Log Consolidation with ELK Stack

Release 1.2

Feb 21, 2020

1 ELK Introduction **3**
1.1 What is ELK? 3
1.2 Main Components 4

2 ELK Installation **9**
2.1 Elasticsearch 9
2.2 Kibana 11
2.3 Logstash 11

3 Logstash Pipelines **13**
3.1 ELK Data Flow 13
3.2 Logstash Pipeline 14
3.3 Logstash Configuration 15
3.4 Pipeline Configuration 16
3.5 Conclusion 25

4 ELK Stack End to End Practice **27**
4.1 Production Enviroment 27
4.2 ELK Deployment 27
4.3 Data Source Configuration 34
4.4 Conclusion 38

5 ELK Stack + Kafka End to End Practice **39**
5.1 Architecture 39
5.2 Demonstration Enviroment 40
5.3 Deployment 41
5.4 Data Source Configuration 50
5.5 Conclusion 50

6 Check Logs with Kibana **51**
6.1 Index Patterns 51
6.2 KQL Basics 53
6.3 Explore Real Data 54
6.4 Save Search/Query 57
6.5 Conclusion 58

7 Grok Debugger **59**

7.1 Logstash pipeline behavior for failed filters 59
7.2 Kibana Grok Debugger 60
7.3 Conclusion 64

8 Tips **65**
8.1 How to add tags based on filed content with pipelines? 65
8.2 Integrate Kafka 65
8.3 Add Tags to Different Kafka Topics 67
8.4 Rename the Host Field while Sending Filebeat Events to Logstash 67

This document provides a simple tutorial on Elastic stack usage, including some tips. All knowledge is based on the author's own experience, and should work as well on anyone's setup. However, because of OS differences and ELK stack software version updates, some inforamtion maybe not suitable for your setup. Anyway, the knowledge is common:)

This document is mainly for training and learning, please do not take it as a best practice. There is no responsibility from the author if you meet serious problems following the document.

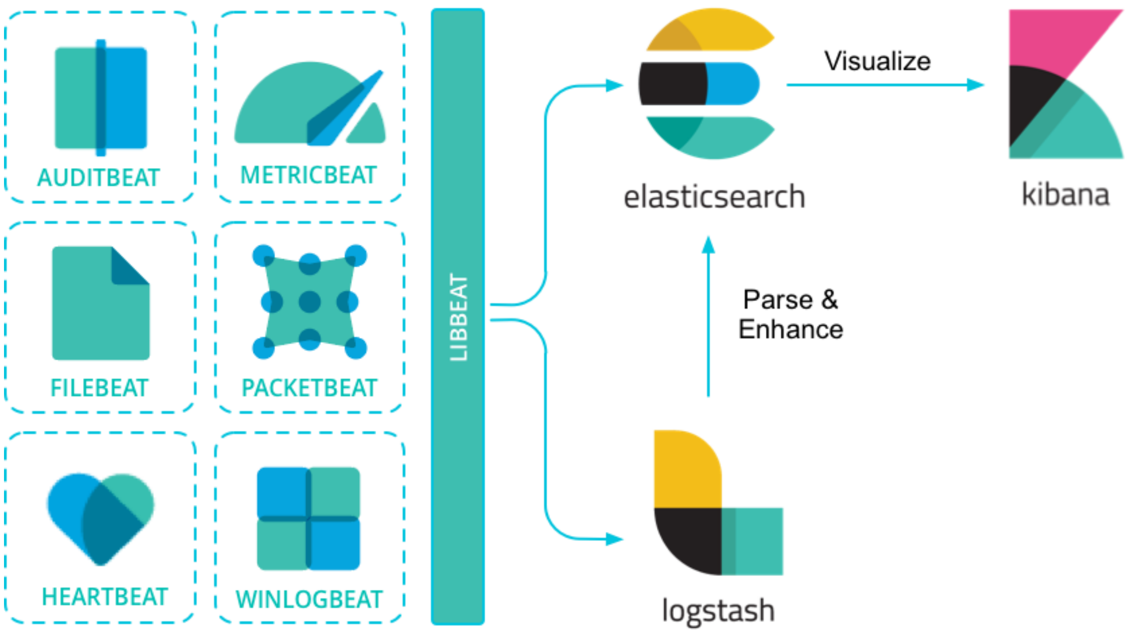
In the meanwhle, for anything unclear or needing enhancement, please help submit a issue/PR here on github

CHAPTER 1

ELK Introduction

1.1 What is ELK?

- ELK stands for the monitoring solution which is mainly consist of **Elasticsearch**, **Logstash** and **Kibana**;
- It has been renamed as **Elastic Stack**, since it has expanded its functions greatly through the use of **beats** and some other addons like APM servers. But people still tend to call it ELK;
- It is a distributed monitoring solution suiteable for almost any **structured** and **unstructured** data source, but not limited to log;
- It supports centralized **logging/metric/APM** monitoring;
- It is open source and can be extended easily.



1.2 Main Components

1.2.1 Elasticsearch

Elasticsearch is the **distributed search and analytics engine** at the heart of the Elastic Stack. It provides real-time search and analytics for all types of data (structuredi and unstructured):

- Store
- Index
- Search
- Analyze

Data In

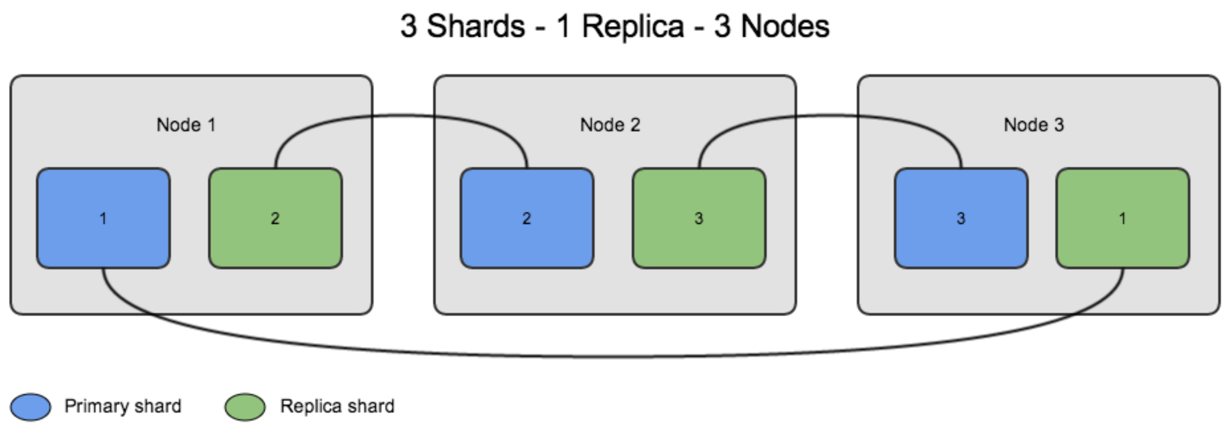
- Elasticsearch stores complex data structures that have been serialized as **JSON documents**;
- Documents are **distributed across the cluster** and can be accessed immediately from any node;
- Documents are indexed in **near real time** (without 1 second), and full-text searches are supported through **inverted index**;
- Elasticsearch indexes all data in **every field**;
- Dynamic mapping makes **schema-less** possible through detecting and adding new fields;

Information Out

- REST API
- Aggregation
- Machine learning

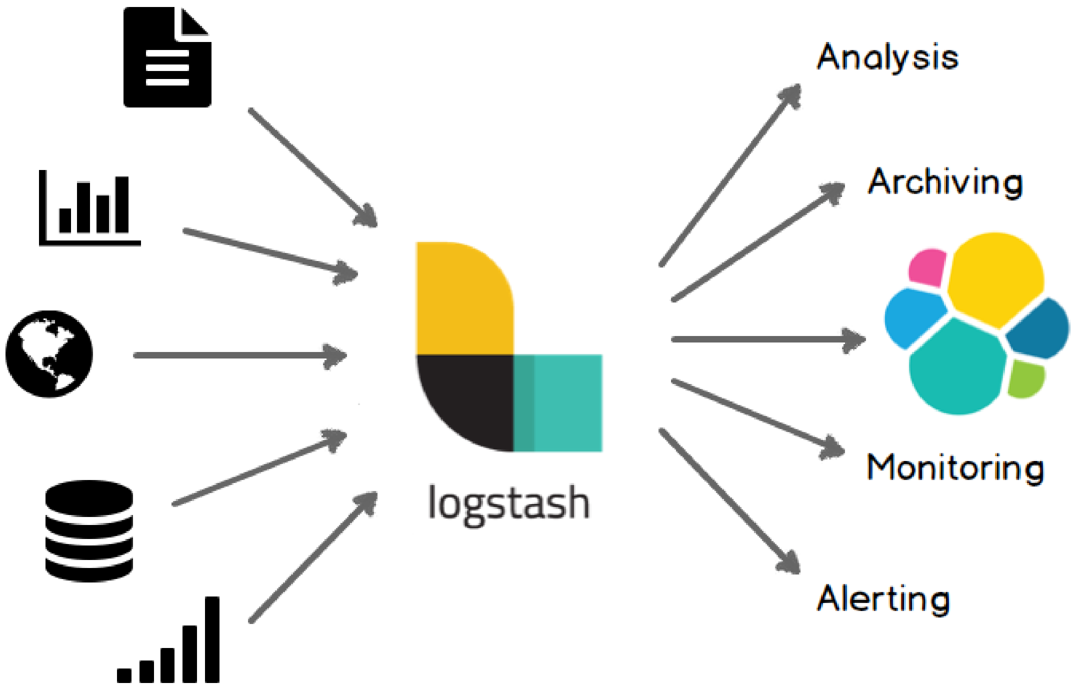
Scalability and Resilience

- Scalability: nodes can be added dynamically;
- Resilience : through the use of primary and replica shards;



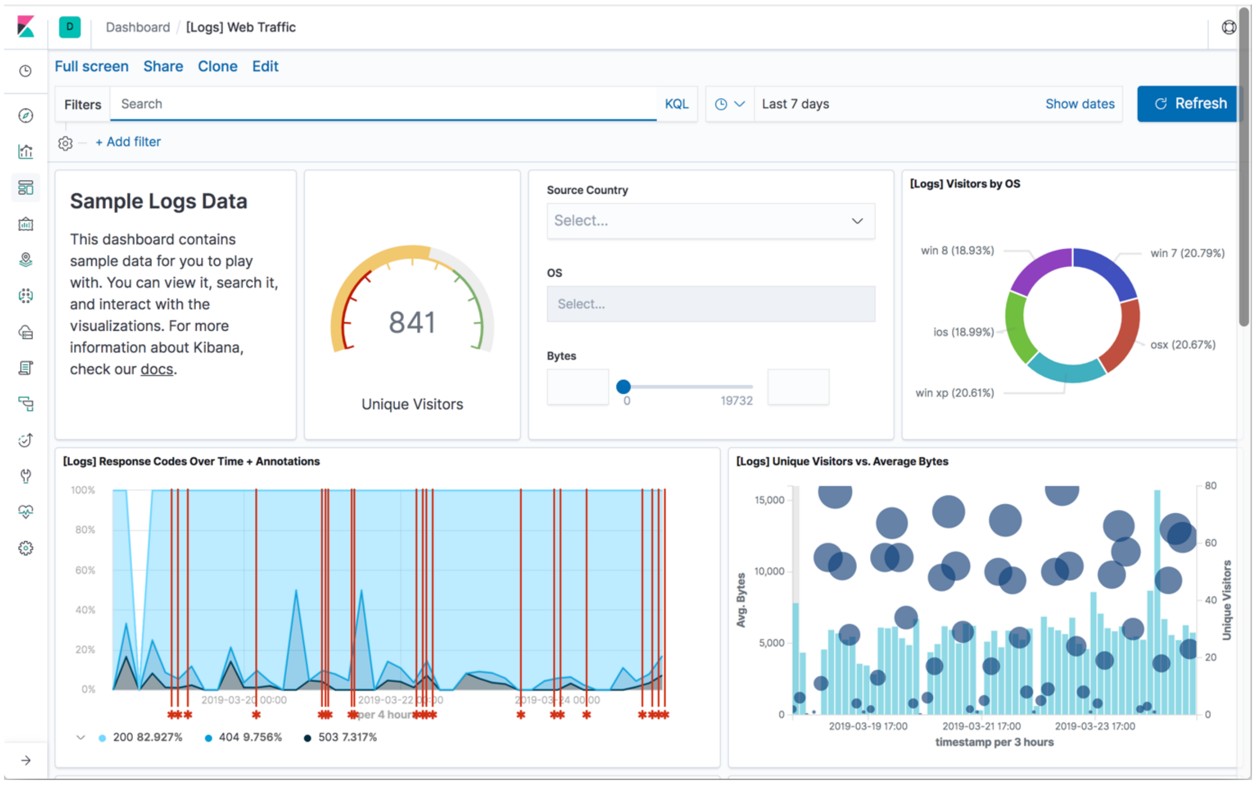
1.2.2 Logstash

- Unify data from disparate sources;
- Normalize data into destinations;



1.2.3 Kibana

Kibana is the front end GUI for Elasticsearch.



Sample Logs Data

This dashboard contains sample data for you to play with. You can view it, search it, and interact with the visualizations. For more information about Kibana, check our docs.

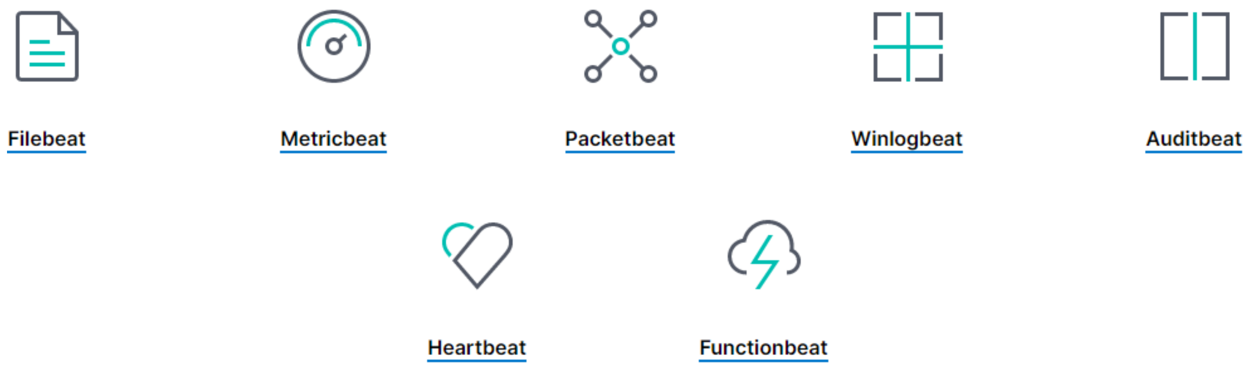
1.2.4 Beats

Beside the main three components mentioned above, there exists a kind of lightweight data collectors which are called beats. They are installed directly (for most beats and their modules) on the data sources, and collect data for specilized purposes, which are then forwarded to Elasticsearch or Logstash.

The most frequently used beats are:

- **filebeat** : sends local file records to Logstash or ELasticsearch (work as "tail -f");
- **winlogbeat** : sends windows event logs to Logstash or Elasticsearch;
- **metricbeat** : sends system or application performance metrics to Logstash or Elasticsearch.

Because of its nature, **filebeat** is extremely suitable for consolidating application logs, e.g., MongoDB, apache, etc.



CHAPTER 2

ELK Installation

As we know, ELK is mainly consit of Elasticsearch, Logstash and Kibana, hence the instalaltion consists of 3 corresponding sections. ELK stack can be installed on bare-metal/VM, and can also be run by using docker and kunernetes, it even can be serviced through public clouds like AWS and GCP.

For non-public cloud based IT solutions, it is recommended to maintain a **local deployment**. At the same time, installing Elasticsearch on **bare-metal/VM** are recommdned since Elasticsearch needs to store and index data frequently although docker/kubernetes also support data persistence (unless there is already a working kubernetes setup including well defined CSI support, it is not cost effective to maintain a kubernetes cluster just because of ELK setup).

2.1 Elasticsearch

2.1.1 Installation

Elasticsearch can be installed through using tarball on Linux, but the prefered way is to use a package manager, such as **rpm/yum/dnf on RHEL/CentOS**, **apt on Ubuntu**.

The detailed installation step won't be covered in this document since it is well documented by its official guide.

Notes: An Elasticsearch cluster, which consists of several nodes, should be used to provide scalability and resilience for most use cases, therefore the pacakge should be installed by following the same steps on all involved cluster nodes.

2.1.2 Configuration

After installing Elasticsearch on all cluster nodes, we need to configure them to form a working cluster. Fortunatelly, Elasticsearch ships with good defaults and requires very little configuration.

- **Config file location** : **/etc/elasticsearch/elasticsearch.yml** is the primary config file for Elasticsearch if it is installed through a pacakge manager, such as rpm;
- **Config file format** : the config file is in YAML format, with a simple Syntax;

The **detailed configuration** is straightforward, let's explain them one by one:

1. **cluster.name** : the name of the Elasticsearch cluster. All nodes should be configured with an identical value;
2. **node.name** : the name for node. It is recommended to use a resolvable (through /etc/hosts or DNS) hostname;
3. **path.data** : where the data will be stored. If a different path than the default is specified, remember to change the permission accordingly;
4. **path.logs** : where the Elasticsearch log will be stored. If a different path than the default is specified, remember to change the permission accordingly;
5. **network.host** : the IP/FQDN each node will listen on. It is recommended to use **0.0.0.0** to bind all available IP address;
6. **discovery.seed_hosts** : the list of nodes which will form the cluster;
7. **cluster.initial_master_nodes** : the list of nodes which may act as the master of the cluster;

Here is a sample configuration file:

```
cluster.name: elab-elasticsearch
node.name: e2e-l4-0680-240

path.data: /home/elasticsearch/data
path.logs: /home/elasticsearch/log

network.host: 0.0.0.0

discovery.seed_hosts: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-0680-242"]
cluster.initial_master_nodes: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-0680-242
↪"]
```

2.1.3 Startup

After configuration, the cluster can be booted. Before that, please make sure the port **9200**, which is the default port Elasticsearch listens at, has been opened on firewall. Of course, one can specify a different port and configure the firewall accordingly.

The cluster can be booted easily by starting the service on each node with systemctl if Elasticsearch is installed through a package manger. Below are the sample commands:

```
# Run below commands on all nodes
# Disable the firewall directly - not recommended for production setup
systemctl stop firewalld
systemctl disable firewalld
# Start elasticsearch
systemctl enable elasticsearch
systemctl start elasticsearch
systemctl status elasticsearch
```

If everything is fine, your cluster should be up and running within a minute. It is easy to verify if the cluster is working as expected by checking its status:

```
curl -XGET 'http://<any node IP/FQDN>:9200/_cluster/state?pretty'
```

2.2 Kibana

Kibana is the front end GUI for Elasticsearch showcase. Since it does not store or index data, it is suitable to run as a docker container or a kubernetes service. However, since we have already privisioned bare-metal/VM for the Elasticsearch cluster setup, installing kibana on any/all nodes of the cluster is also a good choice.

The detailed installation step is straightforward, please refer to the official installation document The **configuration for kibana** is also quite easy:

- **server.host** : specify the IP address Kibana will bind to. **0.0.0.0** is recommended;
- **server.name** : specify a meaningful name for the Kibana instance. A resolvable hostname is recommended;
- **elasticsearch.hosts** : specify a list of elasticsearch cluster Kibana will connect to.

Below is a sample config file:

```
server.host: "0.0.0.0"
server.name: "e2e-l4-0680-242"

elasticsearch.hosts: ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
```

After configuring Kibana, it can be started with systemctl:

```
systemctl disable firewalld
systemctl stop firewalld
systemctl enable kibana
systemctl start kibana
```

If everyghing goes fine, Kibana can be accessed through **http://<IP or FQDN>:5601/**

2.3 Logstash

The instalaltion of Logstash is also pretty easy and straightforward. We won't waste any words here for it, please refer to the official installation guide.

Please **keep in mind** : although Logstash can be installed together on the same server(s) as elasticsearch and Kibana, it is not wise to do so. It is highly recommended to install Logstash near to the soures where logs/metrics are generated.

In the meanwhile, since Logstash is the central place to foward logs/metrics to Elasticsearch cluster, its capability and resilience is important for a smoothly working setup. Generally speacking, this can be achived by particioning and load balancing (we won't provide the guide within this document):

- **Particioning** : leverage different Logstash deployment for differnet solutions/applications. Let's say there are web servers and databases within a production environment, then deploying different Logstash instances for them is a good choice - the capactiy of Logstash is extended, and each solution won't impact each other if its assocaiated Logstash fails;
- **Load balancing** : for each solution/application, it is recommended to deploy several Logstash instances and expose them with a load balancer (such as **HAProxy**) for high availability.

Regarding the configuration of Logstash, we will cover it in the introdcution section of Logstash pipelines.

CHAPTER 3

Logstash Pipelines

After bringing up the ELK stack, the next step is feeding data (logs/metrics) into the setup.

Based on our previous introduction, it is known that Logstash act as the **bridge/forwarder** to consolidate data from sources and forward it to the Elasticsearch cluster. But how?

3.1 ELK Data Flow

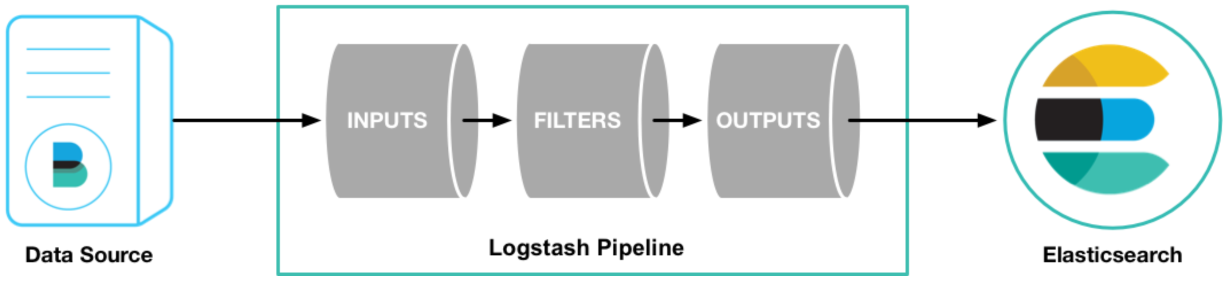
The data flow in ELK is as below:

1. Something happens on the monitored targets/sources:
    - A path down happens on a Linux OS
    - A port flap happens on a switch
    - A LUN is deleted on a storage array
    - A new event is triggered on an application
    - Etc.
2. The event/metric/activity gets recorded by:
    - syslog
    - filebeat
    - metricbeat
    - Etc.
3. Based on the configuration of syslog/filebeat/metricbeat/etc., event(s) are forwarded to Logstash (or to Elasticsearch directly, but we prefer using Logstash in the middle);
4. Logstash:
    a. Get data through its licensing port(s);

   b. Filter/Consolidate/Modify/Enhance data;
   c. Forward data to the Elasticsearch cluster or other supported destinations;
5. Elasticsearch store and index data;
6. Kibana visualize data.

3.2 Logstash Pipeline

Based on the "ELK Data Flow", we can see Logstash sits at the middle of the data process and is responsible for data gathering (input), filtering/aggregating/etc. (filter), and forwarding (output). The process of event processing (**input -> filter -> output**) works as a pipe, hence is called pipeline.



Pipeline is the core of Logstash and is the most important concept we need to understand during the use of ELK stack. Each component of a pipeline (input/filter/output) actually is implemented by using plugins. The most **frequently used plugins** are as below:

- **Input** :
  - file : reads from a file directly, working like "tail -f" on Unix like OS;
  - syslog : listens on defined ports (514 by default) for syslog message and parses based on syslog RFC3164 definition;
  - beats : processes events sent by beats, including filebeat, metricbeat, etc.
- **Filter** :
  - grok : parses and structures arbitrary text;
  - mutate : modifies event fields, such as rename/remove/replace/modify;
  - drop : discards a event;
- **Output** :
  - elasticsearch : sends event data to Elasticsearch cluster;
  - file : writes event data to a file;
  - graphite : sends event data to graphite for graphing and metrics.

Notes :

- Multiple pipelines can be defined;
- Multiple input sources, filters, and output targets can be defined within the same pipeline;

For more information, please refer to Logstash Processing Pipeline.

3.3 Logstash Configuration

We only ontroduced the instalaltion of Logstash in previous chapters without saying any word on its configuration, since it is the most complicated topic in ELK stack. Loosely speaking, Logstash provides two types of configuration:

- **settings** : control the behavior of how Logstash executes;
- **pipelines** : define the flows how data get processed.

If Logstash is installed with a pacakge manager, such as rpm, its configuration files will be as below:

- **/etc/logstash/logstash.yml** : the default setting file;
- **/etc/logstash/pipelines.yml** : the default pipeline config file.

3.3.1 logstash.yml

There are few options need to be set (other options can use the default values):

- **node.name** : specify a node name for the Logstash instance;
- **config.reload.automatic** : whether Logstash detects config changes and reload them automatically.

It is recommended to set config.reload.automatic as **true** since this will make it handy during pipeline tunings.

3.3.2 pipelines.yml

The default pipeline config file. It consists of a list of pipeline reference, each with:

- **pipeline.id** : a meaningful pipeline name specified by the end users;
- **path.config** : the detailed pipeline configuration file, refer to *Pipeline Configuration*.

Below is a simple example, which defines 4 x pipelines:

```
- pipeline.id: syslog.unity
  path.config: "/etc/logstash/conf.d/syslog_unity.conf"
- pipeline.id: syslog.xio
  path.config: "/etc/logstash/conf.d/syslog_xio.conf"
- pipeline.id: syslog.vsphere
  path.config: "/etc/logstash/conf.d/syslog_vsphere.conf"
- pipeline.id: beats
  path.config: "/etc/logstash/conf.d/beats.conf"
```

This config file only specifies pipelines to use, but not define/configure pipelines. We will cover the details with *Pipeline Configuration*.

3.3.3 Service Startup Script

After Logstash installation (say it is installed through a package manager), a service startup script won't be created by default. In other words, it is not possible to control Logstash as a service with systemctl. The reason behind is that Logstash gives end users the ability to further tune how Logstash will act before making it as a serive.

The options can be tuned are defined in **/etc/logstash/startup.options**. Most of times, there is no need to tune it, hence we can install the service startup script directly as below:

```
/usr/share/logstash/bin/system-install
```

After running the script, a service startup script will be installed as **/etc/systemd/system/logstash.service**. Now, one can control Logstash service with systemctl as other services.

3.4 Pipeline Configuration

It is time to introduce how to configure a pipeline, which is the core of Logstash usage. It is really abstractive to understand pipelines without an example, so our introduction will use examples from now on.

3.4.1 Pipeline Skeleton

Pipeline shares the same configuration skeleton (3 x sections: input, filter and output) as below:

```
# This is a comment. You should use comments to describe
# parts of your configuration.
input {
  ...
}

filter {
  ...
}

output {
  ...
}
```

The details of each section are defined through the usage of different plugins. Here are some examples:

- Define a file as the input source:

```
input {
  file {
    path => "/var/log/apache/access.log"
  }
}
```

- Multiple input soures can be specified:

```
input {
  file {
    path => "/var/log/messages"
  }

  file {
    path => "/var/log/apache/access.log"
  }
}
```

- Additional fields can be added as part of the data comming from the sources (these fields can be used for search once forwarded to destinations):

```
input {
  file {
    path => "/var/log/messages"
```

(continues on next page)

(continued from previous page)

```
    type => "syslog"
    tags => ["file", "local_syslog"]
  }

  file {
    path => "/var/log/apache/access.log"
    type => "apache"
    tags => ["file", "local_apache"]
  }
}
```

- Different kinds of plugins can be used for each section:

```
input {
  file {
    path => "/var/log/messages"
    type => "syslog"
    tags => ["file", "local_syslog"]
  }

  file {
    path => "/var/log/apache/access.log"
    type => "apache"
    tags => ["file", "local_apache"]
  }

  beats {
    type => "beats"
    port => 5044
    type => "beats"
    tags => ["beats", "filebeat"]
  }

  tcp {
    port => 5000
    type => "syslog"
    tags => ["syslog", "tcp"]
  }

  udp {
    port => 5000
    type => "syslog"
    tags => ["syslog", "udp"]
  }
}
```

- An empty filter can be defined, which means no data modification will be made:

```
filter {}
```

- Grok is the most powerful filter plugin, especially for logs:

```
# Assume the log format of http.log is as below:
# 55.3.244.1 GET /index.html 15824 0.043
#
# The grok filter will match the log record with a pattern as below:
# %{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %
↪{NUMBER:duration}
```

(continues on next page)

(continued from previous page)

```
#
# After processing, the log will be parsed into a well formated JSON document␣
↪with below fields:
# client  : the client IP
# method  : the request method
# request : the request URL
# bytes   : the size of request
# duration: the time cost for the request
# message : the original raw message
input {
  file {
    path => "/var/log/http.log"
  }
}
filter {
  grok {
    match => { "message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %
↪{NUMBER:bytes} %{NUMBER:duration}" }
  }
}
```

- Multiple plugins can be used within the filter section, and they will process data with the order as they are defined:

```
filter {
    grok {
        match => { "message" => "%{COMBINEDAPACHELOG}"}
    }

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

- Conditions are supported while define filters:

```
filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{DATA:syslog_
↪hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %
↪{GREEDYDATA:syslog_message}" }
    }
    date {
      match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
    }
  }
}
```

- Multiple output destinations can be defined too:

```
output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

By reading above examples, you should be ready to configure your own pipelines. We will introduce the filter plugin grok in more details since we need to use it frequently.o

3.4.2 Output Index for Elasticsearch

If the output plugin is "elasticsearch", the target Elastcisearch index should be specified. To smooth user expereince, Logstash provides default values. For example, **logstash-%{+YYYY.MM.dd}** will be used as the default target Elasticsearch index. However, we may need to change the default values sometimes, and the default won't work if the input is filebeat (due to mapping).

Below is several examples how we change the index:

- Customize indices based on input source difference:

```
output {
  if "vsphere" in [tags] {
    elasticsearch {
      hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
      index => "logstash-vsphere-%{+YYYY.MM.dd}"
    }
  }

  if "san" in [tags] {
    elasticsearch {
      hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
      index => "logstash-san-%{+YYYY.MM.dd}"
    }
  }
}
```

- Specify index name for beats:

```
output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
  }
}
```

3.4.3 The Grok Filter Plugin

Predefined Patterns

Grok defines quite a few patterns for usage directly. They are actually just regular expressions. The definitions of them can be checked here.

Grok Fundamental

The most basic and most important concept in Grok is its syntax:

```
%{SYNTAX:SEMANTIC}
```

- **SYNTAX** : the name of the pattern that will match your text;

- **SEMANTIC** : the identifier you give to the piece of text being matched.

Let's explain it with an example:

- Assume we have a log record as below:

```
Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php /usr/share/cacti/site/poller.
↪php >/dev/null 2>/var/log/cacti/poller-error.log)
```

- By deault, the whole string will be forwarded to destinations (such as Elasticsearch) without any change. In other words, it will be seen by the end user as a JSON document with only one filed "message" which holds the raw string. This is not easy for end users to do search and classify.
- To make the unstructured log record as a meaningful JSON document, below grok pattern can be leveraged to parse it:

```
%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_
↪program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}
```

- SYSLOGTIMESTAMP, SYSLOGHOST, DATA, POSINT and GREEDYDATA are all predefined patterns
- syslog_timestamp, syslog_hostname, syslog_program, syslog_pid and syslog_message are fields names added based on the pattern matching
- After parsing, the log record becomes a JSON document as below:

```
{
                 "message" => "Dec 23 14:30:01 louis CRON[619]: (www-data) CMD␣
↪(php /usr/share/cacti/site/poller.php >/dev/null 2>/var/log/cacti/poller-error.
↪log)",
              "@timestamp" => "2013-12-23T22:30:01.000Z",
                "@version" => "1",
                    "type" => "syslog",
                    "host" => "0:0:0:0:0:0:0:1:52617",
        "syslog_timestamp" => "Dec 23 14:30:01",
         "syslog_hostname" => "louis",
          "syslog_program" => "CRON",
              "syslog_pid" => "619",
          "syslog_message" => "(www-data) CMD (php /usr/share/cacti/site/poller.
↪php >/dev/null 2>/var/log/cacti/poller-error.log)",
             "received_at" => "2013-12-23 22:49:22 UTC",
           "received_from" => "0:0:0:0:0:0:0:1:52617",
    "syslog_severity_code" => 5,
    "syslog_facility_code" => 1,
         "syslog_facility" => "user-level",
         "syslog_severity" => "notice"
}
```

- The full pipeline configuration for this example is as below:

```
input {
  tcp {
    port => 5000
    type => syslog
  }
  udp {
    port => 5000
    type => syslog
  }
```

(continues on next page)

(continued from previous page)

```
}

filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %
↪{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?
↪: %{GREEDYDATA:syslog_message}" }
      add_field => [ "received_at", "%{@timestamp}" ]
      add_field => [ "received_from", "%{host}" ]
    }
    date {
      match => [ "syslog_timestamp", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
  }
}

output {
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

The example is from the official document, please go through it for more details.

3.4.4 Single Pipeline vs. Multiple Pipelines

Based on the previous introduction, we know multiple plugins can be used for each pipeline section (input/filter/output). In other words, there are always two methods to achieve the same data processing goal:

1. Define a single pipeline containing all configurations:
    - Define multiple input sources
    - Define multiple filters for all input sources and make decision based on conditions
    - Define multiple output destinations and make decision based on conditions
2. Define multiple pipelines with each:
    - Define a single input source
    - Define filters
    - Define a single output destination

Here is the example for these different implementations:

1. Define a single pipeline:

```
input {
  beats {
    port => 5044
    type => "beats"
  }
  tcp {
    port => 5000
    type => "syslog"
  }
  udp {
```

(continues on next page)

(continued from previous page)

```
    port => 5000
    type => "syslog"
  }
  stdin {
    type => "stdin"
  }
}

filter {
  if [type] == "syslog" {
    grok {
      match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{DATA:syslog_
↪hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %
↪{GREEDYDATA:syslog_message}" }
    }
    date {
       match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
    }
  } else if [type] == "beats" {
    json {
      add_tag => ["beats"]
    }
  } else {
    prune {
      add_tag => ["stdin"]
    }
  }
}

output {
  if [type] == "syslog" or [type] == "beats" {
    elasticsearch {
      hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    }
  } else {
      stdout { codec => json }
  }
}
```

2. Here is the example implementing the same goal with multiple pipelines:

   a. Define a pipeline configuration for beats:

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

filter {
  json {
    add_tag => ["beats"]
  }
}
```

(continues on next page)

(continued from previous page)

```
output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
  }
}
```

b. Define a pipeline configuration for syslog:

```
input {
  tcp {
    port => 5000
    type => "syslog"
  }
  udp {
    port => 5000
    type => "syslog"
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %
↪{DATA:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?:
↪%{GREEDYDATA:syslog_message}" }
  }
  date {
     match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
  }
}
```

c. Define a pipeline configuration for stdin:

```
input {
  stdin {
    type => "stdin"
  }
}

filter {
  prune {
    add_tag => ["stdin"]
  }
}

output {
  stdout { codec => json }
}
```

d. Enable all pipelines in pipelines.yml

```
- pipeline.id: beats
  path.config: "/etc/logstash/conf.d/beats.conf"
- pipeline.id: syslog
  path.config: "/etc/logstash/conf.d/syslog.conf"
- pipeline.id: stdin
  path.config: "/etc/logstash/conf.d/stdin.conf"
```

The same goal can be achived with both methods, but which method should be used? The answer is **multiple pipelines should always be used whenever possible**:

- Maintaining everything in a single pipeline leads to conditional hell - lots of conditions need to be declared which cause complication and potential errors;
- When multiple output destinations are defined in the same pipeline, congestion may be triggered.

3.4.5 Configuration Pitfall

Based on previous introduction, it is known the file **pipelines.yml** is where pipelines are controlled(enable/disable). However, there exists a pitfall. Logstash supoorts defining and enabling multiple pipelines as below:

```
- pipeline.id: syslog.unity
  path.config: "/etc/logstash/conf.d/syslog_unity.conf"
- pipeline.id: syslog.xio
  path.config: "/etc/logstash/conf.d/syslog_xio.conf"
...
```

However, with the default main pipeline as below, all configurations also seems to work:

```
- pipeline.id: main
  path.config: "/etc/logstash/conf.d/*.conf"
```

This is the pitfall:

- By using a single main pipeline to enable all pipeline configurations(*.conf), acutally only one pipeline is working. All configurations are merged together. In other words, it is the same as you define a single pipeline configuration file containing all logics - all power of multiple pipelines are silenced;
- Some input/output plugin may not work with such configuration, e.g. Kafka. When Kafka is used in the middle of event sources and logstash, Kafka input/output plugin needs to be seperated into different pipelines, otherwise, events will be merged into one Kafka topic or Elasticsearch index.

3.4.6 Reference

- Variables and Condtions
- Input Plugins
- Filter Plugins
- Output Plugins
- Time Format

3.5 Conclusion

After reading this chapter carefully, one is expected to get enough skills to implement pipelines for production setup. We will provide a full example for a production setup end to end in next chapter.

CHAPTER 4

ELK Stack End to End Practice

This chapter will demonstrate an end to end ELK Stack configuration demo for an imaginary production environment.

4.1 Production Enviroment

4.1.1 Environment

The production environment consists of 4 x ESXi servers, 1 x Unity and 1 x XtremIO:

- **4 x vSphere ESXi servers** : 10.226.68.231-234 (hostnames: e2e-l4-0680-231/232/233/234)
- **1 x Cisco MDS FC switch** : 10.228.225.202 (hostname: e2e-l4-sw7-202)
- **1 x Brocade FC switch** : 10.228.225.203 (hostname: e2e-l4-sw8-203)
- **1 x Dell EMC Unity storage array** : 10.226.49.236 (hostname : uni0839)
- **1 x Dell EMC XtremIO storage array** : 10.226.49.222 (hostname : e2es-xio-02)

4.1.2 Monitoring Goals

- Consolidate all logs from servers, switches and storage arrays;
- Consolidate logs of ELK stack itself.

4.2 ELK Deployment

4.2.1 Environment

The ELK stack will be deployed on VMs:

- **3 x VMs for Elasticsearch cluster** : 10.226.68.240-242 (hostnames: e2e-l4-0680-240/241/242)

- **1 x VM for Logstash installation** : 10.226.68.186 (hostname : e2e-l4-0680-186)

4.2.2 Elasticsearch Deployment

The installation process has already been documented by this document, please refer to previous chapters. We will only list configurations and commands in this section.

1. Install Elasticsearch on all nodes;
2. Configs on each node (/etc/elasticsearch/elasticsearch.yml):
   - e2e-l4-0680-240

```
cluster.name: elab-elasticsearch
node.name: e2e-l4-0680-240
path.data: /home/elasticsearch/data
path.logs: /home/elasticsearch/log
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-0680-242
↪"]
cluster.initial_master_nodes: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-
↪0680-242"]
```

   - e2e-l4-0680-241

```
cluster.name: elab-elasticsearch
node.name: e2e-l4-0680-241
path.data: /home/elasticsearch/data
path.logs: /home/elasticsearch/log
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-0680-242
↪"]
cluster.initial_master_nodes: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-
↪0680-242"]
```

   - e2e-l4-0680-242

```
cluster.name: elab-elasticsearch
node.name: e2e-l4-0680-242
path.data: /home/elasticsearch/data
path.logs: /home/elasticsearch/log
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-0680-242
↪"]
cluster.initial_master_nodes: ["e2e-l4-0680-240", "e2e-l4-0680-241", "e2e-l4-
↪0680-242"]
```

3. Start Elasticsearch service on each node:

```
systemctl disable firewalld
systemctl enable elasticsearch
systemctl start elasticsearch
```

4. Verify (on any node): 3 x alive nodes should exist and one master node is elected successfully

```
[root@e2e-l4-0680-240]# curl -XGET 'http://localhost:9200/_cluster/state?pretty'
  "cluster_name" : "elab-elasticsearch",
```

(continues on next page)

(continued from previous page)

```
  "cluster_uuid" : "oDELRsi4QLi8NMH09UfolA",
  "version" : 301,
  "state_uuid" : "0oP2HuyWQyGUOxhr7iPr8A",
  "master_node" : "2sobqFxLRaCft3m3lasfpg",
  "blocks" : { },
  "nodes" : {
    "2sobqFxLRaCft3m3lasfpg" : {
      "name" : "e2e-l4-0680-241",
      "ephemeral_id" : "b39_hgjWTIWfEwY_D3tAVg",
      "transport_address" : "10.226.68.241:9300",
      "attributes" : {
        "ml.machine_memory" : "8192405504",
        "ml.max_open_jobs" : "20",
        "xpack.installed" : "true"
      }
    },
    "9S6jr4zCQBCfEiL8VoM4DA" : {
      "name" : "e2e-l4-0680-242",
      "ephemeral_id" : "vx_SsACWTfml6BJCQJuW4A",
      "transport_address" : "10.226.68.242:9300",
      "attributes" : {
        "ml.machine_memory" : "8192405504",
        "ml.max_open_jobs" : "20",
        "xpack.installed" : "true"
      }
    },
    "NR7jA8AeT3CuDvvNW3s3Eg" : {
      "name" : "e2e-l4-0680-240",
      "ephemeral_id" : "ZOrq8zUiRxavr1F5bzDvQQ",
      "transport_address" : "10.226.68.240:9300",
      "attributes" : {
        "ml.machine_memory" : "8192405504",
        "ml.max_open_jobs" : "20",
        "xpack.installed" : "true"
      }
    }
  },
  ......
```

4.2.3 Kibana Deployment

Kibana is the front end GUI for Elasticsearch. It won't take part in data processing and it does not waste too much computing resouce, hence we can deploy it on the same node(s) as Elasticsearch clusters. Since we have 3 x nodes for Elasticsearch cluster, we can install Kibana on all of them. In other words, people can access the setup from any IP address - this will avoid single point of failure and leave us the potential to configure a front end load balancer for Kibana (e.g. with HAProxy).

The installation process has already been documented by this document, please refer to previous chapters. We will only list configurations and commands in this section.

1. Install Kibana on all Elasticsearch nodes;
2. Configure Kibana on each node (/etc/kibana/kibana.yml):
   - e2e-l4-0680-240

```
server.host: "0.0.0.0"
server.name: "e2e-l4-0680-240"
elasticsearch.hosts: ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-
↪241:9200", "http://e2e-l4-0680-242:9200"]
```

- e2e-l4-0680-241

```
server.host: "0.0.0.0"
server.name: "e2e-l4-0680-241"
elasticsearch.hosts: ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-
↪241:9200", "http://e2e-l4-0680-242:9200"]
```

- e2e-l4-0680-242

```
server.host: "0.0.0.0"
server.name: "e2e-l4-0680-242"
elasticsearch.hosts: ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-
↪241:9200", "http://e2e-l4-0680-242:9200"]
```

3. Start the service on each node:

```
systemctl enable kibana
systemctl start kibana
```

4. Verify: access http://<10.226.68.240-242>:5601 to verify that Kibana is up and running.

4.2.4 Logstash Deployment

The installation process has already been documented by this document, please refer to previous chapters. We will only list configurations and commands in this section.

1. Install Logstash on the prepared VM;
2. Configure Logstash settings (/etc/logstash/logstash.yml):

```
node.name: e2e-l4-0680-186
config.reload.automatic: true
```

3. Initial pipeline definitions (/etc/logstash/pipelines.yml):

```
- pipeline.id: syslog.vsphere
  path.config: "/etc/logstash/conf.d/syslog_vsphere.conf"
- pipeline.id: syslog.fabric
  path.config: "/etc/logstash/conf.d/syslog_fabric.conf"
- pipeline.id: syslog.unity
  path.config: "/etc/logstash/conf.d/syslog_unity.conf"
- pipeline.id: syslog.xio
  path.config: "/etc/logstash/conf.d/syslog_xio.conf"
- pipeline.id: beats
  path.config: "/etc/logstash/conf.d/beats.conf"
```

4. Configure pipelines:

   - syslog_vsphere.conf

```
input {
  tcp {
    type => "syslog"
    port => 5002
    tags => ["syslog", "tcp", "vsphere"]
  }
  udp {
    type => "syslog"
    port => 5002
    tags => ["syslog", "udp", "vsphere"]
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %
↪{DATA:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?:
↪%{GREEDYDATA:syslog_message}" }
    add_field => [ "received_from", "%{host}" ]
  }
  date {
     match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "logstash-vsphere-%{+YYYY.MM.dd}"
    ilm_rollover_alias => "logstash-vsphere"
    ilm_policy => "cweek_policy1"
  }
}
```

- syslog_fabric.conf

```
input {
  tcp {
    type => "syslog"
    port => 514
    tags => ["syslog", "tcp", "fabric"]
  }
  udp {
    type => "syslog"
    port => 514
    tags => ["syslog", "udp", "fabric"]
  }
}

filter {
  mutate {
    add_field => [ "received_from", "%{host}" ]
  }
}

output {
```

(continues on next page)

(continued from previous page)

```
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "logstash-fabric-%{+YYYY.MM.dd}"
    ilm_rollover_alias => "logstash-fabric"
    ilm_policy => "cweek_policy1"
  }
}
```

- syslog_unity.conf

```
input {
  tcp {
    type => "syslog"
    port => 5000
    tags => ["syslog", "tcp", "unity"]
  }
  udp {
    type => "syslog"
    port => 5000
    tags => ["syslog", "udp", "unity"]
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %
↪{DATA:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?:
↪%{GREEDYDATA:syslog_message}" }
    add_field => [ "received_from", "%{host}" ]
  }
  date {
     match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "logstash-unity-%{+YYYY.MM.dd}"
    ilm_rollover_alias => "logstash-unity"
    ilm_policy => "cweek_policy1"
  }
}
```

- syslog_xio.conf

```
input {
  tcp {
    type => "syslog"
    port => 5001
    tags => ["syslog", "tcp", "xio"]
  }
  udp {
    type => "syslog"
    port => 5001
```

(continues on next page)

(continued from previous page)

```
    tags => ["syslog", "udp", "xio"]
  }
}

filter {
  grok {
    match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %
↪{DATA:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?:
↪%{GREEDYDATA:syslog_message}" }
    add_field => [ "received_from", "%{host}" ]
  }
  date {
     match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "logstash-xio-%{+YYYY.MM.dd}"
    ilm_rollover_alias => "logstash-xio"
    ilm_policy => "cweek_policy1"
  }
}
```

- beats.conf

  Notes: the output index must be set if the output destination is elasticsearch

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

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200",
↪"http://e2e-l4-0680-242:9200"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
    ilm_rollover_alias => "filebeat"
    ilm_policy => "cweek_policy1"
  }
}
```

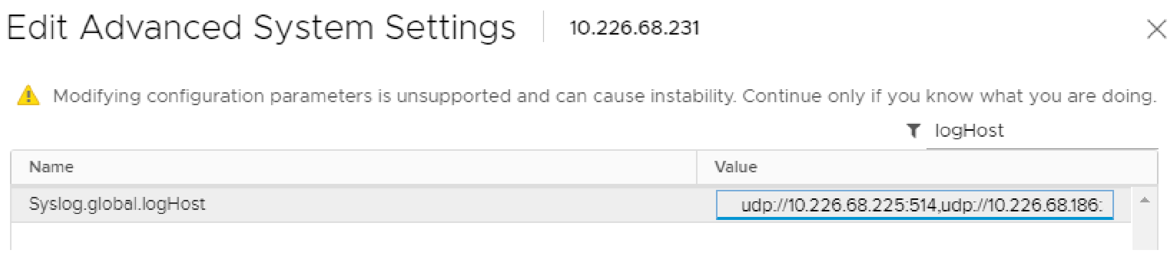
5. Start Logstash

```
/usr/share/logstash/bin/system-install
systemctl disable firewalld
systemctl enable logstash
systemctl start logstash
```

4.3 Data Source Configuration

4.3.1 vSphere Syslog Configuration

1. Select the vSphere ESXi server under vCenter;
2. Click "Configure->System->Advanced System Settings->EDIT";
3. Find the option "Syslog.global.logHost";
4. Add the Logstash syslog listening address "udp://10.226.68.186:5002":


Edit Advanced System Settings | 10.226.68.231

Modifying configuration parameters is unsupported and can cause instability. Continue only if you know what you are doing.

logHost

Name	Value
Syslog.global.logHost	udp://10.226.68.225:514,udp://10.226.68.186:

4.3.2 Switch Syslog Configuration

All network equipment, including Ethernet switches, FC switches, routers, firewalls, etc., support syslog as a kind of de facto standard. Therefor, their logs can be consolidated easily with ELK stack. However, most of network equipment uses UDP port 514 for syslog and does not provide the option to change it, hence we should create a Logstash pipeline listening at the port, just as what did above.

Note: the commands for enabling syslog on different switches may be far from each other. Please refer to their official documents for detailed commands.

Below are configurations for our switches (10.228.225.202/203):

- Cisco Switch

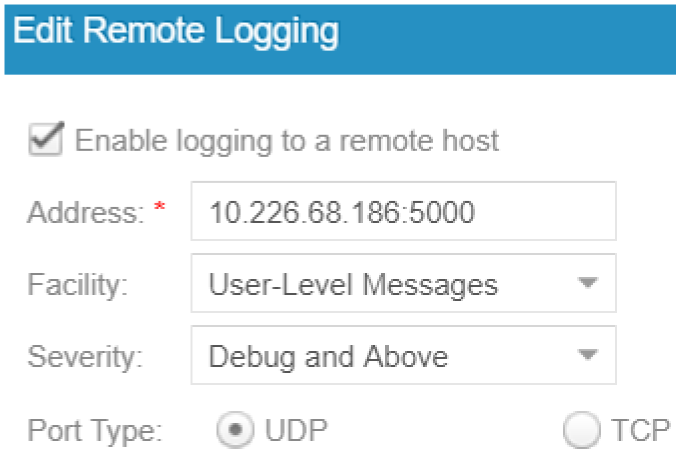
```
conf t
logging server 10.226.68.186 6 facility syslog
end
copy running startup
```

- Brocade Switch

```
syslogdipadd 10.226.68.186
syslogdipshow
```

4.3.3 Unity Storage Array Configuration

1. Login Unisphere of the storage array;
2. Click "Update system settings->Management->Remote Logging->+";
3. Add the Logstash syslog listening address "10.226.68.186:5000":

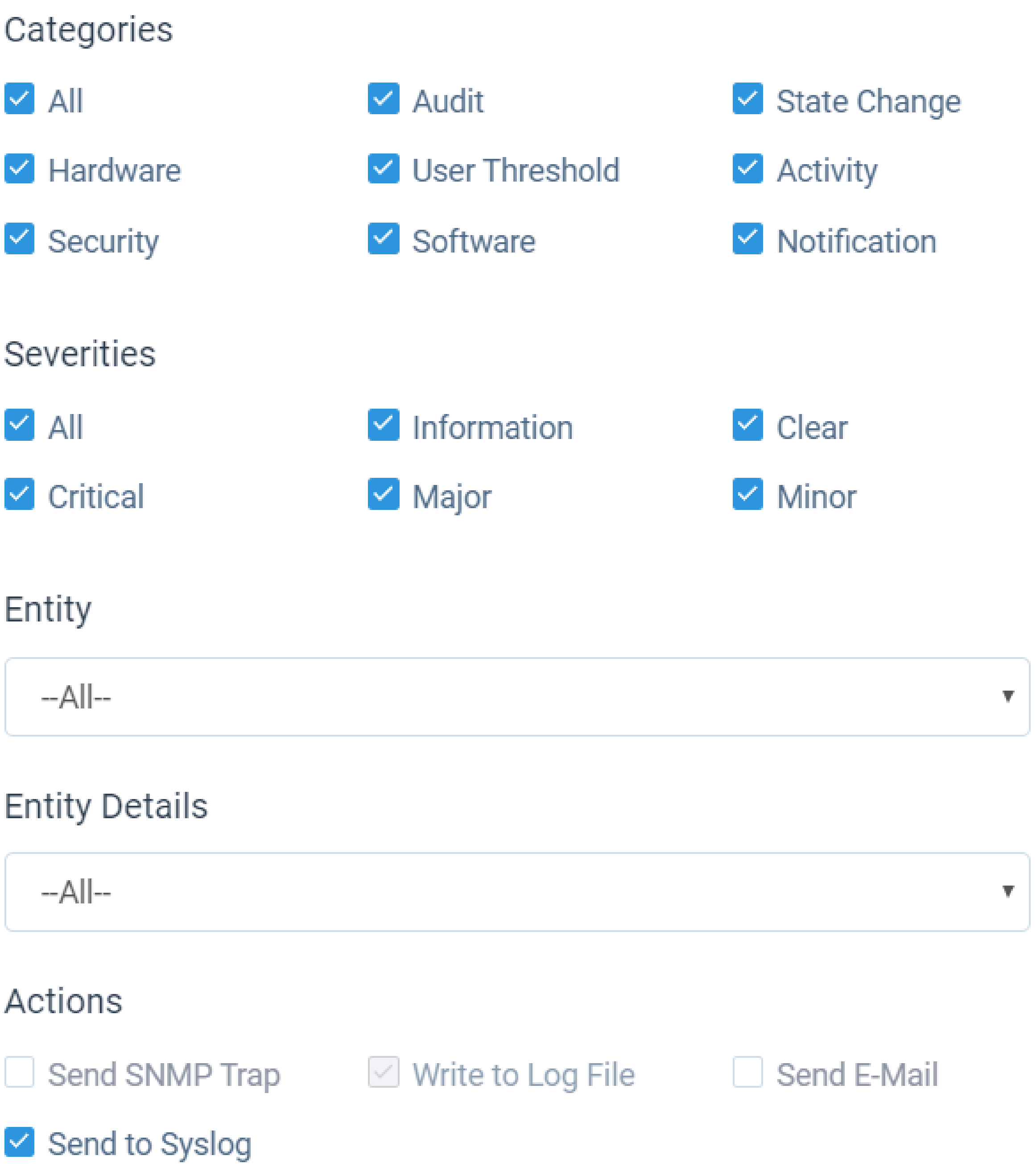

Edit Remote Logging

✓ Enable logging to a remote host

Address: *	10.226.68.186:5000
Facility:	User-Level Messages
Severity:	Debug and Above
Port Type:	● UDP ○ TCP

4.3.4 XtremIO Storage Array Configuration

1. Login Unisphere of the storage array;
2. Click “System Settings->Notifications->Event Handlers->New”;
3. Enable events should be forwarded to syslog and select “Send to Syslog”:



Categories

✓ All	✓ Audit	✓ State Change
✓ Hardware	✓ User Threshold	✓ Activity
✓ Security	✓ Software	✓ Notification

Severities

✓ All	✓ Information	✓ Clear
✓ Critical	✓ Major	✓ Minor

Entity

--All--

Entity Details

--All--

Actions

Send SNMP Trap	✓ Write to Log File	Send E-Mail
✓ Send to Syslog		

4. Click "Syslog Notifications->New" and specify the Logstash syslog listening address "10.226.68.186:5001"

4.3.5 ELK Stack Filebeat Configuraion

Since we are leveraging ELK stack mainly for logging here in the document, we will use filebeat only. Currently, filebeat supports Linux, Windows and Mac, and provide well pacakged binary (deb, rpm, etc.). The installation is pretty easy, we won't cover the details, please refer to the offical instalaltion guide.

After installation, filebeat needs to be configured. The steps can be refered here.

Our target is monitoring ELK stack itself with filebeat. Since ELK stack consists of Elasticsearch cluster, Logstash and Kibana, and Kibana is only a GUI front end (with lots of features), we will only monitor Elasticsearch cluster and

Logstash.

To make the daily configuration work more smoothly, filebeat provides a mechanism to simplify the collection, parsing, and visualization of common log formats, which is called **modules** (refer here for the introduction and supported modules).

Elasticsearch and Logstash have supported modules in filebeat, hence we will leverage them to ease the configuration:

1. Configure (/etc/filebeat/filebeat.yml) all nodes (e2e-l4-0680-240/241/242, e2e-l4-0680-186)

```
output.logstash:
  # The Logstash hosts
  hosts: ["e2e-l4-0680-186:5044"]
```

2. Enable modules:

   - Enable filebeat elasticsearch module on all Elasticsearch cluster nodes:

   ```
   filebeat modules enable elasticsearch
   filebeat modules list
   ```

   - Enable filebeat elasticsearch module on Logstash nodes:

   ```
   filebeat modules enable logstash
   filebeat modules list
   ```

3. Configure filebeat modules:

   - Elasticsearch nodes (/etc/filebeat/modules.d/elasticsearch.yml):

   ```
   - module: elasticsearch
     server:
       enabled: true
       var.paths: ["/home/elasticsearch/log/*.log"]
     gc:
       enabled: false
     audit:
       enabled: false
     slowlog:
       enabled: false
     deprecation:
       enabled: false
   ```

   - Logstash nodes (/etc/filebeat/modules.d/logstash.yml):

   ```
   - module: logstash
     log:
       enabled: true
     slowlog:
      enabled: true
   ```

5. Start filebeat

```
systemctl enable filebeat
systemctl start filebeat
```

4.4 Conclusion

We have completed all the setup work for the production environment. The next step is leveraging the powerful ELK stack checking our logs, which will be covered in a separate chapter.

CHAPTER 5

ELK Stack + Kafka End to End Practice

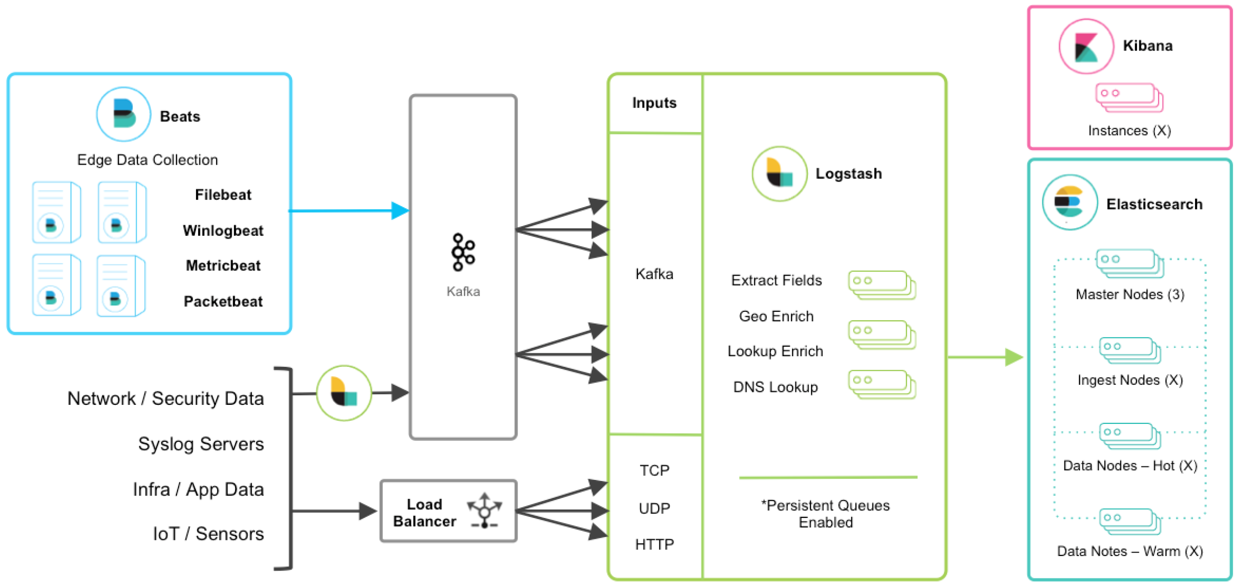
We have learned how to configure an ELK Stack from end to end with the previous chapter. Such a configuration is able to support most use cases. However, for a production environment which scales out unlimitedly, bottlenecks still exists:

- Logstash needs to process logs with pipelines and filters which cost considerable time, it may become a bottleneck if log bursts exist;
- Elasticsearch needs to index logs which cost time too, and it becomes a bottleneck when log bursts happen.

The above mentioned bottlenecks can be smoothed by adding more Logstash deployment and scaling Elasticsearch cluster of course, they can also be smoothed by introduction a cache layer in the middle like all other IT solutions (such as introducing Redis in the middle of the database access path). One of the most popular solutions leveraging a cache layer is integrating Kafka into the ELK stack. We will cover how to set up such an environemnt in this chapter.

5.1 Architecture

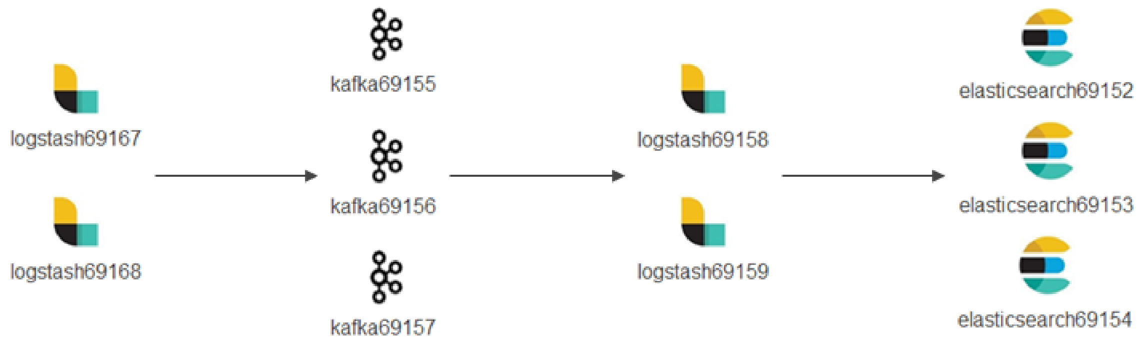
When Kafka is leveraged as a cache layer in ELK Stack, an architecture as below will be used:



The details of this can be found from Deploying and Scaling Logstash

5.2 Demonstration Enviroment

Based on the knowledge introducted above, our demonstration environment will be architectured as below:



The detailed enviroment is as below:

- logstash69167/69168 (hostnames: e2e-l4-0690-167/168): receive logs from syslog, filebeat, etc. and forward/produce logs to Kafka topics;
- kafka69155/156/157 (hostnames: e2e-l4-0690-155/156/157): kafka cluster
  - zookeeper will also be installed on these 3 x nodes;
  - kafka manager will be installed on kafka69155;
- logstash69158/69159 (hostnames: e2e-l4-0690-158/159): consume logs from kafka topics, process logs with pipelines, and send logs to Elasticsearch;
- elasticsearch69152/69153/69154 (hostnames: e2e-l4-0690-152/153/154): Elasticsearch cluster
  - Kibana will be installed on elasticsearch69152

- Data sources such as syslog, filebeat, etc. follow the same configuration as when Kafka is not used, hence we ignore their configuration in this chapter.

5.3 Deployment

5.3.1 Elasticsearch Deployment

The installation process has already been documented by this document, please refer to previous chapters. We will only list configurations and commands in this section.

1. Install Elasticsearch on elasticsearch69152/69153/69154;
2. Configs on each node (/etc/elasticsearch/elasticsearch.yml):
   - elasticsearch69152

```
cluster.name: edc-elasticsearch
node.name: e2e-l4-0690-152
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-0690-154
↪"]
cluster.initial_master_nodes: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-
↪0690-154"]
```

   - elasticsearch69153

```
cluster.name: edc-elasticsearch
node.name: e2e-l4-0690-153
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-0690-154
↪"]
cluster.initial_master_nodes: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-
↪0690-154"]
```

   - elasticsearch69154

```
cluster.name: edc-elasticsearch
node.name: e2e-l4-0690-154
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 0.0.0.0
discovery.seed_hosts: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-0690-154
↪"]
cluster.initial_master_nodes: ["e2e-l4-0690-152", "e2e-l4-0690-153", "e2e-l4-
↪0690-154"]
```

3. Start Elasticsearch service on each node:

```
systemctl disable firewalld
systemctl enable elasticsearch
systemctl start elasticsearch
```

4. Verify (on any node): 3 x alive nodes should exist and one master node is elected successfully

```
[root@e2e-l4-0690-152]# curl -XGET 'http://localhost:9200/_cluster/state?pretty'
```

5.3.2 Kibana Deployment

The installation process has already been documented by this document, please refer to previous chapters. We will only list configurations and commands in this section.

1. Install Kibana on elasticsearch69152;
2. Configure Kibana(/etc/kibana/kibana.yml):

```
server.host: "0.0.0.0"
server.name: "e2e-l4-0690-152"
elasticsearch.hosts: ["http://e2e-l4-0690-152:9200", "http://e2e-l4-0690-153:9200
↪", "http://e2e-l4-0690-154:9200"]
```

3. Start the service on each node:

```
systemctl enable kibana
systemctl start kibana
```

4. Verify: access http://10.226.69.152:5601 to verify that Kibana is up and running.

5.3.3 Zookeeper Deployment

Zookeeper is a must before running a Kafka cluster. For demonstration purpose, we deploy a Zookeeper cluster on the same nodes as the Kafka cluster, A.K.A kafka69155/69156/69157.

1. Download zookeeper;
2. There is no need to do any installation, decompressing the package is enough;
3. Configure zookeeper on each node(conf/zoo.cfg):

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/var/lib/zookeeper
clientPort=2181

server.1=10.226.69.155:2888:3888
server.2=10.226.69.156:2888:3888
server.3=10.226.69.157:2888:3888
```

4. Create file /var/lib/zookeeper/myid with content 1/2/3 on each node:

```
echo 1 > /var/lib/zookeeper/myid # kafka69155
echo 2 > /var/lib/zookeeper/myid # kafka69156
echo 3 > /var/lib/zookeeper/myid # kafka69157
```

5. Start Zookeeper on all nodes:

```
./bin/zkServer.sh start
./bin/zkServer.sh status
```

6. Connect to Zooper for verification:

```
./bin/zkCli.sh -server 10.226.69.155:2181,10.226.69.156:2181,10.226.69.157:2181
```

5.3.4 Kafka Deployment

A Kafka cluster will be deployed on kafka69155/69156/69157.

1. Kafka does not need any installation, downloading and decompressing a tarball is enough. Please refer to Kafka Quickstart for reference;
2. The Kafka cluster will run on kafka69155/156/157 where a Zookeeper cluster is already running. To enable the Kafka cluster, configure each node as below(config/server.properties):
   - kafka69155:

     ```
     broker.id=0
     listeners=PLAINTEXT://0.0.0.0:9092
     advertised.listeners=PLAINTEXT://10.226.69.155:9092
     zookeeper.connect=10.226.69.155:2181,10.226.69.156:2181:10.226.69.157:2181
     ```

   - kafka69156:

     ```
     broker.id=1
     listeners=PLAINTEXT://0.0.0.0:9092
     advertised.listeners=PLAINTEXT://10.226.69.156:9092
     zookeeper.connect=10.226.69.155:2181,10.226.69.156:2181:10.226.69.157:2181
     ```

   - kafka69157:

     ```
     broker.id=1
     listeners=PLAINTEXT://0.0.0.0:9092
     advertised.listeners=PLAINTEXT://10.226.69.157:9092
     zookeeper.connect=10.226.69.155:2181,10.226.69.156:2181:10.226.69.157:2181
     ```

3. Start Kafka on all nodes:

   ```
   ./bin/kafka-server-start.sh -daemon config/server.properties
   ```

Once the Kafka cluster is running, we can go ahead configuring Logstash. When it is required to make changes to the Kafka cluster, we should shut down the cluster gracefully as below, then make changes and start the cluster again:

```
./bin/kafka-server-stop.sh
```

5.3.5 Kafka Manager Deployment

A Kafka cluster can be managed with CLI commands. However, it is not quit handy. Kafka Manager is a web based tool which makes the basic Kafka management tasks straightforward. The tool currently is maintained by Yahoo and has been renamed as CMAK (Cluster Management for Apache Kafka). Anyway, we prefer calling it Kafka Manager.

1. Download the application from its github repo;
2. TBD

5.3.6 Logstash Deployment

Based on our introduction of the demonstration environemnt, we have 2 sets of Logstash deployment:

- Log Producers: logstash69167/69168

  Collect logs from data sources (such as syslog, filebeat, etc.) and forward log entries to corresponding Kafka topics. The num. of such Logstash instances can be determined based on the amount of data generated by data sources.

  Actually, such Logstash instances are separated from each other. In other words, they work as standalone instances and have no knowledge on others.

- Log Consumers: logstash69158/69159

  Consume logs from Kafka topics, modify logs based on pipeline definitions and ship modified logs to Elasticsearch.

  Such Logstash instances have the identical pipeline configurations (except for client_id) and belong to the same Kafka consumer group which load balance each other.

The installation of Logstash has been covered in previous chapters, we won't cover them again in this chapter, instead, we will focus our effort on the clarification of pipeline definitions when Kafka is leveraged in the middle.

Logstash Which Produce Logs to Kafka

We are going to configure pipelines for logstash69167/69168. Each Logstash instance is responsible for consolidating logs for some specified data sources.

- logstash69167: consolidate logs for storage arrays and application solutions based on Linux;
- logstash69168: consolidate logs for ethernet switches and application solutions based on Windows.

1. Define pipelines(/etc/logstash/conf.d)

   - logstash69167

```
# /etc/logstash/conf.d/ps_rhel.conf
input {
  beats {
    port => 5045
    tags => ["server", "filebeat", "ps", "rhel"]
  }
}

filter {
  mutate {
    rename => ["host", "server"]
  }
}

output {
  kafka {
    id => "ps-rhel"
    topic_id => "ps-rhel"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}

# /etc/logstash/conf.d/sc_sles.conf
input {
  beats {
```

(continues on next page)

(continued from previous page)

```
    port => 5044
    tags => ["server", "filebeat", "sc", "sles"]
  }
}

filter {
  mutate {
    rename => ["host", "server"]
  }
}

output {
  kafka {
    id => "sc-sles"
    topic_id => "sc-sles"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}


# /etc/logstash/conf.d/pssc.conf
input {
  udp {
    port => 514
    tags => ["array", "syslog", "sc", "ps"]
  }
}

output {
  kafka {
    id => "pssc"
    topic_id => "pssc"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}

# /etc/logstash/conf.d/unity.conf
input {
  udp {
    port => 5000
    tags => ["array", "syslog", "unity"]
  }
}

output {
  kafka {
    id => "unity"
    topic_id => "unity"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}
```

(continues on next page)

(continued from previous page)

```
# /etc/logstash/conf.d/xio.conf
input {
  udp {
    port => 5002
    tags => ["array", "syslog", "xio"]
  }
}

output {
  kafka {
    id => "xio"
    topic_id => "xio"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}
```

- logstash69168

```
# /etc/logstash/conf.d/ethernet_switch.conf
input {
  udp {
    port => 514
    tags => ["switch", "syslog", "network", "ethernet"]
  }
}

output {
  kafka {
    id => "ether-switch"
    topic_id => "ether-switch"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}

# /etc/logstash/conf.d/vnx_exchange.conf
input {
  beats {
    port => 5044
    tags => ["server", "winlogbeat", "vnx", "windows", "exchange"]
  }
}

filter {
  mutate {
    rename => ["host", "server"]
  }
}

output {
  kafka {
    id => "vnx-exchange"
    topic_id => "vnx-exchange"
```

(continues on next page)

(continued from previous page)

```
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}

# /etc/logstash/conf.d/vnx_mssql.conf
input {
  beats {
    port => 5045
    tags => ["server", "winlogbeat", "vnx", "windows", "mssql"]
  }
}

filter {
  mutate {
    rename => ["host", "server"]
  }
}

output {
  kafka {
    id => "vnx-mssql"
    topic_id => "vnx-mssql"
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.
↪157:9092"
  }
}
```

2. Enable pipelines (/etc/logstash/pipelines.yml):

   - logstash69167:

```
- pipeline.id: ps_rhel
  path.config: "/etc/logstash/conf.d/ps_rhel.conf"
- pipeline.id: sc_sles
  path.config: "/etc/logstash/conf.d/sc_sles.conf"
- pipeline.id: pssc
  path.config: "/etc/logstash/conf.d/pssc.conf"
- pipeline.id: unity
  path.config: "/etc/logstash/conf.d/unity.conf"
- pipeline.id: xio
  path.config: "/etc/logstash/conf.d/xio.conf"
```

   - logstash69168:

```
- pipeline.id: ethernet_switch
  path.config: "/etc/logstash/conf.d/ethernet_switch.conf"
- pipeline.id: vnx_exchange
  path.config: "/etc/logstash/conf.d/vnx_exchange.conf"
- pipeline.id: vnx_mssql
  path.config: "/etc/logstash/conf.d/vnx_mssql.conf"
```

3. Start Logstash servers on all nodes:

```
systemctl start logstash
```

4. Verify topics are successfully created on Kafka:

```
ssh root@kafka69155/156/157
./bin/kafka-topics.sh -bootstrap-server "10.226.69.155:9092,10.226.69.156:9092,10.
↪226.69.157:9092" --list
```

5. Verify logs are sent to Kafka successfully:

```
ssh root@kafka69155/156/157
./bin/kafka-console-consumer.sh -bootstrap-server "10.226.69.155:9092,10.226.69.
↪156:9092,10.226.69.157:9092" --topic <topic name>
```

Now, we have our Logstash instances configured as Kafka producers. Before moving forward, it is worthwhile to introduce some tips on pipeline configurations when Kafka is used as the output plugin.

- Never define complicated filters for pipelines of such Logstash instances since they may increase latency;
- Add tags to the input section to ease the effort of log search/classification with Kibana;
- Specify different **id** with meaningful names for different pipelines;
- Rename the **host** field to some other meaningful name if syslog is also a data source in the setup. Refer to the **tips** chapter on the explanation about this.

Logstash Which Consume Logs from Kafka

We are going to configure pipelines for logstash69158/69159. These two Logstash instances have identical pipeline definitions (except for client_id) and consume messages from Kafka topics evenly by leveraging the consumer group feature of Kafka.

Since logs are cached in Kafka safely, it is the right place to define complicated filters with pipelines to modify log entires before sending them to Elasticsearch. This won't lead to bottlenecks since logs are already there in Kafka, the only impact is that you may need to wait for a while before you are able to see the logs in Elasticsearch/Kibana. And if it is time senstive to see logs from Elasticsearch/Kibana, more Logstash instances belong to the same consumer group can be added to load balance the processing.

1. Define pipelines(/etc/logstash/conf.d): **client_id** should always be set with different values

```
# /etc/logstash/conf.d/kafka_array.conf
input {
  kafka {
    client_id => "logstash69158-array"
    # client_id => "logstash69159-array"
    group_id => "logstash-array"
    topics => ["unity", "vnx", "xio", "pssc", "powerstore"]
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.157:9092
↪"
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0690-152:9200", "http://e2e-l4-0690-153:9200",
↪"http://e2e-l4-0690-154:9200"]
    index => "edc-storage-%{+YYYY.MM.dd}"
```

(continues on next page)

(continued from previous page)

```
  }
}

# /etc/logstash/conf.d/kafka_server.conf
input {
  kafka {
    client_id => "logstash69158-server"
    # client_id => "logstash69159-server"
    group_id => "logstash-server"
    topics => ["sc-sles", "ps-rhel", "vnx-exchange", "vnx-mssql"]
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.157:9092
↪"
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0690-152:9200", "http://e2e-l4-0690-153:9200",
↪"http://e2e-l4-0690-154:9200"]
    index => "edc-server-%{+YYYY.MM.dd}"
  }
}

# /etc/logstash/conf.d/kafka_switch.conf
input {
  kafka {
    client_id => "logstash69158-switch"
    # client_id => "logstash69159-switch"
    group_id => "logstash-switch"
    topics => ["ether-switch"]
    codec => "json"
    bootstrap_servers => "10.226.69.155:9092,10.226.69.156:9092,10.226.69.157:9092
↪"
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0690-152:9200", "http://e2e-l4-0690-153:9200",
↪"http://e2e-l4-0690-154:9200"]
    index => "edc-ethernet-%{+YYYY.MM.dd}"
  }
}
```

2. Enable pipelines on all nodes(/etc/logstash/pipelines.yml):

```
- pipeline.id: kafka_array
  path.config: "/etc/logstash/conf.d/kafka_array.conf"
- pipeline.id: kafka_server
  path.config: "/etc/logstash/conf.d/kafka_server.conf"
- pipeline.id: kafka_switch
  path.config: "/etc/logstash/conf.d/kafka_switch.conf"
```

3. Start logstash on all nodes:

```
systemctl start logstash
```

After configuring and starting Logstash, logs should be able to be sent to Elasticsearch and can be checked from Kibana.

Now, we have our Logstash instances configured as Kafka consumers. Before moving forward, it is worthwhile to introduce some tips on pipeline configurations when Kafka is used as the input plugin.

- **client_id** should always be set with *different values* for each pipeline on differnt Logstash instances. This field is used to identify consumers on Kafka;
- **group_id** should be set with the *idenfical value* for the same pipeline on different Logstsh instances. This field is used to identify consumer groups on Kafka, and load balance won't work if the value are different.

5.4 Data Source Configuration

Data sources are servers, switches, arrays, etc. which send logs to Logstash through beat, syslog, etc. Configuring them follows the same steps as when there is no Kafka integrated, please refer to previous chapter accordingly.

5.5 Conclusion

We have configured a demonstration environment with Kafka integrated with ELK Stack. By integrating Kafka, log processing performance can be boosted(adding a cache layer) and more potential applications can be integrated (consume log messages from Kafka and perform some special operations such as ML).

CHAPTER 6

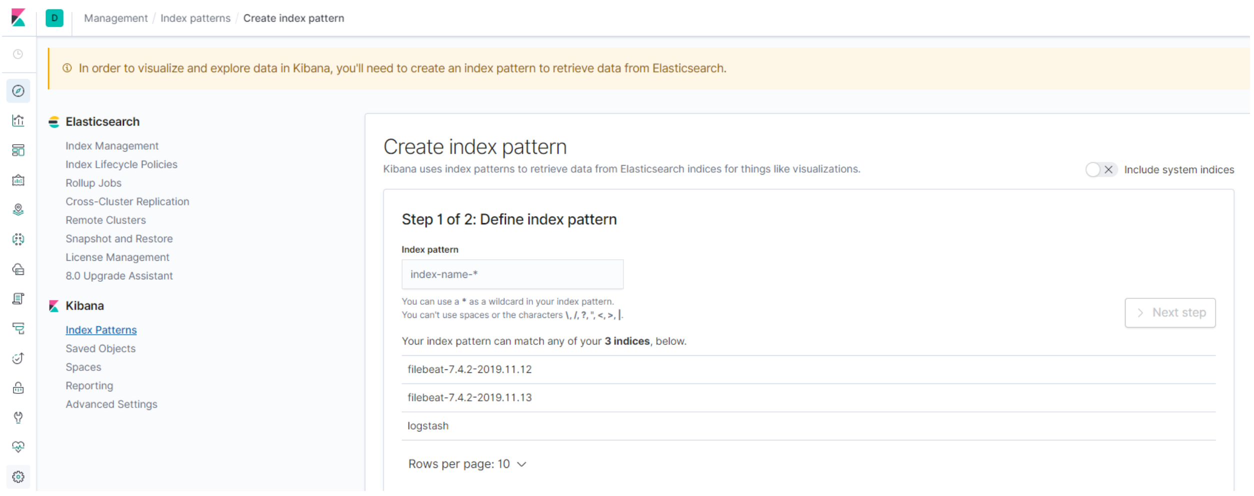
Check Logs with Kibana

Kibana is the web based front end GUI for Elasticsearch. It can be used to search, view, and interact with data stored in Elasticsearch indices. Advanced data analysis and visualize can be performed with the help of Kibana smoothly.

We have completed an end to end production environement ELK stack configuration with the previous chapter. In this chapter, we will use Kibana to explore the collcted data.

6.1 Index Patterns

The first time you login Kibana (http://<IP or FQDN>:5601), a hint as **In order to visualize and explore data in Kibana, you'll need to create an index pattern to retrieve data from Elasticsearch** will be shown on the top of the page and a shortcut to create an index pattern is shown:



In order to visualize and explore data in Kibana, you'll need to create an index pattern to retrieve data from Elasticsearch.

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Step 1 of 2: Define index pattern

Index pattern

index-name-*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

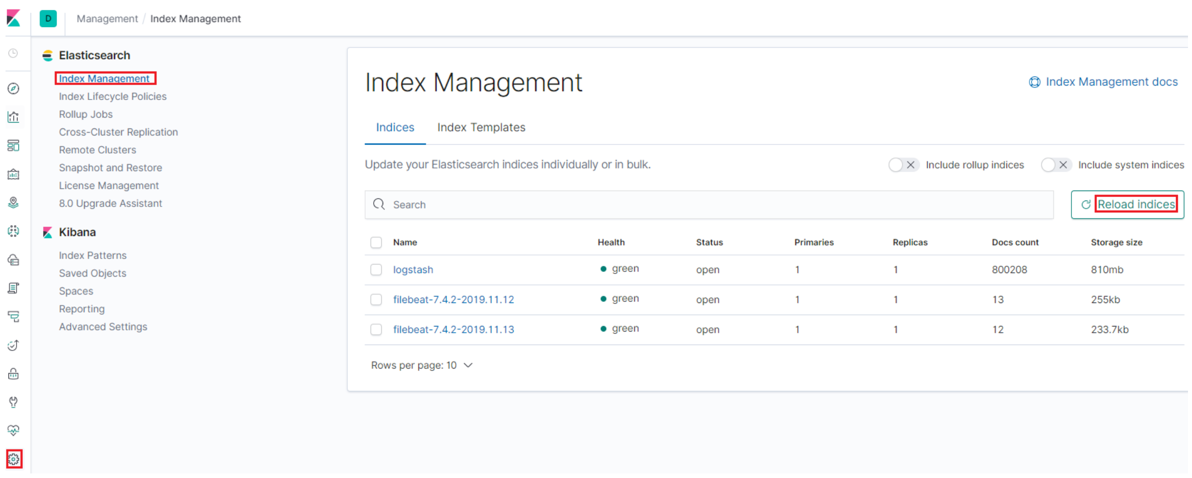
Your index pattern can match any of your **3 indices**, below.

filebeat-7.4.2-2019.11.12
filebeat-7.4.2-2019.11.13
logstash

An index pattern tells Kibana which Elasticsearch indices you want to explore. It can match the name of a single index, or include a wildcard (*) to match multiple indices. But wait, what is an index? An index is a kind of data organization mechanism on how your data is stored and indexed. Every single piece of data sent to Elasticsearch actually is targging at an index (stored and indexed). To retrieve data, we of course need to let Kibana know the data souce (index patterns). Please refer to this blog for more details on index.

To **create index patterns**, it is recommended to conduct the operation from the **Management** view of Kibana:

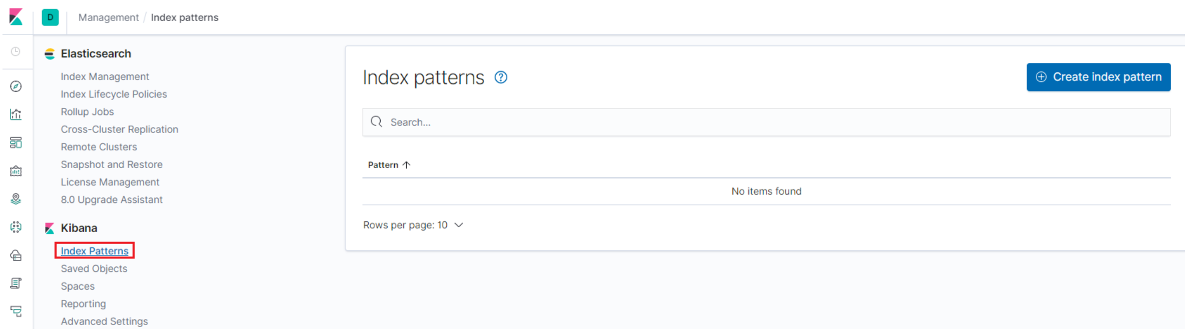
1. Go to the "Management" view, then check available indices (reload indices if there is none):



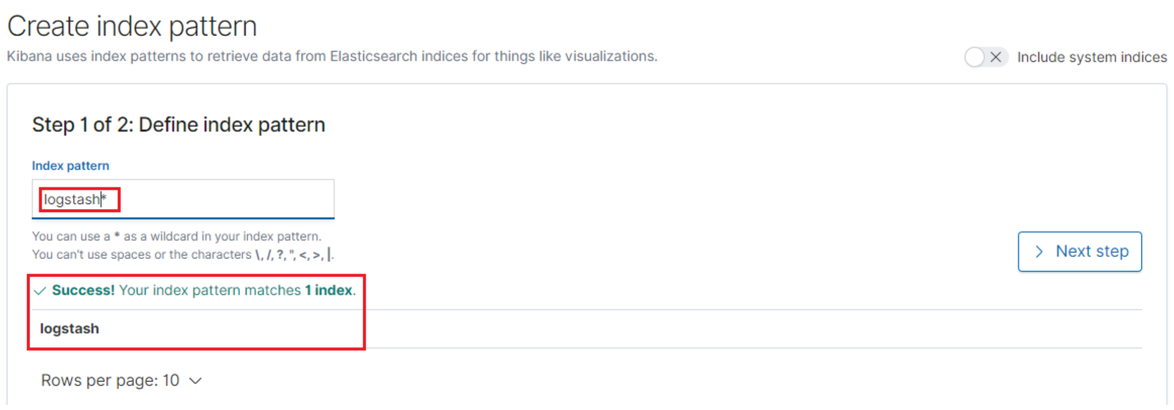
Update your Elasticsearch indices individually or in bulk.

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
logstash	● green	open	1	1	800208	810mb
filebeat-7.4.2-2019.11.12	● green	open	1	1	13	255kb
filebeat-7.4.2-2019.11.13	● green	open	1	1	12	233.7kb

2. Based on the name of existing indices, created index patterns:



No items found



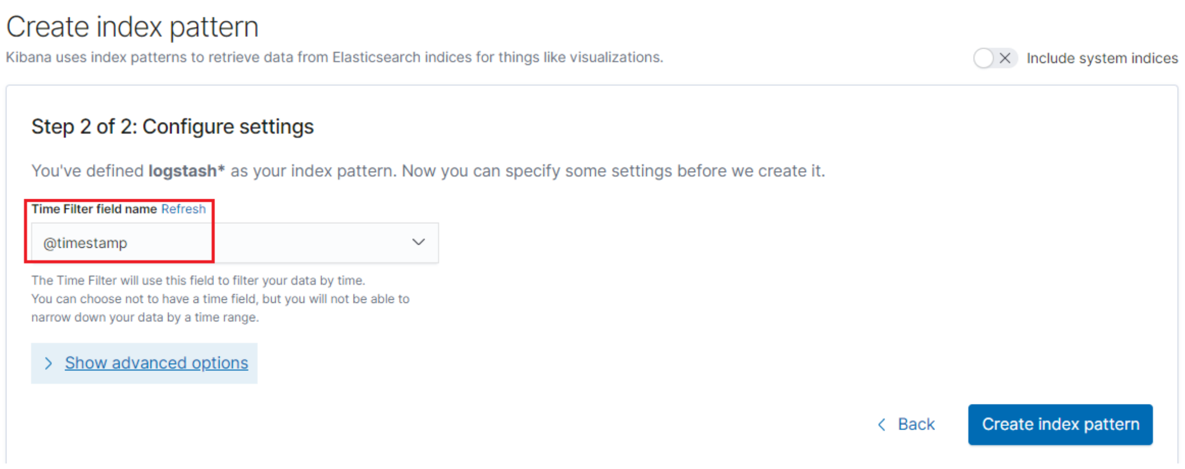
Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Index pattern: logstash*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

✓ **Success!** Your index pattern matches **1 index**.

logstash


Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

Step 2 of 2: Configure settings

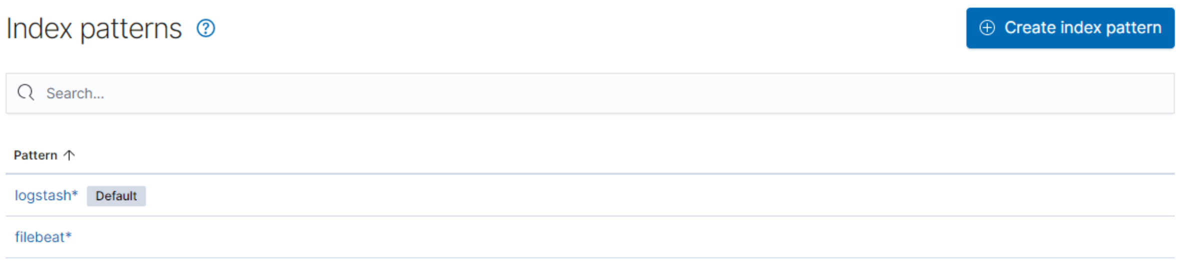
You've defined **logstash*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

@timestamp

The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

3. We create index patterns for **logstash** and **filebeat**:


Index patterns

Pattern ↑
logstash* Default
filebeat*

After creating index patterns, we can start exploring data from the **Discover** view by selecting a pattern:

6.2 KQL Basics

To smooth the exprience of filtering logs, Kibana provides a simple language named **Kibana Query Lanagure (KQL for short)**. The syntax is really straightforward, we will introduce the basics in this section.

6.2.1 Simple Match

Syntax:

```
<field name>: <word to match>
```

Example:

- response:200
  - Search documents (log records) which have a field named "response" and its value is "200"

6.2.2 Quotes Match

Syntax:

```
<filed name>: "<words to match>"
```

Example:

- message:"Quick brown fox"
  - Search the quoted string "Quick brown fox" in the "message" field;
  - If quotes are not used, search documents which have word "Quick" in the "message" field, and have fields "brown" and "fox"

6.2.3 Complicated Match

Syntax:

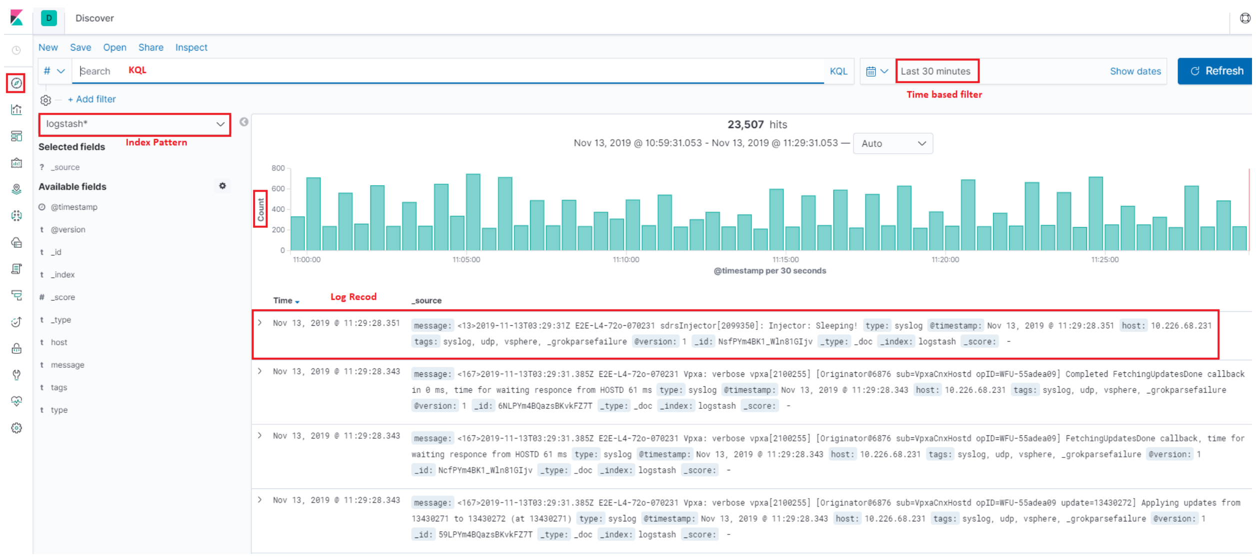
- Logical combination: and, or, not
- Grouping : ()
- Range : >, >=, <, <=
- Wildcard : *

Examples:

- response:200 and extension:php
  - Match documents whose "response" field is "200" **and** "extension" field is "php"
- response:200 and (extension:php or extension:css)
  - Match documents whose "response" field is 200 **and** "extension" field is "php" **or** "css"
- response:200 and not (extension:php or extension:css)
  - Match documents whose "response" field is 200 **and** "extension" field is **not** "php" **or** "css"
- response:200 and bytes > 1000
  - Match documents whose "response" field is 200 and "bytes" field is in **range** larger than "1000"
- machine.os:win*
  - Match documents whose "machine" field has a subfield "os" and its value start with "win", such as "windows", "windows 2016"
- machine.os.*:"windows 10"
  - Match documents whose "machine" field has a subfiled "os" which also has subfileds and any of such subfields' value contains "windows 10"

6.3 Explore Real Data

We have introduced index patterns and KQL, it is time to have a look at real data in our production setup. All log records will be structured as JSON documents as we previously introduced, and Kibana will show a summary for related indices as below once an index pattern is selected:



23,507 hits

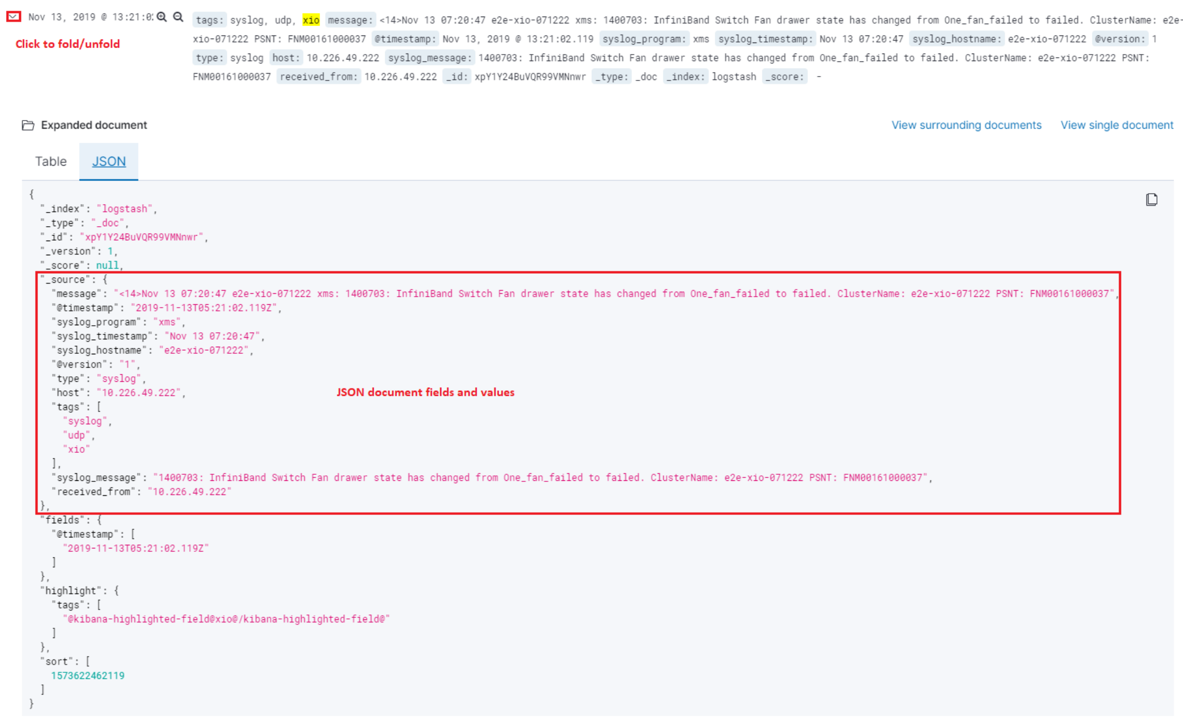
Nov 13, 2019 @ 10:59:31.053 - Nov 13, 2019 @ 11:29:31.053

```
Nov 13, 2019 @ 11:29:28.351  message: <13>2019-11-13T03:29:31Z E2E-L4-72o-070231 sdrsInjector[2099350]: Injector: Sleeping! type: syslog @timestamp: Nov 13, 2019 @ 11:29:28.351 host: 10.226.68.231
                             tags: syslog, udp, vsphere, _grokparsefailure @version: 1 _id: NsfPYm4BK1_Wln81GIjv _type: _doc _index: logstash _score: -

Nov 13, 2019 @ 11:29:28.343  message: <167>2019-11-13T03:29:31.385Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2100255] [Originator@6876 sub=VpxaCnxHostd opID=WFU-55adea09] Completed FetchingUpdatesDone callback
                             in 0 ms, time for waiting responce from HOSTD 61 ms type: syslog @timestamp: Nov 13, 2019 @ 11:29:28.343 host: 10.226.68.231 tags: syslog, udp, vsphere, _grokparsefailure
                             @version: 1 _id: 6NLPYm4BQazsBKvkFZ7T _type: _doc _index: logstash _score: -

Nov 13, 2019 @ 11:29:28.343  message: <167>2019-11-13T03:29:31.385Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2100255] [Originator@6876 sub=VpxaCnxHostd opID=WFU-55adea09] FetchingUpdatesDone callback, time for
                             waiting responce from HOSTD 61 ms type: syslog @timestamp: Nov 13, 2019 @ 11:29:28.343 host: 10.226.68.231 tags: syslog, udp, vsphere, _grokparsefailure @version: 1
                             _id: NcfPYm4BK1_Wln81GIjv _type: _doc _index: logstash _score: -

Nov 13, 2019 @ 11:29:28.343  message: <167>2019-11-13T03:29:31.385Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2100255] [Originator@6876 sub=VpxaCnxHostd opID=WFU-55adea09 update=13430272] Applying updates from
                             13430271 to 13430272 (at 13430271) type: syslog @timestamp: Nov 13, 2019 @ 11:29:28.343 host: 10.226.68.231 tags: syslog, udp, vsphere, _grokparsefailure @version: 1
                             _id: 59LPYm4BQazsBKvkFZ7T _type: _doc _index: logstash _score: -
```

As we said, log records will be formated/structured as JSON documents. Bug how? Actually, there is term called **mapping**, which performs the translation work from the original format (such as text) to JSON. Since logstash and filebeat already have internal mapping defined, we do not need to care about the details. What we should know is that the JSON documents from different data input (logstash, filebeat, etc.) may be different because of the mapping. For more information on mapping, please refer to the offical introduction.

Below are JSON document samples from different input type:

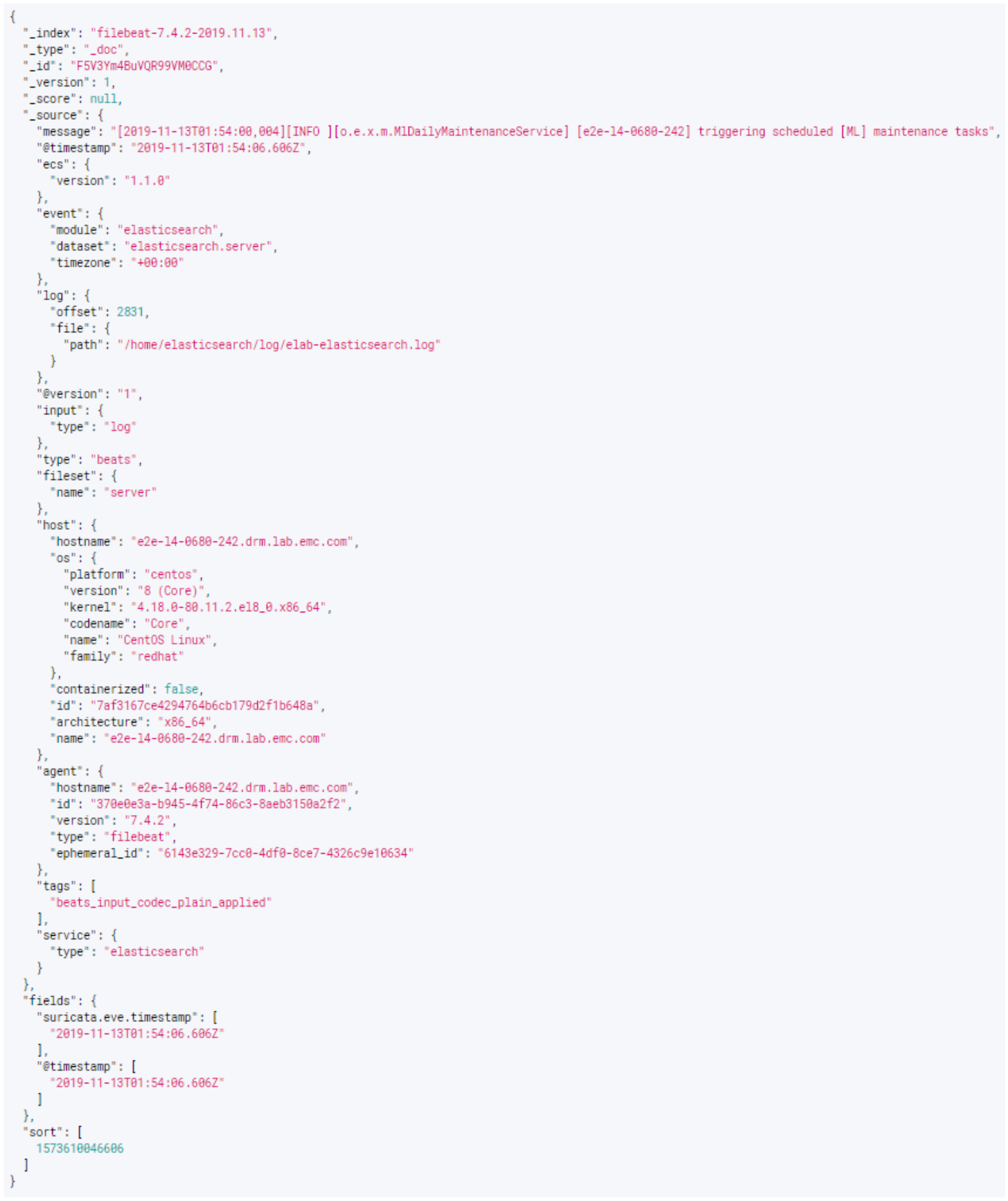
- logstash:



```
Nov 13, 2019 @ 13:21:0  tags: syslog, udp, xio message: <14>Nov 13 07:20:47 e2e-xio-071222 xms: 1400703: InfiniBand Switch Fan drawer state has changed from One_fan_failed to failed. ClusterName: e2e-
                        xio-071222 PSNT: FNM00161000037 @timestamp: Nov 13, 2019 @ 13:21:02.119 syslog_program: xms syslog_timestamp: Nov 13 07:20:47 syslog_hostname: e2e-xio-071222 @version: 1
                        type: syslog host: 10.226.49.222 syslog_message: 1400703: InfiniBand Switch Fan drawer state has changed from One_fan_failed to failed. ClusterName: e2e-xio-071222 PSNT:
                        FNM00161000037 received_from: 10.226.49.222 _id: xpY1Y24BuVQR99VMNnwr _type: _doc _index: logstash _score: -
```

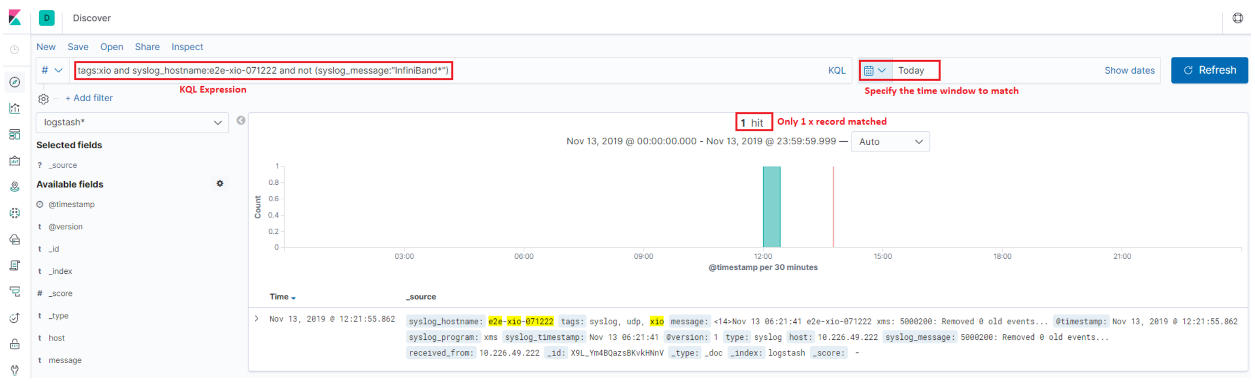
```
{
  "_index": "logstash",
  "_type": "_doc",
  "_id": "xpY1Y24BuVQR99VMNnwr",
  "_version": 1,
  "_score": null,
  "_source": {
    "message": "<14>Nov 13 07:20:47 e2e-xio-071222 xms: 1400703: InfiniBand Switch Fan drawer state has changed from One_fan_failed to failed. ClusterName: e2e-xio-071222 PSNT: FNM00161000037",
    "@timestamp": "2019-11-13T05:21:02.119Z",
    "syslog_program": "xms",
    "syslog_timestamp": "Nov 13 07:20:47",
    "syslog_hostname": "e2e-xio-071222",
    "@version": "1",
    "type": "syslog",
    "host": "10.226.49.222",
    "tags": [
      "syslog",
      "udp",
      "xio"
    ],
    "syslog_message": "1400703: InfiniBand Switch Fan drawer state has changed from One_fan_failed to failed. ClusterName: e2e-xio-071222 PSNT: FNM00161000037",
    "received_from": "10.226.49.222"
  },
  "fields": {
    "@timestamp": [
      "2019-11-13T05:21:02.119Z"
    ]
  },
  "highlight": {
    "tags": [
      "@kibana-highlighted-field@xio@/kibana-highlighted-field@"
    ]
  },
  "sort": [
    1573622462119
  ]
}
```

- filebeat:



```
{
  "_index": "filebeat-7.4.2-2019.11.13",
  "_type": "_doc",
  "_id": "F5V3Ym4BuVQR99VM0CCG",
  "_version": 1,
  "_score": null,
  "_source": {
    "message": "[2019-11-13T01:54:00,004][INFO ][o.e.x.m.MlDailyMaintenanceService] [e2e-14-0680-242] triggering scheduled [ML] maintenance tasks",
    "@timestamp": "2019-11-13T01:54:06.606Z",
    "ecs": {
      "version": "1.1.0"
    },
    "event": {
      "module": "elasticsearch",
      "dataset": "elasticsearch.server",
      "timezone": "+00:00"
    },
    "log": {
      "offset": 2831,
      "file": {
        "path": "/home/elasticsearch/log/elab-elasticsearch.log"
      }
    },
    "@version": "1",
    "input": {
      "type": "log"
    },
    "type": "beats",
    "fileset": {
      "name": "server"
    },
    "host": {
      "hostname": "e2e-14-0680-242.drm.lab.emc.com",
      "os": {
        "platform": "centos",
        "version": "8 (Core)",
        "kernel": "4.18.0-80.11.2.el8_0.x86_64",
        "codename": "Core",
        "name": "CentOS Linux",
        "family": "redhat"
      },
      "containerized": false,
      "id": "7af3167ce4294764b6cb179d2f1b648a",
      "architecture": "x86_64",
      "name": "e2e-14-0680-242.drm.lab.emc.com"
    },
    "agent": {
      "hostname": "e2e-14-0680-242.drm.lab.emc.com",
      "id": "370e0e3a-b945-4f74-86c3-8aeb3150a2f2",
      "version": "7.4.2",
      "type": "filebeat",
      "ephemeral_id": "6143e329-7cc0-4df0-8ce7-4326c9e10634"
    },
    "tags": [
      "beats_input_codec_plain_applied"
    ],
    "service": {
      "type": "elasticsearch"
    }
  },
  "fields": {
    "suricata.eve.timestamp": [
      "2019-11-13T01:54:06.606Z"
    ],
    "@timestamp": [
      "2019-11-13T01:54:06.606Z"
    ]
  },
  "sort": [
    1573610046606
  ]
}
```

Based on the samples, we see each document consist of a few fields. These fields are the key for filtering. For example, we can filter logs which are from "xio" with hostname "e2e-xio-071222" and not related with "InfiniBand" as below:



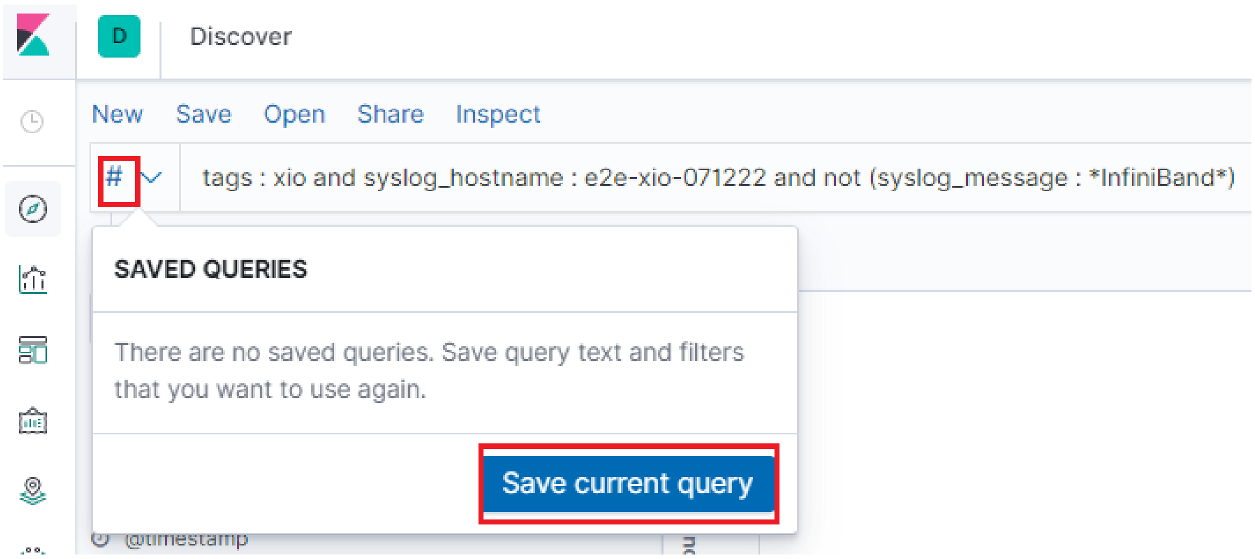
```
tags:xio and syslog_hostname:e2e-xio-071222 and not (syslog_message:"InfiniBand*")
```

```
Nov 13, 2019 @ 12:21:55.862  syslog_hostname: e2e-xio-071222 tags: syslog, udp, xio message: <14>Nov 13 06:21:41 e2e-xio-071222 xms: 5000200: Removed 0 old events... @timestamp: Nov 13, 2019 @ 12:21:55.862
                             syslog_program: xms syslog_timestamp: Nov 13 06:21:41 @version: 1 type: syslog host: 10.226.49.222 syslog_message: 5000200: Removed 0 old events...
                             received_from: 10.226.49.222 _id: X9L_Ym4BQazsBKvkHNnV _type: _doc _index: logstash _score: -
```

Pretty easy, right? There is no more magic for this! Just specify your KQL with fields and value expressions, that is all!

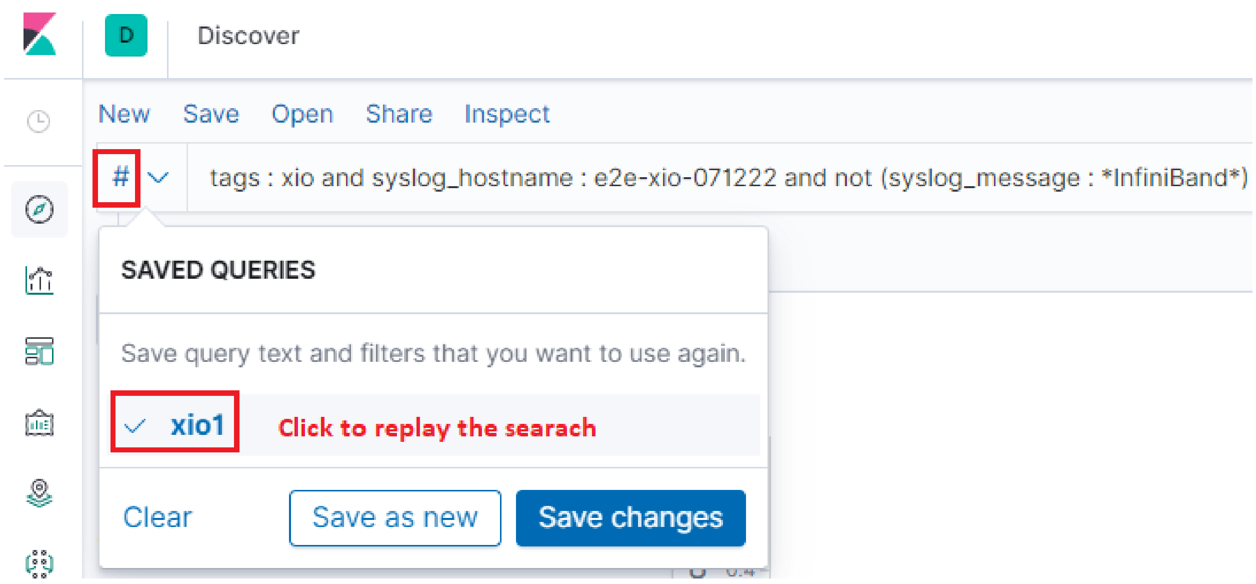
6.4 Save Search/Query

It is a frequent request that we want to classify logs based on different condtions. Of course, we can achieve this by using different KQL expressions, but keeping inputting KQL expressions is not a comfortable way. Kibana provides the functions to save your search/query and replay them on demand.



```
tags : xio and syslog_hostname : e2e-xio-071222 and not (syslog_message : *InfiniBand*)
```

There are no saved queries. Save query text and filters that you want to use again.


Discover
New
Save
Open
Share
Inspect
#
SAVED QUERIES
Save query text and filters that you want to use again.
xio1
Click to replay the seaarch
Clear
Save as new
Save changes

tags : xio and syslog_hostname : e2e-xio-071222 and not (syslog_message : *InfiniBand*)

6.5 Conclusion

Using Kibana explore logs is as easy as we introcued above. Although its usage is easy and straightforward, it is powerful enough covering our daily log processing tasks.

CHAPTER 7

Grok Debugger

In the previous chapter on configuring ELK stack for the end to end production environment, someone may notice that we use the same grok expression for all syslog pipelines, no matter if they are from XtremIO, vSphere, or Unity.

The pattern is as below:

```
%{SYSLOGTIMESTAMP:syslog_timestamp} %{DATA:syslog_hostname} %{DATA:syslog_program}(?
↪:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}
```

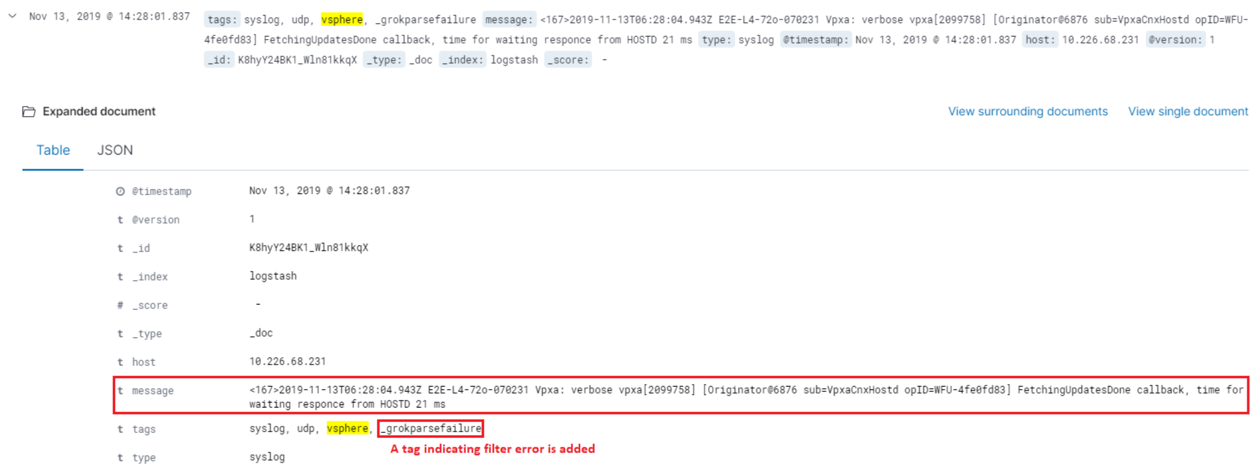
Although, this is the standard format for syslog, but it is a kind of soft requirement. A syslog record may contain much more fields, but may contain fewer fields also. Under such condtion, the expression won't work.

In fact, the expression does not work for vSphere syslog on our production environment. We will show how to fix grok expression issue with "Grok Debugger" provided by Kibana in this chapter.

7.1 Logstash pipeline behavior for failed filters

When data (logs, metrics, etc.) comes into the process of a Logstash pipeline (input), Logstash will modify data based on configured filter (filter plugins). If a filter fail to process data, Logstash won't get blocked but just add some tags and go ahead forward the original data to destionations (output).

Here is an example of log record which cannot be processed correctly by the grok pattern we previously defined:



```
Nov 13, 2019 @ 14:28:01.837   tags: syslog, udp, vsphere, _grokparsefailure message: <167>2019-11-13T06:28:04.943Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099758] [Originator@6876 sub=VpxaCnxHostd opID=WFU-4fe0fd83] FetchingUpdatesDone callback, time for waiting responce from HOSTD 21 ms type: syslog @timestamp: Nov 13, 2019 @ 14:28:01.837 host: 10.226.68.231 @version: 1 _id: K8hyY24BK1_Wln81kkqX _type: _doc _index: logstash _score: -
```

```
@timestamp   Nov 13, 2019 @ 14:28:01.837
@version     1
_id          K8hyY24BK1_Wln81kkqX
_index       logstash
_score        -
_type        _doc
host         10.226.68.231
message      <167>2019-11-13T06:28:04.943Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099758] [Originator@6876 sub=VpxaCnxHostd opID=WFU-4fe0fd83] FetchingUpdatesDone callback, time for
             waiting responce from HOSTD 21 ms
tags         syslog, udp, vsphere, _grokparsefailure
type         syslog
```

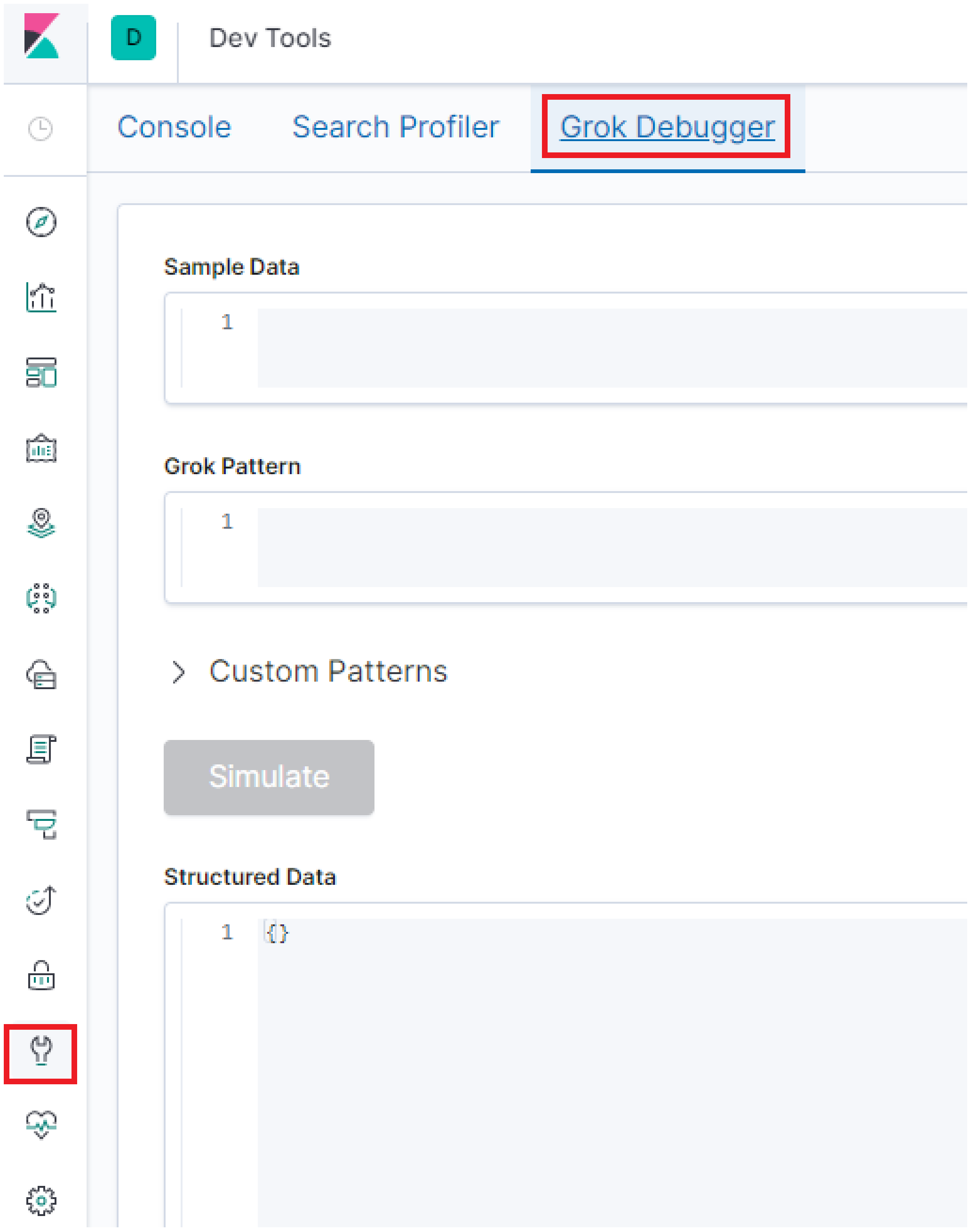
Based on the example, we can see the orignal log record as below is sent to Elasticsearch directly. Fields (syslog_timestamp, syslog_hostname, etc.) which should be added after a successfuly grok prcoessing do not exist for the record:

```
2019-11-13T06:30:20.511Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099751] [Originator@6876 sub=VpxaCnxHostd opID=WFU-6ae80d08] Completed FetchingUpdatesDone callback in 0 ms, time for waiting responce from HOSTD 71 ms
```

We will fix this in the coming sections.

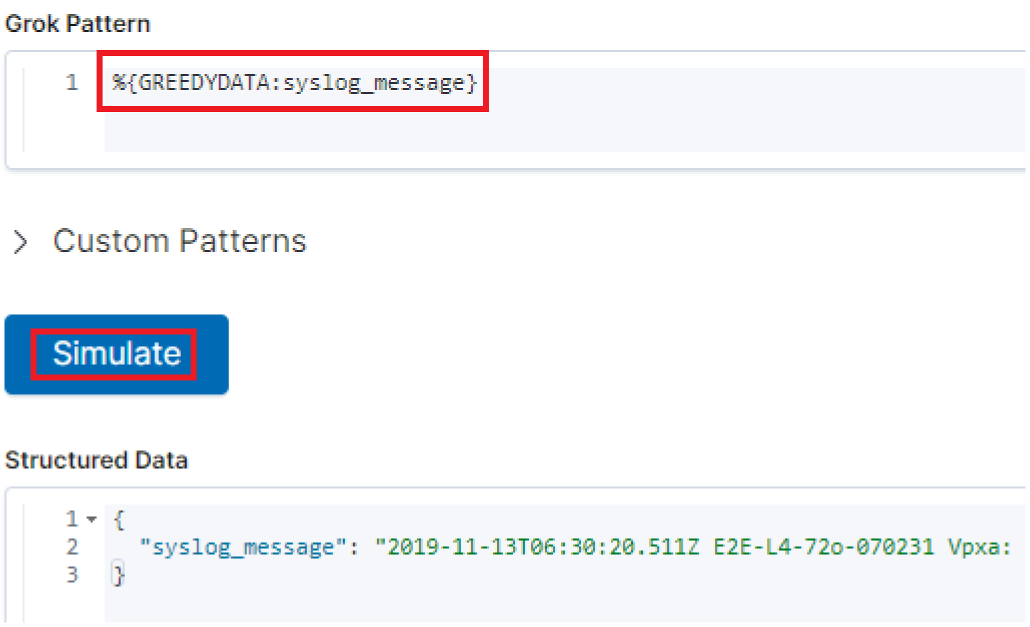
7.2 Kibana Grok Debugger

Since it is a frequent task tuning grok expressions, Kibana provides a debugger with it:



With this tool, we can debug our grok expression lively. Let's fix our above mentioned issues with it.

1. Paste our log record into the **Sample Data** field of the Grok Debugger;
2. Open the predefined grok patterns for reference;
3. Take all the inforamtion into field syslog_message:


Grok Pattern

```
%{GREEDYDATA:syslog_message}
```

Structured Data

```
{
  "syslog_message": "2019-11-13T06:30:20.511Z E2E-L4-72o-070231 Vpxa:
}
```

4. Spit out the syslog_timestamp field:

```
%{DATA:syslog_timestamp} %{GREEDYDATA:syslog_message}
```

5. Spit out the syslog_hostname field:

```
%{DATA:syslog_timestamp} %{DATA:syslog_hostname} %{GREEDYDATA:syslog_message}
```

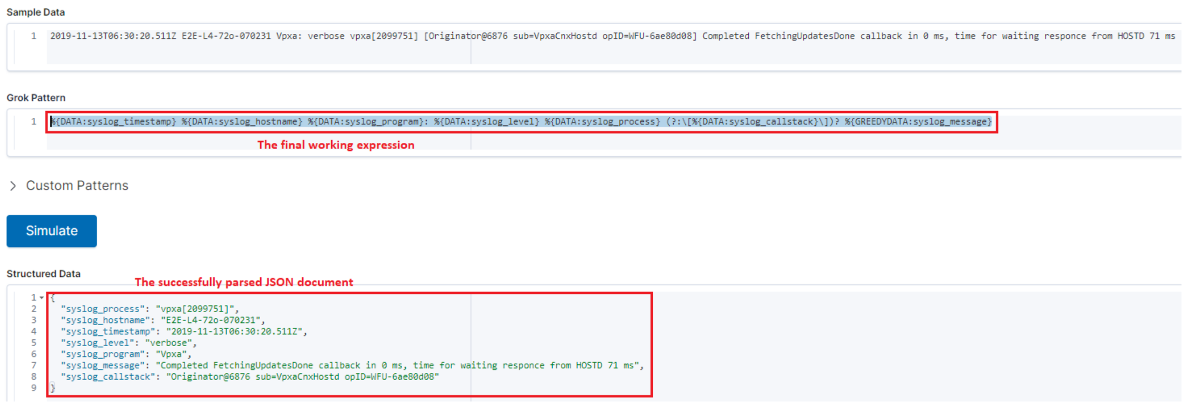
6. Spit out the syslog_program field:

```
%{DATA:syslog_timestamp} %{DATA:syslog_hostname} %{DATA:syslog_program}: %
↪{GREEDYDATA:syslog_message}
```

7. Continue similar operations until the full record is parsed successfully:

```
%{DATA:syslog_timestamp} %{DATA:syslog_hostname} %{DATA:syslog_program}: %
↪{DATA:syslog_level} %{DATA:syslog_process} (?:\[%{DATA:syslog_callstack}\])? %
↪{GREEDYDATA:syslog_message}
```

8. The full result is as below:


Sample Data

```
2019-11-13T06:30:20.511Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099751] [Originator@6876 sub=VpxaCnxHostd opID=WFU-6ae80d08] Completed FetchingUpdatesDone callback in 0 ms, time for waiting responce from HOSTD 71 ms
```

Grok Pattern

```
%{DATA:syslog_timestamp} %{DATA:syslog_hostname} %{DATA:syslog_program}: %{DATA:syslog_level} %{DATA:syslog_process} (?:\[%{DATA:syslog_callstack}\])? %{GREEDYDATA:syslog_message}
```

The final working expression

Structured Data

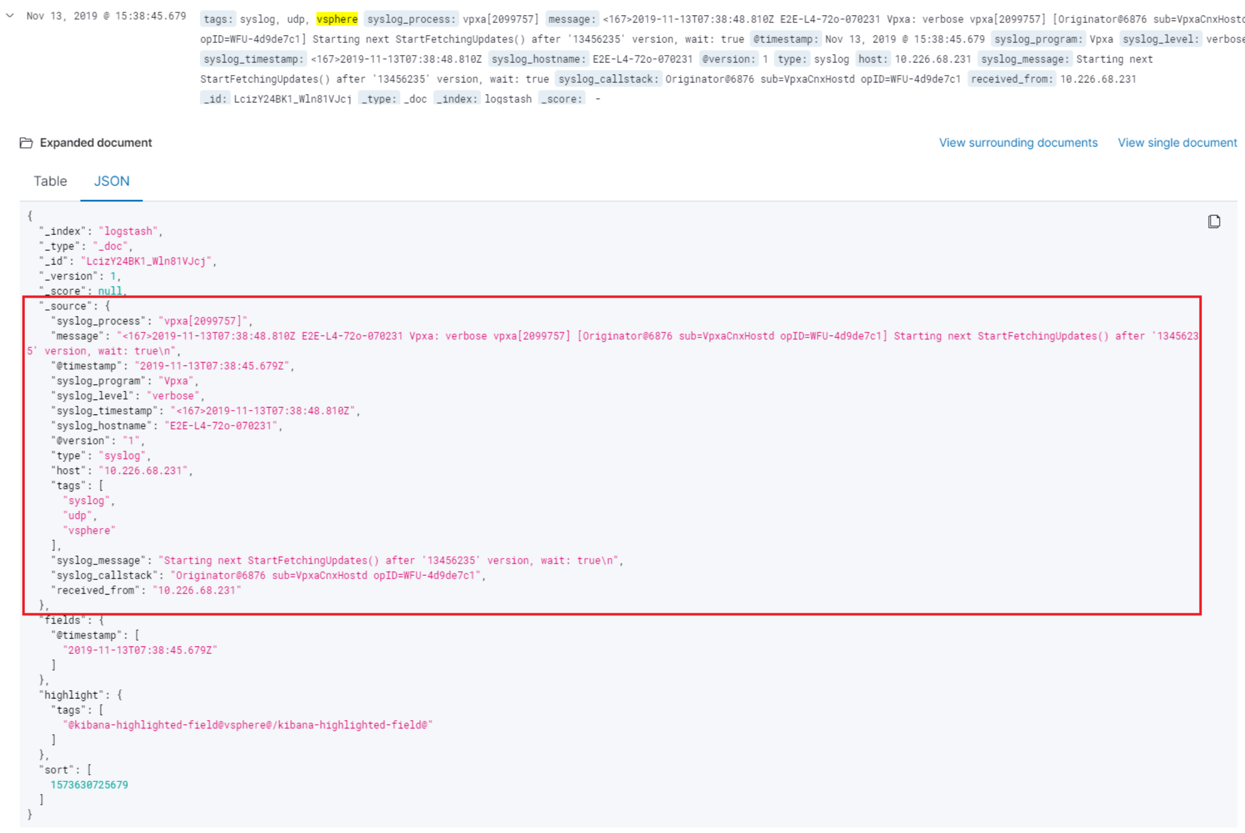
The successfully parsed JSON document

```
{
  "syslog_process": "vpxa[2099751]",
  "syslog_hostname": "E2E-L4-72o-070231",
  "syslog_timestamp": "2019-11-13T06:30:20.511Z",
  "syslog_level": "verbose",
  "syslog_program": "Vpxa",
  "syslog_message": "Completed FetchingUpdatesDone callback in 0 ms, time for waiting responce from HOSTD 71 ms",
  "syslog_callstack": "Originator@6876 sub=VpxaCnxHostd opID=WFU-6ae80d08"
}
```

Our new updated grok expression is ready. We can go to the Logstash pipe configuraiton (/etc/logstash/conf.d/syslog_vsphere.conf) and make changes accordingly:

```
input {
  tcp {
    type => "syslog"
    port => 5002
    tags => ["syslog", "tcp", "vsphere"]
  }
  udp {
    type => "syslog"
    port => 5002
    tags => ["syslog", "udp", "vsphere"]
  }
}

filter {
  grok {
    match => { "message" => "%{DATA:syslog_timestamp} %{DATA:syslog_hostname} %
↪{DATA:syslog_program}: %{DATA:syslog_level} %{DATA:syslog_process} (?:\[%
↪{DATA:syslog_callstack}\])? %{GREEDYDATA:syslog_message}" }
    add_field => [ "received_from", "%{host}" ]
  }
  date {
     match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    hosts => ["http://e2e-l4-0680-240:9200", "http://e2e-l4-0680-241:9200", "http://
↪e2e-l4-0680-242:9200"]
  }
}
```

From Kibana “Discover” view, all log records similar to the original one are parsed successfully:



```
Nov 13, 2019 @ 15:38:45.679   tags: syslog, udp, vsphere syslog_process: vpxa[2099757] message: <167>2019-11-13T07:38:48.810Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099757] [Originator@6876 sub=VpxaCnxHostd
opID=WFU-4d9de7c1] Starting next StartFetchingUpdates() after '13456235' version, wait: true @timestamp: Nov 13, 2019 @ 15:38:45.679 syslog_program: Vpxa syslog_level: verbose
syslog_timestamp: <167>2019-11-13T07:38:48.810Z syslog_hostname: E2E-L4-72o-070231 @version: 1 type: syslog host: 10.226.68.231 syslog_message: Starting next
StartFetchingUpdates() after '13456235' version, wait: true syslog_callstack: Originator@6876 sub=VpxaCnxHostd opID=WFU-4d9de7c1 received_from: 10.226.68.231
_id: LcizY24BK1_Wln81VJcj _type: _doc _index: logstash _score: -
```

```
{
  "_index": "logstash",
  "_type": "_doc",
  "_id": "LcizY24BK1_Wln81VJcj",
  "_version": 1,
  "_score": null,
  "_source": {
    "syslog_process": "vpxa[2099757]",
    "message": "<167>2019-11-13T07:38:48.810Z E2E-L4-72o-070231 Vpxa: verbose vpxa[2099757] [Originator@6876 sub=VpxaCnxHostd opID=WFU-4d9de7c1] Starting next StartFetchingUpdates() after '1345623
5' version, wait: true\n",
    "@timestamp": "2019-11-13T07:38:45.679Z",
    "syslog_program": "Vpxa",
    "syslog_level": "verbose",
    "syslog_timestamp": "<167>2019-11-13T07:38:48.810Z",
    "syslog_hostname": "E2E-L4-72o-070231",
    "@version": "1",
    "type": "syslog",
    "host": "10.226.68.231",
    "tags": [
      "syslog",
      "udp",
      "vsphere"
    ],
    "syslog_message": "Starting next StartFetchingUpdates() after '13456235' version, wait: true\n",
    "syslog_callstack": "Originator@6876 sub=VpxaCnxHostd opID=WFU-4d9de7c1",
    "received_from": "10.226.68.231"
  },
  "fields": {
    "@timestamp": [
      "2019-11-13T07:38:45.679Z"
    ]
  },
  "highlight": {
    "tags": [
      "@kibana-highlighted-field@vsphere@/kibana-highlighted-field@"
    ]
  },
  "sort": [
    1573630725679
  ]
}
```

7.3 Conclusion

With Grok Debugger, correct grok patterns can be defined for different log sources. Now, it is your turn to define your own expressions.

CHAPTER 8

Tips

8.1 How to add tags based on filed content with pipelines?

If the filed's name is known, it can be used directly. If not, use "message" which holds everything.

```
filter {
  if [message] =~ /regexp/ {
    mutate {
      add_tag => [ "tag1", "tag2" ]
    }
  }
}
```

8.2 Integrate Kafka

Kafka can be integrated into the middle of an Elastic Stack. The simplest implementation is leveraging the kafka input/output plugin of logstash directly. With that, the data flow looks as below:

- data source -> logstash -> kafka
- kafka -> logstash -> elasticsearch

More information on scaling Logstash can be found from Deploying and Scaling Logstash

8.2.1 Send into Kafka

1. Create a logstash pipeline as below:

```
input {
  tcp {
    port => 5000
```

(continues on next page)

(continued from previous page)

```
    tags => ["syslog", "topic1"]
  }
}

output {
  kafka {
    id => "topic1" # It is recommended to use different id for different Logstash␣
↪pipelines
    topic_id => "topic1"
    codec => json
    bootstrap_servers => "kafka_server1:9092,kafka_server2:9092,kafka_server3:9092
↪"
  }
}
```

2. Send a test information:

```
telnet logstash_server 5000
message1
message2
```

3. Verify the messages have been sent to Kafka successfully:

```
bin/kafka-console-consumer.sh --bootstrap-server "kafka_server1:9092,kafka_
↪server2:9092,kafka_server3:9092" --topic topic1 --from-beginning
```

8.2.2 Read from Kafka and Send to Elasticsearch

1. Create a logstash pipeline as below:

```
input {
  kafka {
    client_id => "logstash_server" # It is recommended to use different client_id␣
↪for different Logstash pipelines
    group_id => "logstash_server"
    topics => ["topic1"]
    codec => "json"
    bootstrap_servers => "kafka_server1:9092,kafka_server2:9092,kafka_server3:9092
↪"
  }
}

output {
  elasticsearch {
    hosts => ["http://elasticsearch1:9200", "http://elasticsearch2:9200", "http://
↪elasticsearch3:9200"]
    index => "topic1-%{+YYYY.MM.dd}"
  }

}
```

2. From Kibana, the informaiton should be able to be seen

8.3 Add Tags to Different Kafka Topics

Notes: [@metadata][kafka][topic] will be empty sometimes due to unknown issues. Hence this tip is listed here for reference.

```
input {
  kafka {
    client_id => "logstash_server"
    group_id => "logstash"
    topics => ["unity", "xio"]
    codec => "json"
    bootstrap_servers => "kafka_server1:9092,kafka_server2:9092,kafka_server3:9092"
  }
}

filter {
  if [@metadata][kafka][topic] == "unity" {
    mutate { add_tag => ["unity"] }
  }
  if [@metadata][kafka][topic] == "xio" {
    mutate { add_tag => ["xio"] }
  }
}

output {
  elasticsearch {
    hosts => ["http://elasticsearch1:9200", "http://elasticsearch2:9200", "http://
↪elasticsearch3:9200"]
    index => "storagebox-%{+YYYY.MM.dd}"
  }
}
```

8.4 Rename the Host Field while Sending Filebeat Events to Logstash

If filebeat is sending events to Elasticsearch directly, everything works fine. However, if filebeat is sending events to an index already used by Logstash where syslog(TCP/UDP input) is also sending events to, error on the host filed will be raised:

- TCP/UDP input plugin of Logstash will add a field **host** to stand for where the information is generated. This field is a string;
- Filebeat sends events with a filed **host** which is an object(dict);
- Because of the difference, Elasticsearch cannot map the host field correctly and generate index accordingly.

To fix this, the mutate filter plugin can be used to rename the host field of Filebeat to a new name as below:

```
filter {
  mutate {
    rename => ["host", "server"]
  }
}
```