NetWitness® Platform

Version 12.4.1.0

Logstash and NetWitness Integration Guide

NETWITNESS
Platform

Contact Information

NetWitness Community at https://community.netwitness.com contains a knowledge base that answers common questions and provides solutions to known problems, product documentation, community discussions, and case management.

Trademarks

License Agreement

It is advised not to deploy third-party repos or perform any change to the underlying NetWitness Operating System that is not part of the supported NetWitness version. Any such change outside of the NetWitness approved image may result in a service or functionality conflict and require a reimage of the NetWitness system to bring NetWitness back to an optimized functional state. In the event a third-party repo is deployed, or other non-supported change is made by the customer without NetWitness approval, the customer takes full responsibility for any system malfunction until the issue can be remediated through troubleshooting efforts or a reimage of the service.

Third-Party Licenses

This product may include software developed by parties other than RSA. The text of the license agreements applicable to third-party software in this product may be viewed on the product documentation page on NetWitness Community. By using this product, a user of this product agrees to be fully bound by terms of the license agreements.

Note on Encryption Technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when using, importing or exporting this product.

Distribution

Miscellaneous

This product, this software, the associated documentations as well as the contents are subject to NetWitness' standard Terms and Conditions in effect as of the issuance date of this documentation and which can be found at https://www.netwitness.com/standard-form-agreements/.

June, 2024

Contents

Overview

This document is intended to provide a general overview of Logstash and NetWitness integration. The intention is to provide enough implementation detail that users can have comfort using and troubleshooting these integrations on their own.

To describe Logstash, here is some introductory text from Logstash reference documentation:

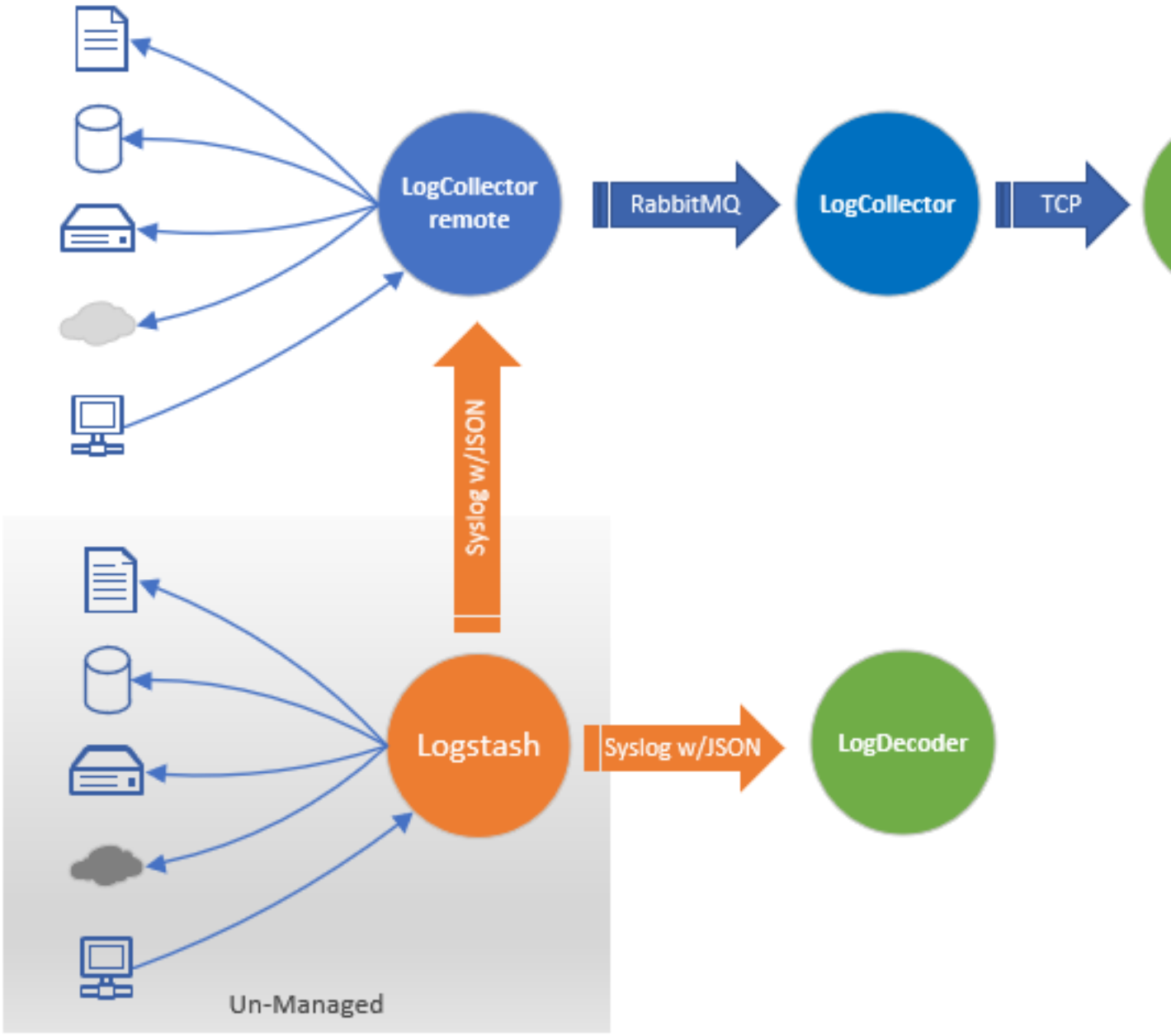
> *Logstash is an open source data collection engine with real-time pipelining capabilities. Logstash can dynamically unify data from disparate sources and normalize the data into destinations of your choice…*

From a NetWitness standpoint, there are two basic use cases:

- For customers that have an event source for which NetWitness does not already provide an integration, or if you want a customized integration that is different from the one provided by NetWitness.
- For customers that already have an existing Logstash configuration, you can use Logstash to integrate as many of your event sources as you like. Integrating your event sources should be a matter of updating the destination for where you currently send the log information: either adding NetWitness as a destination, or changing your current output destination to NetWitness.

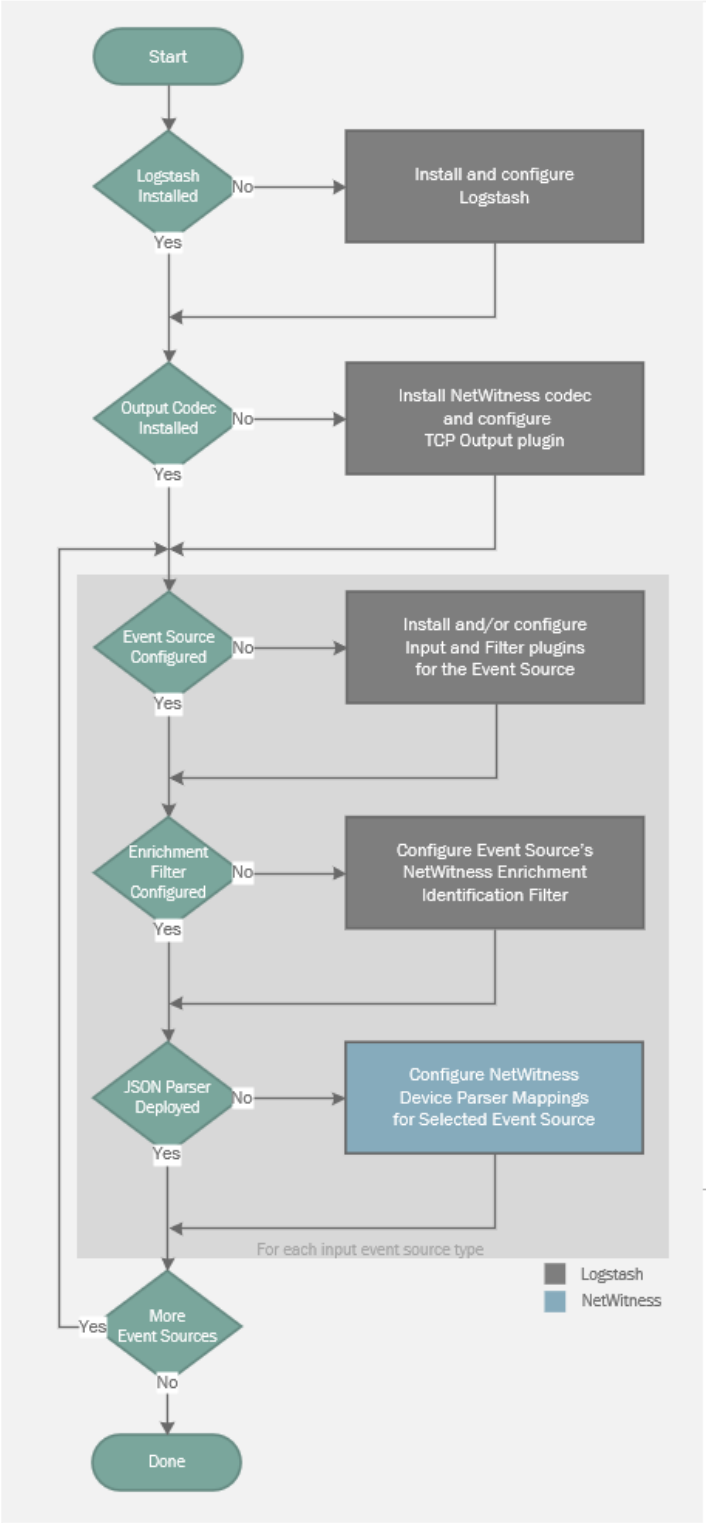
From latest version onwards, the Logstash server is packaged and supported along with the NetWitness Log Collector or Virtual Log Collector (VLC) service to provide easy access to Logstash. This is referred to as Managed Logstash and it eliminates the need for a sep-arate Logstash server outside of the NetWitness Platform. For more information, see Con-figure Logstash Event Sources in NetWitness in the *Log Collection Configuration Guide*.

The following diagram displays a view of how Logstash integrates with the NetWitness.


LogCollector
remote
RabbitMQ
LogCollector
TCP
Syslog w/JSON
Logstash
Syslog w/JSON
LogDecoder
Un-Managed

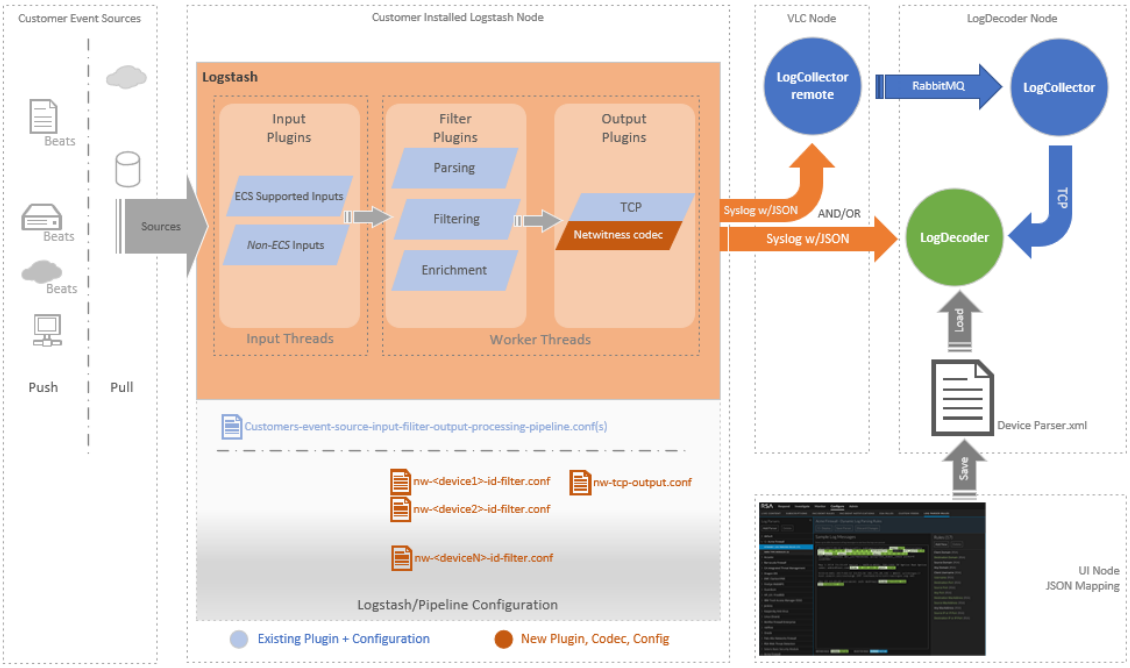
Configuration Process

The following flowchart describes the steps customers take to integrate Logstash with NetWitness, depending on their prior familiarity with and use of Logstash.



The following sequence describes the data flow from an event until it becomes NetWitness meta in a Log Decoder.

1. An event source generates events.
2. The collection plugin (for example a Beats plugin) collects events from the event source.
3. Logstash processes the data from the events.
4. A NetWitness codec encodes the Logstash-processed data into a format that can be consumed by NetWitness.
5. An output plugin sends the processed event data to the NetWitness.
6. A JSON parser populates meta from the processed event data.



Install Logstash

Skip this section if you already have Logstash installed and configured.

Please ensure that you follow all the security-related best practices and guidelines outlined in the Logstash documentation to avoid any potential security risks.

You can install either the free, open source version of Logstash (OSS) or the paid version (Elastic).

Information on released versions of Logstash is available at Logstash Reference. Links in the following steps show an example of installing current version of Logstash on Linux.

1. Install the service: Installing Logstash (OSS free version)
2. Based on your OS, after installation do one of the following:
    - Linux: set Logstash to run as a service: Running Logstash
    - Windows: see Running Logstash on Windows
3. Next, enable Logstash to start when the system boots up:
    - For CentOS, see https://www.unix.com/man-page/centos/1/systemctl/
    - For Debian, see https://wiki.debian.org/systemd/documentation

    Those are 2 examples: adjust instructions according to your particular Operating System.

For Generic Troubleshooting Instructions for Logstash, follow this link: Logstash Troubleshooting

If you are using CentOS, note the following:

- Logstash logs are stored in /var/log/logstash/logstash-plain.log
- If you install logstash using rpm install, make sure it installs as logstash user and folders get created with the same user: **not** the root user.

Install and Configure the NetWitness Codec

To forward Logstash events to the NetWitness in RFC-5424 format, you need to install the NetWitness codec on your system and refer to it in your output plugin configuration.

To install or update the codec:

The following procedure can be performed on either Linux or Windows: instructions that are specific to an OS are noted.

1. Download offline installer from NetWitness Link in the following location: NetWitness Codec Installer
2. Copy the downloaded NetWitness ZIP archive to the system where Logstash runs.
3. Open a command prompt and change directory to Logstash home:
    - On Linux: cd /usr/share/logstash.
    - On Windows: cd: ***logstash_directory***

      For example: cd: c:\Logstash\
4. Stop the logstash service, if it is running.
    - On Linux, run the following command:

      systemctl stop logstash
    - On Windows, open the Services window (you can search for it or find it from the Start menu), then locate the Logstash service in the list and click **Stop the service**.
5. Run the following command and check to see if **logstash-codec-netwitness** is listed.
    - On Linux:

      bin/logstash-plugin list
    - On Windows:

      .\bin\logstash-plugin list
6. If it is listed, delete it by running the following command:
    - On Linux:

      bin/logstash-plugin remove logstash-codec-netwitness
    - On Windows:

      .\bin\logstash-plugin remove logstash-codec-netwitess
7. To install the latest package, run the following command:

- On Linux:

  bin/logstash-plugin install file:///***<path-to-file>***/logstash-codec-netwitness-offline-**<version>**.zip

- On Windows:

  .\bin\logstash-plugin install file:///***<path-to-file>***/logstash-codec-netwitness-offline-**<version>**.zip

  Make sure to use forward slashes (/) as a separator between Windows subfolders for the path to the logstash installation package.

If the codec is installed correctly, you receive a confirmation message. The following is an example:

Installing file: /usr/share/logstash/logstash-codec-netwitness-offline-1.0.0.zip

Install successful

8. Below are default paths for logstash configurations. All output, filter, and input configuration files are under these folders.

   we would go over those configuration files in later sections.

   - For Linux: /etc/logstash/conf.d/
   - For Windows: ***logstash_directory***/config/

   For details on output files, see "Configure Logstash Output Plugins" on page 13. For details on input and filter plugins, see "Logstash Input and Filter plugins" on page 19.

9. Start the logstash service:

   - On Linux, run the following command:

     systemctl start logstash

   - On Windows, open the Services window (you can search for it or find it from the Start menu), then locate the Logstash service in the list and click **Start the service**.

Configure Logstash Output Plugins

Logstash TCP Output

In order to send the events from Logstash to NetWitness, we use the TCP output plugin: https://www.elastic.co/guide/en/logstash/current/plugins-outputs-tcp.html

The TCP output is configured with the NetWitness codec, which formats the outgoing events to be consumable by a NetWitness Log Decoder or Virtual Log Collector (VLC).

The following is an example of a properly configured output block using TCP & the NetWitness codec:

Output Block

```
output {
  tcp {
    id => "nw-output-tcp"
    host => "10.10.1.2" #IP or Hostname of destination Log Decoder or VLC
    port => 514
    codec => netwitness
  }
}
```

Logstash TLS Output

The output block can be further configured to allow for TLS communication between Logstash and NetWitness. An example of a properly configured output block using TLS and the NetWitness codec:

TLS Output Block

```
output {
  tcp {
    id => "nw-output-tcp"
    host => "10.10.1.2" #IP or Hostname of destination Log Decoder or VLC
    port => 6514
    ssl_enable => true
    codec => netwitness
  }
}
```

TLS with Log Decoder (or Virtual Log Collector) Verification

TLS can also be set up to verify the Log Decoder or Virtual Log Collector (VLC) to which it will be communicating. To do this, the Root and Intermediate CA certificates need to be obtained and stored in a truststore for Logstash.

1. On the Log Decoder (or VLC) to which you will be sending events, run the following command:

   cat /etc/pki/nw/ca/nwca-cert.pem /etc/pki/nw/ca/ssca-cert.pem > nw-truststore.pem
2. Copy the nw-truststore.pem file to the Logstash machine and store it in a known location.
3. Create a certificate for the Logstash machine using a self-signed CA or your own CA.
4. Store the cert and private key files in a location of your choosing.

   You need to specify the locations of these files in your TLS output block.

The following code snippet shows an example of a properly configured output block using TLS and the NetWitness codec

Output Block with Verification

```
output {
  tcp {
    id => "nw-output-tcp"
    host => "10.10.1.2" #IP or Hostname of destination Log Decoder or VLC
    port => 6514
    ssl_enable => true
    ssl_verify => true
    ssl_cacert => "/path/to/certs/nw-truststore.pem"
    ssl_key => "/path/to/certs/privkey.pem"
    ssl_cert => "/path/to/certs/cert.pem"
    codec => netwitness
  }
}
```

Configure the Event Source

If you have logstash already install and configured to collect events from desired event-sources you can skip this section.

As a starting point, view the list of input plugins here: Input Plugins. This is not meant to be an exhaustive list, just as a starting point to view some of the available input plugins.

The remainder of this section describes some examples.

Collect Apache File Logs

Typically, Apache logs are written to files on the disk, so to collect events from file you could use the **Filebeat** plugin.

To install and configure Filebeat, see the following websites:

1. Install the Filebeat service on Linux. Refer to the following link: Filebeat Installation
2. Configure Filebeat to collect from specific logs. Refer to the following link: Filebeat Configuration
3. Configure Filebeat to send the output to Logstash. Refer to the following link: Filebeat Logstash Output

Collect CentOS Audit Logs

To collect audit events from an operating system (for example CentOS), you could use the **Auditbeat** plugin.

To install and configure Auditbeat, see the following websites:

1. Install the Auditbeat service on Linux. Refer to the following link: Auditbeat Installation
2. Configure Auditbeat to collect from specific logs. Refer to the following link: Auditbeat Configuration
3. Configure Auditbeat to send the output to Logstash. Refer to the following link: Auditbeat Logstash Output

Configure Parameters for Filebeat or Auditbeat

For Filebeat or Auditbeat plugin, make sure to configure input and output parameters. Modify /etc/filebeats/filebeat.yml as shown below.

To enable file collection, modify the **Filebeat inputs** section as shown in the following image:

```
#=========================== Filebeat inputs =============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/log/*.log
    #- c:\programdata\elasticsearch\logs\*
```

Modify the Outputs as follows:

- Comment out settings in the **Elasticsearch Output** section
- Uncomment **Logstash output** settings and provide logstash IP and Port.

Below is an example of the sections after changes are made:

```
#================================== Outputs ===================================

# Configure what output to use when sending the data collected by the beat.

#-------------------------- Elasticsearch output ------------------------------
#output.elasticsearch:
  # Array of hosts to connect to.
  # hosts: ["localhost:9200"]

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  #username: "elastic"
  #password: "changeme"

#----------------------------- Logstash output --------------------------------
output.logstash:
  # The Logstash hosts
  hosts: ["localhost:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"
```

Configure Logstash Filters to Add NetWitness Meta

In order for an event to be processed in the Log Decoder as a specific data type, you need to add some meta key information to the event in Logstash.

- [@metadata][nw_type] — NetWitness device parser content name
- [@metadata][nw_msgid] — NetWitness message pattern to parse message
- [@metadata][nw_source_host] — Originating event source's IP or host value

The value for nw_type *must* match the device parser name. It should be composed of lowercase characters, numbers, or underscore and be less than 29 characters in length.

Optionally, you can add the following meta key:

[@metadata][nw_collection_host] — Collection system identifier (**lc.cid**)

By default, the NetWitness codec sends the complete JSON event as payload to the NetWitness Log Decoder. If the NetWitness nw_type device parser type has a custom payload format and failover payload format, the NetWitness codec plugin must be configured to use them. Please see the "Configure the Event Source" on page 15 section for more details.

The following code snippet contains an example of adding the required meta:

Code to Populate NetWitness Meta

```
filter {
  if ![@metadata][nw_type] {
    if [agent][type] == "filebeat" {
      mutate {
        add_field => {
          "[@metadata][nw_type]" => "linux"
          "[@metadata][nw_msgid]" => "LOGSTASH001"
          "[@metadata][nw_source_host]" => "%{[host][hostname]}"
        }
      }
    }
  }
}
```

Advanced NetWitness Configuration

Grok Filter Plugin

You can use Grok to parse incoming logs and extract valuable meta information. For details, see Grok filter plugin: match. The meta extracted using grok is part of the full event package sent to NetWitness, where it can be mapped to NetWitness meta.

Resources:

- List of Grok patterns: https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns
- Grok debugger: https://grokdebug.herokuapp.com/

Logstash Input and Filter plugins

See the following URLs:

- Input Plugins
- Filter Plugins

You can use the beats input plugin for Logstash to receive events from beats sources, including Filebeat & Auditbeat: Beats Input Plugin

You can use the drop filter plugin for Logstash to filter out unwanted logs being passed through Logstash: Drop Filter Plugin

Filter out unwanted logs

You can also filter out unwanted logs using the drop-event processor for Filebeat & Auditbeat.

- Examples for Filebeat:

  Filebeat Reference: Filtering and Enhancing Data

  Filebeat Reference: Drop Events

- Examples for Auditbeat:

  Auditbeat Reference: Filter and Enhance the Exported Data

  Auditbeat Reference: Drop Events

Configure heartbeat plugin to send test logs to NetWitness

The **Heartbeat** plugin can be used to send a test message to verify connectivity between logstash and NetWitness. For details, see the Heartbeat Plugin configuration guide: Heartbeat Input Plugin.

Example of Heartbeat Plugin

```
...

heartbeat {
  id => "sample_plugin"
  #interval => 60
  count => 3
  message => "sequence"
  add_field => {
  "[@metadata][nw_type]" => "logstash_testlog"
  "msg" => "This is test log from some eventsource"
}

...
```

Configure logstash to persist events in case of failure (Recommended)

By default, Logstash uses in-memory queues to buffer events. The size of these in-memory queues is not configurable. If there is a machine failure, or if the service is forcibly stopped, the contents of these queues are lost. To protect against data loss in these situations, Logstash supports persistent queues that are stored on disk and thus can survive failures. For details, see Logstash Persistent Queues.

NetWitness Codec Advanced Configuration

By default, the NetWitness codec sends the complete JSON event as payload to the NetWitness Log Decoder. If the NetWitness nw_type device parser type has a custom payload format, you must configure the NetWitness codec plugin to recognize this custom format.

The payload_format and payload_format_failover mappings use nw_type as the key. The payload_format mapping is searched first for the device type (nw_type). If the device type is not set, or no format is specified for nw_type, or the system fails to make all configured variable substitutions, the complete JSON output is used as the payload. If the primary format from the payload_format mapping fails, the payload_format_failover mapping is tried. If that also fails, the complete JSON output is used as the payload. You can use The format can use Logstash event field syntax for this custom configuration.

The following code snippet shows example of adding meta with custom payload formats.

Add meta with custom payload formats

```
output {
   if [@metadata][nw_type] { # Only targeted NetWitness items
   tcp {
```

```
        id => "netwitness-tcp-output-conf-output"
        host => "127.0.0.1"
        port => 514
        ssl_enable => false
        codec => netwitness {
           # Payload format mapping by nw_type.
           # If nw_type is absent or formatting fails,
           # JSON event is used as the payload
           payload_format => {
              "apache" => "%APACHE-4-%{verb}: %{message}"
           }
           # Failover format, if above format fails
           # If nw_type is absent or formatting fails,
           # JSON event is used as the payload
           payload_format_failover => {
              "apache" => "%APACHE-4: %{message}"  # When verb is missing
              }
           }
        }
     }
  }
```

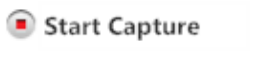
Troubleshoot Installation Issues

If you encounter any issues during installation of Logstash, see Logstash Installation and Setup.

Configure NetWitness to Collect Events

You need to start capture on the Log Decoder to which your are sending your Logstash data.

To start or restart network capture on a Log Decoder:

1. Log in to **NetWitness** and click the **ADMIN** icon > **Services**.
   The Services view is displayed.
2. Select a **Log Decoder** service.
3. Under  (actions), select **View** > **System**.
4. In the toolbar, click 

.

   If the toolbar is displaying the Stop Capture (

) icon, then capture has already been started.

By default, Log Decoders support events that are up to 32 KB in size. If your events are getting truncated on the Log Decoder, use the following procedure to change the event size:

1. Change LogDecoder REST config at **http://*LogDecoder_IP*:50102/-decoder/config**, where ***LogDecoder_IP*** is the IP address of your Log Decoder.
2. Set pool.packet.page.size to 64 KB.
3. Restart the Log Decoder: this is required after you change the pool.packet.page value.

If you are collecting events larger than 64 KB in size, follow instructions above in the "Filter out unwanted logs" on page 19 section. You can drop unwanted logs or fields for a specific event source, to reduce the size of the incoming data.

Linux Event Source Example

This section shows sample input, filters and output configuration to collect system and audit events from CentOS.

Input Plugin

An input plugin enables a specific source of events to be read by Logstash. The following code represents an example input plugin.

input-beats.conf

```
# Below input block collects events using beats plugins (e.g filebeats, auditbeats)
# Skip this block if it's already defined in another pipeline.
input {
    beats {
        port => 5044
    }
}
```

Make sure that port 5044 is open on the Logstash machine. As an example, if Logstash is on a CentOS system, run the following commands to open port 5044:

firewall-cmd --add-port=5044/tcp
firewall-cmd --add-port=5044/tcp --permanent
firewall-cmd --reload

Output Plugin

An output plugin sends event data to a particular destination. Outputs are the final stage in the event pipeline.

output-netwitness-tcp.conf

```
# Below is tcp output plugin with netwitness codec to tranform events in syslog and send
it to LogDecoder
# Only one of these configurations can be within the same pipeline.
output {
  #if [@metadata][nw_type] { # Only targeted Netwitness items
    tcp {
      id => "netwitness-tcp-output-conf-output"
      host => "10.10.100.100"  ## LogDecoder IP
      port => 514
      ssl_enable => false
      #ssl_verify => true
      #ssl_cacert => "/path/to/certs/nw-truststore.pem"
      #ssl_key => "/path/to/certs/privkey.pem"
      #ssl_cert => "/path/to/certs/cert.pem"
```

```
      codec => netwitness {
        # Payload format mapping by nw_type.
        # If nw_type is absent or formatting fails, JSON event is used as the payload
        payload_format => {
          "apache" => "%APACHE-4-%{verb}: %{message}"
        }
        # Failover format, if above format fails
        # If nw_type is absent or formatting fails, JSON event is used as the payload
        payload_format_failover => {
          "apache" => "%APACHE-4: %{message}"  # When verb is missing
        }
      }
    }
  #}
}
```

Filter Plugin

A filter plugin performs intermediary processing on an event. Below is a filter plugin configuration for system events collected from linux using the Filebeat plugin.

linux-system.conf

```
# Filters are often applied conditionally depending on the characteristics of the events.
# Requires these additional configurations within the same pipeline:
#   input-beats.conf
#   output-netwitness-tcp.conf

filter {
if ![@metadata][nw_type] {
  if [ecs][version] and [host][hostname] and [agent][type] == "filebeat" {
    if [event][module] == "system" {
      mutate {
        add_field => {
          "[@metadata][nw_type]" => "linux"
          "[@metadata][nw_msgid]" => "LOGSTASH001"
          "[@metadata][nw_source_host]" => "%{[host][hostname]}"
        }
      }
    }
  }
 }
}
```

Below is filter plugin configuration for audit events collected from linux using the Auditbeat plugin.

linux-audit.conf

```
filter {
    if ![@metadata][nw_type] { # Update Once
```

```
    if [ecs][version] and [host][hostname] and [agent][type] == "auditbeat" {
      if [event][module] == "audit" {
        mutate {
          add_field => {
            "[@metadata][nw_type]" => "linux"
            "[@metadata][nw_msgid]" => "LOGSTASH002"
            "[@metadata][nw_source_host]" => "%{[host][hostname]}"
          }
        }
      }
    }
  }
```

Create a Pipeline

It is recommended to have one pipeline for each input type. For example, all beats collection should be in the same pipeline. To run collection as separate pipeline, create a directory and add above input, filters, and output configuration files to it.

Example Pipeline for Beats

```
/etc/logstash/pipeline1/
/etc/logstash/pipeline1/input-beats.conf
/etc/logstash/pipeline1/output-netwitness-tcp.conf
/etc/logstash/pipeline1/linux-system.conf
/etc/logstash/pipeline1/linux-audit.conf
```

Modify /etc/logstash/pipeline.yml and add the following entries:

Add to pipeline.yml

```
- pipeline.id: my-sample-pipeline-1
  path.config: "/etc/logstash/pipeline1/*.conf"
```

Build Custom JSON Parser

This section is intended for advanced programmers who want to build their own JSON parser. It describes how to build a Logstash parser for a sample device. We use the Linux device as an example throughout.

This chapter is optional: you do not need to build a custom JSON parser from scratch to input logs from Logstash to NetWitness.

Major sections in this document:

- Configure a filter by defining several required pieces of metadata.
- Examine a sample log message from the Linux device
- Walk through creating the parser, based on the sample log message
- View the parsed meta from the sample log message, as it appears on the Log Decoder

Sample JSON Log Received on Log Decoder

Let's examine a sample log and discuss its contents.

<13>1 - Centos7 linux - LOGSTASH001 [lc@36807 lc.ctime="1585886465037" lc.cid="Centos7" lc.ctype="logstash"] {"message": "msg='op=PAM:accounting grantors=pam_access,pam_unix,pam_localuser acct=root exe=/usr/sbin/crond hostname=? addr=? terminal=cron res=success'", "user":{ "email":"john.deaux@test.com", "username":"CORP\\deauxj" }, "host": { "name": "Centos7", "hostname": "Centos7", "containerized": false, "architecture": "x86_64", "id": "d1059ac783b24eb7bbde70a41fa572c9", "os": { "name": "CentOS Linux", "kernel": "3.10.0-1062.el7.x86_64", "version": "7 (Core)", "codename": "Core", "platform": "centos", "family": "redhat" } }, "@timestamp": "2020-04-03T04:01:05.037Z", "files": ["test1.log", "test2.log", "test3.log"],"machine_details" : { "1" : { "hostname" : "USXXLinux" }, "2" : { "hostname" : "USXXWindows" }}}

The first portion of the log is the RFC-5424 header:

<13>1 - **Centos7 linux** - **LOGSTASH001** [lc@36807 lc.ctime="1585886465037" lc.cid="Centos7" lc.ctype="logstash"]

This header contains the information that we used in setting our fields above:

- **nw_source_host**: Centos7 (Hostname)
- **nw_type**: linux (Device Type)
- **nw_msgid**: LOGSTASH001 (Message ID)

The remainder of the log is the JSON Payload.

JSON Payload

```
{
```

```
    "message": "msg='op=PAM:accounting grantors=pam_access,pam_unix,pam_localuser acct-
t=root exe=/usr/sbin/crond hostname=? addr=? terminal=cron res=success'",

    "user": {
        "email": "john.deaux@test.com",
        "username": "CORP\\deauxj"
    },

    "host": {
        "name": "Centos7",
        "hostname": "Centos7",
        "containerized": false,
        "architecture": "x86_64",
        "id": "d1059ac783b24eb7bbde70a41fa572c9",
        "os": {
            "name": "CentOS Linux",
            "kernel": "3.10.0-1062.el7.x86_64",
            "version": "7 (Core)",
            "codename": "Core",
            "platform": "centos",
            "family": "redhat"
        }
    },

    "@timestamp": "2020-04-03T04:01:05.037Z",

    "files": [
        "test1.log",
        "test2.log",
        "test3.log"
    ],

    "machine_details": {
        "1": { "hostname": "USXXLinux"},
        "2": { "hostname": "USXXWindows"}
    }

}
```


Create the JSON Parser for a Linux Device

Now that we have the sample log from the Linux device, we can construct a filter plugin for this device.

Initial Parser to Match Message ID and Device Type

The parser name should match the device type. We call this initial parser v20_linuxmsg.xml, which matches the message ID from the event. We set content to a variable, logstash_json_payload, which represents the JSON payload. We will parse the payload later in the process.

Message ID and Device Type Parsing

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<DEVICEMESSAGES
    name="linux"
    displayname="Linux"
    group="Unix">
    <VERSION device="2.0"/>

    <MESSAGE
        id1="LOGSTASH001"
        id2="LOGSTASH001"
        content="&lt;logstash_json_payload&gt;" />

    <!-- Additional logic to parse JSON payload -->

</DEVICEMESSAGES>
```

Map Payload Contents to Datatypes

We create datatypes to map each element from the payload to meta that can be saved to the NetWitness database.

The entire payload is assigned to the **FileBeatsEvent** datatype.

```
<VARTYPE name="logstash_json_payload" dataType="FileBeatsEvent"/>
```

The timestamp is parsed an assigned to the **InternetTime** datatype.

```
<DataType name="InternetTime"dateTime="%W-%M-%DT%H:%T:%S.%V%E" />
```

Parse the Message String

Using the parser above, we parse (using the FineParse type defined above) the message string from the log file.

"message": "msg='**op**=PAM:accounting grantors=pam_access,pam_unix,pam_localuser **acct**=\"root\"
exe=\"/usr/sbin/crond\" hostname=? addr=? terminal=cron **res**=success'",

The following code extracts the values from the string and saves them to meta keys:

- **op** is saved to **operation.id**
- **acct** is saved to **service.account**
- **res** is saved to **result**
- **exe** is saved to **process.src**

Note that he **search** flag is set to **true** to parse key/value pars regardless of their order in the string.

Extract Values: save to Meta

```
<DataType name="TagValParse" regex="(?: |^)(?:exe=(\S+)|acct=(\S+)|res=(\S+)|op=(\S+))"
```

```
search="true">
    <Capture index="1" meta="process.src" />
    <Capture index="2" meta="service.account" />
    <Capture index="3" meta="result" />
    <Capture index="4" meta="operation.id" />
</DataType>
```

The following code:

- Assigns the whole key as **message** and maps it to the **FineParse** type.
- The **FineParse** type is then mapped to **TagValParse** type.

```
<DataType name="FineParse" regex="msg='(.*)'">
    <Capture index="1" type="TagValParse" />
</DataType>

<DataType name="FileBeatsEvent" type="ElasticCommonSchemaSubset">
    <Capture key="/message" type="FineParse" meta="message"/>
    <Capture key="/user/email" meta="email"    />
    <Capture key="/user/username" type="DomainUser"/>
    <Capture key="/files/" meta="sourcefile" />
    <Capture key="/machine_details//hostname"   meta="host.dst"/>
</DataType>
```

Give the previous string and the code above, the output on the Log Decoder is as follows:

message: msg='op=PAM:accounting grantors=pam_access,pam_unix,pam_localuser acct=root exe=/usr/sbin/crond
  hostname=? addr=? terminal=cron res=success'
service.account: root
process.src: /usr/sbin/crond
result: success
operation.id: PAM:accounting

Parse an Array in JSON

In our sample JSON log file from earlier, one section contained an array object:

```
"files": [
  "test1.log",
  "test2.log",
  "test3.log"
]
```

To fetch all the values of an array, you need to define a capture key enclosed in forward slashes to fetch all values, for example **/files/**.

```
<DataType name="FileBeatsEvent" type="ElasticCommonSchemaSubset">
   <Capture key="/files/" meta="sourcefile" />
```

```
  </DataType>
```

Using the code above on the sample array, the following would be the output on the Log Decoder:

sourcefile: test1.log
sourcefile: test2.log
sourcefile: test3.log

Parse a Nested JSON Object

Let's look at an example nested object from our sample log file from earlier:

```
"host": {
  "name": "Centos7",
  "hostname": "Centos7",
  "containerized": false,
  "architecture": "x86_64",
  "id": "d1059ac783b24eb7bbde70a41fa572c9",
  "os": {
    "name": "CentOS Linux",
    "kernel": "3.10.0-1062.el7.x86_64",
    "version": "7 (Core)",
    "codename": "Core",
    "platform": "centos",
    "family": "redhat"
  }
}
```

To fetch nested values, you need to build a path that contains the keys from each nested level. For example, to fetch the OS name from our example, you use the following code:

```
  <DataType name="ElasticCommonSchemaSubset">
     <Capture key="/host/os/name"> meta="OS" />
  </DataType>
```

Using the code above on the sample nested object, the following would be the output on the Log Decoder:

OS: CentOS Linux

Capture Data That Has Varying Parent Key

When capturing structured data types like JSON, instead of a numbered capture index, you can provide a field name path that uses the key attribute. For example, assume we want to capture the **hostname** from **machine_details** and ignore the indexed key:

```
"machine_details": {
  "1": { "hostname": "USXXLinux"},
  "2": { "hostname": "USXXWindows"}
}
```

To fetch the required values, which have a varying parent key name, we leave the parent key empty in the path:

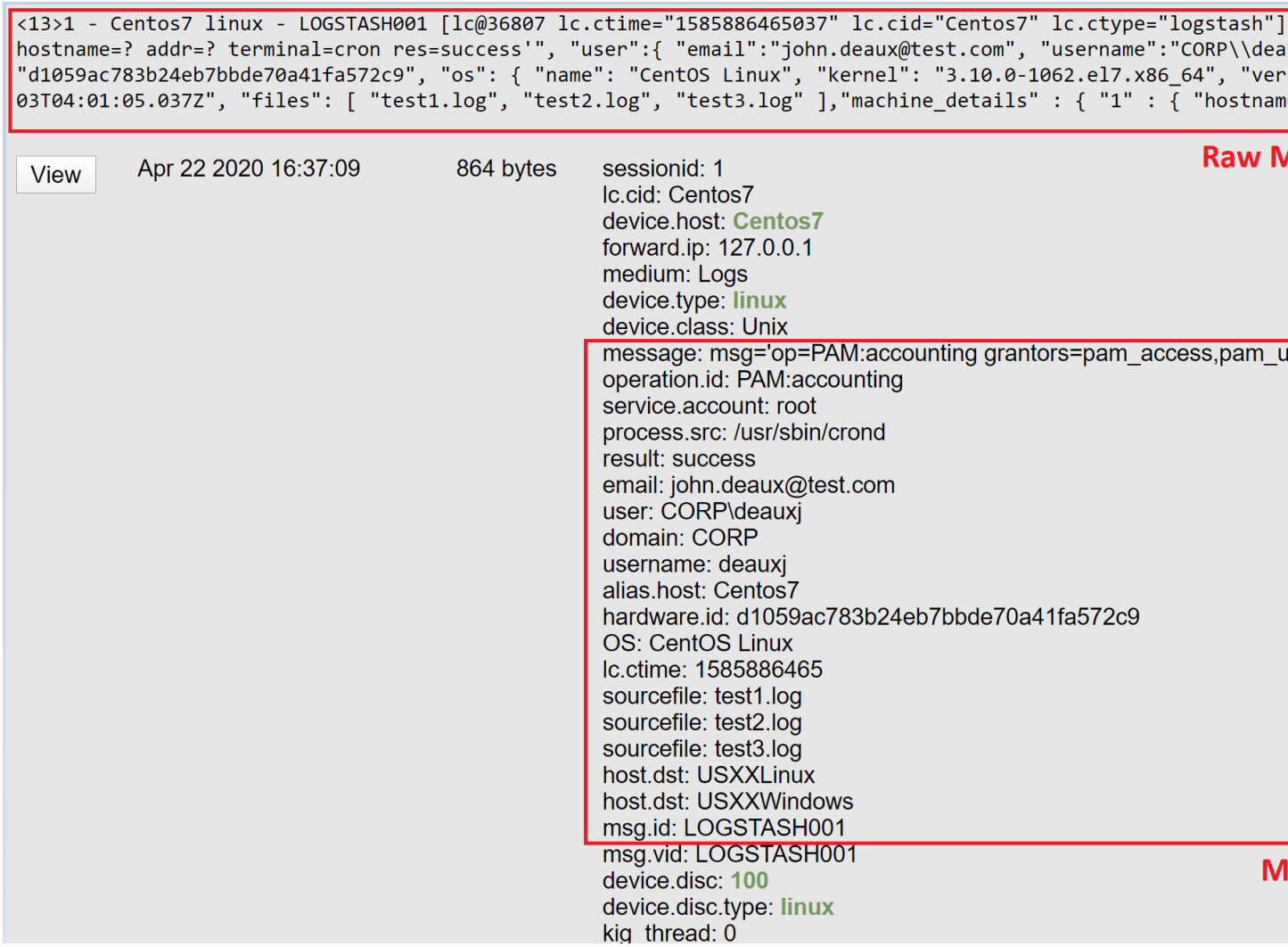
```
<DataType name="FileBeatsEvent" type="ElasticCommonSchemaSubset">
   <Capture key="/machine_details//hostname">  meta="host.dst" />
</DataType>
```

Using the code above on the sample, the following would be the output on the Log Decoder:

host.dst: USXXLinux
host.dst: USXXWindows

The Parsed Example Event on the Log Decoder

Assuming the sample log message from the beginning of this document, and using the parser that we have built, the image below details the event as it would appear on the Log Decoder.



The following representation of the sample log has meta values highlighted.

<13>1 - Centos7 linux - **LOGSTASH001** [lc@36807 **lc.ctime="1585886465037"** lc.-cid="Centos7" lc.ctype="logstash"] {**"message":** "msg='**op=PAM:accounting** grant-ors=pam_access,pam_unix,pam_localuser **acct=root exe=/usr/sbin/crond** hostname=? addr=? terminal=cron **res=success**'", "user":{ **"email":"-john.deaux@test.com"**, **"username":"CORP\\deauxj"** }, "host": { "name": "Centos7", **"hostname": "Centos7"**, "containerized": false, "architecture": "x86_64", **"id": "d1059ac783b24eb7bbde70a41fa572c9"**, "os": { **"name": "CentOS Linux"**, "kernel": "3.10.0-1062.el7.x86_64", "version": "7 (Core)", "codename": "Core", "platform": "centos", "family": "redhat" } }, "@timestamp": "2020-04-03T04:01:05.037Z", "files": [**"test1.log", "test2.log", "test3.log"**],"machine_details" : { "1" : { **"hostname" : "USXXLinux"** }, "2" : { **"hostname" : "USXXWindows"** }} }

Example Parser Listing

The following code represents the complete parser, including the components we built earlier in this document.

Example Parser Listing

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<DEVICEMESSAGES
   name="linux"
   displayname="Linux"
   group="Unix">
   <VERSION device="2.0" />

<MESSAGE
      id1="LOGSTASH001"
      id2="LOGSTASH001"
      content="&lt;logstash_json_payload&gt;" />

<VARTYPE name="logstash_json_payload" dataType=FileBeatsEvent"/>

<DataType name="InternetTime" dateTime="%W-%M-%DT%H:%T:%S.%V%E" />

<DataType name="CollectionTime" type="InternetTime" meta="lc.ctime"/>

<DataType name="ElasticCommonSchemaSubset" format="JSON">
   <Capture key="/@timestamp">     type="CollectionTime" />
   <Capture key="/host/hostname">  meta="alias.host" />
   <Capture key="/host/id">        meta="hardware.id" />
   <Capture key="/host/os/name">   meta="OS" />
</DataType>

<DataType name="DomainUser" regex="(?:(\w+)\\)?(\w+)">
   <Capture index="0" meta="user" />
   <Capture index="1" meta="domain" />
   <Capture index="2" meta="username" />
</DataType>

<DataType name="TagValParse" regex="(?: |^)(?:exe=(\S+)|acct=(\S+)|res=(\S+)|op=(\S+))"
search="true">
   <Capture index="1" meta="process.src" />
   <Capture index="2" meta="service.account" />
   <Capture index="3" meta="result" />
   <Capture index="4" meta="operation.id" />
</DataType>

<DataType name="FineParse" regex="msg='(.*)'">
   <Capture index="1" type="TagValParse" />
</DataType>

<DataType name="FileBeatsEvent" type="ElasticCommonSchemaSubset">
   <Capture key="/message" type="FineParse" meta="message"/>
   <Capture key="/user/email" meta="email" />
   <Capture key="/user/username" type="DomainUser"/>
   <Capture key="/files/" meta="sourcefile" />
   <Capture key="/machine_details//hostname" meta="host.dst" />
```

```
</DataType>

</DEVICEMESSAGES>
```

Deploy JSON parser

After you have built or changed a JSON parser, you need to upload it to the NetWitness Log Decoder.

1. SSH to the Log Decoder system.
2. Copy the custom parser file to the following folder:

   /etc/netwitness/ng/envision/etc/devices/***eventsource***

   where ***eventsource*** is the name of the event source. You may need to create the folder if it doesn't already exist.

   For example, we need to create **linux** folder under /etc/net-witness/ng/envision/etc/devices directory and copy the **v20_linuxmsg.xml** parser file to /etc/netwitness/ng/envision/etc/devices/linux directory.
3. To get the new parser loaded into memory, you need to reload the parsers on the Log Decoder.

Reload Parsers from REST

From a browser, run the REST reload command by entering the following URL:

http://***<logdecoder_ip>***:50102/decoder/parsers?msg=reload

For example, if your Log Decoder IP address is **10.10.100.101**, use the following string:

http://10.10.100.101:50102/decoder/parsers?msg=reload

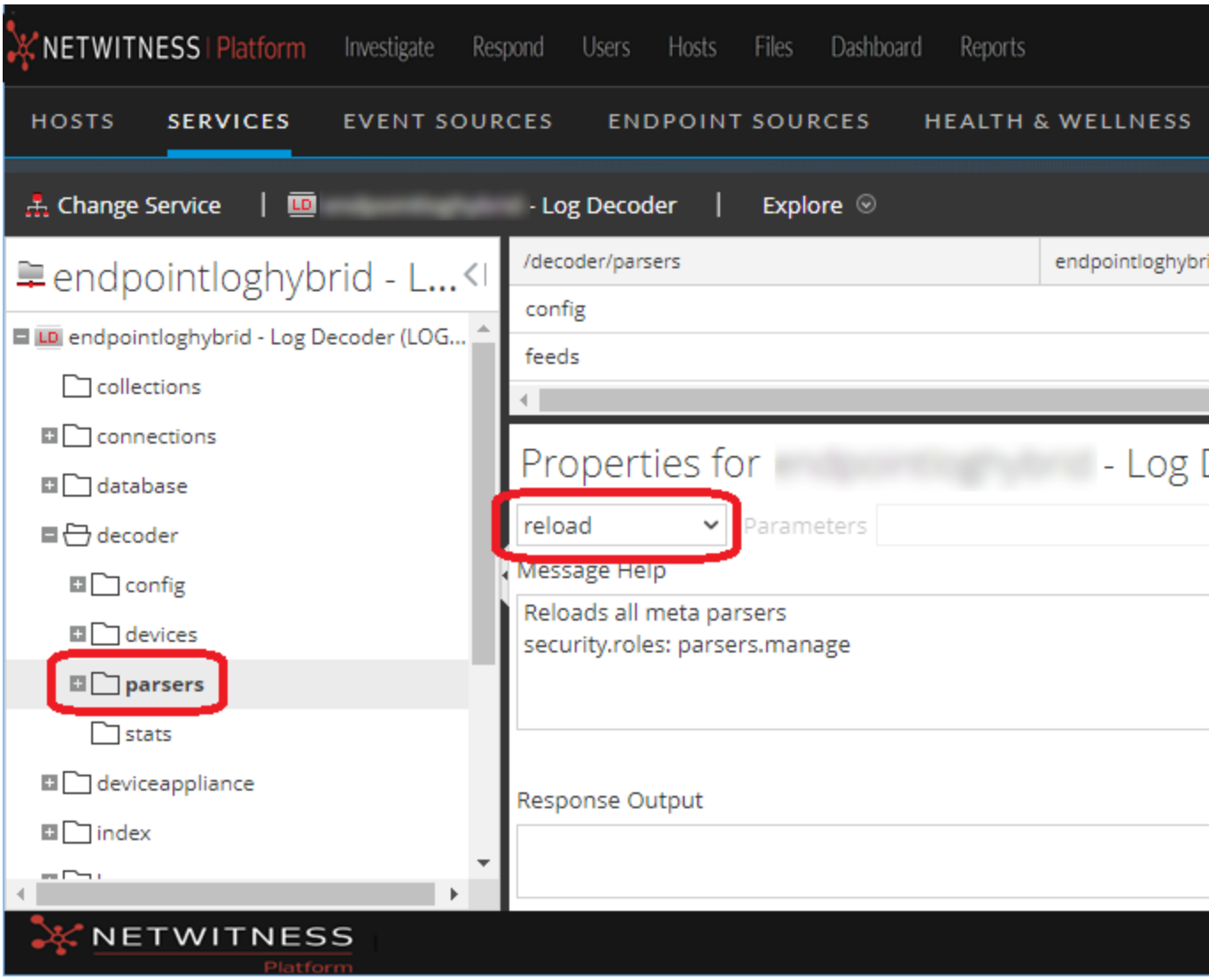
If the call is successful, you should see a REST response, "The parsers have been reloaded."

Reload Parsers from NetWitness UI

You can also reload your parsers from the UI as follows.

1. In the NetWitness UI, navigate to  (Admin) **> Services**.

   The Services view is displayed.
2. Select the Log Decoder to which your want to reload the parsers, and click **View > Explore**.
3. In the left pane, navigate to **decoder > parsers**.
4. Right-click **parsers** and select **Properties**.
5. From the drop-down menu in the Properties panel, select **reload**.


NETWITNESS Platform
Investigate
Respond
Users
Hosts
Files
Dashboard
Reports
HOSTS
SERVICES
EVENT SOURCES
ENDPOINT SOURCES
HEALTH & WELLNESS
Change Service
- Log Decoder
Explore
endpointloghybrid - L...
endpointloghybrid - Log Decoder (LOG...
collections
connections
database
decoder
config
devices
parsers
stats
deviceappliance
index
/decoder/parsers
config
feeds
Properties for
reload
Parameters
Message Help
Reloads all meta parsers
security.roles: parsers.manage
Response Output
NETWITNESS
Platform

6. Click **Send**.