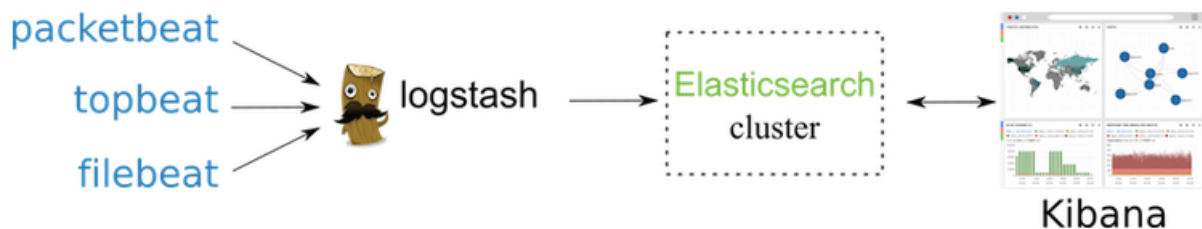


What's logstash?

Logstash is used to gather logging messages, convert them into json documents and store them in an Elasticsearch cluster.

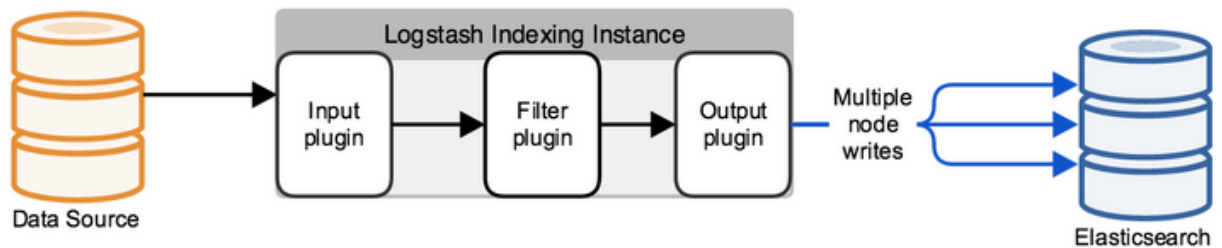


The minimal Logstash installation has one Logstash instance and one Elasticsearch instance. These instances are directly connected.

Logstash uses an input plugin to ingest data and an Elasticsearch output plugin to index the data in Elasticsearch, following the Logstash processing pipeline.

A Logstash instance has a fixed pipeline constructed at startup, based on the

instance's configuration file.



=

We must specify an input plugin.

Output defaults to stdout.

Log data is typically unstructured, often contains extraneous information. We may want to use a filter plugin to parse the log into fields, remove unnecessary information, and derive additional information from the existing fields.

For example, filters can derive geolocation information from an IP address and add that information to the logs, or parse and structure arbitrary text with the grok (see **patterns that are distributed with Logstash**) filter.

The Kibana is used as a frontend client to search for and display messages from Elasticsearch cluster.

Download logstash

We need to add Logstash to our source list, and then update our apt package database:

```
$ echo "deb http://packages.elastic.co/  
logstash/2.4/debian stable main" | sudo tee  
-a /etc/apt/sources.list  
$ sudo apt-get update
```

Install Logstash with this command:

```
$ sudo apt-get install logstash
```

```
$ /opt/logstash/bin/logstash --version  
logstash 2.4.1
```

Configuring logstash

The Logstash configuration file (/etc/logstash/conf.d/logstash.conf) defines our Logstash pipeline. We're going to configure out Logstash in later section. But in this section, we may want to just check how it looks like.

The following text represents the skeleton of a configuration pipeline:

```
input {  
}  
# filter {  
# }  
output {  
}
```

The "filter" section is optional.

Note that the skeleton configuration is non-functional, because the input and output sections don't have any valid options defined.

A Logstash pipeline in most use cases has one or more input, filter, and output plugins.

Sample : Beat input

In the following setup example, the Beat sends events to Logstash.

Logstash receives these events by using the Beats input plugin for Logstash and then sends the transaction to Elasticsearch by using the Elasticsearch output plugin for Logstash.

The Elasticsearch output plugin uses the bulk API, making indexing very efficient.

To install the required plugin, run the following command:

```
$ sudo /opt/logstash/bin/logstash-plugin  
install logstash-input-beats
```

Validating logstash-input-beats

Installing logstash-input-beats

Installation successful

Configure Logstash to listen on port 5044 for incoming Beats connections and to index into Elasticsearch.

We configure Logstash by creating a configuration file. For example, we can save the following example configuration to a file called `/etc/logstash/conf.d/logstash.conf`:

```
input {  
  beats {  
    port => 5044  
  }  
}
```

```
output {  
  elasticsearch {
```

```
hosts => "localhost:9200"
manage_template => false
index => "%{[@metadata][beat]}-%
{+YYYY.MM.dd}"
document_type => "%{[@metadata]
[type]}"
}
}
```

Logstash uses this configuration to index events in Elasticsearch in the same way that the Beat would, but we get additional buffering and other capabilities provided by Logstash.

To test the configuration:

```
$ sudo service logstash configtest
Configuration OK
```

Running logstash

Run the most basic Logstash pipeline:

```
$ /opt/logstash/bin/logstash -e 'input
```

```
{ stdin { } } output { stdout { } }
```

Note that we used the "-e" command line flag to make Logstash to accept a configuration directly from the command line. This is very useful to test configurations without having to edit a file between iterations.

This pipeline takes input from the standard input, stdin, and moves that input to the standard output, stdout, in a structured format.

When it prompts:

Settings: Default pipeline workers: 2

Pipeline main started

Just type "Hello Logstash" as the input:

```
$ /opt/logstash/bin/logstash -e 'input
```

```
{ stdin { } } output { stdout { } }'
```

Settings: Default pipeline workers: 2

Pipeline main started

Hello Logstash

2017-02-14T21:34:07.488Z laptop Hello

Logstash

Note that Logstash added timestamp and IP address information to the message.

Exit Logstash by issuing a CTRL-D command in the shell where Logstash is running.

Output Format

The rubydebug codec will output our Logstash event data using the ruby-awesome-print library.

So, by re-configuring the "stdout" output by adding a "codec", we can change the output of Logstash. By adding inputs, outputs and filters to our configuration, it is possible to massage the log data in many ways, in order to

maximize flexibility of the stored data when we are querying it.

```
$ /opt/logstash/bin/logstash -e 'input
{ stdin { } } output { stdout { codec =>
rubydebug } }'
```

Settings: Default pipeline workers: 2

Pipeline main started

Hello Logstash!

```
{
  "message" => "Hello Logstash!",
  "@version" => "1",
  "@timestamp" =>
"2017-02-15T02:24:16.236Z",
  "host" => "laptop"
}
```

Storing logs with Elasticsearch

By default Elasticsearch runs on 9200 port.

For testing purpose, we'll still take the

stdin, but the output will not be displayed on the stdout. Instead, it goes to ElasticSearch.

In other words, we can configure for Logstash to use Elasticsearch as its backend.

```
$ /opt/logstash/bin/logstash -e 'input { stdin {} } output { elasticsearch { hosts => localhost } }'
```

Type something, and Logstash will process it as before, however, this time we won't see any output, since we don't have the stdout output configured:

"Storing logs with Elasticsearch!"

We can confirm that ElasticSearch actually received the data with a curl request and inspecting the return:

```
$ curl 'http://localhost:9200/_search?pretty'
```

This should return something like this:

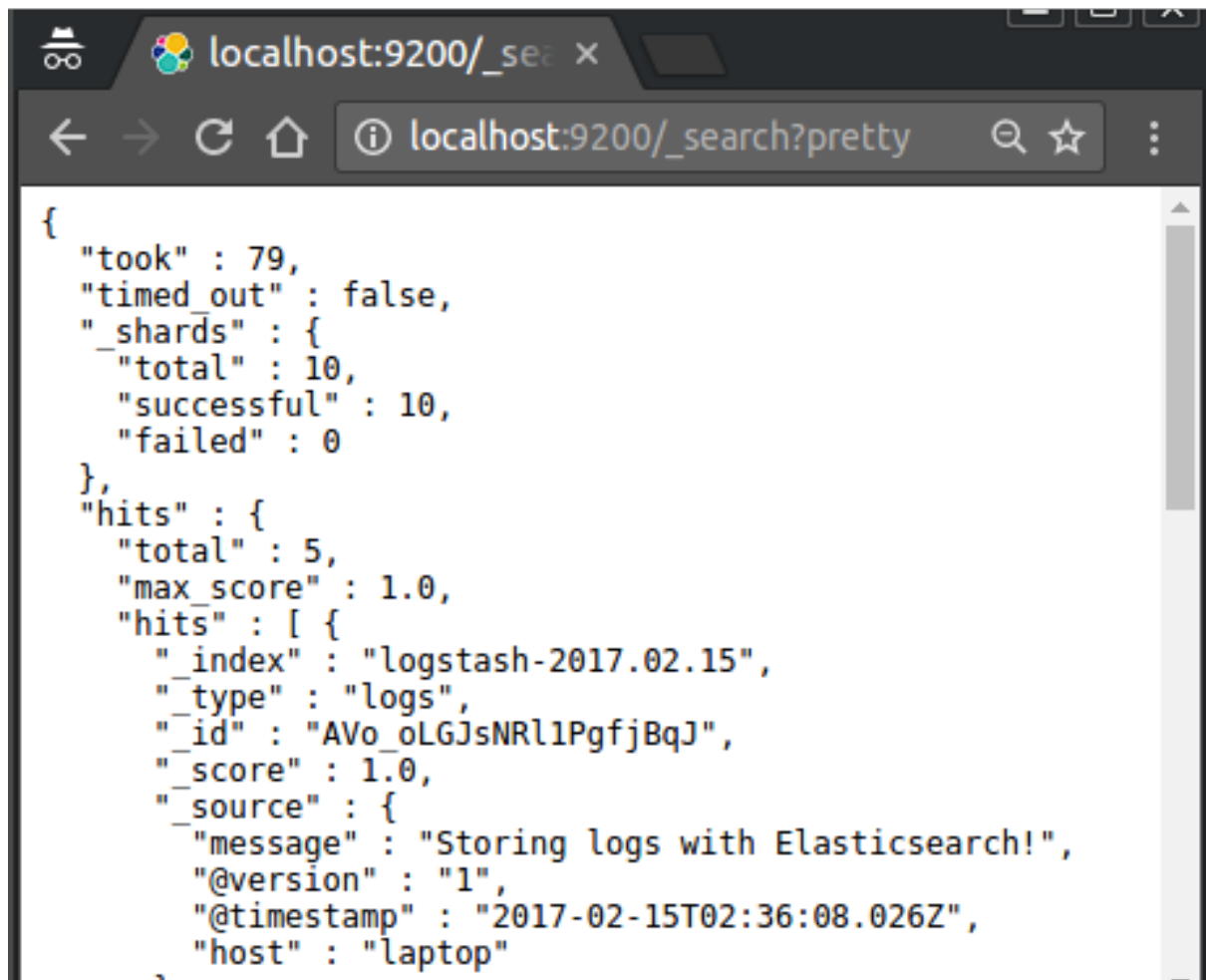
```
{
  "took" : 1980,
  "timed_out" : false,
```

```
"_shards" : {
  "total" : 10,
  "successful" : 10,
  "failed" : 0
},
"hits" : {
  "total" : 5,
  "max_score" : 1.0,
  "hits" : [ {
    "_index" : "logstash-2017.02.15",
    "_type" : "logs",
    "_id" : "AVo_oLGJsNRl1PgFjBqJ",
    "_score" : 1.0,
    "_source" : {
      "message" : "Storing logs with
Elasticsearch!",
      "@version" : "1",
      "@timestamp" :
"2017-02-15T02:36:08.026Z",
      "host" : "laptop"
    }
  }
  ...
] ]
}
```

}

For this to work, the Elasticsearch server should be running as either a background daemon or a simple foreground process, otherwise we may get an error something like "Master Not Discovered".

Go to http://localhost:9200/_search?pretty:



The screenshot shows a web browser window with the address bar displaying `localhost:9200/_search?pretty`. The page content is a JSON response from the Elasticsearch search API, formatted for readability. The response indicates that 5 hits were found in the `logstash-2017.02.15` index. The first hit is a log entry with the message "Storing logs with Elasticsearch!".

```
{
  "took" : 79,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
    "failed" : 0
  },
  "hits" : {
    "total" : 5,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "logstash-2017.02.15",
      "_type" : "logs",
      "_id" : "AVo_oLGJsNRl1PgFjBqJ",
      "_score" : 1.0,
      "_source" : {
        "message" : "Storing logs with Elasticsearch!",
        "@version" : "1",
        "@timestamp" : "2017-02-15T02:36:08.026Z",
        "host" : "laptop"
      }
    }
  ]
}
```

We've successfully stashed logs in Elasticsearch via Logstash!

Sample : Apache logs

Let's create a Logstash pipeline that takes Apache web logs as input, parses those logs to create specific, named fields from the logs, and writes the parsed data to an Elasticsearch cluster.

To get started, go [here](#) to download the sample data set (logstash-tutorial.log.gz) used in this example.

Unpack the file.

Ref : [Setting Up an Advanced Logstash Pipeline](#)

Here is our config file (/etc/logstash/conf.d/pipeline.conf):

```
input {
```

```
file {
    path => "/home/k/Downloads/
logstash-tutorial.log"
    start_position => beginning
    sincedb_path => "/dev/null"
    ignore_older => 0
}
}
```

```
filter {
    grok {
        match => { "message" => "%
{COMBINEDAPACHELOG}" }
    }
    date {
        match => [ "timestamp" , "dd/MMM/
yyyy:HH:mm:ss Z" ]
    }
    geoip {
        source => "clientip"
    }
}
```

```
output {
```

```
    elasticsearch {  
    }  
}
```

Note that we changed two default behaviors:

1. The default behavior of the file **input plugin** is to monitor a file for new information, in a manner similar to the "tail -f" command.

To change this default behavior and process the entire file, we need to specify the position where Logstash starts processing the file.

2. The default behavior of the file input plugin is to ignore files whose last modification is greater than 86400s.

To change this default behavior and process the sample input file (which date can be much older than a day), we need to specify not to ignore old files.

A representative line from the web server log sample looks like this:


```
83.149.9.216 - - [04/Jan/2015:05:13:42
+0000] "GET /presentations/logstash-
monitorama-2013/images/kibana-
search.png
HTTP/1.1" 200 203023 "http://
semicomplete.com/presentations/logstash-
monitorama-2013/" "Mozilla/5.0
(Macintosh; Intel
Mac OS X 10_9_1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/32.0.1700.77
Safari/537.36"
```

We need to parse the logs with the grok filter plugin.

The grok filter plugin is one of several plugins that are available by default in Logstash.

Because the grok filter plugin looks for patterns in the incoming log data, configuration requires us to make decisions about how to identify the patterns that are of interest to our use case.

After grok processing, the sample line has the following JSON representation:

```
clientip: "86.1.76.62",
ident: "-",
auth: "-",
timestamp: "04/Jan/2015:05:30:37
+0000",
verb: "GET",
request: "/projects/xdotool/",
httpversion: "1.1",
response: "200",
bytes: "12292",
referrer: ""http://www.haskell.org/
haskellwiki/Xmonad/
Frequently_asked_questions"",
agent: ""Mozilla/5.0 (X11; Linux x86_64; rv:
24.0) Gecko/20140205 Firefox/24.0
Iceweasel/24.3.0"",
```

In this sample, we're going to use the `%{COMBINEDAPACHELOG}` grok pattern, which structures lines from the Apache log using the following schema:

Information	Field Name
IP Address	clientip
User ID	ident
User Authentication	auth
timestamp	timestamp
HTTP Verb	verb
Request body	request
HTTP Version	httpversion
HTTP Status Code	response
Bytes served	bytes
Referrer URL	referrer
Bytes served	bytes
User agent	agent

After processing, the sample line has the following JSON representation:

```
{  
  "clientip" : "83.149.9.216",  
  "ident" : ,  
  "auth" : ,  
  "timestamp" : "04/Jan/2015:05:13:42"
```

```
+0000",  
"verb" : "GET",  
"request" : "/presentations/logstash-  
monitorama-2015/images/kibana-  
search.png",  
"httpversion" : "HTTP/1.1",  
"response" : "200",  
"bytes" : "203023",  
"referrer" : "http://semicomplete.com/  
presentations/logstash-  
monitorama-2013/",  
"agent" : "Mozilla/5.0 (Macintosh; Intel Mac  
OS X 10_9_1) AppleWebKit/537.36 (KHTML,  
like Gecko) Chrome/32.0.1700.77 Safari/  
537.36"  
}
```

Now that the web logs are broken down into specific fields, the Logstash pipeline can index the data into an Elasticsearch cluster. That's why we have the following for the output section:

```
output {  
  elasticsearch {  
  }  
}
```

Logstash uses http protocol to connect to Elasticsearch.

In this sample, we assume Logstash and Elasticsearch to be running on the same instance.

Otherwise, we can specify a remote Elasticsearch instance using hosts configuration like `hosts => "es-machine:9092"`.

Now, we may want to enhancing our data with the geoip filter plugin.

In addition to parsing log data for better searches, filter plugins can derive supplementary information from existing data.

As an example, the geoip plugin looks up IP addresses, derives geographic

location information from the addresses, and adds that location information to the logs.

That's why we added the following lines to the filter section of the pipeline.conf file:

```
geoip {  
    source => "clientip"  
}
```

The geoip plugin configuration requires data that is already defined as separate fields. Make sure that the geoip section is after the grok section of the configuration file.

We specify the name of the field that contains the IP address to look up, and the field name is clientip.

To verify our configuration, run the following command:

```
$ /opt/logstash/bin/logstash -f /etc/  
logstash/conf.d/pipeline.conf --configtest  
Configuration OK
```

Since the configuration file passed the configuration test, we can start Logstash with the following command:

```
$ /opt/logstash/bin/logstash -f /etc/logstash/conf.d/pipeline.conf
```

Now we can try a test query to Elasticsearch based on the fields created by the grok filter plugin:

```
$ curl -XGET 'localhost:9200/logstash-2015.01.04/_search?pretty&q=response=200'
```

The hit we get:

```
{  
  "took" : 63,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 5,  
    "successful" : 5,  
    "failed" : 0  
  },  
}
```

```
"hits" : {
  "total" : 98,
  "max_score" : 5.2220316,
  "hits" : [ {
    "_index" : "logstash-2015.01.04",
    "_type" : "logs",
    "_id" : "AVpApy7ug6wEMkiUMyGn",
    "_score" : 5.2220316,
    "_source" : {
      "message" : "83.149.9.216 - - [04/Jan/
2015:05:13:45 +0000] \"GET /
presentations/logstash-monitorama-2013/
images/frontend-response-codes.png
HTTP/1.1\" 200 52878 \"http://
semicomplete.com/presentations/logstash-
monitorama-2013/\" \"Mozilla/5.0
(Macintosh; Intel Mac OS X 10_9_1)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/32.0.1700.77 Safari/537.36\"",
      "@version" : "1",
      "@timestamp" :
"2015-01-04T05:13:45.000Z",
      "path" : "/home/k/Downloads/
logstash-tutorial.log",
```



```
"host" : "laptop",
"clientip" : "83.149.9.216",
"ident" : "-",
"auth" : "-",
"timestamp" : "04/Jan/2015:05:13:45
+0000",
"verb" : "GET",
"request" : "/presentations/logstash-
monitorama-2013/images/frontend-
response-codes.png",
"httpversion" : "1.1",
"response" : "200",
"bytes" : "52878",
"referrer" : "\"http://semicomplete.com/
presentations/logstash-
monitorama-2013/\"",
"agent" : "\"Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_9_1) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/
32.0.1700.77 Safari/537.36\"",
"geoip" : {
  "ip" : "83.149.9.216",
  "country_code2" : "RU",
  "country_code3" : "RUS",
```

```
    "country_name" : "Russian Federation",
    "continent_code" : "EU",
    "region_name" : "48",
    "city_name" : "Moscow",
    "latitude" : 55.752199999999999,
    "longitude" : 37.6156,
    "timezone" : "Europe/Moscow",
    "real_region_name" : "Moscow City",
    "location" : [ 37.6156,
55.752199999999999 ]
  }
}
}, {
  "_index" : "logstash-2015.01.04",
  "_type" : "logs",
  "_id" : "AVpApx4wg6wEMkiUMyGZ",
  "_score" : 0.17731485,
  ...
```

Another search for the geographic information derived from the IP address:

```
$ curl -XGET 'http://localhost:9200/
```

logstash-2015.01.04/_search?
pretty&q=geoip.city_name=Buffalo'

The hit we got:

```
{
  "took" : 47,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.8422426,
    "hits" : [ {
      "_index" : "logstash-2015.01.04",
      "_type" : "logs",
      "_id" : "AVpApy75g6wEMkiUMyH1",
      "_score" : 0.8422426,
      "_source" : {
        "message" : "108.174.55.234 - - [04/
Jan/2015:05:27:45 +0000] \"GET /?"
```

```
flav=rss20 HTTP/1.1\" 200 29941 \"-\" \"-  
\"\",  
    \"@version\" : \"1\",  
    \"@timestamp\" :  
    \"2015-01-04T05:27:45.000Z\",  
    \"path\" : \"/home/k/Downloads/  
logstash-tutorial.log\",  
    \"host\" : \"laptop\",  
    \"clientip\" : \"108.174.55.234\",  
    \"ident\" : \"-\",  
    \"auth\" : \"-\",  
    \"timestamp\" : \"04/Jan/2015:05:27:45  
+0000\",  
    \"verb\" : \"GET\",  
    \"request\" : \"/?flav=rss20\",  
    \"httpversion\" : \"1.1\",  
    \"response\" : \"200\",  
    \"bytes\" : \"29941\",  
    \"referrer\" : \"\"-\"\",  
    \"agent\" : \"\"-\"\",  
    \"geoip\" : {  
        \"ip\" : \"108.174.55.234\",  
        \"country_code2\" : \"US\",  
        \"country_code3\" : \"USA\",
```

```
"country_name" : "United States",
"continent_code" : "NA",
"region_name" : "NY",
"city_name" : "Buffalo",
"postal_code" : "14221",
"latitude" : 42.9864,
"longitude" : -78.7279,
"dma_code" : 514,
"area_code" : 716,
"timezone" : "America/New_York",
"real_region_name" : "New York",
"location" : [ -78.7279, 42.9864 ]
}
}
} ]
}
}
```

Note - the subsequent sections are largely based on official guide: [Getting Started with Logstash](#)

Parsing logs with Filebeat

We'll create a Logstash pipeline that uses Filebeat to take Apache web logs as input, parses those logs to create specific, named fields from the logs, and writes the parsed data to an Elasticsearch cluster.

Rather than defining the pipeline configuration at the command line, we'll define the pipeline in a config file.

To get started, go [here](#) to download the sample data set (logstash-tutorial.log.gz) used in this example. Unpack the file.

Configuring Filebeat to Send Log Lines to Logstash

Before creating the Logstash pipeline, we may want to configure Filebeat to send log lines to Logstash.

The Filebeat client is a lightweight, resource-friendly tool that collects logs from files on the server and forwards these logs to our Logstash instance for processing.

Filebeat is designed for reliability and low latency. Filebeat has a light resource footprint on the host machine, so the Beats input plugin minimizes the resource demands on the Logstash instance.

Usually, Filebeat runs on a separate machine from the machine running our Logstash instance. For the purposes of this tutorial, Logstash and Filebeat are running on the same machine.

The default Logstash installation includes the **Beats input** plugin.

The Beats input plugin enables

Logstash to receive events from the Elastic Beats framework, which means that any Beat written to work with the Beats framework, such as Packetbeat and Metricbeat, can also send event data to Logstash.

To install Filebeat on our data source machine, download the appropriate package from the Filebeat [product page](#). We can also refer to [Getting Started with Filebeat](#) in the Beats documentation for additional installation instructions.

To download and install Filebeat on Ubuntu 16.04, use the following commands:

```
$ curl -L -O https://download.elastic.co/beats/filebeat/filebeat_1.2.3_amd64.deb
```

```
$ sudo dpkg -i filebeat_1.2.3_amd64.deb
```

Before starting Filebeat, we should look at the configuration options in the configuration file, for example `/etc/`

filebeat/filebeat.yml. For more information about these options, see [Configuration Options](#).

Since we want to use Logstash to perform additional processing on the data collected by Filebeat, we need to check [Step 3 \(Optional\): Configuring Filebeat to Use Logstash](#).

As we guessed "(Optional)", we want to use Logstash to perform additional processing on the data collected by Filebeat, we need to configure Filebeat to use Logstash.

To do this, we edit the Filebeat configuration file to disable the Elasticsearch output by commenting it out and enable the Logstash output by uncommenting the logstash section (/etc/filebeat/filebeat.yml):

```
#####  
Filebeat  
#####  
#####
```

filebeat:

- # List of prospectors to fetch data.

prospectors:

- # Each - is a prospector. Below are the prospector specific configurations

-

- # Paths that should be crawled and fetched. Glob based paths.

- # To fetch all ".log" files from a specific level of subdirectories

- # /var/log/*/*.log can be used.

- # For each file found under this path, a harvester is started.

- # Make sure not file is defined twice as this can lead to unexpected behaviour.

- paths:

- #- /var/log/*.log

- /home/k/Downloads/logstash-tutorial.log

...

#####

Output

#####

#####

Configure what outputs to use when
sending the data collected by the beat.
Multiple outputs may be used.
output:

Elasticsearch as output
#elasticsearch:
Array of hosts to connect to.
Scheme and port can be left out and
will be set to the default (http and 9200)
In case you specify an additional path,
the scheme is required: http://localhost:
9200/path
IPv6 addresses should always be
defined as: https://[2001:db8::1]:9200
hosts: ["localhost:9200"]

Optional protocol and basic auth
credentials.
#protocol: "https"
#username: "admin"
#password: "s3cr3t"

...

```
#### Logstash as output
```

```
logstash:
```

```
# The Logstash hosts
```

```
hosts: ["localhost:5044"]
```

```
# Number of workers per Logstash host.
```

```
#worker: 1
```

In this configuration, `hosts` specifies the Logstash server and the port (5044) where Logstash is configured to listen for incoming Beats connections. Note that we set paths to point to the example Apache log file, `logstash-tutorial.log`, that we downloaded earlier. To keep the configuration simple, we did not specify TLS/SSL settings as we would in a real world scenario. To test our configuration file, run Filebeat in the foreground with the following options specified:

```
$ filebeat -configtest -e -c /etc/filebeat/
```

filebeat.yml

To use this configuration, we must also **set up Logstash** to receive events from Beats.

Setting up Logstash to receive event from Beat

In this setup, the Beat sends events to Logstash. Logstash receives these events by using the **Beats input plugin for Logstash** and then sends the transaction to Elasticsearch by using the **Elasticsearch output plugin for Logstash**.

The Elasticsearch output plugin uses the bulk API, making indexing very efficient.

To set up Logstash:

1. Make sure we have the latest compatible version of the Beats input plugin for Logstash installed. To install the required plugin, run the following command inside the logstash directory (for deb and rpm installs, the directory is **/opt/logstash**).
\$ sudo ./bin/logstash-plugin install logstash-input-beats
2. Validating logstash-input-beats
3. Installing logstash-input-beats
4. Installation successful

5. Configure Logstash to listen on port 5044 for incoming Beats connections and to index into Elasticsearch. We need to configure Logstash by creating a configuration file. For example, we can save the following example configuration to a file called **/etc/logstash/conf.d/logstash.conf**:
input {

6. beats {
7. port => 5044
8. }
9. }

```
10. output {
11.   elasticsearch {
12.     hosts => "localhost:9200"
13.     manage_template => false
14.     index => "%{[@metadata][beat]}-%
        {+YYYY.MM.dd}"
15.     document_type => "%{[@metadata]
        [type]}"
16.   }
17. }
18.
```

Logstash uses this configuration to index events in Elasticsearch in the same way that the Beat would, but we get additional buffering and other capabilities provided by Logstash.

Updating the Beats Input Plugin for Logstash

Plugins have their own release cycle and are often released independent of Logstash's core release cycle. To ensure that we have the latest version of the Beats input plugin for Logstash, run the following command from our Logstash installation (/opt/logstash/):

```
$ sudo ./bin/logstash-plugin update  
logstash-input-beats
```

```
Updated logstash-input-beats 3.1.8 to  
3.1.12
```

Keep in mind that we can update to the latest version of the plugin without having to upgrade to a newer version of Logstash. More details about working with input plugins in Logstash are available [here](#).

Let's start Logstash:

```
$ sudo /etc/init.d/logstash start  
logstash started.
```


Query into Elasticsearch

Let's check if Logstash indexed the log from filebeat successfully, and put it into Elasticsearch:

```
$ curl -XGET 'http://localhost:9200/  
logstash-2015.01.04/_search?  
pretty&q=geoip.city_name=Buffalo'
```

```
{  
  "took" : 25087,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 5,  
    "successful" : 5,  
    "failed" : 0  
  },  
  "hits" : {  
    "total" : 1,  
    "max_score" : 0.8422426,  
    "hits" : [ {  
      "_index" : "logstash-2015.01.04",  
      "_type" : "logs",
```

```
"_id" : "AVpApy75g6wEMkiUMyH1",
"_score" : 0.8422426,
"_source" : {
  "message" : "108.174.55.234 - - [04/
Jan/2015:05:27:45 +0000] \"GET /?
flav=rss20 HTTP/1.1\" 200 29941 \"-\" \"-
\"",
  "@version" : "1",
  "@timestamp" :
"2015-01-04T05:27:45.000Z",
  "path" : "/home/k/Downloads/
logstash-tutorial.log",
  "host" : "laptop",
  "clientip" : "108.174.55.234",
  "ident" : "-",
  "auth" : "-",
  "timestamp" : "04/Jan/2015:05:27:45
+0000",
  "verb" : "GET",
  "request" : "/?flav=rss20",
  "httpversion" : "1.1",
  "response" : "200",
  "bytes" : "29941",
  "referrer" : "\"-\"",
```

```
"agent" : "\"-\"",
"geoip" : {
  "ip" : "108.174.55.234",
  "country_code2" : "US",
  "country_code3" : "USA",
  "country_name" : "United States",
  "continent_code" : "NA",
  "region_name" : "NY",
  "city_name" : "Buffalo",
  "postal_code" : "14221",
  "latitude" : 42.9864,
  "longitude" : -78.7279,
  "dma_code" : 514,
  "area_code" : 716,
  "timezone" : "America/New_York",
  "real_region_name" : "New York",
  "location" : [ -78.7279, 42.9864 ]
}
}
} ]
}
}
```

