GÉANT Project GN4-3

Accelerating Research, Driving Innovation, and Enriching Education

Work Package 6

Network Technologies and Services Development

Task 3

Monitoring and Management

WiFiMon

Wireless Crowdsourced Performance Monitoring and Verification

Streaming Logs Into ELK Cluster - Simulation

Revision	Date	Author
v1.0 - Added streaming for DHCP logs	2020-07-08	Sokol Gjeçi <sgjeci@rash.al>
v0.9 - Curator replaced by ILM	2020-06-27	Sokol Gjeçi <sgjeci@rash.al>
v0.8c - Added cover page	2020-02-04	Sokol Gjeçi <sgjeci@rash.al>
v0.8b - Fixed old HTTP config, Wordings corrections	2020-02-03	Sokol Gjeçi <sgjeci@rash.al>
v0.8a - Logstash: Cipher replaced by Fingerprint	2020-01-29	Sokol Gjeçi <sgjeci@rash.al>
v0.8 - SSL/TLS, Keystores, X-Pack, ELK v7.x	2019-12-15	Sokol Gjeçi <sgjeci@rash.al>
Initial document - Basic configuration	2019-05-02	Sokol Gjeçi <sgjeci@rash.al>

Table of Contents

1. Introduction

To achieve its purpose, correlating user information with network performance data, WiFiMon needs RADIUS and/or DHCP logs to be streamed in an Elasticsearch structure. For that purpose, an ELK cluster was built on VMs. A total of five VMs were used, with three of them defined as Elasticsearch master-eligible and data nodes, one VM configured as coordinating node where Kibana was installed too, and another one dedicated to Logstash.

The sources generating log files are a freeRadius and dhcp server where Filebeat was installed as an agent. Thus the data flow starts with Filebeat collecting log events and forwarding them to Logstash. At Logstash, the logs are filtered/enriched according to the needs of WiFiMon, before sending them toward Elasticsearch nodes in the cluster.

2. VMs Specifications

This setup consists of five VMs each of them having the following specifications:

- CPUs: 4
- Memory: 8 GB
- Storage: 100 GB
- Network: 1 Gbps
- Architecture: x86_64
- OS: CentOS 7

3. Cluster Setup

Setting up an ELK cluster means installing the software packages implementing its components. Some configuration must also be done as a preparation, before starting with the configuration of the cluster itself.

3.1. DNS and Roles

The following list shows the DNS configuration and the role each machine plays in the cluster.

- wifimon-node1.rash.al ↔ 10.254.24.230 → master-eligible / data node
- wifimon-node2.rash.al ↔ 10.254.24.232 → master-eligible / data node
- wifimon-node3.rash.al ↔ 10.254.24.237 → master-eligible / data node
- wifimon-kibana.rash.al ↔ 10.254.24.148 → coordinating node
- wifimon-logstash.rash.al ↔ 10.254.24.233 → pipeline node

Cluster node is considered to be the one that joins the cluster. In this setup, cluster nodes are the three master-eligible/data nodes and the coordinating node. The pipeline node is not, it doesn’t join the cluster.

3.2. Package Installation

A cluster is a collection of nodes. Being a cluster of Elasticsearch nodes, Java (at least version 8) is required, so the `java-1.8.0-openjdk` package was installed on each node.

Having the Java dependency satisfied, the next step was to install the `elasticsearch` package on each cluster node, that is not in the pipeline node. For more information see Install Elasticsearch with RPM.

On coordinating node, along with elasticsearch, the `kibana` package was installed, too. For more information see Install Kibana with RPM.

On pipeline node was installed the `logstash` package. For more information see Installing Logstash.

The `filebeat` package was installed in a dhcp server and in the freeRadius server which implements the eduroam Service Provider. For more information see Repositories for APT and YUM.

All the packages implementing the cluster's components (elasticsearch, logstash, kibana, filebeat) must be of the same version. This setup is about version `7.8.0`.

3.3. System Configuration

Each node’s hostname is set to its FQDN, according to the values shown in the VMs DNS list. This value is referenced in the configuration file of Elasticsearch.

It is recommended to disable system swapping, which can result in parts of JVM Heap or even its executable pages being swapped out to disk.

Various communications take place in a cluster, with their connections requiring specific ports being opened in the firewall. The following list represents our situation.

- wifimon-node{1,2,3}.rash.al: 9200/tcp, 9300/tcp
- wifimon-kibana.rash.al: 9200/tcp, 9300/tcp, 5601/tcp
- wifimon-logstash.rash.al: 5044/tcp

Port `9200/tcp` is used to query the cluster using the Elasticsearch REST API. Port `9300/tcp` is used for internal communication between cluster nodes. Port `5044/tcp` is where Logstash listens for beats of log events sent from Filebeat. Port `5601/tcp` is used to access Kibana platform from the browser.

3.4. SSL/TLS Certificates

The cluster communication is secured by configuring SSL/TLS. The `elasticsearch-certutil` was used to generate a CA certificate utilized for signing while generating the cluster components certificates. This utility comes with the elasticsearch installation, and in this case was used the one installed in the wifimon-kibana.rash.al node.

Create the `instances.yml` file with the following contents:

```
instances:
    - name: node1
      dns: wifimon-node1.rash.al
      ip: 10.254.24.230

    - name: node2
      dns: wifimon-node2.rash.al
      ip: 10.254.24.232

    - name: node3
      dns: wifimon-node3.rash.al
      ip: 10.254.24.237

    - name: kibana
      dns: wifimon-kibana.rash.al
      ip: 10.254.24.148

    - name: logstash
      dns: wifimon-logstash.rash.al

    - name: filebeat
```

Generate the CA certificate and key:

```
# /usr/share/elasticsearch/bin/elasticsearch-certutil ca \
--ca-dn CN='WiFiMon CA' --days 3650 --keysize 4096 \
--out $(pwd)/wifimon-ca.zip --pass --pem
```

The above will create the `wifimon-ca.zip` file in the current directory. Unzip it.

Generate components certificates and keys:

```
# /usr/share/elasticsearch/bin/elasticsearch-certutil cert \
--ca-cert $(pwd)/ca/ca.crt --ca-key $(pwd)/ca/ca.key --days 1000 \
--in $(pwd)/instances.yml --keysize 4096 --out $(pwd)/wifimon-certs.zip \
--pass --pem
```

The above will create the `wifimon-certs.zip` file in the current directory. Unzip it.

NOTE

In the commands above, the paths have been prefixed with `$(pwd)` because the version `7.8.0` of Elasticsearch, at the moment of this writing, falls back on using the `/usr/share/elasticsearch` if you don't use the full path.

The key to configure Logstash must be in the PKCS#8 format:

```
# cd logstash
# openssl pkcs8 -topk8 -in logstash.key -out logstash.pkcs8.key
# cd ..
```

At this point the current directory should have the following layout:

```
├── ca
│   ├── ca.crt
│   └── ca.key
├── filebeat
│   ├── filebeat.crt
│   └── filebeat.key
├── instances.yml
├── kibana
│   ├── kibana.crt
│   └── kibana.key
├── logstash
│   ├── logstash.crt
│   ├── logstash.key
│   └── logstash.pkcs8.key
├── node1
│   ├── node1.crt
│   └── node1.key
├── node2
│   ├── node2.crt
│   └── node2.key
├── node3
│   ├── node3.crt
│   └── node3.key
├── wifimon-ca.zip
└── wifimon-certs.zip
```

Create a directory named `certs` under each component's configuration directory, and copy there the `certificate authority` and the corresponding component's `certificate` and `key`. At the end, the certs directories on each component should look like the layouts shown below.

On wifimon-kibana.rash.al node:

```
/etc/elasticsearch/certs/
├── ca.crt
├── kibana.crt
└── kibana.key

/etc/kibana/certs/
├── ca.crt
├── kibana.crt
└── kibana.key
```

On wifimon-node1.rash.al node:

```
/etc/elasticsearch/certs/
├── ca.crt
├── node1.crt
└── node1.key
```

On wifimon-node2.rash.al node:

```
/etc/elasticsearch/certs/
├── ca.crt
├── node2.crt
└── node2.key
```

On wifimon-node3.rash.al node:

```
/etc/elasticsearch/certs/
├── ca.crt
├── node3.crt
└── node3.key
```

On wifimon-logstash.rash.al node:

```
/etc/logstash/certs/
├── ca.crt
├── logstash.crt
└── logstash.pkcs8.key
```

On freeRadius and dhcp server where filebeat is installed:

```
/etc/filebeat/certs/
├── ca.crt
├── filebeat.crt
└── filebeat.key
```

For more information on elasticsearch-certutil see its documentation page.

4. Cluster Configuration

Configuring a cluster means configuring the nodes it consists of, which in turn means defining cluster-general and node-specific settings. Elasticsearch defines these settings in configuration files located under the `/etc/elasticsearch` directory.

4.1. JVM Options

JVM options are defined in the `/etc/elasticsearch/jvm.options` file.

By default Elasticsearch tells JVM to use a `heap` of minimum and maximum of 1 GB size. The more heap available, the more memory it can use for caching, however it is recommended to use no more than 50% of the total memory.

NOTE

Tests with heap size set to 4 GB which indeed isn't more than 50% of the 8 GB the total memory, generated some out of memory exceptions, so it's better to think in terms of "*less than*" rather than "*no more than*".

On each node, configure the heap size to be 3 GB by setting the `-Xms3g` and `-Xmx3g` options. For more information see Setting the heap size.

NOTE

On a running elasticsearch instance:

If the command:

```
systemctl -l status elasticsearch.service
```

produces the following warning:

```
OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was
deprecated in version 9.0 and will likely be removed in a future
release.
```

then comment out the option:

```
-XX:+UseConcMarkSweepG
```

and set the option:

```
-XX:+UseG1GC
```

according to JEP 291.

If the file `/var/log/elasticsearch/wifimon_deprication.log` contains warnings like the following:

```
transport.publish_address was printed as [ip:port] instead of
[hostname/ip:port]. This format is deprecated and will change to
[hostname/ip:port] in a future version. Use
-Des.transport.cname_in_publish_address=true to enforce
non-deprecated formatting.
```

then proceed with the recommendation:

```
-Des.transport.cname_in_publish_address=true
```

4.2. Master-Eligible / Data Nodes

In a heavy data traffic cluster of many nodes, it is recommended to have the master-eligible and data nodes separated and dedicated to their own role. In this setup, however, there are three nodes configured as having both functionalities.

By default a node is a master-eligible, data, and ingest node, which means (a) it can be elected as master node to control the cluster, (b) it can hold data and perform operations on them, and (c) it is able to filter and enrich a data document before being indexed. Having a dedicated pipeline node with filtering/enriching capabilities there's no need for the ingest feature, it has been however enabled because it is used for monitoring purposes.

> **NOTE**
>
> Elasticsearch keystore should be configured before running this configuration.

On wifimon-node1.rash.al node:

```
# cat /etc/elasticsearch/elasticsearch.yml
cluster.name: wifimon
node.name: ${HOSTNAME}
node.master: true
node.voting_only: false
node.data: true
node.ingest: true
node.ml: false
cluster.remote.connect: false
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: wifimon-node1.rash.al
discovery.seed_hosts: [
    "wifimon-node1.rash.al",
    "wifimon-node2.rash.al",
    "wifimon-node3.rash.al"
]
#cluster.initial_master_nodes: [
#    "wifimon-node1.rash.al",
#    "wifimon-node2.rash.al",
#    "wifimon-node3.rash.al"
#]
xpack.security.enabled: true
xpack.security.http.ssl.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: full
xpack.security.http.ssl.key: /etc/elasticsearch/certs/node1.key
xpack.security.http.ssl.certificate: /etc/elasticsearch/certs/node1.crt
xpack.security.http.ssl.certificate_authorities: /etc/elasticsearch/certs/ca.crt
xpack.security.transport.ssl.key: /etc/elasticsearch/certs/node1.key
```

```
xpack.security.transport.ssl.certificate: /etc/elasticsearch/certs/node1.crt
xpack.security.transport.ssl.certificate_authorities:
/etc/elasticsearch/certs/ca.crt
xpack.monitoring.enabled: true
xpack.monitoring.collection.enabled: true
```

On wifimon-node2.rash.al node:

```
# cat /etc/elasticsearch/elasticsearch.yml
cluster.name: wifimon
node.name: ${HOSTNAME}
node.master: true
node.voting_only: false
node.data: true
node.ingest: true
node.ml: false
cluster.remote.connect: false
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: wifimon-node2.rash.al
discovery.seed_hosts: [
    "wifimon-node1.rash.al",
    "wifimon-node2.rash.al",
    "wifimon-node3.rash.al"
]
#cluster.initial_master_nodes: [
#    "wifimon-node1.rash.al",
#    "wifimon-node2.rash.al",
#    "wifimon-node3.rash.al"
#]
xpack.security.enabled: true
xpack.security.http.ssl.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: full
xpack.security.http.ssl.key: /etc/elasticsearch/certs/node2.key
xpack.security.http.ssl.certificate: /etc/elasticsearch/certs/node2.crt
xpack.security.http.ssl.certificate_authorities: /etc/elasticsearch/certs/ca.crt
xpack.security.transport.ssl.key: /etc/elasticsearch/certs/node2.key
xpack.security.transport.ssl.certificate: /etc/elasticsearch/certs/node2.crt
xpack.security.transport.ssl.certificate_authorities:
/etc/elasticsearch/certs/ca.crt
xpack.monitoring.enabled: true
xpack.monitoring.collection.enabled: true
```

On wifimon-node3.rash.al node:

```
# cat /etc/elasticsearch/elasticsearch.yml
cluster.name: wifimon
node.name: ${HOSTNAME}
node.master: true
node.voting_only: false
```

```
node.data: true
node.ingest: true
node.ml: false
cluster.remote.connect: false
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: wifimon-node3.rash.al
discovery.seed_hosts: [
    "wifimon-node1.rash.al",
    "wifimon-node2.rash.al",
    "wifimon-node3.rash.al"
]
#cluster.initial_master_nodes: [
#    "wifimon-node1.rash.al",
#    "wifimon-node2.rash.al",
#    "wifimon-node3.rash.al"
#]
xpack.security.enabled: true
xpack.security.http.ssl.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: full
xpack.security.http.ssl.key: /etc/elasticsearch/certs/node3.key
xpack.security.http.ssl.certificate: /etc/elasticsearch/certs/node3.crt
xpack.security.http.ssl.certificate_authorities: /etc/elasticsearch/certs/ca.crt
xpack.security.transport.ssl.key: /etc/elasticsearch/certs/node3.key
xpack.security.transport.ssl.certificate: /etc/elasticsearch/certs/node3.crt
xpack.security.transport.ssl.certificate_authorities:
/etc/elasticsearch/certs/ca.crt
xpack.monitoring.enabled: true
xpack.monitoring.collection.enabled: true
```

Each node has the same value for `cluster.name` which is a unique name identifying the cluster. This is how a node joins a cluster.

The `node.name` is set to the value of `${HOSTNAME}`, that is the value of the node's FQDN. This setting can also be configured explicitly to some value.

The `node.master` makes this node eligible to be elected as a master node which controls the cluster. Every master-eligible node, which is not a `voting_only` node, can be the master node of the cluster.

The `node.ml` feature is set to false. This is a machine learning feature which by default is set to true by x-pack extension, which comes installed by elastisearch package since version 6.3.

The `cluster.remote.connect` setting makes this node function as a cross-cluster client able to connect to remote clusters. This is not the case of this setup so it is set to false.

The `network.host` functions as a shortcut for the `network.bind_host` and `network.publish_host`. The former specifies the interface to listen for requests while the later is used to communicate with other nodes in the cluster.

The `discovery.seed_hosts` provides a list of nodes for this node to contact in order to join the cluster. It is safe to set this to the list of master-eligible nodes.

The `cluster.initial_master_nodes` provides a list of master-eligible nodes whose votes count in the very first election of the master node. The nodes in this list must match exactly the `node.name` of the nodes.

NOTE

The `cluster.initial_master_nodes` setting is only used when starting a new cluster for the very first time. This is known as the cluster bootstrapping. This setting should not be used when restarting or adding a new node in an existing cluster.

The `xpack.security.{http,transport}.*` settings enable the SSL/TLS encryption for the HTTP and Transport communication protocol, respectively.

The `xpack.monitoring.enabled` enables/disables the monitoring on the node, while `xpack.monitoring.collection.enabled` enable/disables the collection of monitoring data.

NOTE

It is recommended to setup a separate cluster for monitoring a production cluster. For more information see Monitoring in a production environment.

For more information about the aforementioned settings see Node, Network Settings, Important discovery and cluster formation settings, and Secure a cluster.

4.3. Coordinating Node

A coordinating node is a node that has `node.master`, `node.data`, and `node.ingest` settings set to `false`, which means you are left with a node actually behaving as a load-balancer, routing the requests on the appropriate nodes in the cluster.

A coordinating node is an Elasticsearch node which joins the cluster as every cluster node. In this setup, the coordinating node is named wifimon-kibana.rash.al because the Kibana visualization platform has been installed and configured on it.

Below is the configuration of wifimon-kibana.rash.al as an Elasticsearch coordinating node. It follows the same pattern as the master-eligible/data nodes, but with their functionalities set to false.

NOTE

Elasticsearch keystore should be configured before running this configuration.

On wifimon-kibana.rash.al node:

```
# cat /etc/elasticsearch/elasticsearch.yml
cluster.name: wifimon
node.name: ${HOSTNAME}
node.master: false
node.voting_only: false
node.data: false
node.ingest: false
node.ml: false
cluster.remote.connect: false
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: wifimon-kibana.rash.al
discovery.seed_hosts: [
    "wifimon-node1.rash.al",
    "wifimon-node2.rash.al",
    "wifimon-node3.rash.al"
]
xpack.security.enabled: true
xpack.security.http.ssl.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: full
xpack.security.http.ssl.key: /etc/elasticsearch/certs/kibana.key
xpack.security.http.ssl.certificate: /etc/elasticsearch/certs/kibana.crt
xpack.security.http.ssl.certificate_authorities: /etc/elasticsearch/certs/ca.crt
xpack.security.transport.ssl.key: /etc/elasticsearch/certs/kibana.key
xpack.security.transport.ssl.certificate: /etc/elasticsearch/certs/kibana.crt
xpack.security.transport.ssl.certificate_authorities:
/etc/elasticsearch/certs/ca.crt
xpack.monitoring.enabled: true
xpack.monitoring.collection.enabled: true
```

4.4. Setup Passwords

Elasticsearch comes with built-in users configured, each of them having a set of privileges but with their passwords not set, and consequently unable to be used for authentication.

Passwords setup requires the nodes being up and running in a healthy cluster. Start the elasticsearch instance on each cluster node, and after ensuring each instance is running properly, run the following command in wifimon-kibana.rash.al node to setup the passwords.

```
# /usr/share/elasticsearch/bin/elasticsearch-setup-passwords auto \
-u "https://wifimon-kibana.rash.al:9200"
```

The above command will randomly generate passwords for each built-in user.
Save the output!

For more information on Built-in users follow the link.

4.5. Kibana Platform

Kibana is a browser-based interface that allows for searching, viewing, and interacting with the data stored in the cluster. It's a visualization platform for creating charts, tables, and maps to represent the data. Kibana should be configured in an Elasticsearch node. The configuration of Kibana is done by editing the `/etc/kibana/kibana.yml` file.

> **NOTE**
>
> Kibana keystore should be configured before running this configuration.

On wifimon-kibana.rash.al node:

```
# cat /etc/kibana/kibana.yml
server.port: 5601
server.host: "wifimon-kibana.rash.al"
server.name: "wifimon-kibana"
elasticsearch.hosts: ["https://wifimon-kibana.rash.al:9200"]
server.ssl.enabled: true
server.ssl.certificate: /etc/kibana/certs/kibana.crt
server.ssl.key: /etc/kibana/certs/kibana.key
elasticsearch.ssl.certificateAuthorities: ["/etc/kibana/certs/ca.crt"]
elasticsearch.ssl.verificationMode: full
```

The `elasticsearch.hosts` is an array of URLs of nodes to send the queries. It is set to the `https://wifimon-kibana.rash.al:9200` which is the coordinating node.

Kibana application can be optionally configured to write log entries:

```
pid.file: /var/run/kibana/kibana.pid
logging.dest: /var/log/kibana/kibana.log
```

This needs two `kibana` directories created under the `/var/run` and `/var/log` directories:

```
# mkdir /var/run/kibana && chown kibana:kibana /var/run/kibana
# mkdir /var/log/kibana && chown kibana:kibana /var/log/kibana
```

The file `/lib/tmpfiles.d/kibana.conf` should also be created, otherwise the directory `/var/run/kibana` won't survive on system reboot – more on `tmpfiles.d(5)` man page.

```
# cat /lib/tmpfiles.d/kibana.conf
d  /run/kibana  0755  kibana  kibana  -  -
```

With logging enabled, rotation is needed. The following will perform a daily rotation of kibana logs.

```
# cat /etc/logrotate.d/kibana
/var/log/kibana/*.log {
```

```
    notifempty
    missingok
    compress
    daily
    rotate 7
    create 0644 kibana kibana
    sharedscripts
    postrotate
        /bin/kill -s SIGHUP $(cat /var/run/kibana/kibana.pid) > /dev/null 2>&1
    endscript
}
```

Start the kibana service, access the platform at `https://wifimon-kibana.rash.al:5601` and authenticate with the `elastic` superuser and its password.

For more information on Kibana configuration settings see Configuring Kibana.

5. Cluster Exploration

Even though it is possible to explore the cluster by using the Kibana platform, this section is about querying the cluster by using the REST API provided by Elasticsearch. The querying commands are executed in wifimon-kibana.rash.al node and the user `elastic` is used for authentication.

Display nodes:

```
# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt \
--user elastic 'https://wifimon-kibana.rash.al:9200/_cat/nodes?v'
```

Each node is represented by a row consisting of node's IP, heap and memory % usage, average loads as in uptime command output, the roles (m)aster, (d)ata, (i)ngest, which node is elected (*) as master, and node's name.

Display master node:

```
# curl --cacert /etc/elasticsearch/certs/ca.crt \
--user elastic -XGET 'https://wifimon-kibana.rash.al:9200/_cat/master?v'
```

Display health:

```
# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt \
--user elastic 'https://wifimon-kibana.rash.al:9200/_cat/health?v'
```

Our cluster is of green status, but this will change to yellow after stopping the elasticsearch instance in the master node, which was intentionally chosen in order to see the election of the new master.

The output of health query shows useful information about the replica-shards, primary-shards, etc. These values change according to the cluster's state. The replicas are stored in different nodes from them of primary shards, thus providing the ability for a fail-over mechanism.

On wifimon-node1.rash.al (the master node) run `systemctl stop elasticsearch.service` to stop the elasticsearch instance.

Querying the cluster again from wifimon-kibana.rash.al node shows that the wifimon-node1.rash.al has gone and the wifimon-node3.rash.al has been elected as the new master. The cluster status is now yellow.

Start the elasticsearch instance on wifimon-node1.rash.al node and query the cluster again. The wifimon-node1.rash.al will join the cluster and the status of the cluster will become green, while wifimon-node3.rash.al continues to be the master node.

6. Filebeat Configuration

Filebeat monitors log files for new content, collect log events, and forwards them to Elasticsearch, either directly or via Logstash. In Filebeat terms one speaks about a) the input which looks in the configured log data locations, b) the harvester which reads a single log for new content and sends new log data to libbeat, and c) the output which aggregates and sends data to the configured output. For more information see Filebeat overview.

The configuration of Filebeat is done by editing the `/etc/filebeat/filebeat.yml` file. Tests will be performed by triggering it manually with sample logs, in order to see how it works and what the results look like. Filebeat will be firstly configured to dump the output in a file, and then will have it forwarding the data toward Logstash.

During the tests, the same samples of logs will be used. For this to be possible, the registry Filebeat uses to store tracking information of last reading, must be deleted. It is recommended to stop the filebeat service before removing the registry. The following little script makes it easier while experimenting with different settings.

```
# cat test_filebeat.sh
#!/bin/bash
systemctl stop filebeat.service && \
rm -rf /var/lib/filebeat/registry/filebeat && \
rm -f /tmp/sample_logs_output.json && \
systemctl start filebeat.service
exit 0
```

Below are the sample log files to be used in tests. It's about a log event when a user interact with the eduroam Service Provider and another one while interacting with the dhcp server.

```
# cat /tmp/radius_sample_logs
Sun Mar 10 08:16:05 2019
        Service-Type = Framed-User
        NAS-Port-Id = "wlan2"
        NAS-Port-Type = Wireless-802.11
        User-Name = "sgjeci@rash.al"
        Acct-Session-Id = "82c000cd"
        Acct-Multi-Session-Id = "CC-2D-E0-9A-EB-A3-88-75-98-6C-31-AA-82-C0-00-00-
00-00-00-CD"
        Calling-Station-Id = "88-75-98-6C-31-AA"
        Called-Station-Id = "CC-2D-E0-9A-EB-A3:eduroam"
        Acct-Authentic = RADIUS
        Acct-Status-Type = Start
        NAS-Identifier = "Eduroam"
        Acct-Delay-Time = 0
        NAS-IP-Address = 192.168.192.111
        Event-Timestamp = "Mar 8 2019 08:16:05 CET"
        Tmp-String-9 = "ai:"
        Acct-Unique-Session-Id = "e5450a4e16d951436a7c241eaf788f9b"
        Realm = "rash.al"
        Timestamp = 1552029365

# cat /tmp/dhcp_sample_logs
Jun 18 19:15:20 centos dhcpd[11223]: DHCPREQUEST for 192.168.1.200 from
a4:c4:94:cd:35:70 (galliumos) via wlp6s0
Jun 18 19:15:20 centos dhcpd[11223]: DHCPACK on 192.168.1.200 to a4:c4:94:cd:35:70
(galliumos) via wlp6s0
```

6.1. File Output

As mentioned above, Filebeat will be firstly configured to dump the output in a file. Below is shown the configuration file of Filebeat for each agent. It configures an `input` of type `log`, which is needed to read lines from log files. There's also the `output` which configures the `path` and the `filename` to dump the data in, and finally the section of `processors` to drop some fields Filebeat adds by default, and to add the `logtype` field used in the Logstash beats-pipeline output.

6.1.1. RADIUS Server

The following is the Filebeat configuration on the radius server, which dumps the data in the `/tmp/sample_logs_output.json` file.

```
# cat /etc/filebeat/filebeat.yml
filebeat.inputs:
- type: log
  enabled: true
  paths: /tmp/radius_sample_logs
  multiline.pattern: '^[[:space:]]'
  multiline.negate: false
  multiline.match: after
```

```
output.file:
  path: /tmp
  filename: sample_logs_output.json
processors:
- add_fields:
    target: ''
    fields:
      logtype: radius
- drop_fields:
    fields: ['input', 'host', 'agent', 'acs', 'log', 'ecs']
```

The important settings here are the `multiline.*` ones which manage multiline formatted logs. The `.pattern` matches lines starting with white-space. The `.negate` and `.match` work together, and combined as `false` and `after` make consecutive lines that match the pattern to be appended to the previous line that doesn't match it. This makes all the lines starting with whites-pace to be appended to the line that hold the date, actually the first line in the radius_sample_logs. For more information see Manage multiline messages.

After executing `test_filebeat.sh` as root user the following output is generated:

```
# cat /tmp/sample_logs_output.json
{"@timestamp":"2020-06-28T13:07:37.183Z","@metadata":
{"beat":"filebeat","type":"_doc","version":"7.8.0"},"logtype":"radius","message":"S
un Mar 10 08:16:05 2019\n\tService-Type = Framed-User\n\tNAS-Port-Id = \"wlan2\"\n\
tNAS-Port-Type = Wireless-802.11\n\tUser-Name = \"sgjeci@rash.al\"\n\tAcct-Session-
Id = \"82c000cd\"\n\tAcct-Multi-Session-Id = \"CC-2D-E0-9A-EB-A3-88-75-98-6C-31-AA-
82-C0-00-00-00-00-00-CD\"\n\tCalling-Station-Id = \"88-75-98-6C-31-AA\"\n\tCalled-
Station-Id = \"CC-2D-E0-9A-EB-A3:eduroam\"\n\tAcct-Authentic = RADIUS\n\tAcct-
Status-Type = Start\n\tNAS-Identifier = \"Eduroam\"\n\tAcct-Delay-Time = 0\n\tNAS-
IP-Address = 192.168.0.22\n\tEvent-Timestamp = \"Mar 8 2019 08:16:05 CET\"\n\tTmp-
String-9 = \"ai:\"\n\tAcct-Unique-Session-Id
= \"e5450a4e16d951436a7c241eaf788f9b\"\n\tRealm = \"rash.al\"\n\tTimestamp =
1552029365"}
```

The logs are located in the `message` field. The `logtype` field holds the `radius` value, thus differentiating these events from the dhcp ones when arriving at Logstash pipeline.

6.1.2. DHCP Server

The following is the Filebeat configuration on the dhcp server, which dumps the data in the `/tmp/sample_logs_output.json` file.

```
# cat /etc/filebeat/filebeat.yml
filebeat.inputs:
- type: log
  enabled: true
  paths: /tmp/dhcp_sample_logs
  include_lines: ['DHCPACK']
output.file:
```

```
  path: /tmp
  filename: sample_logs_output.json
processors:
- add_fields:
    target: ''
    fields:
      logtype: dhcp
- drop_fields:
    fields: ['input', 'host', 'agent', 'acs', 'log', 'ecs']
```

The lines to include from dhcp logs are the ones containing `DHCPACK` string, which represent the final phase of dhcp operations. These lines are filtered with the `include_lines` setting.

After executing `test_filebeat.sh` as root user the following output is generated:

```
# cat /tmp/sample_logs_output.json
{"@timestamp":"2020-06-28T09:20:17.834Z","@metadata":
{"beat":"filebeat","type":"_doc","version":"7.8.0"},"message":"Jun 18 19:15:20
centos dhcpd[11223]: DHCPACK on 192.168.1.200 to a4:c4:94:cd:35:70 (galliumos) via
wlp6s0","logtype":"dhcp"}
```

The `logtype` field contains the `dhcp` value, thus differentiating these events from the radius ones, when arriving at Logstash pipeline.

6.2. Filtering Log Events

Apart from adding or dropping named fields, processors can also be used to filter log events when certain criteria are met. For example, to send out only the log events containing the value `Eduroam` in the `NAS-Identifyer` field, the following configuration could be applied.

```
processors:
  - drop_event:
      when:
        not:
          regexp:
            message: '.*NAS-Identifier.*=.*Eduroam.*'
```

For more information on configuring processors see Filter and enhance the exported data.

6.3. Logstash Output

This section shows how to configure Filebeat's logstash output to feed the pipeline node.

> **NOTE**
>
> Filebeat keystore should be configured before running this configuration.

```
output.logstash:
  hosts: ["wifimon-logstash.rash.al:5044"]
  ssl.certificate_authorities: ["/etc/filebeat/certs/ca.crt"]
  ssl.certificate: "/etc/filebeat/certs/filebeat.crt"
  ssl.key: "/etc/filebeat/certs/filebeat.key"
  ssl.key_passphrase: "${key_passphrase}"
```

The `hosts` setting specifies node and port where Logstash service listens for incoming log events. The `${key_passphrase}` references the passphrase of filebeat.key stored in Filebeat keystore -- it's about mutual SSL identification, the client (filebeat) is forced to provide a certificate to the server (logstash) for the connection to be established.

For this configuration to work, the Elasticsearch index template must be manually loaded. Template autoloading is only supported for the elasticsearch output. Replace `elastic-password-goes-here` with the proper password and run:

```
# set +o history
# filebeat setup --index-management \
-E output.logstash.enabled=false \
-E 'output.elasticsearch.hosts=["wifimon-kibana.rash.al:9200"]' \
-E output.elasticsearch.protocol=https \
-E output.elasticsearch.username=elastic \
-E output.elasticsearch.password=elastic-password-goes-here \
-E 'output.elasticsearch.ssl.certificate_authorities=["/etc/filebeat/certs/
ca.crt"]'
# set -o history
```

The above command loads the template from wifimon-kibana.rash.al node where elasticsearch is installed. Detailed information is written in the Filebeat log file.

6.4. Monitoring

The Kibana platform allows for monitoring the health of Filebeat service. For this to happen, the following configuration must be added in the `/etc/filebeat/filebeat.yml` file.

> **NOTE**
>
> Filebeat keystore should be configured before running this configuration.

```
monitoring.enabled: true
monitoring.cluster_uuid: "cluster-id-goes-here"
monitoring.elasticsearch.ssl.certificate_authorities:
["/etc/filebeat/certs/ca.crt"]
monitoring.elasticsearch.ssl.certificate: "/etc/filebeat/certs/filebeat.crt"
monitoring.elasticsearch.ssl.key: "/etc/filebeat/certs/filebeat.key"
monitoring.elasticsearch.ssl.key_passphrase: "${key_passphrase}"
```

```
monitoring.elasticsearch.hosts: ["https://wifimon-kibana.rash.al:9200"]
monitoring.elasticsearch.username: beats_system
monitoring.elasticsearch.password: "${beats_system_password}"
```

The value of `monitoring.cluster_uuid` must be provided. To get it run:

```
# curl --cacert /etc/elasticsearch/certs/ca.crt --user elastic \
-XGET 'https://wifimon-kibana.rash.al:9200/_cluster/state/all?pretty'
```

The `${beats_system_password}` references the password of the beats_system built-in user which is stored in Filebeat keystore.

7. Logstash Configuration

Logstash is a data collection engine with real-time pipelining capabilities. A Logstash pipeline consists of three elements, `input`, `filter`, and `output`. The input plugins consume data coming from a source, the filter plugins modify the data as specified, and the output plugins send data to a defined destination. In this setup data comes from Filebeat agents, with their logstash output configured to feed the Logstash instance on port 5044/tcp.

> **NOTE**
>
> Logstash keystore should be configured before running the configurations provided here.

7.1. JVM Options

The JVM Options for Logstash are defined in the `/etc/logstash/jvm.options` file. The configuration is the same as the one configuring the JVM Options of Elasticsearch.

7.2. Logstash Settings

Logstash settings are defined in the `/etc/logstash/logstash.yml` file, which contains the following:

```
path.data: /var/lib/logstash
path.logs: /var/log/logstash
queue.type: persisted
xpack.monitoring.enabled: true
xpack.monitoring.elasticsearch.username: "logstash_system"
xpack.monitoring.elasticsearch.password: "${logstash_system_password}"
xpack.monitoring.elasticsearch.hosts: "https://wifimon-kibana.rash.al:9200"
xpack.monitoring.elasticsearch.ssl.certificate_authority:
"/etc/logstash/certs/ca.crt"
xpack.monitoring.elasticsearch.ssl.verification_mode: certificate
xpack.monitoring.elasticsearch.sniffing: true
```

The `path.data` and `path.logs` defines the directories logstash will use to write persistent data and log messages, respectively.

The `queue.type` set the queue to `persisted` to provide protection against data loss by using an on-disk queue. For more information see Persistent Queues.

The other settings configures Logstash to send monitoring data over SSL/TLS.

7.3. Logstash Pipelines

Logstash pipelines are defined in the `/etc/logstash/pipelines.yml` file, which contains:

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

- pipeline.id: radius-pipeline
  path.config: "/etc/logstash/conf.d/radius-pipeline.conf"

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

For each pipeline, an id and the configuration file is defined. The beats-pipeline functions as a gate receiving logs from both (radius, dhcp) streams and then feeds their pipelines, respectively.

7.3.1. Beats Pipeline

As already mentioned, the beats-pipeline acts as receiver / forwarder of log-events coming from radius and dhcp streams. It doesn't configure any filter element, but the input and output ones.

```
# cat /etc/logstash/conf.d/beats-pipeline.conf
input {
    beats {
        port => 5044
        ssl => true
        ssl_certificate_authorities => ["/etc/logstash/certs/ca.crt"]
        ssl_certificate => "/etc/logstash/certs/logstash.crt"
        ssl_key => "/etc/logstash/certs/logstash.pkcs8.key"
        ssl_key_passphrase => "${pkcs8_key_passphrase}"
        ssl_verify_mode => "force_peer"
    }
}

output {
    if ([logtype] == "radius") {
        pipeline { send_to => radiuslogs }
    } else { # logtype is dhcp
        pipeline { send_to => dhcplogs }
    }
}
```

The `beats` plugin configures Logstash to listen on port `5044`. It also provides settings for SSL/TLS encryption and forces the peer (filebeat) to provide a certificate for identification.
The output defines which pipeline to forward the data to, based on the value of `logtype` field sent from filebeat agent.

7.3.2. RADIUS Pipeline

The radius-pipeline is configured in the `/etc/logstash/conf.d/radius-pipeline.conf` file. It receives radius log-events sent from the beats-pipeline.

```
# cat /etc/logstash/conf.d/radius-pipeline.conf
input {
    pipeline { address => radiuslogs }
}

filter {
    mutate { gsub => [ "message", "[\n\t]+", " " ] }

    kv {
        allow_duplicate_values => false

        include_keys => [
            "Calling-Station-Id",
            "Framed-IP-Address",
            "Timestamp",
            "Called-Station-Id",
            "NAS-IP-Address",
            "Acct-Status-Type"
        ]

        remove_field => [
            "logtype",
            "message",
            "@version"
        ]
    }

    if "beats_input_codec_plain_applied" in [tags] {
        mutate { remove_tag => ["beats_input_codec_plain_applied"] }
    }

    geoip { source => "NAS-IP-Address" }

    fingerprint {
        key => "${cipher_key}"
        method => "SHA512"
        source => "Calling-Station-Id"
        target => "Calling-Station-Id"
    }
```

```
    fingerprint {
        key => "${cipher_key}"
        method => "SHA512"
        source => "Framed-IP-Address"
        target => "Framed-IP-Address"
    }
}

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

The `filter` element defines the filters `mutate`, `kv`, `geoip`, and `fingerprint`. The gsub (global substitute) mutation is used to replace `\n\t` with space. The kv (key value) filter automatically parses messages formatted on an `option=value` pattern. A list of fields needed for the correlation is included, and then no needed fields are removed from the log event. The NAS-IP-Address field disposable from the kv plugin, is passed to the `source` of geoip filter to gather geographical information for that IP. Finally the fingerprint plugin hash-es the Calling-Station-Id (user's mac-address) and the Framed-IP-Address (user's IP address) if available.

The `output` defines the `stdout` plugin which dumps the filtered data in the standard output, allowing for testing a data flow of Filebeat → Logstash → Logstash_STDOUT.

7.3.3. DHCP Pipeline

The dhcp-pipeline is configured in the `/etc/logstash/conf.d/dhcp-pipeline.conf` file. It receives dhcp log-events sent from the beats-pipeline.

```
# cat /etc/logstash/conf.d/dhcp-pipeline.conf
input {
    pipeline { address => dhcplogs }
}

filter {
    dissect {
        mapping => {
            "message" => "%{} DHCPACK on %{ip} to %{mac} %{}"
        }

        remove_field => [
            "logtype",
            "message",
            "@version"
        ]
    }

    if "beats_input_codec_plain_applied" in [tags] {
        mutate { remove_tag => ["beats_input_codec_plain_applied"] }
    }
```

```
    fingerprint {
        key => "${cipher_key}"
        method => "SHA512"
        source => "ip"
        target => "ip"
    }

    fingerprint {
        key => "${cipher_key}"
        method => "SHA512"
        source => "mac"
        target => "mac"
    }
}

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

The `filter` element defines the filters `dissect` and `fingerprint`. The dissect parses the DHCPACK entries and create the fields map and ip populated with the matched values. The fingerprint plugin hash-es these values, for them to be securely stored in the cluster.

The `output` defines the `stdout` plugin which dumps the filtered data in the standard output, allowing for testing a data flow of Filebeat → Logstash → Logstash_STDOUT.

7.4. Streaming to STDOUT

Having Filebeat agents configured to feed Logstash, whose pipelines are configured to dump data to STDOUT, makes it possible to test a data flowing through Filebeat → Logstash → Logstash_STDOUT.

On wifimon-logstash.rash.al start the logstash service:

```
# systemctl start logstash.service
```

Set the journal to follow recently appended entries for logstash:

```
# journalctl --follow --unit logstash.service
```

On radius server run the `test_filebeat.sh` script as root user.

On wifimon-logstash.rash.al terminal should be shown something like:

```
{
"Called-Station-Id" => "CC-2D-E0-9A-EB-A3:eduroam",
"Acct-Status-Type" => "Start",
"NAS-IP-Address" => "162.13.218.132",
"@timestamp" => 2019-12-10T17:35:38.054Z,
```

```
"Calling-Station-Id" =>
"UFWjPNUDSNkBYirsfcaZlkPrY0UOddLORId8boq59FTAhE3fM8xyV2uShOIf5y8W",
"Timestamp" => "1552029365",
"geoip" => {
"country_code3" => "GB",
"ip" => "162.13.218.132",
"timezone" => "Europe/London",
"country_code2" => "GB",
"continent_code" => "EU",
"latitude" => 51.4964,
"country_name" => "United Kingdom",
"location" => {
"lat" => 51.4964,
"lon" => -0.1224
},
"longitude" => -0.1224
},
"tags" => []
}
```

On dhcp server run the `test_filebeat.sh` script as root user.

On wifimon-logstash.rash.al terminal should be shown something like:

```
{
"mac" =>
"8db8b992e5a9686e0113b1f885ff485e274d3824847a11c6a371ad873eea2959198199068472f84dc8
9a9489380b6cd8ff02cb97c32dfb849c43a8ed86898b76",
"@timestamp" => 2020-06-28T09:46:36.638Z,
"tags" => [],
"ip" =>
"a5b40b78fb8b1062ba2464f2d5d15e05bde353beae313d67a6caabf7d219f7905377f706b13f5bc863
20e6784b97bcad25a90d120bb64137d605a67313b2c415"
 }
```

The outputs verify the tests were successful, the fields of interest are populated with their values, with some of them being hash-ed. The traffic Filebeat → Logstash was sent over SSL/TLS.

You may have noticed in the output of radius-pipeline that the value of NAS-IP-Address have been changed from private IP to 162.13.218.132 (www.geant.org). This was done intentionally in order to see the results of geoip filter, which gives nothing for private Ips.

8. Streaming Logs Into Cluster

Until now the streaming of data has been triggered manually by using the sample data. This allowed for testing the configuration of Filebeat and Logstash, and also having a first view of results.

This section is about configuring the components to use real data and implement a streaming through the path Filebeat → Logstash → Elasticsearch.

8.1. Filebeat Inputs

In the `/etc/filebeat/filebeat.yml` file under the `filebeat.inputs`, the `paths` should now point to the full path in the filesystem where the RADIUS or the DHCP logs are located.

`paths: /tmp/radius_sample_logs`
and
`paths: /tmp/dhcp_sample_logs`

become:

`paths: /path/to/your/radius/logs`
and
`paths: /path/to/your/dhcp/logs`

respectively. Multiple files can be given to `paths` setting as a list or as a glob-based pattern.

8.2. Create User and Role

In order to send log events to the cluster, the user `logstash_writer` with the role `logstash_writer_role` must be created. The role assigns the cluster permissions of `monitor` and `manage_index_templates` and privileges of `write` and `create_index` for `radiuslogs` and `dhcplogs` indices. Granted with these permissions the logstash_writer user is able to write data into the index.

To create the role `logstash_writer_role` run:

```
# curl -X POST --cacert /etc/elasticsearch/certs/ca.crt --user elastic \
'https://wifimon-kibana.rash.al:9200/_security/role/logstash_writer_role?pretty' \
-H 'Content-Type: application/json' -d'
{
    "cluster": [
      "monitor",
      "manage_index_templates"
    ],
    "indices": [
      {
        "names": [
          "radiuslogs",
          “dhcplogs”
        ],
        "privileges": [
          "write",
          "create_index"
        ],
        "field_security": {
```

```
          "grant": [
            "*"
          ]
        }
      }
    ],
    "run_as": [],
    "metadata": {},
    "transient_metadata": {
      "enabled": true
    }
}
'
```

To create the user `logstash_user` replace `some-password-goes-here` and run:

```
# set +o history
# curl -X POST --cacert /etc/elasticsearch/certs/ca.crt --user elastic \
'https://wifimon-kibana.rash.al:9200/_security/user/logstash_writer?pretty' \
-H 'Content-Type: application/json' -d'
{
  "username": "logstash_writer",
  "roles": ["logstash_writer_role"],
  "full_name": null,
  "email": null,
  "password": "some-password-goes-here",
  "enabled": true
}
'
# set -o history
```

8.3. Logstash Output

On radius-pipeline and dhcp-pipeline configuration files, the output should be configured to send data to Elasticsearch cluster. This is done by configuring the Logstash output elasticsearch plugin.

On radius-pipeline, the output becomes:

```
output {
    elasticsearch {
        ssl => true
        ssl_certificate_verification => true
        cacert => "/etc/logstash/certs/ca.crt"
        user => "logstash_writer"
        password => "${logstash_writer_password}"
        hosts => ["https://wifimon-kibana.rash.al"]
        index => "radiuslogs"
    }
}
```

On dhcp-pipeline, the output becomes:

```
output {
    elasticsearch {
        ssl => true
        ssl_certificate_verification => true
        cacert => "/etc/logstash/certs/ca.crt"
        user => "logstash_writer"
        password => "${logstash_writer_password}"
        hosts => ["https://wifimon-kibana.rash.al"]
        index => "dhcplogs"
    }
}
```

Logstash is now able to send the data over SSL/TLS toward the coordinating node. The logs will be stored in `radiuslogs` and `dhcplogs` indices, respectively.

After restarting the components, query the cluster to get information about indices:

```
# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt --user elastic \
'https://wifimon-kibana.rash.al:9200/_cat/indices/radiuslogs?v'

# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt --user elastic \
'https://wifimon-kibana.rash.al:9200/_cat/indices/dhcplogs?v'
```

9. ILM Configuration

The intention of WiFiMon is not to keep the RADIUS logs forever, they are only needed for a limited period of time. New log events keep coming so, after that time period has passed, the old logs should be deleted.

Logs are stored in the radiuslogs and dhcplogs indices. The index lifecycle management is achieved by creating and applying ILM policies, which can trigger actions upon indexes based on certain criteria. More information about ILM can be found at ILM Overview page.

9.1. Create Policy

This setup is about deleting the index when it’s one day old. Run the following command in the wifimon-kibana.rash.al node to create the `wifimon_policy` policy.

```
# curl -X PUT --cacert /etc/elasticsearch/certs/ca.crt --user elastic
"https://wifimon-kibana.rash.al:9200/_ilm/policy/wifimon_policy?pretty" -H
'Content-Type: application/json' -d'
{
    "policy": {
        "phases": {
            "delete": {
```

```
                "min_age": "1d",
                "actions": {
                    "delete": {}
                }
            }
        }
    }
}
'
```

Verify the policy was created:

```
# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt --user elastic
"https://wifimon-kibana.rash.al:9200/_ilm/policy/wifimon_policy?pretty"
```

9.2. Apply Policy

The policy must be associated with the indexes upon which it will trigger the configured actions. For this to happen the policy must be configured in the index template used to create the index.

On wifimon-kibana.rash.al node run the following command to apply the `wifimon_policy` to the `wifimon_template` index template matching the `radiuslogs` and `dhcplogs` indices.

```
# curl -X PUT --cacert /etc/elasticsearch/certs/ca.crt --user elastic
"https://wifimon-kibana.rash.al:9200/_template/wifimon_template?pretty" -H
'Content-Type: application/json' -d'
{
    "index_patterns": ["radiuslogs", “dhcplogs”],
    "settings": {"index.lifecycle.name": "wifimon_policy"}
}
'
```

Verify the policy was applied to the template:

```
# curl -XGET --cacert /etc/elasticsearch/certs/ca.crt --user elastic
"https://wifimon-kibana.rash.al:9200/_template/wifimon_template?pretty"
```

9.3. Logstash Output

The Logstash elasticsearch output plugin provides settings to control the Index Lifecycle Management. Include the ILM settings on radius-pipeline and dhcp-pipeline configuration files, so that the elasticsearch output plugin becomes:

On radius-pipeline:

```
output {
    elasticsearch {
        ssl => true
        cacert => "/etc/logstash/certs/ca.crt"
        ssl_certificate_verification => true
        user => "logstash_writer"
        password => "${logstash_writer_password}"
        hosts => ["https://wifimon-kibana.rash.al"]
        ilm_enabled => true
        ilm_policy => "wifimon_policy"
        index => "radiuslogs"
    }
}
```

On dhcp-pipeline:

```
output {
    elasticsearch {
        ssl => true
        cacert => "/etc/logstash/certs/ca.crt"
        ssl_certificate_verification => true
        user => "logstash_writer"
        password => "${logstash_writer_password}"
        hosts => ["https://wifimon-kibana.rash.al"]
        ilm_enabled => true
        ilm_policy => "wifimon_policy"
        index => "dhcplogs"
    }
}
```

Restart the logstash service to apply the new settings.

10. Keystores

In order not to have sensitive information hardcoded in the configuration files and just protecting that information with filesystem permissions, it is recommended to make use of keystores provided by the Elasticsearch components.

10.1. Elasticsearch

To configure Elasticsearch keystore run the following commands on each cluster node.

Create keystore:

```
# /usr/share/elasticsearch/bin/elasticsearch-keystore create
```

Add certificate key passphrase for HTTP communication protocol:

```
# /usr/share/elasticsearch/bin/elasticsearch-keystore add \
xpack.security.http.ssl.secure_key_passphrase
```

Add certificate key passphrase for Transport communication protocol:

```
# /usr/share/elasticsearch/bin/elasticsearch-keystore add \
xpack.security.transport.ssl.secure_key_passphrase
```

Verify:

```
# /usr/share/elasticsearch/bin/elasticsearch-keystore list
keystore.seed
xpack.security.http.ssl.secure_key_passphrase
xpack.security.transport.ssl.secure_key_passphrase
```

10.2. Kibana

To configure Kibana keystore run the following commands on wifimon-kibana.rash.al node.

Create keystore:

```
# sudo -u kibana /usr/share/kibana/bin/kibana-keystore create
```

Add server.ssl.keyPassphrase:

```
# sudo -u kibana /usr/share/kibana/bin/kibana-keystore add server.ssl.keyPassphrase
```

Add elasticsearch.username:

```
# sudo -u kibana /usr/share/kibana/bin/kibana-keystore add elasticsearch.username
```

Enter `kibana` as username.

Add elasticsearch.password:

```
# sudo -u kibana /usr/share/kibana/bin/kibana-keystore add elasticsearch.password
```

Enter the password generated for kibana built-in user.

Verify:

```
sudo -u kibana /usr/share/kibana/bin/kibana-keystore list
server.ssl.keyPassphrase
elasticsearch.username
elasticsearch.password
```

10.3. Logstash

To configure Logstash keystore run the following commands on wifimon-logstash.rash.al node.

For more security, protect the Logstash keystore with a password stored in the environment variable `LOGSTASH_KEYSTORE_PASS`. This variable must be available to the running logstash instance, otherwise the keystore cannot be accessed.

The LOGSTASH_KEYSTORE_PASS variable is sourced from the `/etc/sysconfig/logstash` file. Create it to hold the following contents:

```
LOGSTASH_KEYSTORE_PASS=yourLogstashKestorePassword
```

Set file owners and permissions as:

```
-rw------- 1 root root 44 Oct 15 09:54 /etc/sysconfig/logstash
```

Export the variable:

```
# export $(cat /etc/sysconfig/logstash)
```

Create keystore:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ create
```

Add fingerprint_key:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ \
add fingerprint_key
```

Add logstash_system password:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ \
add logstash_system_password
```

Add logstash_writer_password:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ \
add logstash_writer_password
```

Add pkcs8_key_passphrase:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ \
add pkcs8_key_passphrase
```

Verify:

```
# /usr/share/logstash/bin/logstash-keystore --path.settings /etc/logstash/ list
fingerprint_key
logstash_system_password
logstash_writer_password
pkcs8_key_passphrase
```

10.4. Filebeat

To configure Filebeat keystore run the following commands on the freeRadius server where Filebeat is installed.

Create keystore:

```
# filebeat keystore create
```

Add key_passphrase:

```
# filebeat keystore add key_passphrase
```

Enter the passphrase for filebeat.key

Add beats_system_password:

```
# filebeat keystore add beats_system_password
```

Enter the password of your beats_system built-in user.

Verify:

```
# filebeat keystore list
beats_system_password
key_passphrase
```

11. References

The following links were very useful while writing this material and performing the tests mentioned in it.

Elasticsearch Reference - https://www.elastic.co/guide/en/elasticsearch/reference/7.8/index.html
Logstash Reference - https://www.elastic.co/guide/en/logstash/7.8/index.html
Filebeat Reference - https://www.elastic.co/guide/en/beats/filebeat/7.8/index.html
Kibana Guide - https://www.elastic.co/guide/en/kibana/7.8/index.html
Elastic Blog - https://www.elastic.co/blog/