.4
    weight 60
    standby
  </server>
  ...
</match>


## Buffer Parameters
For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

### buffer_type
The buffer type is `memory` by default ([buf_memory](#buf_memory)). The `file` ([buf_file](#buf_file)) buff

### buffer_queue_limit, buffer_chunk_limit
The length of the chunk queue and the size of each chunk, respectively. Please see the [Buffer Plugin Overvie

### flush_interval
```

```
The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (ho

### retry_wait, retry_limit and max_retry_wait
The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectivel

### num_threads
The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) de


#### log_level option (Fluentd v0.10.43 and above)

The `log_level` option allows the user to set different levels of logging for each plugin. The supported log l

Please see the [logging article](#logging) for further details.
```

Troubleshooting

"no nodes are available"

Please make sure that you can communicate with port 24224 using **not only TCP, but also UDP**. These commands will be useful for checking the network configuration.

```
$ telnet host 24224
$ nmap -p 24224 -sU host
```

Please note that there is one known issue where VMware will occasionally lose small UDP packets used for heartbeat.

Secure Forward Output

The `out_secure_forward` output plugin sends messages via **SSL with authentication** (cf. in_secure_forward).

Example Configurations

This section provides some example configurations for `out_secure_forward`.

Minimalist Configuration

```
<match secret.data.**>
  type secure_forward
  shared_key secret_string
  self_hostname client.fqdn.local
  <server>
    host server.fqdn.local  # or IP
    # port 24284
  </server>
</match>
```

NOTE: Without hostname ACL (not yet implemented), `self_hostname` is not checked in any state. The `${hostname}` placeholder is available for such cases.

```
<match secret.data.**>
  type secure_forward
  shared_key secret_string
  self_hostname ${hostname}
  <server>
    host server.fqdn.local  # or IP
    # port 24284
  </server>
</match>
```

Multiple Forward Destinations over SSL

When two or more `<server>...</server>` clauses are specified, `out_secure_forward` uses these server nodes in a round-robin order. The servers with `standby yes` are NOT selected until all non-standby servers go down.

NOTE: If a server requires username & password, set `username` and `password` in the `<server>` section:

```
<match secret.data.**>
  type secure_forward
  shared_key secret_string
  self_hostname client.fqdn.local
  <server>
    host first.fqdn.local
    username repeatedly
    password sushi
  </server>
  <server>
    host second.fqdn.local
    username sasatatsu
    password karaage
  </server>
  <server>
    host standby.fqdn.local
    username kzk
    password hawaii
    standby  yes
  </server>
</match>
```

Use the `keepalive` parameter to specify keepalive timeouts. For example, the configuration below disconnects and re-connects its SSL connection every hour. By default, `keepalive` is set to 0 and the connection does NOT get disconnected unless there is a connection issue (This feature is for DNS name updates and refreshing SSL common keys).

```
<match secret.data.**>
  type secure_forward
  shared_key secret_string
  self_hostname client.fqdn.local
  keepalive 3600
```

```
  <server>
    host server.fqdn.local  # or IP
    # port 24284
  </server>
</match>
```

Parameters

type This parameter is required. Its value must be `secure_forward`.

port (integer) The default value is 24284.

bind (sring) The default value is 0.0.0.0.

self_hostname (string) Default value of the auto-generated certificate common name (CN).

shared_key (string) Optional shared key.

keepalive (time) The duration for keepalive. If this parameter is not specified, keepalive is disabled.

send_timeout (time) The send timeout value for sockets. The default value is 60 seconds.

allow_self_signed_certificate (bool) Enables self-signed CA. The default is `true`.

ca_file_path (string) The path to the certificate file.

reconnect_interval (time) The interval between SSL reconnects. The default value is 5 seconds.

read_length (integer) The number of bytes read per nonblocking read. The default value is 8MB=8*1024*1024 bytes.

read_interval_msec (integer) The interval between the non-blocking reads, in milliseconds. The default value is 50.

socket_interval_msec (integer) The inteval between SSL reconnects in milliseconds. The default value is 200.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

exec Output Plugin

The `out_exec` Buffered Output plugin passes events to an external program. The program receives the path to a file containing the incoming events as its last argument. The file format is tab-separated values (TSV) by default.

Example Configuration

`out_exec` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type exec
  command cmd arg arg
  format tsv
  keys k1,k2,k3
  tag_key k1
  time_key k2
  time_format %Y-%m-%d %H:%M:%S
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `exec`.

command (required)

The command (program) to execute. The exec plugin passes the path of a TSV file as the last argument.

format

The format used to map the incoming events to the program input.

The following formats are supported:

- tsv (default)

When using the tsv format, please also specify the comma-separated `keys` parameter.

`in_keys k1,k2,k3`

- json
- msgpack

tag_key

The name of the key to use as the event tag. This replaces the value in the event record.

time_key

The name of the key to use as the event time. This replaces the the value in the event record.

time_format

The format for event time used when the `time_key` parameter is specified. The default is UNIX time (integer).

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

exec_filter Output Plugin

The `out_exec_filter` Buffered Output plugin (1) executes an external program using an event as input and (2) reads a new event from the program output. It passes tab-separated values (TSV) to stdin and reads TSV from stdout by default.

Example Configuration

`out_exec_filter` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type exec_filter
  command cmd arg arg
  in_keys k1,k2,k3
  out_keys k1,k2,k3,k4
  tag_key k1
  time_key k2
  time_format %Y-%m-%d %H:%M:%S
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `exec_filter`.

command (required)

The command (program) to execute. The `out_exec_filter` plugin passes the incoming event to the program input and receives the filtered event from the program output.

in_format

The format used to map the incoming event to the program input.

The following formats are supported:

- tsv (default)

When using the tsv format, please also specify the comma-separated `in_keys` parameter.

```
in_keys k1,k2,k3
```

- json
- msgpack

out_format

The format used to process the program output.

The following formats are supported:

- tsv (default)

When using the tsv format, please also specify the comma-separated `out_keys` parameter.

```
out_keys k1,k2,k3,k4
```

- json
- msgpack

NOTE: When using the json format, this plugin uses the Yajl library to parse the program output. Yajl buffers data internally so the output isn't always instantaneous.

tag_key

The name of the key to use as the event tag. This replaces the value in the event record.

time_key

The name of the key to use as the event time. This replaces the the value in the event record.

time_format

The format for event time used when the `time_key` parameter is specified. The default is UNIX time (integer).

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

copy Output Plugin

The `copy` output plugin copies events to multiple outputs.

Example Configuration

`out_copy` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type copy
  <store>
    type file
    path /var/log/fluent/myapp1
    ...
  </store>
  <store>
    ...
  </store>
  <store>
    ...
  </store>
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Here is an example set up to send events to both a local file under `/var/log/fluent/myapp` and the collection `fluentd.test` in a local MongoDB instance (Please see the out_file and out_mongo articles for more details about the respective plugins.)

```
<match myevent.file_and_mongo>
  type copy
  <store>
    type file
    path /var/log/fluent/myapp
    time_slice_format %Y%m%d
    time_slice_wait 10m
    time_format %Y%m%dT%H%M%S%z
    compress gzip
    utc
  </store>
  <store>
    type mongo
    host fluentd
    port 27017
    database fluentd
    collection test
  </store>
</match>
```

Parameters

type (required)

The value must be `copy`.

deep_copy

`out_copy` shares a record between `store` plugins by default.

When `deep_copy` is true, `out_copy` passes different record to each `store` plugin.

<store> (at least one required)

Specifies the storage destinations. The format is the same as the <match> directive.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

GeoIP Output Plugin

The `out_geoip` Buffered Output plugin adds geographic location information to logs using the Maxmind GeoIP databases.

Prerequisites

- The GeoIP library.

```
# for RHEL/CentOS
$ sudo yum install geoip-devel --enablerepo=epel

# for Ubuntu/Debian
$ sudo apt-get install libgeoip-dev

# for MacOSX (brew)
$ brew install geoip
```

Install

`out_geoip` is not included in td-agent. All users must install the fluent-plugin-geoip gem using the following command.

```
$ fluent-gem install fluent-plugin-geoip
OR
$ sudo /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-geoip
```

Example Configuration

The configuration shown below adds geolocation information to apache.access

“‘text type geoip geoip_lookup_key host enable_key_country_code geoip_country enable_key_city geoip_city enable_key_latitude geoip_lat enable_key_longitude geoip_lon remove_tag_prefix test. add_tag_prefix geoip. flush_interval 5s

```
# original record
test.message {
  "host":"66.102.9.80",
  "message":"test"
}

# output record
geoip.message: {
  "host":"66.102.9.80",
  "message":"test",
  "geoip_country":"US",
  "geoip_city":"Mountain View",
  "geoip_lat":37.4192008972168,
  "geoip_lon":-122.05740356445312
}
```

NOTE: Please see the fluent-plugin-geoip README for further details.

Parameters

geoip_lookup_key (required)

Specifies the geoip lookup field (default: host) If accessing a nested hash value, delimit the key with '.', as in 'host.ip'.

remove_tag_prefix / add_tag_prefix (requires one or the other)

Set tag replace rule.

enable_key_*** (requires at least one)

Specifies the geographic data that will be added to the record. The supported parameters are shown below:

- enable_key_city
- enable_key_latitude
- enable_key_longitude
- enable_key_country_code3
- enable_key_country_code
- enable_key_country_name
- enable_key_dma_code
- enable_key_area_code
- enable_key_region

include_tag_key

Set to `true` to include the original tag name in the record. (default: false)

tag_key

Adds the tag name into the record using this value as the key name When `include_tag_key` is set to `true`.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. Unlike many other output plugins, the `buffer_path` parameter MUST be specified when using `buffer_type file`.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 256m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between forced data flushes. The default is nil (don't force flush and wait until the end of time slice + time_slice_wait). The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Use Cases

Plot real time access statistics on a world map using Elasticsearch and Kibana The `country_code` field is needed to visualize access statistics on a world map using Kibana.

Note: The following plugins are required: * fluent-plugin-geoip * fluent-plugin-elasticsearch

```
<match td.apache.access>
  type geoip

  # Set key name for the client ip address values
  geoip_lookup_key     host

  # Specify key name for the country_code values
  enable_key_country_code  geoip_country

  # Swap tag prefix from 'td.' to 'es.'
  remove_tag_prefix    td.
  add_tag_prefix       es.
</match>

<match es.apache.access>
  type            elasticsearch
  host            localhost
  port            9200
```

```
  type_name       apache
  logstash_format true
  flush_interval  10s
</match>
```

Further Reading

- fluent-plugin-geoip repository

roundrobin Output Plugin

The `roundrobin` Output plugin distributes events to multiple outputs using a round-robin algorithm.

Example Configuration

`out_roundrobin` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type roundrobin

  <store>
    type tcp
    host 192.168.1.21
    ...
  </store>
  <store>
    ...
  </store>
  <store>
    ...
  </store>
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `roundrobin`.

<store> (required at least one)

Specifies the storage destinations. The format is the same as the <match> directive.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

stdout Output Plugin

The `stdout` output plugin prints events to stdout (or logs if launched with daemon mode). This output plugin is useful for debugging purposes.

Example Configuration

`out_stdout` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type stdout
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required) The value must be `stdout`.

output_type Output format. The following formats are supported:

- json
- hash (Ruby's hash)

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

null Output Plugin

The `null` output plugin just throws away events.

Example Configuration

`out_null` is included in Fluentd's core. No additional installation process is required.

```
<match pattern>
  type null
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `null`.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Amazon S3 Output Plugin

The `out_s3` Buffered Output plugin writes records into the Amazon S3 cloud object storage service. By default, it creates files on an hourly basis. This means that when you first import records using the plugin, no file is created immediately. The file will be created when the `time_slice_format` condition has been met. To change the output frequency, please modify the `time_slice_format` value.

Installation

`out_s3` is included in td-agent by default. Fluentd gem users will need to install the fluent-plugin-s3 gem using the following command.

```
$ fluent-gem install fluent-plugin-s3
```

Example Configuration

```
<match pattern>
  type s3

  aws_key_id YOUR_AWS_KEY_ID
  aws_sec_key YOUR_AWS_SECRET/KEY
  s3_bucket YOUR_S3_BUCKET_NAME
  s3_endpoint s3-us-west-1.amazonaws.com
  path logs/
  buffer_path /var/log/fluent/s3

  time_slice_format %Y%m%d%H
  time_slice_wait 10m
  utc

  buffer_chunk_limit 256m
</match>
```

Please see the Store Apache Logs into Amazon S3 article for real-world use cases.

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

NOTE: Please make sure that you have enough space in the buffer_path directory. Running out of disk space is a problem frequently reported by users.

Parameters

type (required)

The value must be `s3`.

aws_key_id (required/optional)

The AWS access key id. This parameter is required when your agent is not running on an EC2 instance with an IAM Instance Profile.

aws_sec_key (required/optional)

The AWS secret key. This parameter is required when your agent is not running on an EC2 instance with an IAM Instance Profile.

s3_bucket (required)

The Amazon S3 bucket name.

buffer_path (required)

The path prefix of the log buffer files.

s3_endpoint

The Amazon S3 endpoint name. Please select the appropriate endpoint name from the list below and confirm that your bucket has been created in the correct region.

- s3.amazonaws.com
- s3-us-west-1.amazonaws.com
- s3-us-west-2.amazonaws.com
- s3.sa-east-1.amazonaws.com
- s3-eu-west-1.amazonaws.com
- s3-ap-southeast-1.amazonaws.com
- s3-ap-northeast-1.amazonaws.com

The most recent versions of the endpoints can be found here.

time_slice_format

The time format used as part of the file name. The following characters are replaced with actual values when the file is created:

- %Y: year including the century (at least 4 digits)
- %m: month of the year (01..12)
- %d: Day of the month (01..31)
- %H: Hour of the day, 24-hour clock (00..23)
- %M: Minute of the hour (00..59)
- %S: Second of the minute (00..60)

The default format is `%Y%m%d%H`, which creates one file per hour.

time_slice_wait

The amount of time Fluentd will wait for old logs to arrive. This is used to account for delays in logs arriving to your Fluentd node. The default wait time is 10 minutes ('10m'), where Fluentd will wait until 10 minutes past the hour for any logs that occured within the past hour.

For example, when splitting files on an hourly basis, a log recorded at 1:59 but arriving at the Fluentd node between 2:00 and 2:10 will be uploaded together with all the other logs from 1:00 to 1:59 in one transaction, avoiding extra overhead. Larger values can be set as needed.

time_format

The format of the time written in files. The default format is ISO-8601.

path

The path prefix of the files on S3. The default is "" (no prefix).

NOTE: The actual path on S3 will be: "{path}{time_slice_format}*{sequential*number}.gz"

utc

Uses UTC for path formatting. The default format is localtime.

store_as

The compression type. The default is "gzip", but you can also choose "lzo", "json", or "txt".

proxy_uri

The proxy url. The default is nil.

use_ssl

Enable/disable SSL for data transfers between Fluentd and S3. The default is "yes".

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Further Reading

This page doesn't describe all the possible configurations. If you want to know about other configurations, please check the link below.

- fluent-plugin-s3 repository

MongoDB Output Plugin

The `out_mongo` Buffered Output plugin writes records into MongoDB, the emerging document-oriented database system.

NOTE: If you're using ReplicaSet, please see the out_mongo_replset article instead.

Why Fluentd with MongoDB?

Fluentd enables your apps to insert records to MongoDB asynchronously with batch-insertion, unlike direct insertion of records from your apps. This has the following advantages:

1. less impact on application performance
2. higher MongoDB insertion throughput while maintaining JSON record structure

Install

`out_mongo` is included in td-agent by default. Fluentd gem users will need to install the fluent-plugin-mongo gem using the following command.

```
$ fluent-gem install fluent-plugin-mongo
```

Example Configuration

```
# Single MongoDB
<match mongo.**>
  type mongo
  host fluentd
  port 27017
  database fluentd
  collection test

  # for capped collection
  capped
  capped_size 1024m

  # authentication
  user michael
  password jordan

  # flush
  flush_interval 10s
</match>
```

Please see the Store Apache Logs into MongoDB article for real-world use cases.

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `mongo`.

host (required)

The MongoDB hostname.

port (required)

The MongoDB port.

database (required)

The database name.

collection (required, if not tag_mapped)

The collection name.

capped

This option enables capped collection. This is always recommended because MongoDB is not suited to storing large amounts of historical data.

capped_size Sets the capped collection size.

user

The username to use for authentication.

password

The password to use for authentication.

tag_mapped

This option will allow out_mongo to use Fluentd's tag to determine the destination collection. For example, if you generate records with tags 'mongo.foo', the records will be inserted into the `foo` collection within the `fluentd` database.

```
<match mongo.*>
  type mongo
  host fluentd
  port 27017
  database fluentd

  # Set 'tag_mapped' if you want to use tag mapped mode.
  tag_mapped

  # If the tag is "mongo.foo", then the prefix "mongo." is removed.
  # The inserted collection name is "foo".
  remove_tag_prefix mongo.

  # This configuration is used if the tag is not found. The default is 'untagged'.
  collection misc
</match>
```

This option is useful for flexible log collection.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Further Reading

- fluent-plugin-mongo repository

MongoDB ReplicaSet Output Plugin

The `out_mongo_replset` Buffered Output plugin writes records into MongoDB, the emerging document-oriented database system.

NOTE: This plugin is for users using ReplicaSet. If you are not using ReplicaSet, please see the out_mongo article instead.

Why Fluentd with MongoDB?

Fluentd enables your apps to insert records to MongoDB asynchronously with batch-insertion, unlike direct insertion of records from your apps. This has the following advantages:

1. less impact on application performance
2. higher MongoDB insertion throughput while maintaining JSON record structure

Install

`out_mongo_replset` is included in td-agent by default. Fluentd gem users will need to install the fluent-plugin-mongo gem using the following command.

```
$ fluent-gem install fluent-plugin-mongo
```

Example Configuration

```
# Single MongoDB
<match mongo.**>
  type mongo_replset
  database fluentd
  collection test
  nodes localhost:27017,localhost:27018,localhost:27019

  # flush
  flush_interval 10s
</match>
```

Please see the Store Apache Logs into MongoDB article for real-world use cases.

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `mongo`.

nodes (required)

The comma separated node strings (e.g. host1:27017,host2:27017,host3:27017).

database (required)

The database name.

collection (required if not tag_mapped)

The collection name.

capped

This option enables capped collection. This is always recommended because MongoDB is not suited to storing large amounts of historical data.

capped_size

Sets the capped collection size.

user

The username to use for authentication.

password

The password to use for authentication.

tag_mapped

This option will allow out_mongo to use Fluentd's tag to determine the destination collection.

For example, if you generate records with tags 'mongo.foo', the records will be inserted into the `foo` collection within the `fluentd` database.

```
<match mongo.*>
  type mongo_replset
  database fluentd
  nodes localhost:27017,localhost:27018,localhost:27019

  # Set 'tag_mapped' if you want to use tag mapped mode.
  tag_mapped

  # If the tag is "mongo.foo", then the prefix "mongo." is removed.
  # The inserted collection name is "foo".
  remove_tag_prefix mongo.

  # This configuration is used if the tag is not found. The default is 'untagged'.
  collection misc
</match>
```

name

The ReplicaSet name.

read

The ReplicaSet read preference (e.g. secondary, etc).

refresh_mode

The ReplicaSet refresh mode (e.g. sync, etc).

refresh_interval

The ReplicaSet refresh interval.

num_retries

The ReplicaSet failover threshold. The default threshold is 60. If the retry count reaches this threshold, the plugin raises an exception.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Further Readings

- fluent-plugin-webhdfs mongo

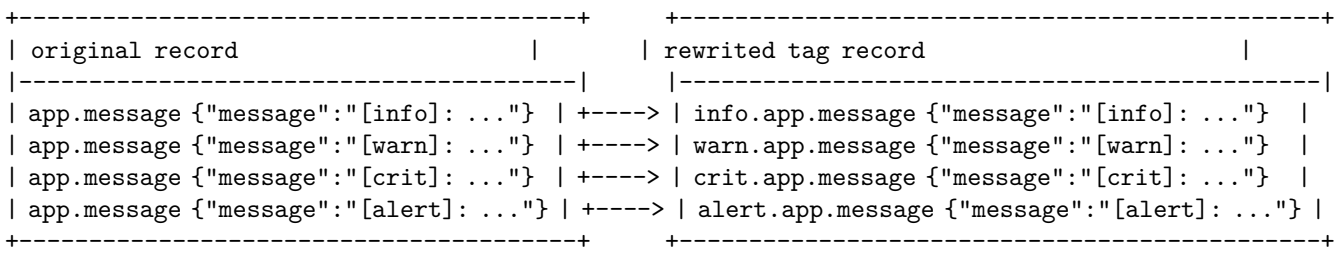
rewrite_tag_filter Output Plugin

The `out_rewrite_tag_filter` Output plugin has designed to rewrite tag like mod_rewrite. Re-emit a record with rewrited tag when a value matches/unmatches with the regular expression. Also you can change a tag from apache log by domain, status-code(ex. 500 error), user-agent, request-uri, regex-backreference and so on with regular expression.

How it works

It is a sample to arrange the tags by the regexp matched value of 'message'.

"'text # Configuration type rewrite_tag_filter rewriterule1 message ^[(\w+)] $1.${tag}



Install

`out_rewrite_tag_filter` is included in td-agent by default (v1.1.18 or later). Fluentd gem users will have to install the fluent-plugin-rewrite-tag-filter gem using the following command.

```
$ fluent-gem install fluent-plugin-rewrite-tag-filter
OR
$ sudo /usr/lib64/fluent/ruby/bin/fluent-gem install fluent-plugin-rewrite-tag-filter
```

Example Configuration

Configuration design is dropping some pattern record first, then re-emit other matched record as new tag name.

```
<match apache.access>
  type rewrite_tag_filter
  capitalize_regex_backreference yes
  rewriterule1 path   \.(gif|jpe?g|png|pdf|zip)$  clear
  rewriterule2 status !^200$                      clear
  rewriterule3 domain !^.+\.com$                  clear
  rewriterule4 domain ^maps\.example\.com$        site.ExampleMaps
  rewriterule5 domain ^news\.example\.com$        site.ExampleNews
  # it is also supported regexp back reference.
  rewriterule6 domain ^(mail)\.(example)\.com$    site.$2$1
  rewriterule7 domain .+                          site.unmatched
</match>

<match clear>
  type null
</match>
```

NOTE: Please see the README.md for further details.

Parameters

rewriteruleN (required at least one)

`rewriterule<num> <key> <regex_pattern> <new_tag>`

NOTE: It works with the order <num> ascending, regexp matching <regex_pattern> for the values of <key> from each record, re-emit with <new_tag>.

capitalize_regex_backreference

Capitalize letter for every matched regex backreference. (ex: maps -> Maps)

hostname_command

Override hostname command for placeholder. (default setting is long hostname)

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Placeholders

It is supported these placeholder for new_tag (rewrited tag). See more details at README.md

- ${tag}
- **TAG**
- {$tag_parts[n]}
- **TAG_PARTS[n]**
- ${hostname}
- **HOSTNAME**

Use cases

- Aggregate + display 404 status pages by URL and referrer to find and fix dead links.
- Send an IRC alert for 5xx status codes on exceeding thresholds.

Aggregate + display 404 status pages by URL and referrer to find and fix dead links.

- Collect access log from multiple application servers (config1)
- Sum up the 404 error and output to mongoDB (config2)

Note: These plugins are required to be installed. * fluent-plugin-rewrite-tag-filter * fluent-plugin-mongo

[Config1] Application Servers

```
# Input access log to fluentd with embedded in_tail plugin
<source>
  type tail
  path /var/log/httpd/access_log
  format apache2
  time_format %d/%b/%Y:%H:%M:%S %z
  tag apache.access
  pos_file /var/log/td-agent/apache_access.pos
</source>

# Forward to monitoring server
<match apache.access>
  type forward
  flush_interval 5s
  <server>
    name server_name
    host 10.100.1.20
  </server>
</match>
```

[Config2] Monitoring Server

```
# built-in TCP input
<source>
  type forward
</source>

# Filter record like mod_rewrite with fluent-plugin-rewrite-tag-filter
<match apache.access>
  type rewrite_tag_filter
  rewriterule1 status ^(?!404)$ clear
  rewriterule2 path .+ mongo.apache.access.error404
</match>

# Store deadlinks log into mongoDB
<match mongo.apache.access.error404>
  type        mongo
  host        10.100.1.30
  database    apache
  collection  deadlinks
  capped
  capped_size 50m
</match>

# Clear tag
<match clear>
  type null
</match>
```

Send an IRC alert for 5xx status codes on exceeding thresholds.

- Collect access log from multiple application servers (config1)
- Sum up the 500 error and notify IRC and logging details to mongoDB (config2)

Note: These plugins are required to be installed. * fluent-plugin-rewrite-tag-filter * fluent-plugin-datacounter * fluent-plugin-notifier * fluent-plugin-parser * fluent-plugin-mongo * fluent-plugin-irc

[Config1] Application Servers

```
# Input access log to fluentd with embedded in_tail plugin
# sample results: {"host":"127.0.0.1","user":null,"method":"GET","path":"/","code":500,"size":5039,"refe
<source>
  type tail
  path /var/log/httpd/access_log
  format apache2
  time_format %d/%b/%Y:%H:%M:%S %z
  tag apache.access
  pos_file /var/log/td-agent/apache_access.pos
</source>

# Forward to monitoring server
<match apache.access>
  type forward
  flush_interval 5s
  <server>
    name server_name
    host 10.100.1.20
  </server>
</match>
```

[Config2] Monitoring Server

```
# built-in TCP input
<source>
  type forward
</source>

# Filter record like mod_rewrite with fluent-plugin-rewrite-tag-filter
<match apache.access>
  type copy
  <store>
  type rewrite_tag_filter
  # drop static image record and redirect as 'count.apache.access'
  rewriterule1 path ^/(img|css|js|static|assets)/ clear
  rewriterule2 path .+ count.apache.access
  </store>
  <store>
  type rewrite_tag_filter
  rewriterule1 code ^5\d\d$ mongo.apache.access.error5xx
  </store>
```

```
</match>

# Store 5xx error log into mongoDB
<match mongo.apache.access.error5xx>
  type        mongo
  host        10.100.1.30
  database    apache
  collection  error_5xx
  capped
  capped_size 50m
</match>

# Count by status code
# sample results: {"unmatched_count":0,"unmatched_rate":0.0,"unmatched_percentage":0.0,"200_count":0,"200
<match count.apache.access>
  type datacounter
  unit minute
  outcast_unmatched false
  aggregate all
  tag threshold.apache.access
  count_key code
  pattern1 200 ^200$
  pattern2 2xx ^2\d\d$
  pattern3 301 ^301$
  pattern4 302 ^302$
  pattern5 3xx ^3\d\d$
  pattern6 403 ^403$
  pattern7 404 ^404$
  pattern8 410 ^410$
  pattern9 4xx ^4\d\d$
  pattern10 5xx ^5\d\d$
</match>

# Determine threshold
# sample results: {"pattern":"code_500","target_tag":"apache.access","target_key":"5xx_count","check_type
<match threshold.apache.access>
  type notifier
  input_tag_remove_prefix threshold
  <def>
    pattern code_500
    check numeric_upward
    warn_threshold  10
    crit_threshold  40
    tag alert.http_5xx_error
    target_key_pattern ^5xx_count$
  </def>
</match>

# Generate message
# sample results: {"message":"HTTP Status warn [5xx_count] apache.access: 1.0 (threshold 1.0)"}
<match alert.http_5xx_error>
  type deparser
  tag irc.http_5xx_error>
  format_key_names level,target_key,target_tag,value,threshold
```

```
  format HTTP Status %s [%s] %s: %s (threshold %s)
  key_name message
  reserve_data no
</match>

# Send IRC message
<match irc.http_5xx_error>
  type irc
  host localhost
  port 6667
  channel fluentd
  nick fluentd
  user fluentd
  real fluentd
  message %s
  out_keys message
</match>

# Clear tag
<match clear>
  type null
</match>
```

HDFS (WebHDFS) Output Plugin

The `out_webhdfs` Buffered Output plugin writes records into HDFS (Hadoop Distributed File System). By default, it creates files on an hourly basis. This means that when you first import records using the plugin, no file is created immediately. The file will be created when the `time_slice_format` condition has been met. To change the output frequency, please modify the `time_slice_format` value.

Install

`out_webhdfs` is included in td-agent by default (v1.1.10 or later). Fluentd gem users will have to install the fluent-plugin-webhdfs gem using the following command.

```
$ fluent-gem install fluent-plugin-webhdfs
```

HDFS Configuration

Append operations are not enabled by default on CDH. Please put these configurations into your hdfs-site.xml file and restart the whole cluster.

```
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.support.append</name>
  <value>true</value>
```

```
</property>

<property>
  <name>dfs.support.broken.append</name>
  <value>true</value>
</property>
```

Example Configuration

```
<match access.**>
  type webhdfs
  host namenode.your.cluster.local
  port 50070
  path /path/on/hdfs/access.log.%Y%m%d_%H.${hostname}.log
  flush_interval 10s
</match>
```

Please see the Fluentd + HDFS: Instant Big Data Collection article for real-world use cases.

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

type (required)

The value must be `webhfds`.

host (required)

The namenode hostname.

port (required)

The namenode port number.

path (required)

The path on HDFS. Please include ${hostname} in your path to avoid writing into the same HDFS file from multiple Fluentd instances. This conflict could result in data loss.

Buffer Parameters

For advanced usage, you can tune Fluentd's internal buffering mechanism with these parameters.

buffer_type

The buffer type is `memory` by default (buf_memory). The `file` (buf_file) buffer type can be chosen as well. The `path` parameter is used as `buffer_path` in this plugin.

buffer_queue_limit, buffer_chunk_limit

The length of the chunk queue and the size of each chunk, respectively. Please see the Buffer Plugin Overview article for the basic buffer structure. The default values are 64 and 8m, respectively. The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for buffer_chunk_limit.

flush_interval

The interval between data flushes. The default is 60s. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

retry_wait, retry_limit and max_retry_wait

The interval between write retries, and the number of retries. The default values are 1.0 and 17, respectively. `retry_wait` doubles every retry (e.g. the last retry waits for 131072 sec, roughly 36 hours), and `max_retry_wait` may be used to limit the maximum retry interval.

num_threads

The number of threads to flush the buffer. This option can be used to parallelize writes into the output(s) designated by the output plugin. The default is 1.

log_level option (Fluentd v0.10.43 and above) The `log_level` option allows the user to set different levels of logging for each plugin. The supported log levels are: `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.

Please see the logging article for further details.

Further Reading

- fluent-plugin-webhdfs repository
- Slides: Fluentd and WebHDFS

Other Output Plugins

Please refer to this list of available plugins to find out about other Output plugins.

- Fluentd plugins

Buffer Plugin Overview

Fluentd has 3 types of plugins: Input, Output, and Buffer. This article gives an overview of Buffer Plugin.

We will first explain how Buffer Plugin works in general. Then, we will explain the mechanism of Time Sliced Plugin, a subclass of Buffer Plugin used by several core plugins.

Buffer Plugin Overview

Buffer plugins are used by buffered output plugins, such as `out_file`, `out_forward`, etc. Users can choose the buffer plugin that best suits their performance and reliability needs.

Buffer Structure

The buffer structure is a queue of chunks like the following:

```
queue
+---------+
|         |
|  chunk <-- write events to the top chunk
|         |
|  chunk  |
|         |
|  chunk  |
|         |
|  chunk --> write out the bottom chunk
|         |
+---------+
```

When the top chunk exceeds the specified size or time limit (`buffer_chunk_limit` and `flush_interval`, respectively), a new empty chunk is pushed to the top of the queue. The bottom chunk is written out immediately when new chunk is pushed.

If the bottom chunk write out fails, it will remain in the queue and Fluentd will retry after waiting several seconds (`retry_wait`). If the retry count exceeds the specified limit (`retry_limit`), the chunk is trashed. The retry wait time doubles each time (1.0sec, 2.0sec, 4.0sec, ...). If the queue length exceeds the specified limit (`buffer_queue_limit`), new events are rejected.

All buffered output plugins support the following parameters:

```
<match pattern>
  buffer_type memory
  buffer_chunk_limit 256m
  buffer_queue_limit 128
  flush_interval 60s
  retry_limit 17
  retry_wait 1s
</match>
```

`buffer_type` specifies the buffer plugin to use. The `memory` Buffer plugin is used by default. You can also specify `file` as the buffer type alongside the `buffer_path` parameter as follows:

```
<match pattern>
  buffer_type file
  buffer_path /var/fluentd/buffer/ #make sure fluentd has write access to the directory!
  ...
</match>
```

The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used for `flush_interval` and `retry_wait`. `retry_wait` can also be a decimal value.

The suffixes "k" (KB), "m" (MB), and "g" (GB) can be used for `buffer_chunk_limit`.

Time Sliced Plugin Overview

Time Sliced Plugin is a type of Buffer Plugin, so, it has the same basic buffer structure as Buffer Plugin.

In addition, each chunk is keyed by time and flushed when that chunk's timestamp has passed. This is different from This immediately raises a couple of questions.

1. How do we specify the granularity of time chunks? This is done through the `time_slice_format` option, which is set to "%Y%m%d" (daily) by default. If you want your chunks to be hourly, "%Y%m%d%H" will do the job.
2. What if new logs come after the time corresponding the current chunk? For example, what happens to an event, timestamped at 2013-01-01 02:59:45 UTC, comes in at 2013-01-01 03:00:15 UTC? Would it make into the 2013-01-01 02:00:00-02:59:59 chunk?

This issue is addressed by setting the `time_slice_wait` parameter. `time_slice_wait` sets, in seconds, how long fluentd waits to accept "late" events into the chunk *past the max time corresponding to that chunk*. The default value is 600, which means it waits for 10 minutes before moving on. So, in the current example, as long as the events come in before 2013-01-01 03:10:00, it will make it in to the 2013-01-01 02:00:00-02:59:59 chunk.

Alternatively, you can also flush the chunks regularly using `flush_interval`. Note that `flush_interval` and `time_slice_wait` are mutually exclusive. If you set `flush_interval`, `time_slice_wait` will be ignored and fluentd would issue a warning.

Notes

If you are curious which core output plugin use Buffered and which are Time Sliced, please see the list here

List of Buffer Plugins

- buf_memory
- buf_file

memory Buffer Plugin

The `memory` buffer plugin provides a fast buffer implementation. It uses memory to store buffer chunks. When Fluentd is shut down, buffered logs that can't be written quickly are deleted.

Example Config

```
<match pattern>
  buffer_type memory
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

buffer_type (required) The value must be `memory`.

buffer_chunk_limit The size of each buffer chunk. The default is 8m. The suffixes “k” (KB), “m” (MB), and “g” (GB) can be used. Please see the Buffer Plugin Overview article for the basic buffer structure.

buffer_queue_limit The length limit of the chunk queue. Please see the Buffer Plugin Overview article for the basic buffer structure. The default limit is 64 chunks.

flush_interval The interval between data flushes. The suffixes “s” (seconds), “m” (minutes), and “h” (hours) can be used

flush_at_shutdown If true, queued chunks are flushed at shutdown process. The default is `true`. If false, queued chunks are dicarded unlike `buf_file`.

retry_wait The interval between retries. The suffixes “s” (seconds), “m” (minutes), and “h” (hours) can be used.

file Buffer Plugin

The `file` buffer plugin provides a persistent buffer implementation. It uses files to store buffer chunks on disk.

Example Config

```
<match pattern>
  buffer_type file
  buffer_path /var/log/fluent/myapp.*.buffer
</match>
```

NOTE: Please see the Config File article for the basic structure and syntax of the configuration file.

Parameters

buffer_type (required) The value must be `file`.

buffer_path (required) The path where buffer chunks are stored. The ‘*’ is replaced with random characters. This parameter is required.

buffer_chunk_limit The size of each buffer chunk. The default is 256m. The suffixes “k” (KB), “m” (MB), and “g” (GB) can be used. Please see the Buffer Plugin Overview article for the basic buffer structure.

buffer_queue_limit The length limit of the chunk queue. Please see the Buffer Plugin Overview article for the basic buffer structure. The default limit is 256 chunks.

flush_interval The interval between data flushes. The suffixes “s” (seconds), “m” (minutes), and “h” (hours) can be used

flush_at_shutdown If true, queued chunks are flushed at shutdown process. The default is `false`.

retry_wait The interval between retries. The suffixes "s" (seconds), "m" (minutes), and "h" (hours) can be used.

Writing plugins

Installing custom plugins

To install a plugin, please put the ruby script in the `/etc/fluent/plugin` directory.

Alternatively, you can create a Ruby Gem package that includes a `lib/fluent/plugin/<TYPE>_<NAME>.rb` file. The *TYPE* is `in` for input plugins, `out` for output plugins, and `buf` for buffer plugins. For example, an eMail Output plugin would have the path: `lib/fluent/plugin/out_mail.rb`. The packaged gem can be distributed and installed using RubyGems. For further information, please see the list of Fluentd plugins.

Writing Input Plugins

Extend the **Fluent::Input** class and implement the following methods.

```
class SomeInput < Fluent::Input
  # First, register the plugin. NAME is the name of this plugin
  # and identifies the plugin in the configuration file.
  Fluent::Plugin.register_input('NAME', self)

  # This method is called before starting.
  # 'conf' is a Hash that includes configuration parameters.
  # If the configuration is invalid, raise Fluent::ConfigError.
  def configure(conf)
    super
    @port = conf['port']
    ...
  end

  # This method is called when starting.
  # Open sockets or files and create a thread here.
  def start
    super
    ...
  end

  # This method is called when shutting down.
  # Shutdown the thread and close sockets or files here.
  def shutdown
    ...
  end
end
```

To submit events, use the `Fluent::Engine.emit(tag, time, record)` method, where `tag` is the String, `time` is the UNIX time integer and `record` is a Hash object.

```
tag = "myapp.access"
time = Time.now.to_i
record = {"message"=>"body"}
Fluent::Engine.emit(tag, time, record)
```

The RDoc of the Engine class can be found in the Fluentd RDoc.

Writing Buffered Output Plugins

Extend the **Fluent::BufferedOutput** class and implement the following methods.

```
class SomeOutput < Fluent::BufferedOutput
  # First, register the plugin. NAME is the name of this plugin
  # and identifies the plugin in the configuration file.
  Fluent::Plugin.register_output('NAME', self)

  # This method is called before starting.
  # 'conf' is a Hash that includes configuration parameters.
  # If the configuration is invalid, raise Fluent::ConfigError.
  def configure(conf)
    super
    @path = conf['path']
    ...
  end

  # This method is called when starting.
  # Open sockets or files here.
  def start
    super
    ...
  end

  # This method is called when shutting down.
  # Shutdown the thread and close sockets or files here.
  def shutdown
    super
    ...
  end

  # This method is called when an event reaches to Fluentd.
  # Convert the event to a raw string.
  def format(tag, time, record)
    [tag, time, record].to_json + "\n"
    ## Alternatively, use msgpack to serialize the object.
    # [tag, time, record].to_msgpack
  end

  ## Instead of the above method, you could instead implement this method,
  ## which is called by fluentd to process a collection of events.
  #def format_stream(tag, es)
  #  es.each {|time, record|
  #    [tag, time, record].to_json + "\n"
  #    # Alternatively, use msgpack to serialize the object.
```

```
  #    # [tag, time, record].to_msgpack
  #  }
  #end

  # This method is called every flush interval. Write the buffer chunk
  # to files or databases here.
  # 'chunk' is a buffer chunk that includes multiple formatted
  # events. You can use 'data = chunk.read' to get all events and
  # 'chunk.open {|io| ... }' to get IO objects.
  def write(chunk)
    data = chunk.read
    print data
  end

  ## Optionally, you can use chunk.msgpack_each to deserialize objects.
  #def write(chunk)
  #  chunk.msgpack_each {|(tag,time,record)|
  #  }
  #end
end
```

Writing Time Sliced Output Plugins

Time Sliced Output plugins are extended versions of buffered output plugins. One example of a time sliced output is the `out_file` plugin.

Note that Time Sliced Output plugins use file buffer by default. Thus the `buffer_path` option is required.

To implement a Time Sliced Output plugin, extend the **Fluent::TimeSlicedOutput** class and implement the following methods.

```
class SomeOutput < Fluent::TimeSlicedOutput
  # configure(conf), start(), shutdown() and format(tag, time, record) are
  # the same as BufferedOutput.
  ...

  # You can use 'chunk.key' to get sliced time. The format of 'chunk.key'
  # can be configured by the 'time_format' option. The default format is %Y%m%d.
  def write(chunk)
    day = chunk.key
    ...
  end
end
```

Writing Non-buffered Output Plugins

Extend the **Fluent::Output** class and implement the following methods.

```
class SomeOutput < Fluent::Output
  # First, register the plugin. NAME is the name of this plugin
  # and identifies the plugin in the configuration file.
```

```
  Fluent::Plugin.register_output('NAME', self)

  # This method is called before starting.
  def configure(conf)
    super
    ...
  end

  # This method is called when starting.
  def start
    super
    ...
  end

  # This method is called when shutting down.
  def shutdown
    super
    ...
  end

  # This method is called when an event reaches Fluentd.
  # 'es' is a Fluent::EventStream object that includes multiple events.
  # You can use 'es.each {|time,record| ... }' to retrieve events.
  # 'chain' is an object that manages transactions. Call 'chain.next' at
  # appropriate points and rollback if it raises an exception.
  def emit(tag, es, chain)
    chain.next
    es.each {|time,record|
      $stderr.puts "OK!"
    }
  end
end
```

Customizing the Tail Input Plugin Parser

You can customize the text parser for the Tail Input plugin by extending the **Fluent::TailInput** class.

Put the following file into **/etc/fluent/plugin/in_mytail.rb**.

```
class MyTailInput < Fluent::TailInput
  Fluent::Plugin.register_input('mytail', self)

  # Override the 'configure_parser(conf)' method.
  # You can get config parameters in this method.
  def configure_parser(conf)
    @time_format = conf['time_format'] || '%Y-%M-%d %H:%M:%S'
  end

  # Override the 'parse_line(line)' method that returns the time and record.
  # This example method assumes the following log format:
  #   %Y-%m-%d %H:%M:%S\tkey1\tvalue1\tkey2\tvalue2...
  #   %Y-%m-%d %H:%M:%S\tkey1\tvalue1\tkey2\tvalue2...
  #   ...
  def parse_line(line)
```

```
    elements = line.split("\t")

    time = elements.shift
    time = Time.strptime(time, @time_format).to_i

    # [k1, v1, k2, v2, ...] -> {k1=>v1, k2=>v2, ...}
    record = {}
    while (k = elements.shift) && (v = elements.shift)
      record[k] = v
    end

    return time, record
  end
end
```

Use the following configuration file:

```
<source>
  type mytail
  path /path/to/myformat_file
  tag myapp.mytail
</source>
```

Logging

In the past, Fluentd used `$log` objects to output logs. Fluentd (v0.10.43 and above) now provides the `log` method to support the plugin-specific `log_level` parameter. Newer plugins should use the `log` method instead of `$log` objects:

```
log.info "setup foo plugin"
```

Supporting Older Fluentd Versions

The `log` method only works with Fluentd v0.10.43 or above. Please use the following code in your plugin to support older Fluentd versions:

```
module Fluent
  module FooPluginOutput < Output
    # Define `log` method for v0.10.42 or earlier
    unless method_defined?(:log)
      define_method(:log) { $log }
    end
    ...
  end
end
```

This code defines the `log` method using `$log` when the `log` method is not defined, so `log.error` becomes `$log.error` for older Fluentd versions.

Debugging plugins

Run `fluentd` with the `-vv` option to show debug messages:

```
$ fluentd -vv
```

The **stdout** and **copy** output plugins are useful for debugging. The **stdout** output plugin dumps matched events to the console. It can be used as follows:

```
# You want to debug this plugin.
<source>
  type your_custom_input_plugin
</source>

# Dump all events to stdout.
<match **>
  type stdout
</match>
```

The **copy** output plugin copies matched events to multiple output plugins. You can use it in conjunction with the stdout plugin:

```
<source>
  type tcp
</source>

# Use the tcp Input plugin and the fluent-cat command to feed events:
#  $ echo '{"event":"message"}' | fluent-cat test.tag
<match test.tag>
  type copy

  # Dump the matched events.
  <store>
    type stdout
  </store>

  # Feed the dumped events to your plugin.
  <store>
    type your_custom_output_plugin
  </store>
</match>
```

Writing test cases

Fluentd provides unit test frameworks for plugins:

```
Fluent::Test::InputTestDriver
  Test driver for input plugins.

Fluent::Test::BufferedOutputTestDriver
  Test driver for buffered output plugins.

Fluent::Test::OutputTestDriver
  Test driver for non-buffered output plugins.
```

Please see Fluentd's source code for details.

Further Reading

- Slides: Dive into Fluentd Plugin

Community

Fluentd has a thriving developer and user community. Here are some ways to come join the conversation!

Get in Touch Whether you have technical questions or just want to mingle with fellow community members, we've got you covered.

- Fluentd Google Groups
- Twitter (#fluentd, [@fluentd](http://www.twitter.com/fluentd))
- Facebook Page
- StackOverflow

Share Help others learn about the value of Fluentd by sharing what you've built!

Whether it be a company/community tech talk, a conference talk proposal, or a blog article, if you have a great story to tell we'll be happy to help spread the word. Tweet us at (#fluentd and [@fluentd](http://www.twitter.com/fluentd)) about how you are using Fluentd!

Meetup Our meetups are a great way to mingle with fellow Fluentd developers and users. Come share your ideas, discuss your challenges, and learn from each other.

- Fluentd User Group in San Francisco

Contribute An Open Source project like Fluentd couldn't exist without contributions from the developer community. Take a look at our source code repository and issue list, and consider submitting a patch.

- Source Code
- Bug/Feature Tracker

We run the documentation as an open source project as well. We encourage and appreciate any improvements, big or small. Make a pull request!

- Documentation Souce Code

Mailing List

Please join the Fluentd Google Group to share your ideas and feedback within the Fluentd community!

Source Code

The Fluentd Source Code is managed on github. The project organization can be found here.

- github.com/fluent/
- github.com/fluent/fluentd

Bug Tracking

Github Issues is used for tracking bugs. Please report any bugs or send pull requests here.

ChangeLog

Fluentd ChangeLog

Fluentd's ChangeLog can be found here.

td-agent ChangeLog

td-agent's ChangeLog can be found here.