The logstash Book

Log management made easy

James Turnbull

The Logstash Book

James Turnbull

May 6, 2016

Version: v2.3.2 (e9c3ebc)

Website: The Logstash Book

Contents

Chapter 1

Shipping Events without the Logstash agent

Our log management project is going well. We've got some of our Syslog messages centralized and searchable but we've hit a snag. We've discovered some hosts and devices in our environment that can't be managed with the Logstash agent. There are a few different devices that all have varying reasons for not being able to run the agent:

- Small virtual machine with limited memory insufficient to run the agent.
- Some embedded devices and appliances without the ability to install Java and hence run the agent.
- Some outsourced managed hosts where you can't install software of your own.

So to address these hosts we're going to make a slight digression in our project and look at alternatives to running the Logstash agent and getting events to our central Logstash server.

We're going to look at three methods:

- Syslog - the traditional Linux/Unix logging framework.

- Logstash Forwarder - A lightweight log forwarding agent written by the team at Elastic.
- Filebeat - A next generation log forwarding agent, also written by the team at Elastic, designed to replace Logstash Forwarder.

Using Syslog

The first way we can get our recalcitrant devices to log to Logstash is using a more traditional logging method: Syslog. Instead of using the Logstash agent to send our logs we can enable existing Syslog daemons or services to do it for us.

To do this we're going to configure our central Logstash server to receive Syslog messages and then configure Syslog on the remote hosts to send to it. We're also going to show you how to configure a variety of Syslog services.

A quick introduction to Syslog

Syslog is one of the original standards for computer logging. It was designed by Eric Allman as part of Sendmail and has grown to support logging from a variety of platforms and applications. It has become the default mechanism for logging on Unix and Unix-like systems like Linux and is heavily used by applications running on these platforms as well as printers and networking devices like routers, switches and firewalls.

As a result of its ubiquity on these types of platforms it's a commonly used means to centralize logs from disparate sources. Each message generated by Syslog (and there are variations between platforms) is roughly structured like so:

Listing 1.1: A Syslog message

```
Dec 15 14:29:31 joker systemd-logind[2113]: New session 31581 of ↩
  user bob.
```

They consist of a timestamp, the host that generated the message (here `joker`), the process and process ID (PID) that generated the message and the content of the message.

Messages also have metadata attached to them in the form of facilities and severities. Messages refer to a facility like:

- AUTH
- KERN
- MAIL
- etcetera

The facility specifies the type of message generated, for example messages from the `AUTH` facility usually relate to security or authorization, the `KERN` facility are usually kernel messages or the `MAIL` facility usually indicates it was generated by a mail subsystem or application. There are a wide variety of facilities including custom facilities, prefixed with `LOCAL` and a digit: `LOCAL0` to `LOCAL7`, that you can use for your own messages.

Messages also have a severity assigned, for example `EMERGENCY`, `ALERT`, and `CRITICAL`, ranging down to `NOTICE`, `INFO` and `DEBUG`.

TIP You can find more details on Syslog here.

Configuring Logstash for Syslog

Configuring Logstash to receive Syslog messages is really easy. All we need to do is add the `syslog` input plugin to our central server's `/etc/logstash/conf.d/↩ central.conf` configuration file. Let's do that now:

Listing 1.2: Adding the 'syslog' input

```
input {
  redis {
    host => "10.0.0.1"
    data_type => "list"
    type => "redis-input"
    key => "logstash"
  }
  syslog {
    type => syslog
    port => 5514
  }
}
output {
  stdout { }
  elasticsearch { }
}
```

You can see that in addition to our `redis` input we've now got `syslog` enabled and we've specified two options:

Listing 1.3: The ‘syslog‘ input

```
syslog {
  type => syslog
  port => 5514
}
```

The first option, `type`, tells Logstash to label incoming events as `syslog` to help us to manage, filter and output these events. The second option, `port`, opens port 5514 for both TCP and UDP and listens for Syslog messages. By default most Syslog servers can use either TCP or UDP to send Syslog messages and when being used to centralize Syslog messages they generally listen on port 514. Indeed, if not specified, the `port` option defaults to 514. We've chosen a different port here to separate out Logstash traffic from any existing Syslog traffic flows you might have. Additionally, since we didn't specify an interface (which we could do using the `host` option) the `syslog` plugin will bind to `0.0.0.0` or all interfaces.

TIP You can find the full list of options for the `syslog` input plugin here.

Now, if we restart our Logstash agent, we should have a Syslog listener running on our central server.

Listing 1.4: Restarting the Logstash server

```
$ sudo service logstash restart
```

You should see in your `/var/log/logstash/logstash.log` log file some lines indicating the `syslog` input plugin has started:

Listing 1.5: Syslog input startup output

```
{:message=>"Starting syslog udp listener", :address=>"0.0.0.0:5514",↩
   :level=>:info}
{:message=>"Starting syslog tcp listener", :address=>"0.0.0.0:5514",↩
   :level=>:info}
```

NOTE To ensure connectivity you will need make sure any host or intervening network firewalls allow connections on TCP and UDP between hosts sending Syslog messages and the central server on port 5514.

Configuring Syslog on remote agents

There are a wide variety of hosts and devices we need to configure to send Syslog messages to our Logstash central server. Some will be configurable by simply specifying the target host and port, for example many appliances or managed devices. In their case we'd specify the hostname or IP address of our central server and the requisite port number.

Central server

- Hostname: smoker.example.com
- IP Address: 10.0.0.1
- Syslog port: 5514

In other cases our host might require its Syslog daemon or service to be specifically configured. We're going to look at how to configure three of the typically used Syslog daemons to send messages to Logstash:

- RSyslog
- Syslog-NG
- Syslogd

We're not going to go into great detail about how each of these Syslog servers works but rather focus on how to send Syslog messages to Logstash. Nor are we going to secure the connections. The `syslog` input and the Syslog servers will be receiving and sending messages unencrypted and unauthenticated.

Assuming we've configured all of these Syslog servers our final environment might look something like:

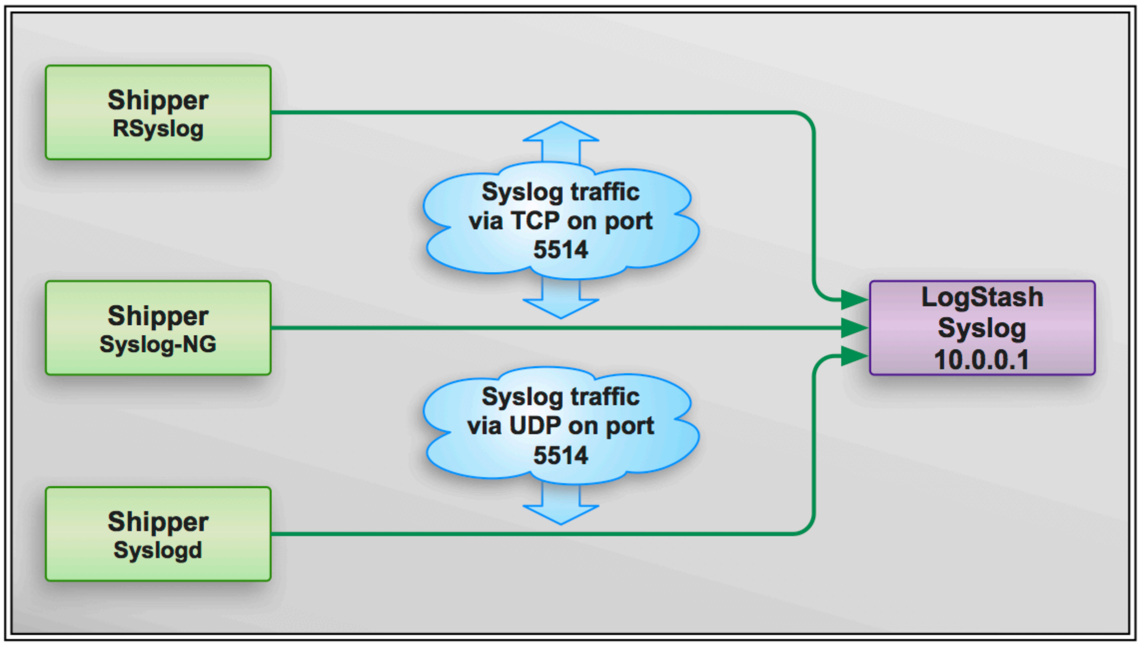
Shipper
RSyslog
Syslog traffic via TCP on port 5514
Shipper
Syslog-NG
LogStash
Syslog
10.0.0.1
Syslog traffic via UDP on port 5514
Shipper
Syslogd

Figure 1.1: Syslog shipping to Logstash

WARNING As I mentioned above Syslog has some variations between platforms. The Logstash `syslog` input plugin supports RFC3164 style syslog with the exception that the date format can either be in the RFC3164 style or in ISO8601. If your Syslog output isn't compliant with RFC3164 then this plugin will probably not work. We'll look at custom filtering in Chapter 5 that may help parse your specific Syslog variant.

Configuring RSyslog

The RSyslog daemon has become popular on many distributions, indeed it has become the default Syslog daemon on recent versions of Ubuntu, CentOS, Fedora, Debian, openSuSE and others. It can process log files, handle local Syslog and comes with a modular plug-in system.

TIP In addition to supporting Syslog output Logstash also supports the RSyslog specific RELP protocol.

We're going to add Syslog message forwarding to our RSyslog configuration file, usually `/etc/rsyslog.conf` (or on some platforms inside the `/etc/rsyslog.d↩ /` directory). To do so we're going to add the following line to the end of our `/etc/rsyslog.conf` file:

Listing 1.6: Configuring RSyslog for Logstash

```
*.* @@smoker.example.com:5514
```

NOTE If you specify the hostname, here `smoker.example.com`, your host will need to be able to resolve it via DNS.

This tells RSyslog to send all messages using `*.*`, which indicates all facilities and priorities. You can specify one or more facilities or priorities if you wish, for example:

Listing 1.7: Specifying RSyslog facilities or priorities

```
mail.* @@smoker.example.com:5514
*.emerg @@joker.example.com:5514
```

The first line would send all `mail` facility messages to our `smoker` host and the second would send all messages of `emerg` priority to the host `joker`.

The `@@` tells RSyslog to use TCP to send the messages. Specifying a single `@` uses UDP as a transport.

TIP I would strongly recommend using the more reliable and resilient TCP protocol to send your Syslog messages.

If we then restart the RSyslog daemon, like so:

Listing 1.8: Restarting RSyslog

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

Our host will now be sending all the messages collected by RSyslog to our central Logstash server.

The RSyslog imfile module One of RSyslog's modules provides another method of sending log entries from RSyslog. You can use the imfile module to transmit the contents of files on the host via Syslog. The `imfile` module works much like Logstash's `file` input and supports file rotation and tracks the currently processed entry in the file.

To send a specific file via RSyslog we need to enable the `imfile` module and then specify the file to be processed. Let's update our `/etc/rsyslog.conf` file (or if your platform supports the `/etc/rsyslog.d` directory then you can create a file-specific configuration file in that directory).

Listing 1.9: Monitoring files with the imfile module

```
module(load="imfile" PollingInterval="10")

input(type="imfile"
      File="/var/log/riemann/riemann.log"
      Tag="riemann")
```

The first line loads the `imfile` module and sets the polling internal for events to 10 seconds. It only needs to be specified once in your configuration.

The next block specifies the file from which to collect events. It has a `type` of imfile, telling RSyslog to use the imfile module. The File attribute specifies the name of the file to poll. The File attribute also supports wildcards.

Listing 1.10: Monitoring files with an imfile wildcard

```
input(type="imfile"
      File="/var/log/riemann/*.log"
      Tag="riemann")
```

This would collect all events from all files in the /var/log/riemann directory with a suffix of .log.

Lastly, the Tag attribute tags these messages in RSyslog with a tag of riemann.

Now, once you've restarted RSyslog, it will be monitoring this file and sending any new lines via Syslog to our Logstash instance (assuming we've configured RSyslog as suggested in the previous section).

TIP You can find the full RSyslog documentation here.

Configuring Syslog-NG

Whilst largely replaced in modern distributions by RSyslog, there are still a lot of platforms that use Syslog-NG including Gentoo, FreeBSD, Arch Linux and HP UX. Like RSyslog, Syslog-NG is a fully featured Syslog server but its configuration is a bit more substantial than what we needed for RSyslog.

Syslog-NG configuration comes in four types:

- source statements - where log messages come from.
- destination statements - where to send log messages.
- filter statements - how to filter or process log messages.
- log statements - actions that combine source, destination and filter statements.

Let's look inside an existing Syslog-NG configuration. Its configuration file is usually `/etc/syslog-ng.conf` or `/etc/syslog-ng/syslog-ng.conf`. You'll usually find a line something like this inside:

Listing 1.11: Syslog-NG s_src source statement

```
source s_src { unix-dgram("/dev/log"); internal(); file("/proc/kmsg"↩
  program_override("kernel"));
};
```

This basic `source` statement collects Syslog messages from the host, kernel messages and any internal messages to Syslog-NG. This is usually the default `source` on most distributions and platforms. If you don't see this `source` your Syslog-NG server may not be collecting Syslog messages and you should validate its configuration. You may also see additional `source` statements, for example collecting messages via the network from other hosts.

We then need to define a new `destination` for our Logstash server. We do this with a line like so:

Listing 1.12: New Syslog-NG destination

```
destination d_logstash { tcp("10.0.0.1" port(5144)); };
```

This tells Syslog-NG to send messages to IP address `10.0.0.1` on port 5144 via TCP. If you have domain name resolution you could instead specify our Logstash server's host name.

Lastly, we will need to specify a `log` action to combine our `source` or sources and our `destination`

Listing 1.13: New Syslog-NG log action

```
log { source(s_src); destination(d_logstash); };
```

This will send all Syslog messages from the `s_src` source to the `d_logstash`↩ destination which is our central Logstash server.

To enable the message transmission you'll need to restart Syslog-NG like so:

Listing 1.14: Restarting Syslog-NG

```
$ sudo /etc/init.d/syslog-ng restart
```

TIP You can find the full Syslog-NG documentation here.

Configuring Syslogd

The last Syslog variant we're going to look at configuring is the older style Syslogd. While less common it's still frequently seen on older distribution versions and especially in the more traditional Unix platforms.

TIP This includes many of the *BSD-based platforms including OSX.

Configuring Syslogd to send on messages is very simple. Simply find your Syslogd configuration file, usually `/etc/syslog.conf` and add the following line at the end of the file:

Listing 1.15: Configuring Syslogd for Logstash

```
*.* @smoker.example.com:5514
```

TIP You can find more details about Syslogd configuration here.

This will send all messages to the host `smoker.example.com` on UDP port 5514.

It is important to note that Syslogd generally does not support sending messages via TCP. This may be a problem for you given UDP is a somewhat unreliable protocol: there is absolutely no guarantee that the datagram will be delivered to the destination host when using UDP. Failure rates are typically low but for certain types of data including log events losing them is potentially problematic. You should take this into consideration when using Syslogd and if possible upgrade to a more fully featured Syslog server like Syslog-NG or RSyslog.

Once you've configured the Syslogd you'll need to restart the daemon, for example:

Listing 1.16: Restarting Syslogd

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

Other Syslog daemons

There are a variety of other Syslog daemons including several for Microsoft Windows. If you need to configure these then please see their documentation.

- Snare for Windows
- KiwiSyslog
- Syslog-Win32
- Cisco devices
- Checkpoint
- Juniper
- F5 BigIP
- HP Jet Direct

WARNING Remember not all of these devices will produce RFC-compliant Syslog output and may not work with the `syslog` input. We'll look at custom filtering in Chapter 5 that may assist in working with your Syslog variant.

Testing with logger

Most Unix and Unix-like platforms come with a handy utility called `logger`. It generates Syslog messages that allow you to easily test if your Syslog configuration is working. You can use it like so:

Listing 1.17: Testing with logger

```
$ logger "This is a syslog message"
```

This will generate a message from the `user` facility of the priority `notice` (`user↩
.notice`) and send it to your Syslog process.

TIP You can see full options to change the facility and priority of logger messages here.

Assuming everything is set up and functioning you should see the resulting log event appear on your Logstash server:

Listing 1.18: Logstash log event from Syslog

```
{
  "host" => "joker.example.com",
  "priority" => 13,
  "timestamp" => "Dec 17 16:00:35",
  "logsource" => "joker.example.com",
  "program" => "bob",
  "pid" => "23262",
  "message" =>"This is a syslog message",
  "severity" => 5,
  "facility" => 1,
  "facility_label" => "user-level",
  "severity_label" => "Notice",
  "@timestamp" => "2012-12-17T16:00:35.000Z",
  "@version => "1",
  "message" => "<13>Dec 17 16:00:35 joker.example.com bob[23262]: ↩
    This is a syslog message",
  "type" => "syslog"
}
```

Filebeat

Filebeat is a lightweight, open source shipper for logs. It replaces the Logstash Forwarder or Lumberjack. It can tail logs, manages log rotation and can send log data on to Logstash or even directly to Elasticsearch.

Filebeat is part of a larger collection of data shipping tools called Beats. Other Beats include Packetbeat, for shipping web and network protocol data and Topbeat, a metrics and monitoring data shipper. There are several other Beats in development, including community contributions, for monitoring things like Docker and Nginx. Beats are licensed with the Apache 2.0 license and written in Golang.

TIP There's also a Windows Event Log beat called Winlogbeat if you're collecting logs on Microsoft Windows.

Configure Filebeat on our central server

Let's first configure our central server to receive data from Filebeat. To do this we use a new input plugin called `beats`. The `beats` plugin was introduced in Logstash version 1.5.4 and later. If you have an earlier Logstash version you should upgrade to version 2.1.0 or later. If you're still missing the plugin you can manually install it using the `logstash-plugin` command. From inside the `/opt/logstash` directory we would run.

Listing 1.19: Manually installing the beats input

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

Now we have our plugin let's add it to our `central.conf` configuration file.

Listing 1.20: Adding the beats input

```
input {
  redis {
    host => "10.0.0.1"
    data_type => "list"
    type => "redis-input"
    key => "logstash"
  }
  syslog {
    type => syslog
    port => 5514
  }
  beats {
    port => 5044
  }
}
output {
  stdout { }
  elasticsearch { }
}
```

We've added the `beats` plugin and specified one option: `port`. The `port` option controls which port Logstash will receive logs from, here `5044`.

TIP You can find the full documentation for the `beats` input on the Elastic site.

If we now restart Logstash we will have the `beats` input enabled.

Listing 1.21: Restarting Logstash for Beats

```
$ sudo service logstash restart
```

Installing Filebeat on the remote host

Now we need to download, compile and install Filebeat on a remote agent. We're going to choose a new Ubuntu host called `gangsteroflove.example.com`.

We can download Filebeat as a DEB (it's also available as an RPM, a tarball or a Windows executable installer) from the Elastic.com download site.

Listing 1.22: Downloading Filebeat

```
$ wget https://download.elastic.co/beats/filebeat/filebeat_1.0.0↩
  _amd64.deb
```

Now we can install Filebeat.

Listing 1.23: Installing Filebeat

```
$ sudo dpkg -i filebeat_1.0.0_amd64.deb
```

After installation you can see that an example configuration file has been created in the `/etc/filebeat` directory.

Configuring Filebeat

Filebeat is configured via a YAML file called `filebeat.yml`, located in the `/etc/↩ filebeat` directory. Filebeat comes with a commented example file that explains all of Filebeat's local options. Let's create our own file now.

Listing 1.24: Our new filebeat.yml file

```
filebeat:
  prospectors:
    -
    paths:
      - /var/log/*.log
    input_type: log
    document_type: beat
  registry: /var/lib/filebeat/registry
output:
  logstash:
    hosts: ["10.0.0.1:5044"]
logging:
  to_files: true
  files:
    path: /var/log/filebeat
    name: filebeat
    rotateeverybytes: 10485760
  level: error
```

The `filebeat.yml` file is divided into stanzas. The most relevant to us are `prospectors`, `output` and `logging`. The `prospectors` tells Filebeat what files to gather logs from and the `output` tells Filebeat where to send those files. The last stanza, `logging`, controls Filebeat's own logging. Let's look at each in turn now, starting with `prospectors`.

Listing 1.25: The prospectors section

```
  prospectors:
    -
      paths:
        - /var/log/*.log
      input_type: log
      document_type: beat
  registry: /var/lib/filebeat/registry
```

Each stanza, marked with a `paths` statement, represents a file or collection of files you want to "prospect". Here we've grabbed all of the files ending in `*.log` in the `/var/log` directory. The `input_type` controls what sort of file is being read, here a standard log file. You can also use this setting to read from `STDIN`. The last option, `document_type`, controls the value of the `type` field in Logstash. The default is `log` but we've updated it to `beat` so we can distinguish where our logs are coming from. The last option, `registry`, records file offsets and we'll talk more about it in a moment.

To match files and directories, Filebeat supports all Golang-style globs. For example, we could also get everything in subdirectories too with a glob.

Listing 1.26: The prospectors section

```
  prospectors:
    -
      paths:
        - /var/log/*/*.log
. . .
```

Or multiple sets of paths like so:

Listing 1.27: The prospectors section

```
  prospectors:
    -
      paths:
        - /var/log/*/*.log
        - /opt/application/logs/*.log
. . .
```

Filebeat will grab all files ending in `*.log` from both these paths.

Filebeat will also take care of log rotation. It recognizes when a file has been rotated and grabs the new file. Filebeat also handles tracking progress reading a file. When Filebeat reads a file it will mark its current read position in the file in a catalogue called a registry. The default registry, which we've defined using the `registry` option, is at `/var/lib/filebeat/registry`. Let's look inside that file.

Listing 1.28: The /var/lib/filebeat/registry file

```
{"/var/log/auth.log":{"source":"/var/log/auth.log","offset":956674,"↩
  FileStateOS":{"inode":1180057,"device":64769}},"/var/log/dpkg.log"↩
  :{"source":"/var/log/dpkg.log","offset":23515,"FileStateOS":{"inode↩
  ":1180391,"device":64769}},"/var/log/kern.log":{"source":"/var/log/↩
  kern.log","offset":54270249,"FileStateOS":{"inode":1180046,"device"↩
  :64769}}}
```

We see a list of files that Filebeat is collecting logs from and their current offset. If we were to restart Filebeat then it would check the `registry` file and resume collecting logs from those offsets. This stops duplicate logs being sent or Filebeat restarting logging from the start of a file rather than the current point. If you need to reset the registry you can just delete the `/var/lib/filebeat/registry` file.

Now we've defined where we want to collect logs from we now need to define where to send those logs. Filebeat can send log entries from the host to Logstash or even directly to Elasticsearch. It does that in the `output` stanza. Let's look at our `output` stanza now.

Listing 1.29: The Filebeat output stanza

```
output:
  logstash:
    hosts: ["10.0.0.1:5044"]
```

We've defined an output type of `logstash` and specified the `hosts` option. This tells Filebeat to connect to a Logstash server. The `hosts` option is an array that can contain one or Logstash hosts running the `beats` input plugin. In our case we're connecting to the Logstash host at `10.0.0.1`.

Lastly, we want Filebeat to log some information about what it is doing. To handle this we configure the `logging` stanza.

Listing 1.30: The Filebeat logging stanza

```
logging:
  to_files: true
  files:
    path: /var/log/filebeat
    name: filebeat
    rotateeverybytes: 10485760
  level: error
```

Here we've configured the `to_files` option to `true` to tell Filebeat to log to a file. We could also log to Syslog or `STDOUT`. We've then told Filebeat where to log, inside the `files` block. We have given Filebeat a `path`, `/var/log/filebeat` (you might need to create this directory), the `name` of the file to log to and controlled when the file will rotate, when it fills up to `rotateeverybytes` of `10485760` or 10Mb. We also specify a `level` which specifies the level of detail in the output. We're going to use `error`, which will show any `error` level output.

TIP Filebeat is hugely configurable. You can send data with TLS, control network and transport options like back-off and manage how files are handled when they rotate. Amongst many other settings. You'll find the commented `filebeat.yml` example file very useful for exploring settings and further documentation is available in the Filebeat documentation.

To start the Filebeat service we can use the `service` command.

Listing 1.31: Starting the Filebeat service

```
$ sudo service filebeat start
```

If we now check out Logstash server we should see JSON-encoded log entries arriving from our Filebeat service with a type of `beat`. We can then use the type field to route and process those logs.

Using the Logstash Forwarder

If you can't use Filebeat, the Logstash agent or Syslog isn't an option then the legacy Logstash Forwarder (formerly Lumberjack) may be an option.

The Logstash Forwarder is largely deprecated and replaced with Filebeat. This documentation exists for backwards compatibility purposes for people with older versions of the book. You should use Filebeat from now on.

The Logstash Forwarder (hereafter Forwarder) is designed to be a lightweight client and server for sending messages to Logstash. It includes a custom-designed protocol and unlike any of our previous transports it also includes some security via SSL encryption of the traffic as well as compression of log traffic. Using the Forwarder you can:

- Follow files (it also respects rename and truncation conditions like log rotation).
- Receive `stdin`, which is useful for things like piping output to the Forwarder.

So why use the Forwarder at all instead of say Syslog? The Forwarder is designed to be tiny, incredibly memory conservative and very, very fast. None of the existing Syslog servers are really designed to scale and transmit large volumes of events and they often break down at large volumes.

To get it running we're going to configure the Forwarder `input` plugin on the central Logstash server and then install and configure the Forwarder on a remote host.

Configure the Logstash Forwarder on our central server

The first step in configuring the Forwarder on our central server is to generate a self-signed SSL certificate to secure our log traffic. This is a mandatory step for configuring the Forwarder. You can only send events with the SSL transport enabled and encrypting your traffic.

NOTE You could also use a real certificate if you wished but this is a simpler and faster way to get started.

Create a self-signed SSL certificate

We're going to quickly step through creating the required SSL certificate and key as it is a pretty standard process on most platforms. It requires the `openssl` binary as a prerequisite.

Listing 1.32: Checking for openssl

```
$ which openssl
/usr/bin/openssl
```

We first generate a private key.

Listing 1.33: Generating a private key

```
$ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
......................................+++
....+++
e is 65537 (0x10001)
```

This creates a new file called `server.key`. This is our SSL certificate key. Don't share it or lose it as it is integral to the security of our solution.

Next we're going to generate a Certificate Signing Request or CSR from which we're going to generate our SSL certificate.

Listing 1.34: Generating a CSR

```
$ openssl req -new -key server.key -batch -out server.csr
```

This will generate a file called `server.csr` which is our signing request.

Lastly we're going to sign our CSR and generate a new certificate.

Listing 1.35: Signing our CSR

```
$ openssl x509 -req -days 3650 -in server.csr -signkey server.key -↩
  out server.crt
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd
Getting Private key
```

This will result in a file called `server.crt` which is our self-signed certificate.

NOTE We've set a very long expiry, 3650 days, for the certificate.

Now let's copy the required files:

- server.key
- server.crt

To our Logstash configuration directory:

Listing 1.36: Copying the key and certificate

```
$ sudo cp server.key server.crt /etc/logstash
```

If you wish to renew the self-signed certificate at some point you'll need to keep the original key and CSR otherwise you can delete the original key and the CSR to keep things tidy.

Listing 1.37: Cleaning up

```
$ rm server.orig.key server.csr
```

WARNING There have been some issues with Forwarder and SSL. I recommend you read this ticket and this documentation if you have problems.

Configuring the Lumberjack input

Now we've got our self-signed key we need to add the `lumberjack` input to our central Logstash server's configuration. To do this we're going to edit our `/etc/logstash/conf.d/central.conf` configuration file.

Listing 1.38: Adding the Lumberjack input

```
input {
  redis {
    host => "10.0.0.1"
    data_type => "list"
    type => "redis-input"
    key => "logstash"
  }
  syslog {
    type => syslog
    port => 5514
  }
  lumberjack {
    port => 6782
    ssl_certificate => "/etc/logstash/server.crt"
    ssl_key => "/etc/logstash/server.key"
    type => "lumberjack"
  }
}
output {
  stdout { }
  elasticsearch { }
}
```

You can see we've added a new input plugin called `lumberjack`:

Listing 1.39: The Lumberjack input

```
lumberjack {
  port => 6782
  ssl_certificate => "/etc/logstash/server.crt"
  ssl_key => "/etc/logstash/server.key"
  type => "lumberjack"
}
```

To configure it we've specified a `port` of `6782`. The `lumberjack` input will listen on this TCP port for incoming events. By default the plugin will be bound to all interfaces but you can specify a specific interface with the `host` option.

NOTE You'll need to ensure any firewalls on the host or between the remote client and the central server allow traffic on this port.

We've also specified the certificate and key we created in the last section in the `ssl_certificate` and `ssl_key` options respectively. If we'd put a pass phrase on the key we could specify it here with the `ssl_key_passphrase` option.

Lastly, we've specified a type of `lumberjack` so we can identify events coming in from this input.

TIP You can find the full documentation for the `lumberjack` input here.

If we now restart Logstash we will have the `lumberjack` input enabled.

Listing 1.40: Restarting Logstash for Lumberjack

```
$ sudo service logstash restart
```

We can tell if the input plugin has loaded from our `/var/log/logstash/↩ logstash.log` log file. Check for the following message:

Listing 1.41: Checking Lumberjack has loaded

```
{
  :timestamp => "2013-08-23T04:09:04.426000+0000",
  :message => "Input registered",
  :plugin=><LogStash::Inputs::Lumberjack ssl_certificate=>"/etc/↩
   logstash/server.crt", ssl_key=>"/etc/logstash/server.key", type=>↩
   "lumberjack", charset=>"UTF-8", host=>"0.0.0.0">,
  :level=>:info
}
```

The `lumberjack` input is now ready to receive events from our remote clients.

Installing the Logstash Forwarder on the remote host

Now we need to download, compile and install the Forwarder on a remote agent. We're going to choose a new Ubuntu host called `gangsteroflove.example.com`.

We can download the Logstash forwarder as a DEB (it's also available as an RPM, a tarball or a Windows executable installer) from the Elastic.com download site.

Listing 1.42: Downloading the Forwarder

```
$ wget https://download.elastic.co/logstash-forwarder/binaries/↩
 logstash-forwarder_0.4.0_amd64.deb
```

Listing 1.43: Installing the Forwarder

```
$ sudo dpkg -i logstash-forwarder_0.4.0_amd64.deb
Selecting previously unselected package logstash-forwarder.
(Reading database ... 45980 files and directories currently ↩
  installed.)
Unpacking logstash-forwarder (from logstash-forwarder_0.4.0_amd64.↩
  deb) ...
Setting up logstash-forwarder (0.4.0) ...
```

From this package the Forwarder will be installed into the `/opt/logstash-↩ forwarder` directory.

An example configuration file has been created at `/etc/logstash-forwarder.↩ conf`.

We now need to copy our SSL server certificate across to the remote host so we can use it to validate our SSL connection.

Listing 1.44: Copying the Forwarder's SSL certificate

```
smoker$ scp /etc/logstash/server.crt bob@gangsteroflove:/etc/
```

As I explained either, the Forwarder works by tailing files or taking input from `STDIN`. We're going to focus on tailing files, which covers most of the logging scenarios you're likely to have.

The Forwarder is configured with a JSON-based configuration file that is specified using the `-config` command line flag.

Let's edit our example file now and replace the contents with:

Listing 1.45: The logstash-forwarder.conf file

```
{
  "network": {
    "servers": [ "10.0.0.1:6782" ],
    "ssl ca": "/etc/server.crt",
    "timeout": 15
  },
  "files": [
{
  "paths": [
    "/var/log/syslog",
    "/var/log/*.log"
  ],
  "fields": { "type": "syslog" }
},
{
  "paths": [
    "/var/log/apache2/*.log"
  ],
  "fields": { "type": "apache" }
}
  ]
}
```

Let's examine the contents of our `logstash-forwarder.conf` configuration file. It's divided into two JSON stanzas: `network` and `files`.

The `network` stanza configures the transport portion of the Forwarder. The first entry `servers` configures the target destination for any Logstash Forwarder log entries, in our case the server at `10.0.0.1` on port `6782` as we configured in our `lumberjack` input above. You can specify an array of servers. The Forwarder will chose one at random and then keep using that server until it becomes unresponsive at which point it will try another server.

We've also defined the location of the SSL server certificate we downloaded from our server. Finally we've specified a server timeout of 15 seconds. This is the time that the Forwarder will wait for a response from a server. If it doesn't receive a response it will select a new server to send to or if no other servers are available it will enter a wait-retry-wait cycle until a server is available.

The next stanza, `files`, controls which files we're monitoring for log events. The `files` stanza is made up of `paths` and optional `fields` blocks. The `paths` blocks specify files or globs of files to watch and receive log entries from. In the case of our example configuration we're monitoring the `/var/log/syslog` file, all files in `/var/log/` ending in `*.log` and all files in the `/var/log/apache2/` directory ending in `*.log`. You can also see that each `path` block also has a `fields` block. This block will add a `type` field of `syslog` and `apache` respectively to any log entries from these files.

Now let's run the Forwarder on the command line to test this out.

Listing 1.46: Testing the Forwarder

```
$ /opt/logstash-forwarder/bin/logstash-forwarder -config /etc/↩
  logstash-forwarder.conf
```

Testing the Logstash Forwarder

Now let's trigger a Syslog message to make sure things are working okay.

Listing 1.47: Test the Forwarder

```
$ logger "This is a message eh?"
```

We should see the connection made on the local client in the Forwarder's `STDOUT`:

Listing 1.48: The Forwarder connection output

```
2013/08/23 04:18:59 publisher init
2013/08/23 04:18:59.444617 Setting trusted CA from file: /etc/↩
  logstash-forwarder/server.crt
2013/08/23 04:18:59.445321 Starting harvester: /var/log/auth.log
. . .
2013/08/23 04:18:59.446050 Starting harvester: /var/log/kern.log
2013/08/23 04:18:59.446459 Starting harvester: /var/log/apache2/↩
  access.log
2013/08/23 04:18:59.505609 Connected to localhost:6782
```

```
2013/08/23 04:18:59.056065 Registrar received 1 events
2013/08/23 04:18.59.057591 Saving registrar state.
```

On the central Logstash server we should see a matching event appear in `/var/↩ log/logstash/logstash.log`:

Listing 1.49: Forwarder events

```
2013-08-23T04:19.00.197Z lumberjack://gangsteroflove.example.com/var↩
 /log/syslog: Aug 23 04:19:00 gangsteroflove.example.com root: This ↩
 is a message eh?
```

Managing the Logstash Forwarder as a service

Running the Forwarder on the command line isn't a viable option so we're going to implement it as a service. We're going to run the Forwarder using an init script.

Let's enable the Forwarder to run on boot.

Listing 1.50: Starting the Forwarder on boot

```
$ sudo update-rc.d logstash-forwarder defaults
```

If we're happy with these files we can start the Forwarder.

Listing 1.51: Starting the Forwarder

```
$ sudo service logstash-forwarder start
logstash-forwarder started
```

We now confirm the Forwarder is running by checking the PID file, `/var/run/↩ logstash-forwarder.pid` or by confirming there is a running process:

Listing 1.52: Checking the Forwarder process

```
$ ps -aux | grep 'logstash-forwarder'
root  1501  0.0  0.2  59736  2832 ?SNl  19:51   0:00 /opt/logstash-↩
  forwarder/bin/logstash-forwarder -config /etc/logstash-forwarder/↩
  logstash-forwarder.conf
```

We can also send a `logger` event from our remote host that should show up on the central Logstash server.

Other log shippers

If the Logstash Forwarder doesn't suit your purposes there are also several other shippers that might work for you.

Beaver

The Beaver project is another Logstash shipper. Beaver is written in Python and available via PIP.

Listing 1.53: Installing Beaver

```
$ pip install beaver
```

Beaver supports sending events via Redis, `STDIN`, or zeroMQ. Events are sent in Logstash's `json` codec.

TIP This is an excellent blog post explaining how to get started with Beaver and Logstash.

Woodchuck

Another potential shipping option is Woodchuck. It's designed to be lightweight and is written in Ruby and deployable as a RubyGem. It currently only supports outputting events as Redis (to be received by Logstash's `redis` input) but plans include ZeroMQ and TCP output support. It has not been recently updated.

Others

- Syslog-shipper
- Remote_syslog
- Message::Passing

Summary

We've now got some of the recalcitrant hosts into our logging infrastructure via some of the methods we've learnt about in this chapter: Syslog, the Logstash Forwarder or some of the other log shippers. That should put our log management project back on track and we can now look at adding some new log sources to our Logstash infrastructure.

List of Figures

Listings

Index

Thanks! I hope you enjoyed the book.

ISBN 978-0-9888202-1-0
9 780988 820210 51200>