# 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. **filter** directives determine the event processing pipelines.
4. **system** directives set system wide configuration.
5. **label** directives group the output and filter for internal

routing

1. **@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.

Fluentd accepts all non-period characters as a part of a tag. However, since the tag is sometimes used in a different context by output destinations (e.g., table name, database name, key name, etc.), **it is strongly recommended that you stick to the lower-case alphabets, digits and underscore**, e.g., ^[a-z0-9\_]+$.

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](/input) 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. See [the section below for more advanced usage](/configuration/config-file#how-match-patterns-work)). 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](/output) article.

**(3) "filter": Event processing pipeline**

The "filter" directive has same syntax as "match" but "filter" could be chained for processing pipeline. Using filters, event flow is like below:

Input -> filter 1 -> ... -> filter N -> Output

Let's add standard record\_transformer filter to "match" example.

# http://this.host:9880/myapp.access?json={"event":"data"}

<source>

@type http

port 9880

</source>

<filter myapp.access>

@type record\_transformer

<record>

host\_param "#{Socket.gethostname}"

</record>

</filter>

<match myapp.access>

@type file

path /var/log/fluent/access

</match>

Received event, {"event":"data"}, goes to record\_transformer filter first. record\_transformer adds "host\_param" field to event and filtered event, {"event":"data","host\_param":"webserver1"}, goes to file output.