Logstash: Data Processing Unit

Introduction

Logstash is an open-source data processing engine that allows you to collect, enrich, and transform data from different sources. It is often used alongside other tools in the Elastic Stack, such as Elasticsearch and Kibana, to create a complete data processing and visualization pipeline. In this room, we will explore Logstash in-depth and how data from different sources can be ingested, parsed, normalized, and sent to various choice outputs.

Learning Objective

Some of the learning objectives this room will cover are:

- Overview of ELK stack
- Installing and configuring ELK stack
- Explore various Input, filter, and output plugins for Logstash
- How to use Grok plugin to parse and normalize unstructured data

Elasticsearch: Installation and Configuration

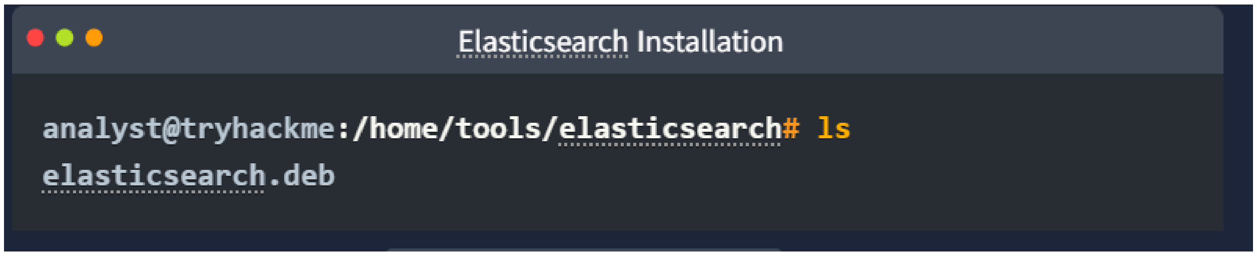
Elasticsearch

Elasticsearch is a distributed, open-source search and analytics engine that allows you to store, search, and analyze large volumes of data in real-time. It is built on top of Apache Lucene and provides a scalable solution for full-text search, structured querying, and data analysis.

In this room, we will use Elasticsearch to store data after being filtered/normalized by Logstash. It's important to know how to install and configure Elasticsearch.

Installing Elasticsearch

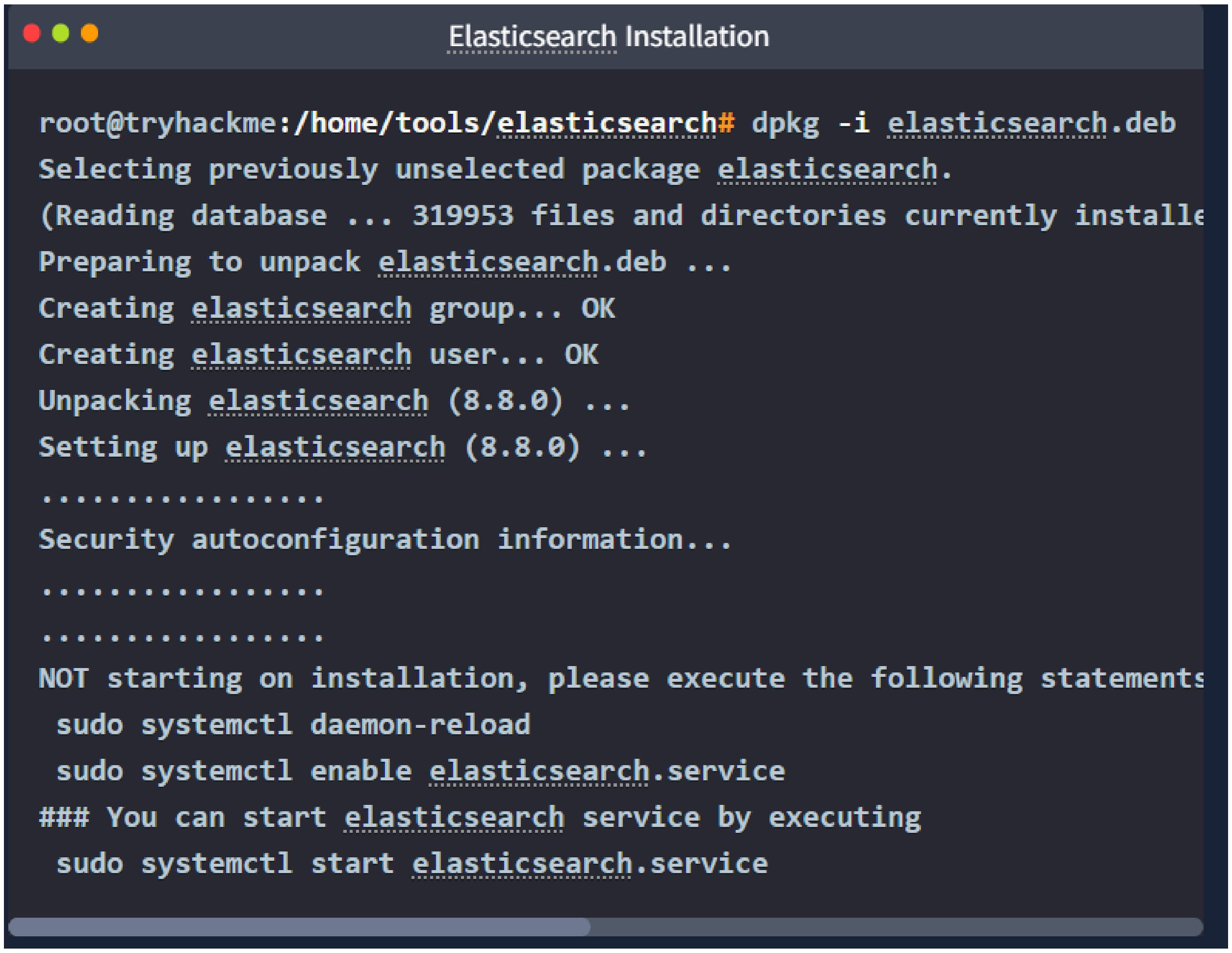
Let's go through the process of installing Elasticsearch. The latest installation instance is placed on the path /home/tools/elasticsearch.



```
analyst@tryhackme:/home/tools/elasticsearch# ls
elasticsearch.deb
```

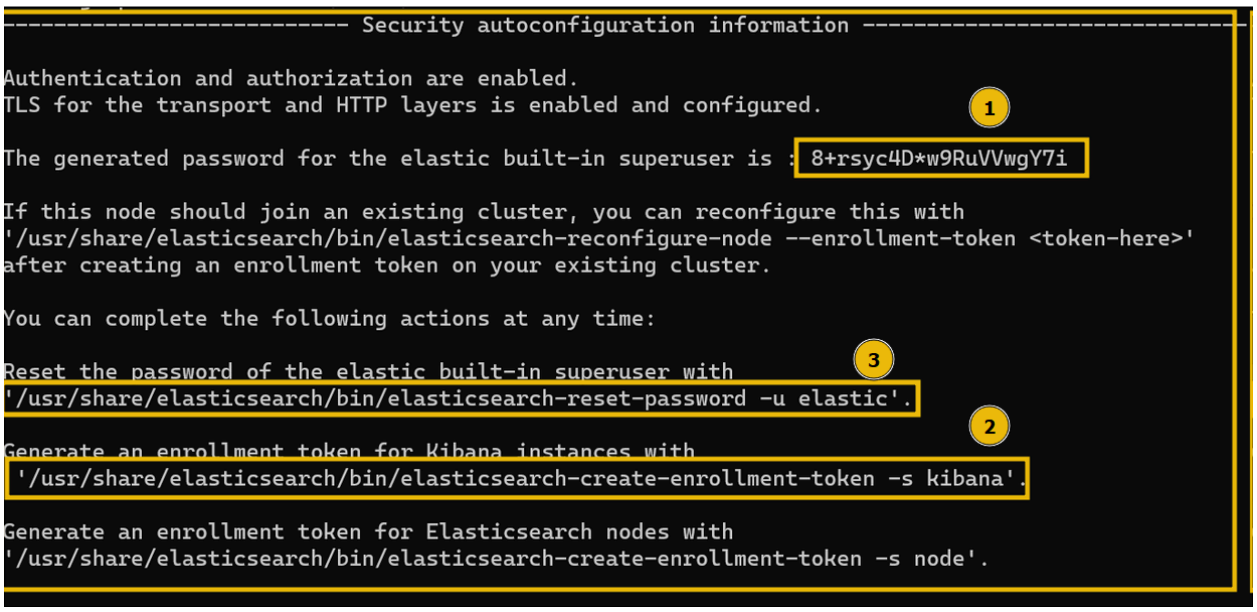
Run the following command *dpkg -i elasticsearch.deb* as a root user to install Elasticsearch on the lab. It will take 1-2 minutes to get installed, as shown below:

Note: Make sure to change the user to root using the command sudo su



```
root@tryhackme:/home/tools/elasticsearch# dpkg -i elasticsearch.deb
Selecting previously unselected package elasticsearch.
(Reading database ... 319953 files and directories currently installe
Preparing to unpack elasticsearch.deb ...
Creating elasticsearch group... OK
Creating elasticsearch user... OK
Unpacking elasticsearch (8.8.0) ...
Setting up elasticsearch (8.8.0) ...
.................
Security autoconfiguration information...
.................
.................
NOT starting on installation, please execute the following statements
 sudo systemctl daemon-reload
 sudo systemctl enable elasticsearch.service
### You can start elasticsearch service by executing
 sudo systemctl start elasticsearch.service
```

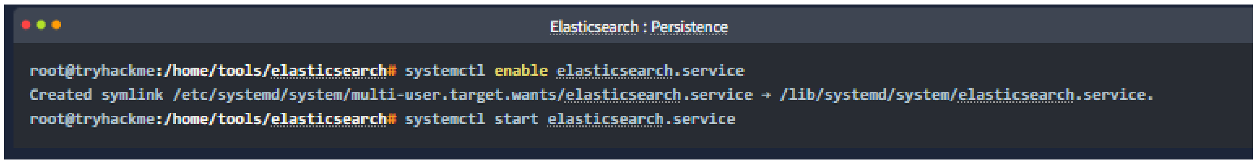
When installing for the first time, security configuration will be displayed, including the default password for the elastic user; note this down as it will be used later.



```
--------------------------- Security autoconfiguration information ------------------------------

Authentication and authorization are enabled.
TLS for the transport and HTTP layers is enabled and configured.

The generated password for the elastic built-in superuser is : 8+rsyc4D*w9RuVVwgY7i

If this node should join an existing cluster, you can reconfigure this with
'/usr/share/elasticsearch/bin/elasticsearch-reconfigure-node --enrollment-token <token-here>'
after creating an enrollment token on your existing cluster.

You can complete the following actions at any time:

Reset the password of the elastic built-in superuser with
'/usr/share/elasticsearch/bin/elasticsearch-reset-password -u elastic'.

Generate an enrollment token for Kibana instances with
 '/usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s kibana'.

Generate an enrollment token for Elasticsearch nodes with
'/usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s node'.
```

If all goes well, Elasticsearch will be installed on the host.

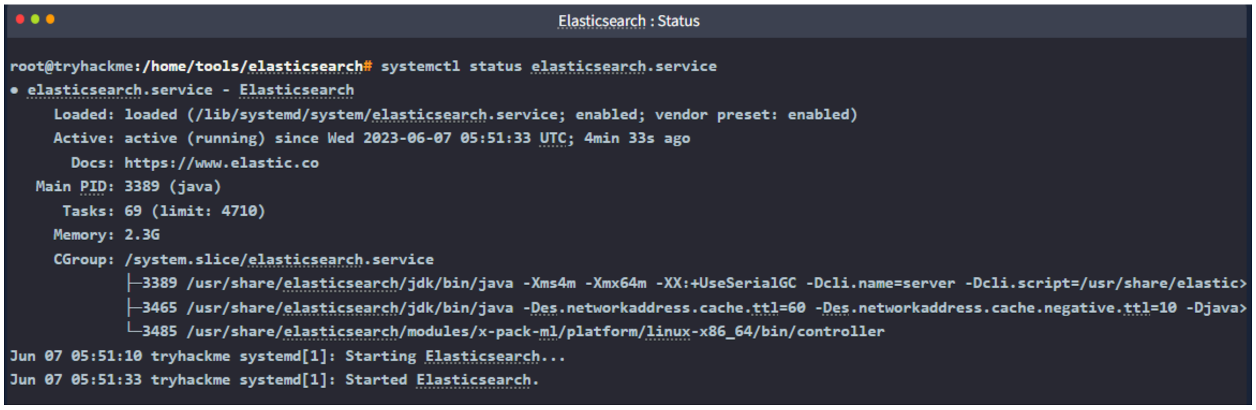
The following commands will be used to make the Elasticsearch service persistent so that it gets started whenever the server restarts.



```
root@tryhackme:/home/tools/elasticsearch# systemctl enable elasticsearch.service
Created symlink /etc/systemd/system/multi-user.target.wants/elasticsearch.service → /lib/systemd/system/elasticsearch.service.
root@tryhackme:/home/tools/elasticsearch# systemctl start elasticsearch.service
```

Elasticsearch Status

Now that we have installed Elasticsearch successfully, let's check its status to see if it's installed and running properly.



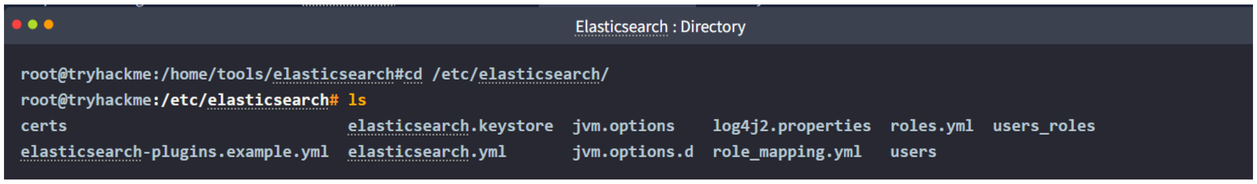
```
root@tryhackme:/home/tools/elasticsearch# systemctl status elasticsearch.service
• elasticsearch.service - Elasticsearch
     Loaded: loaded (/lib/systemd/system/elasticsearch.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-06-07 05:51:33 UTC; 4min 33s ago
       Docs: https://www.elastic.co
   Main PID: 3389 (java)
      Tasks: 69 (limit: 4710)
     Memory: 2.3G
     CGroup: /system.slice/elasticsearch.service
             ├─3389 /usr/share/elasticsearch/jdk/bin/java -Xms4m -Xmx64m -XX:+UseSerialGC -Dcli.name=server -Dcli.script=/usr/share/elastic>
             ├─3465 /usr/share/elasticsearch/jdk/bin/java -Des.networkaddress.cache.ttl=60 -Des.networkaddress.cache.negative.ttl=10 -Djava>
             └─3485 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/controller
Jun 07 05:51:10 tryhackme systemd[1]: Starting Elasticsearch...
Jun 07 05:51:33 tryhackme systemd[1]: Started Elasticsearch.
```

This output shows the elasticsearch service has been successfully installed and running properly.

Configuring Elasticsearch

We have successfully installed Elasticsearch on the Linux host. We will now make some changes to the configuration so that it's accessible to other components as well.

Important Configurations
All important configuration files related to Elasticsearch can be found in the /etc/elasticsearch directory.

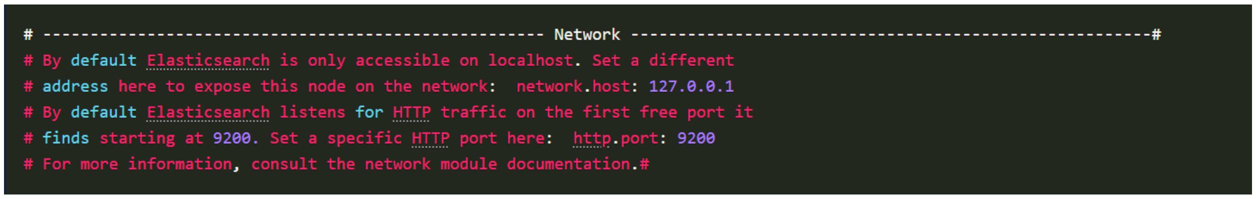


```
root@tryhackme:/home/tools/elasticsearch#cd /etc/elasticsearch/
root@tryhackme:/etc/elasticsearch# ls
certs                             elasticsearch.keystore  jvm.options    log4j2.properties  roles.yml  users_roles
elasticsearch-plugins.example.yml  elasticsearch.yml       jvm.options.d  role_mapping.yml   users
```

Go to this directory and explore the files and directories. These are important configuration files; some of them are explained below:

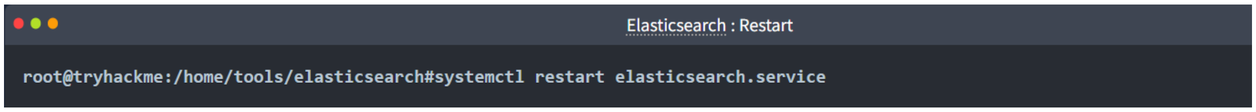
- elasticsearch.yml: This is the main configuration file for Elasticsearch. It contains various settings that determine the behavior of Elasticsearch, such as network configuration, cluster settings, node settings, and paths for data storage and logging. Modifying this file allows you to customize Elasticsearch according to your specific requirements.
- jvm.options: The jvm.options file contains JVM (Java Virtual Machine) configuration settings for Elasticsearch. It allows you to specify parameters related to memory allocation, garbage collection, and other JVM options. Adjusting these settings is crucial for optimizing Elasticsearch's performance and ensuring efficient memory usage.
- log4j2.properties: The log4j2.properties file is the configuration file for Elasticsearch's logging system, Log4j. It defines how Elasticsearch logs different types of messages and sets log levels for different components. You can modify this file to configure the log output format, log rotation, and other logging-related settings.
- users: The users file is used for configuring user authentication and authorization in Elasticsearch. It allows you to define users, roles, and their respective permissions. By managing this file, you can control access to Elasticsearch resources and secure your cluster.
- roles.yml and roles_mapping.yml: These files are used in conjunction with the users file to define roles and their mappings to users and privileges. Roles provide a way to group users and assign common permissions to them. The roles.yml file defines the roles and their privileges, while the roles_mapping.yml file maps roles to users.

Let's open the elasticsearch.yml using the command nano elasticsearch.yml and go to the Network section, as shown below:



```
# ---------------------------------------------------- Network ----------------------------------------------------#
# By default Elasticsearch is only accessible on localhost. Set a different
# address here to expose this node on the network:  network.host: 127.0.0.1
# By default Elasticsearch listens for HTTP traffic on the first free port it
# finds starting at 9200. Set a specific HTTP port here:  http.port: 9200
# For more information, consult the network module documentation.#
```

In this section, we see the two variables network.host and http.port. Uncomment these two and change the value of the network.host variable to 127.0.0.1 . As we are installing all these components on the same host, therefore we will update the network.host to 127.0.0.1. Once these changes are made, save the document and restart the Elasticsearch service using the following command.



Elasticsearch : Restart

```
root@tryhackme:/home/tools/elasticsearch#systemctl restart elasticsearch.service
```

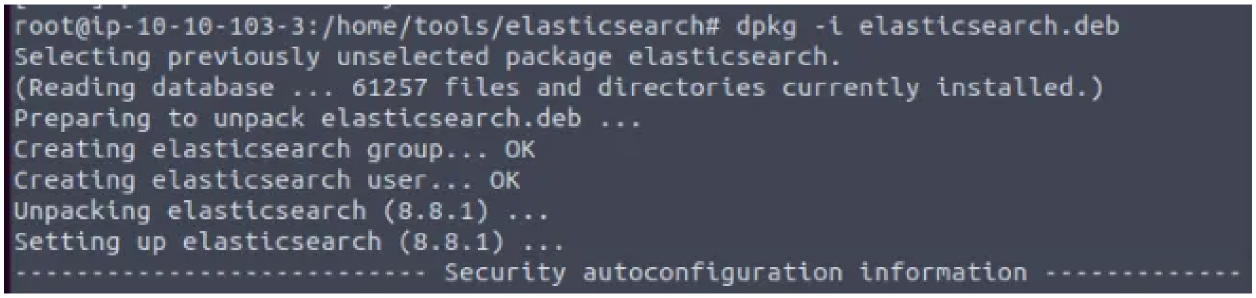
Now that we have installed and configured Elasticsearch properly, let's move on to the next task and install the Logstash component.

****

Answer the questions below:

What is the default port Elasticsearch runs on?
Answer: 9200

What version of Elasticsearch have we installed in this task?
This was shown in the output when installing Elasticsearch.



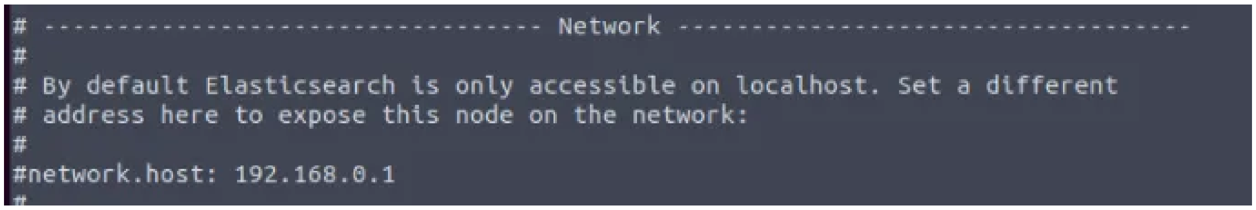
```
root@ip-10-10-103-3:/home/tools/elasticsearch# dpkg -i elasticsearch.deb
Selecting previously unselected package elasticsearch.
(Reading database ... 61257 files and directories currently installed.)
Preparing to unpack elasticsearch.deb ...
Creating elasticsearch group... OK
Creating elasticsearch user... OK
Unpacking elasticsearch (8.8.1) ...
Setting up elasticsearch (8.8.1) ...
--------------------------- Security autoconfiguration information ------------
```

Answer: 8.8.1

What is the command used to check the status of Elasticsearch service?
Answer: systemctl status elasticsearch.service

What is the default value of the network.host variable found in the elasticsearch.yml file?

This was found in the elasticsearch.yml file we modified earlier.



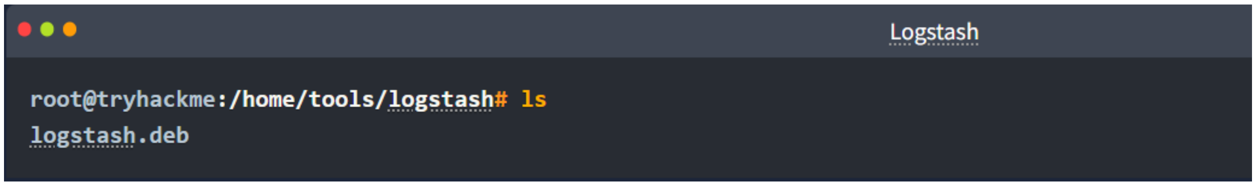
```
# ---------------------------------- Network -----------------------------------
#
# By default Elasticsearch is only accessible on localhost. Set a different
# address here to expose this node on the network:
#
#network.host: 192.168.0.1
```

Answer: 192.168.0.1

Logstash: Installation and Configuration

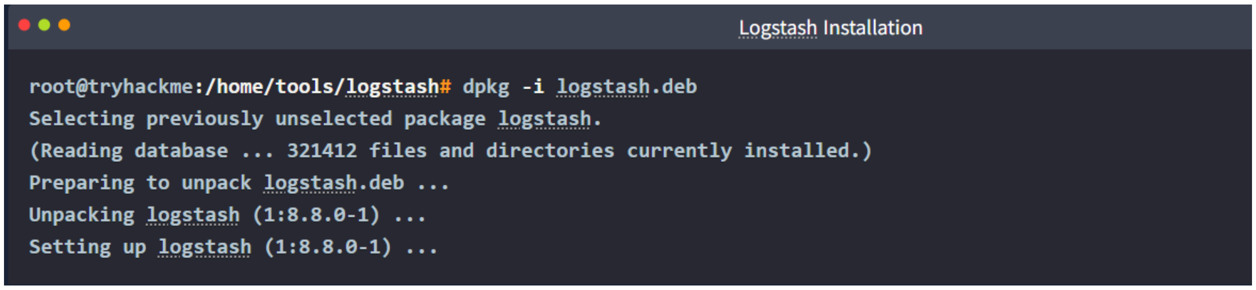
Installing Logstash

Let's go through the process of installing Logstash. The latest installation instance is placed on the path /home/tools/logstash.



```
root@tryhackme:/home/tools/logstash# ls
logstash.deb
```

Run the following command dpkg -i logstash.deb as a root user to install the logstash on the server. It will take 1-2 minutes to get installed, as shown below:

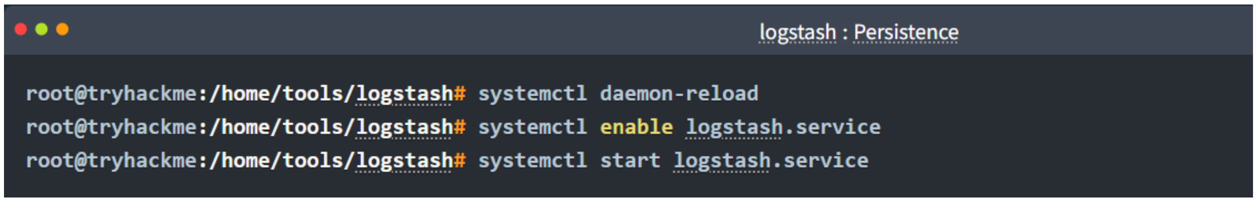


```
root@tryhackme:/home/tools/logstash# dpkg -i logstash.deb
Selecting previously unselected package logstash.
(Reading database ... 321412 files and directories currently installed.)
Preparing to unpack logstash.deb ...
Unpacking logstash (1:8.8.0-1) ...
Setting up logstash (1:8.8.0-1) ...
```

If all goes well, Logstash will be installed on the host.

Logstash: Persistence

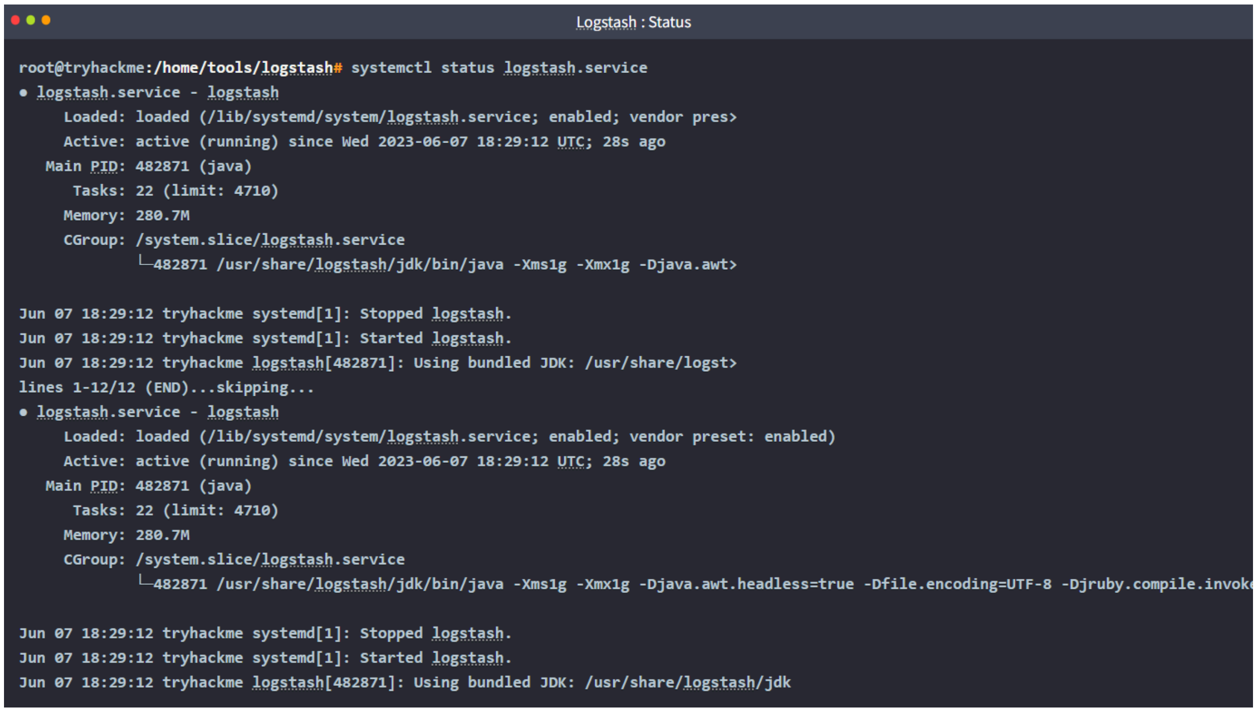
We will use the following commands to make the Logstash service persistent:



```
root@tryhackme:/home/tools/logstash# systemctl daemon-reload
root@tryhackme:/home/tools/logstash# systemctl enable logstash.service
root@tryhackme:/home/tools/logstash# systemctl start logstash.service
```

Logstash Status

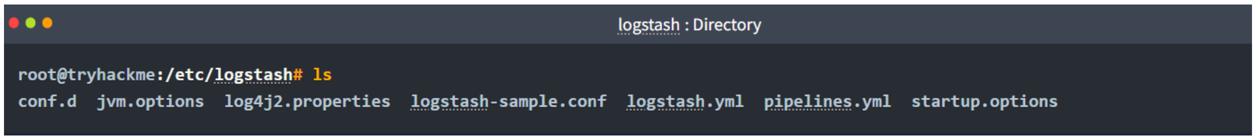
Now that we have installed Logstash successfully, let's check its status to see if it's installed and running properly.



```
root@tryhackme:/home/tools/logstash# systemctl status logstash.service
● logstash.service - logstash
     Loaded: loaded (/lib/systemd/system/logstash.service; enabled; vendor pres>
     Active: active (running) since Wed 2023-06-07 18:29:12 UTC; 28s ago
   Main PID: 482871 (java)
      Tasks: 22 (limit: 4710)
     Memory: 280.7M
     CGroup: /system.slice/logstash.service
             └─482871 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -Djava.awt>

Jun 07 18:29:12 tryhackme systemd[1]: Stopped logstash.
Jun 07 18:29:12 tryhackme systemd[1]: Started logstash.
Jun 07 18:29:12 tryhackme logstash[482871]: Using bundled JDK: /usr/share/logst>
lines 1-12/12 (END)...skipping...
● logstash.service - logstash
     Loaded: loaded (/lib/systemd/system/logstash.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-06-07 18:29:12 UTC; 28s ago
   Main PID: 482871 (java)
      Tasks: 22 (limit: 4710)
     Memory: 280.7M
     CGroup: /system.slice/logstash.service
             └─482871 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djruby.compile.invok

Jun 07 18:29:12 tryhackme systemd[1]: Stopped logstash.
Jun 07 18:29:12 tryhackme systemd[1]: Started logstash.
Jun 07 18:29:12 tryhackme logstash[482871]: Using bundled JDK: /usr/share/logstash/jdk
```

Configuring Logstash

In a typical installation of Logstash, the /etc/logstash directory is the default location for important configuration files.



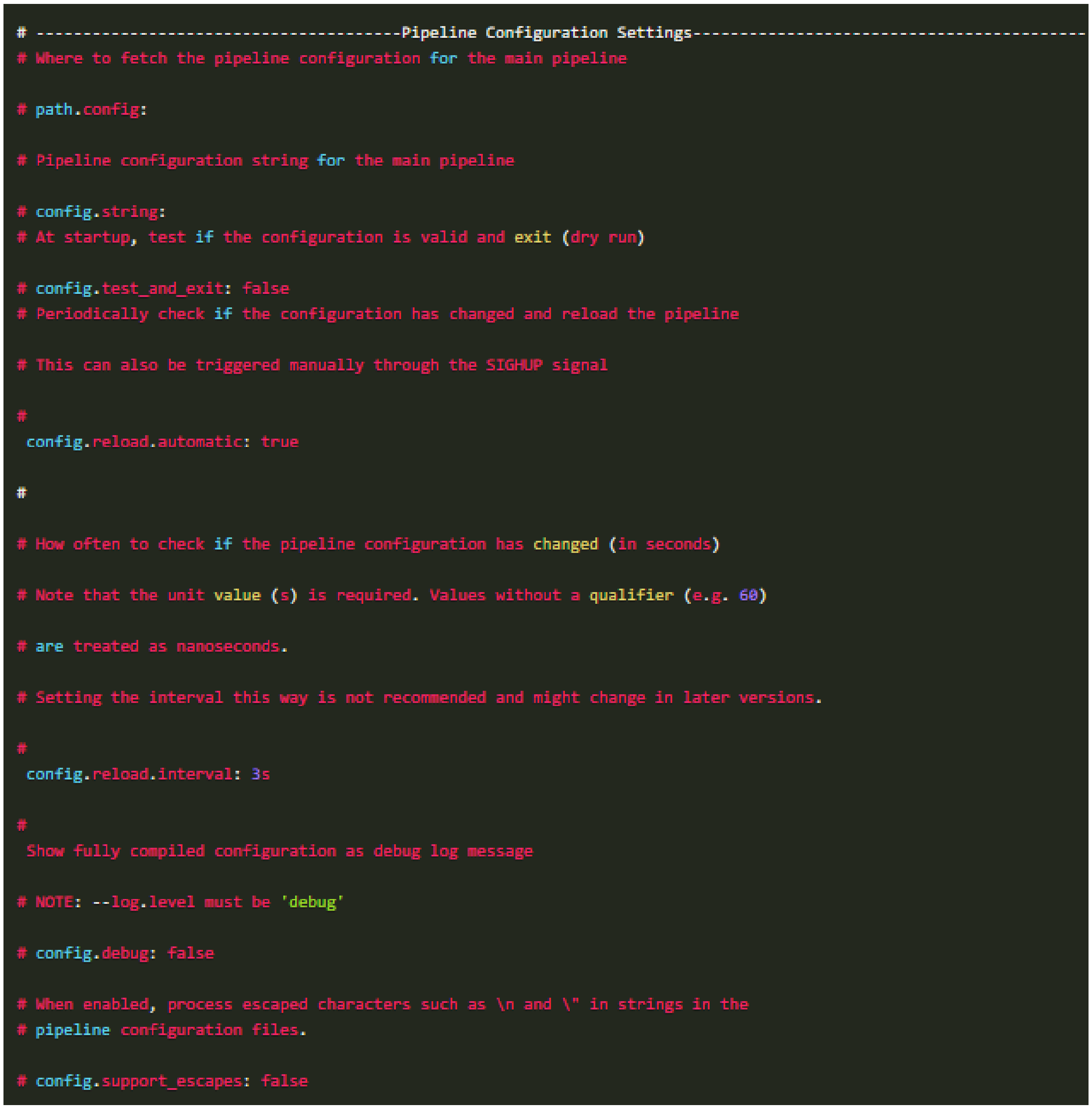
```
root@tryhackme:/etc/logstash# ls
conf.d  jvm.options  log4j2.properties  logstash-sample.conf  logstash.yml  pipelines.yml  startup.options
```

Here are some of the important files you may find in the /etc/logstash directory:

- logstash.yml: This is the main configuration file for Logstash. It contains global settings and options for Logstash, such as network settings, logging configurations, pipeline configuration paths, and more.
- jvm.options: This file contains the Java Virtual Machine (JVM) options for Logstash. You can use it to configure parameters like memory allocation, garbage collection settings, and other JVM-related options.
- log4j2.properties: Logstash uses the Log4j2 framework for logging. This file allows you to configure the logging behavior, including log levels, log outputs (such as console or file), log file locations, and more.
- pipelines.yml: If you are running multiple pipelines in Logstash, this file is used to define and configure them. It allows you to specify each pipeline's different inputs, filters, and outputs.

- conf.d/: This directory is often used to store individual pipeline configuration files. You can create separate configuration files within this directory, each defining a specific data processing pipeline. Logstash will load and process these configuration files in alphabetical order, so it’s common to prefix them with numbers to control the processing order.
- patterns/: This directory stores custom patterns that can be used in Logstash’s grok filter. Grok is a powerful pattern-matching and extraction tool in Logstash, and you can define your own patterns in separate files within this directory.
- startup.options: On some systems, you may find this file which contains additional options and arguments that can be passed to Logstash during startup.

Let’s open the logstash.yml using the command nano logstash.yml and go to the Pipeline Configuration Settings section, as shown below:

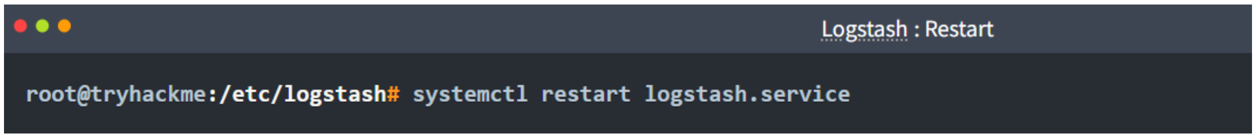


```
# ------------------------------------------Pipeline Configuration Settings------------------------------------------
# Where to fetch the pipeline configuration for the main pipeline

# path.config:

# Pipeline configuration string for the main pipeline

# config.string:
# At startup, test if the configuration is valid and exit (dry run)

# config.test_and_exit: false
# Periodically check if the configuration has changed and reload the pipeline

# This can also be triggered manually through the SIGHUP signal

#
 config.reload.automatic: true

#

# How often to check if the pipeline configuration has changed (in seconds)

# Note that the unit value (s) is required. Values without a qualifier (e.g. 60)

# are treated as nanoseconds.

# Setting the interval this way is not recommended and might change in later versions.

#
 config.reload.interval: 3s

#
 Show fully compiled configuration as debug log message

# NOTE: --log.level must be 'debug'

# config.debug: false

# When enabled, process escaped characters such as \n and \" in strings in the
# pipeline configuration files.

# config.support_escapes: false
```

There are two updates that we need to make. Uncomment both the following variables and change the value of config.reload.automatic: to true, as shown below:

- config.reload.automatic: true
- config.reload.interval: 3s

These changes will ensure Logstash looks at the configuration files every 3 seconds to see if there are any changes to the log sources which are being ingested.



```
root@tryhackme:/etc/logstash# systemctl restart logstash.service
```

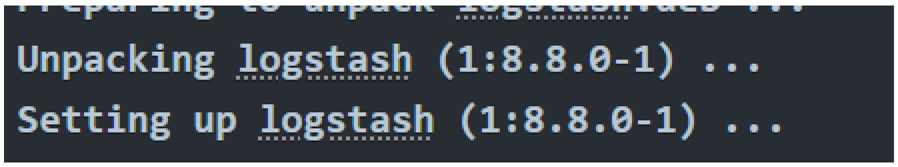
Now that we have installed and configured Logstash properly, let's move on to the next task and install the Kibana component.

****

Answer the questions below:

What is the configuration reload interval set by default in logstash.yml?
Answer: 3s

What is the Logstash version we just installed?
Again this was shown when installing Logstash



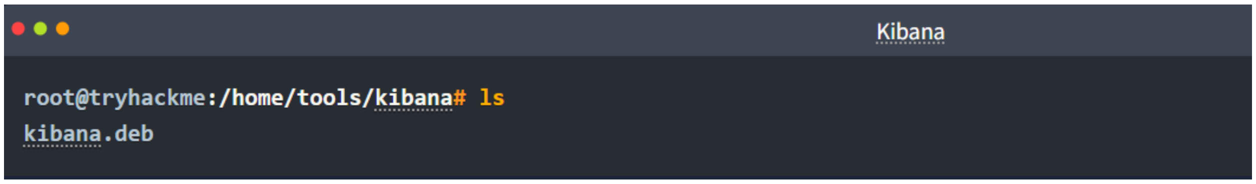
```
Unpacking logstash (1:8.8.0-1) ...
Setting up logstash (1:8.8.0-1) ...
```

Answer: 8.8.1

Kibana: Installation and Configuration

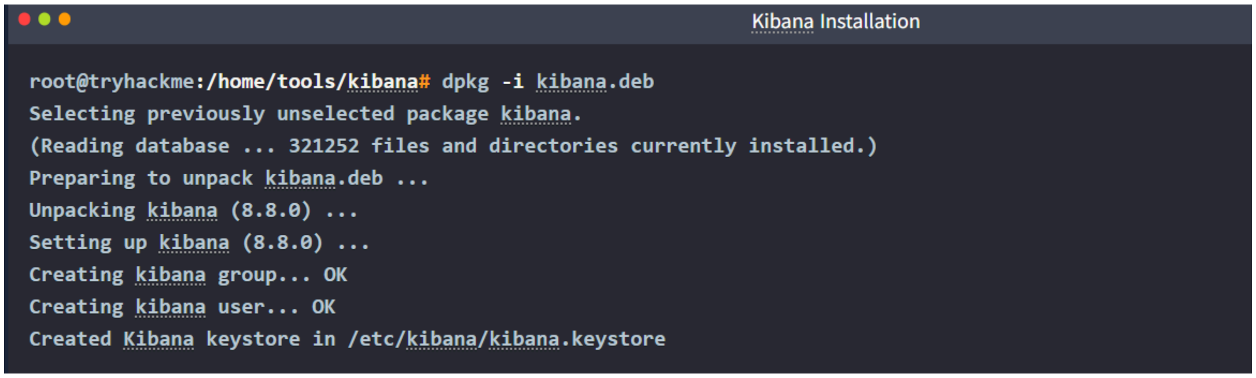
Installing Kibana

We have already explored the Kibana interface in the InvestigatingwithELK room. Let's now go through the process of installing Kibana on the Ubuntu lab. The latest installation instance is placed on the path /home/tools/kibana.



```
root@tryhackme:/home/tools/kibana# ls
kibana.deb
```

Run the following command dpkg -i kibana as a root user to install the Kibana on the server. It will take 1-2 minutes to get installed, as shown below:

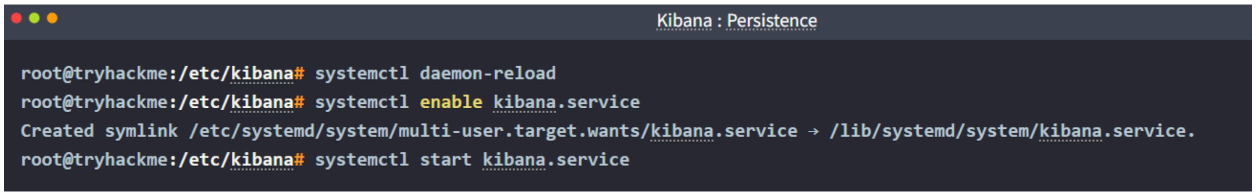


```
root@tryhackme:/home/tools/kibana# dpkg -i kibana.deb
Selecting previously unselected package kibana.
(Reading database ... 321252 files and directories currently installed.)
Preparing to unpack kibana.deb ...
Unpacking kibana (8.8.0) ...
Setting up kibana (8.8.0) ...
Creating kibana group... OK
Creating kibana user... OK
Created Kibana keystore in /etc/kibana/kibana.keystore
```

If all goes well, Kibana will be installed on the host.

Kibana: Persistence

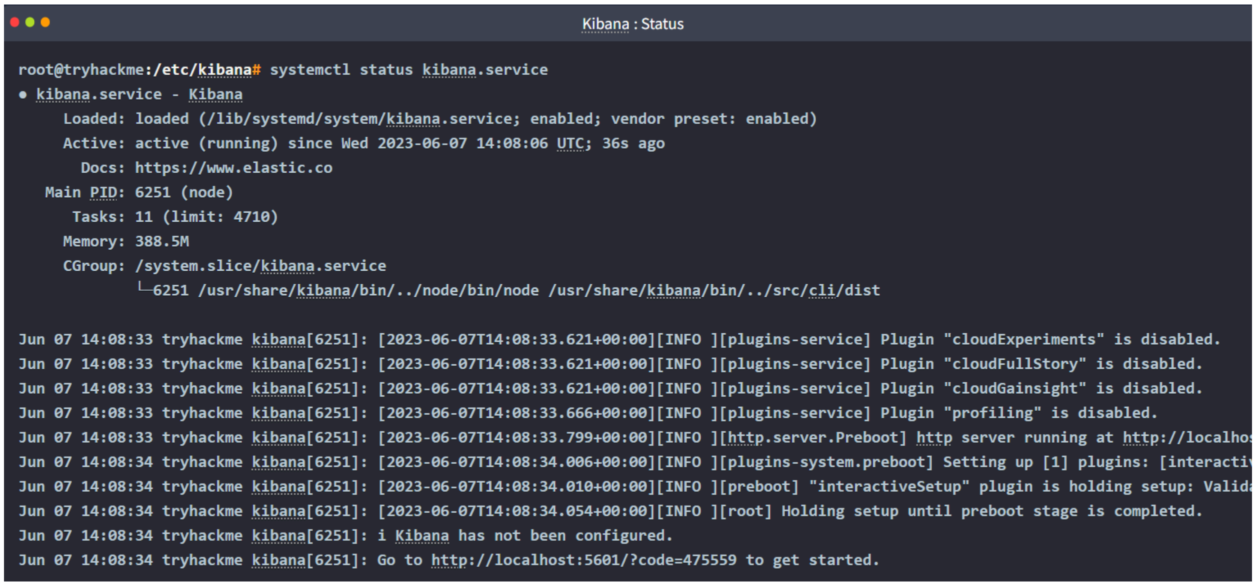
We will use the following commands to make the Kibana service persistent.



```
root@tryhackme:/etc/kibana# systemctl daemon-reload
root@tryhackme:/etc/kibana# systemctl enable kibana.service
Created symlink /etc/systemd/system/multi-user.target.wants/kibana.service → /lib/systemd/system/kibana.service.
root@tryhackme:/etc/kibana# systemctl start kibana.service
```

Kibana Status

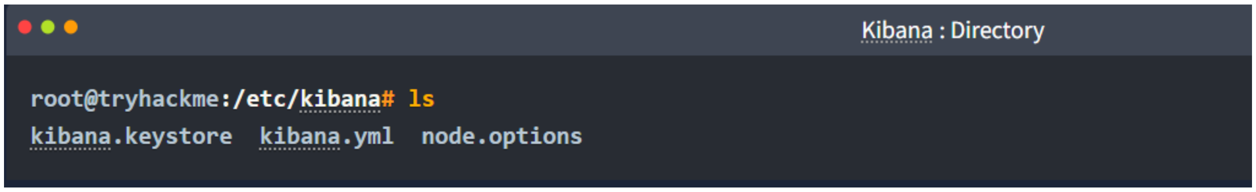
Now that we have installed Kibana successfully, let's check its status to see if it's installed and running properly.



```
root@tryhackme:/etc/kibana# systemctl status kibana.service
● kibana.service - Kibana
     Loaded: loaded (/lib/systemd/system/kibana.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-06-07 14:08:06 UTC; 36s ago
       Docs: https://www.elastic.co
   Main PID: 6251 (node)
      Tasks: 11 (limit: 4710)
     Memory: 388.5M
     CGroup: /system.slice/kibana.service
             └─6251 /usr/share/kibana/bin/../node/bin/node /usr/share/kibana/bin/../src/cli/dist

Jun 07 14:08:33 tryhackme kibana[6251]: [2023-06-07T14:08:33.621+00:00][INFO ][plugins-service] Plugin "cloudExperiments" is disabled.
Jun 07 14:08:33 tryhackme kibana[6251]: [2023-06-07T14:08:33.621+00:00][INFO ][plugins-service] Plugin "cloudFullStory" is disabled.
Jun 07 14:08:33 tryhackme kibana[6251]: [2023-06-07T14:08:33.621+00:00][INFO ][plugins-service] Plugin "cloudGainsight" is disabled.
Jun 07 14:08:33 tryhackme kibana[6251]: [2023-06-07T14:08:33.666+00:00][INFO ][plugins-service] Plugin "profiling" is disabled.
Jun 07 14:08:33 tryhackme kibana[6251]: [2023-06-07T14:08:33.799+00:00][INFO ][http.server.Preboot] http server running at http://localhos
Jun 07 14:08:34 tryhackme kibana[6251]: [2023-06-07T14:08:34.006+00:00][INFO ][plugins-system.preboot] Setting up [1] plugins: [interacti
Jun 07 14:08:34 tryhackme kibana[6251]: [2023-06-07T14:08:34.010+00:00][INFO ][preboot] "interactiveSetup" plugin is holding setup: Valid
Jun 07 14:08:34 tryhackme kibana[6251]: [2023-06-07T14:08:34.054+00:00][INFO ][root] Holding setup until preboot stage is completed.
Jun 07 14:08:34 tryhackme kibana[6251]: i Kibana has not been configured.
Jun 07 14:08:34 tryhackme kibana[6251]: Go to http://localhost:5601/?code=475559 to get started.
```

Configuring Kibana

The /etc/kibana path typically contains the configuration files for Kibana, an open-source data visualization and exploration tool. Here are the commonly found files in the /etc/kibana directory:



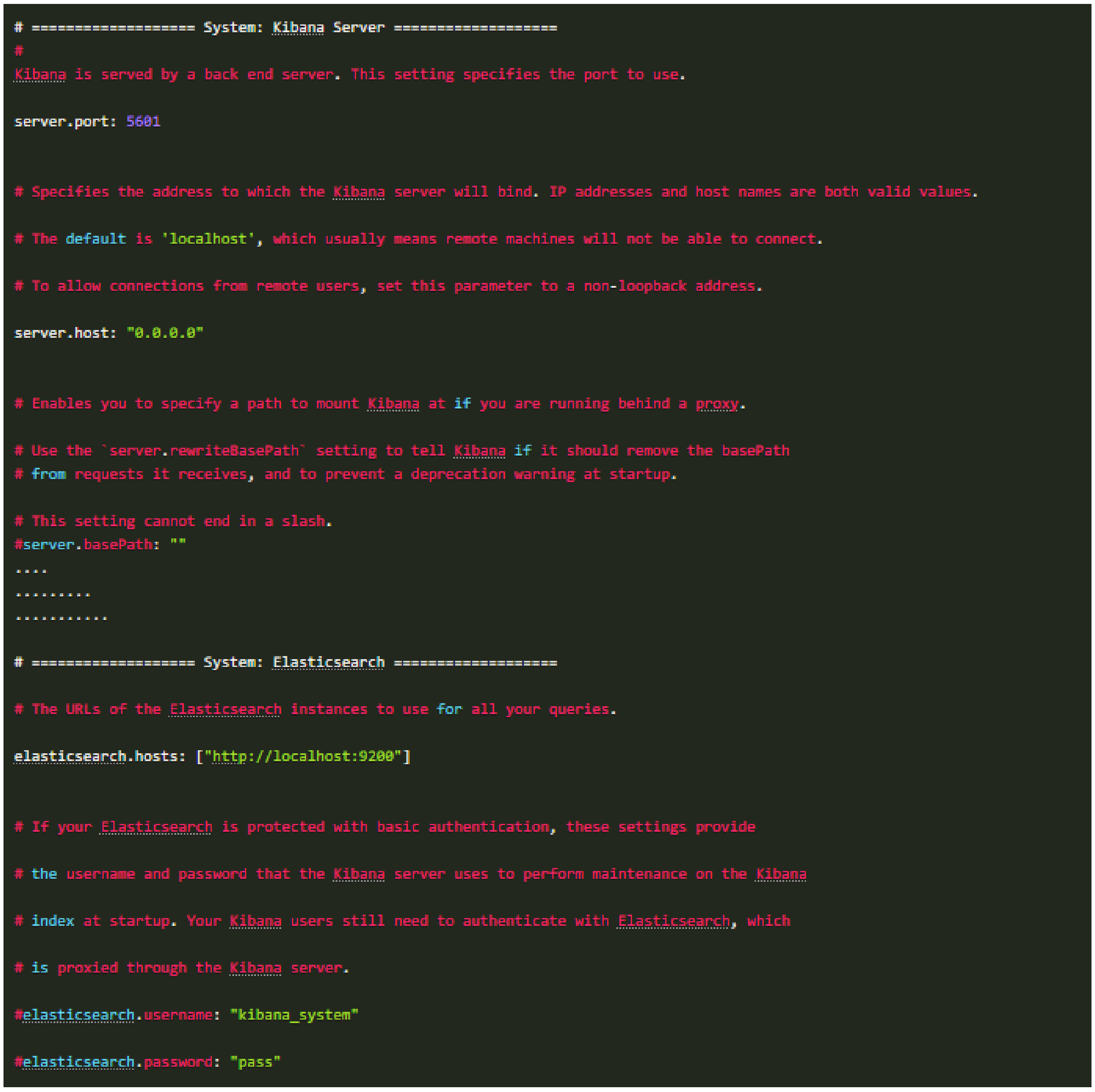
```
root@tryhackme:/etc/kibana# ls
kibana.keystore  kibana.yml  node.options
```

Two important files are explained below:

- kibana.yml: This is the main configuration file for Kibana. It contains various settings to customize Kibana's behavior, such as the Elasticsearch server URL, server host and port, logging options, security configurations, and more. You can modify this file to tailor Kibana to your specific environment.

- kibana.keystore: This file securely stores sensitive configuration settings, such as passwords and API keys. The kibana.keystore file provides a safer alternative to storing sensitive information in plain text within the kibana.yml file. It is encrypted and can be managed using the bin/kibana-keystore command-line tool.

To configure Kibana, open the kibana.yml using the command nano kibana.yml and go to the System: Kibana Server section, as shown below:



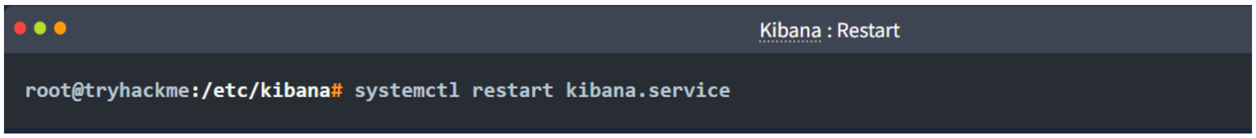
```
# ================== System: Kibana Server ==================
#
Kibana is served by a back end server. This setting specifies the port to use.

server.port: 5601


# Specifies the address to which the Kibana server will bind. IP addresses and host names are both valid values.

# The default is 'localhost', which usually means remote machines will not be able to connect.

# To allow connections from remote users, set this parameter to a non-loopback address.

server.host: "0.0.0.0"


# Enables you to specify a path to mount Kibana at if you are running behind a proxy.

# Use the `server.rewriteBasePath` setting to tell Kibana if it should remove the basePath
# from requests it receives, and to prevent a deprecation warning at startup.

# This setting cannot end in a slash.
#server.basePath: ""
....
.........
...........

# ================== System: Elasticsearch ==================

# The URLs of the Elasticsearch instances to use for all your queries.

elasticsearch.hosts: ["http://localhost:9200"]


# If your Elasticsearch is protected with basic authentication, these settings provide

# the username and password that the Kibana server uses to perform maintenance on the Kibana

# index at startup. Your Kibana users still need to authenticate with Elasticsearch, which

# is proxied through the Kibana server.

#elasticsearch.username: "kibana_system"

#elasticsearch.password: "pass"
```

Uncomment the following two variables and make the changes to server.host as shown below:

- server.port: 5601 => Kibana runs on port 5601 by default.

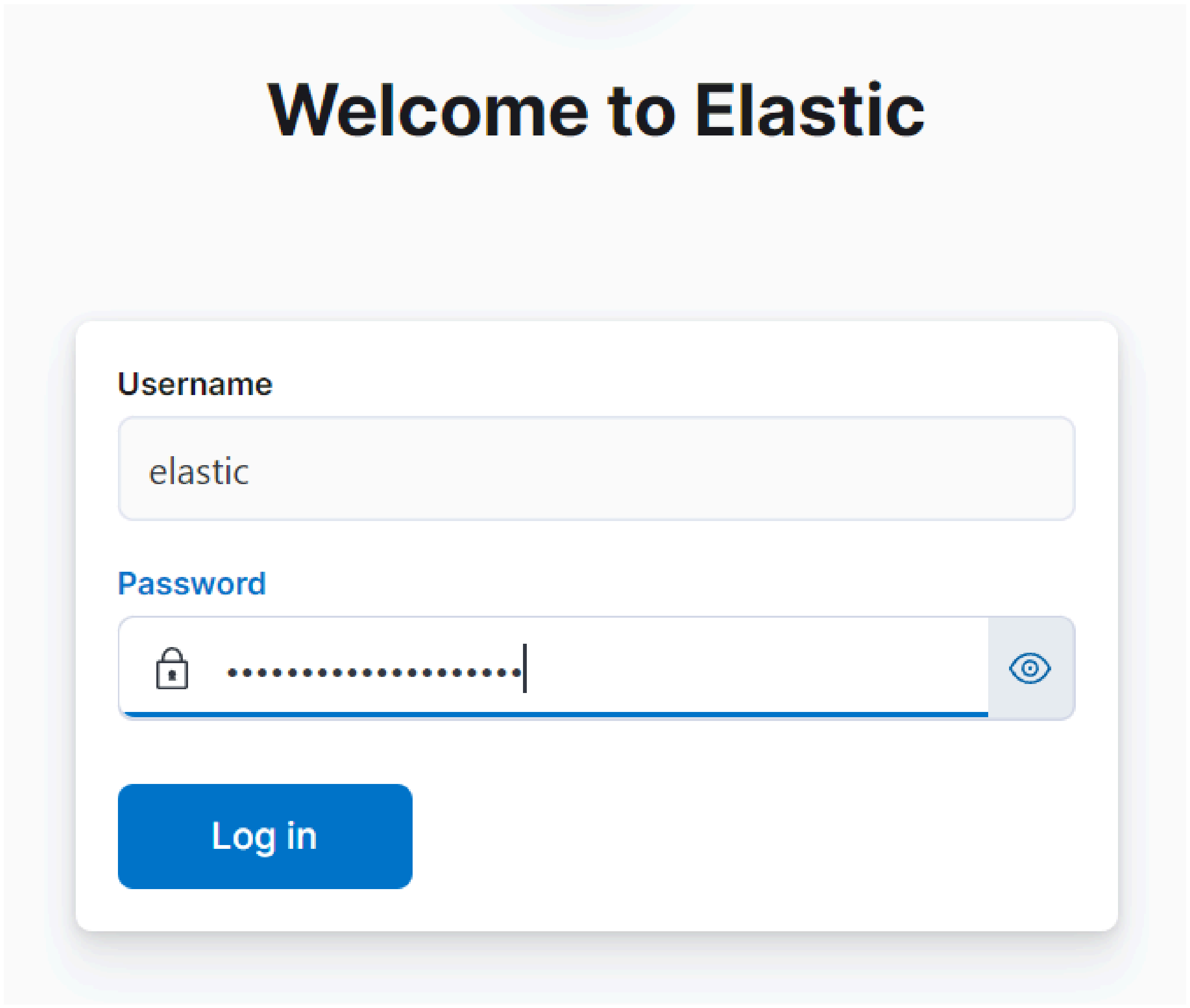
- server.host: "0.0.0.0" => This is important to note that; if the server IP is changed, it should be updated here. The server's IP does not change in a production environment; in that case, this parameter will not be changed often.

Once the changes are made, and the config file is saved, it's time to restart the Kibana server using the following command:

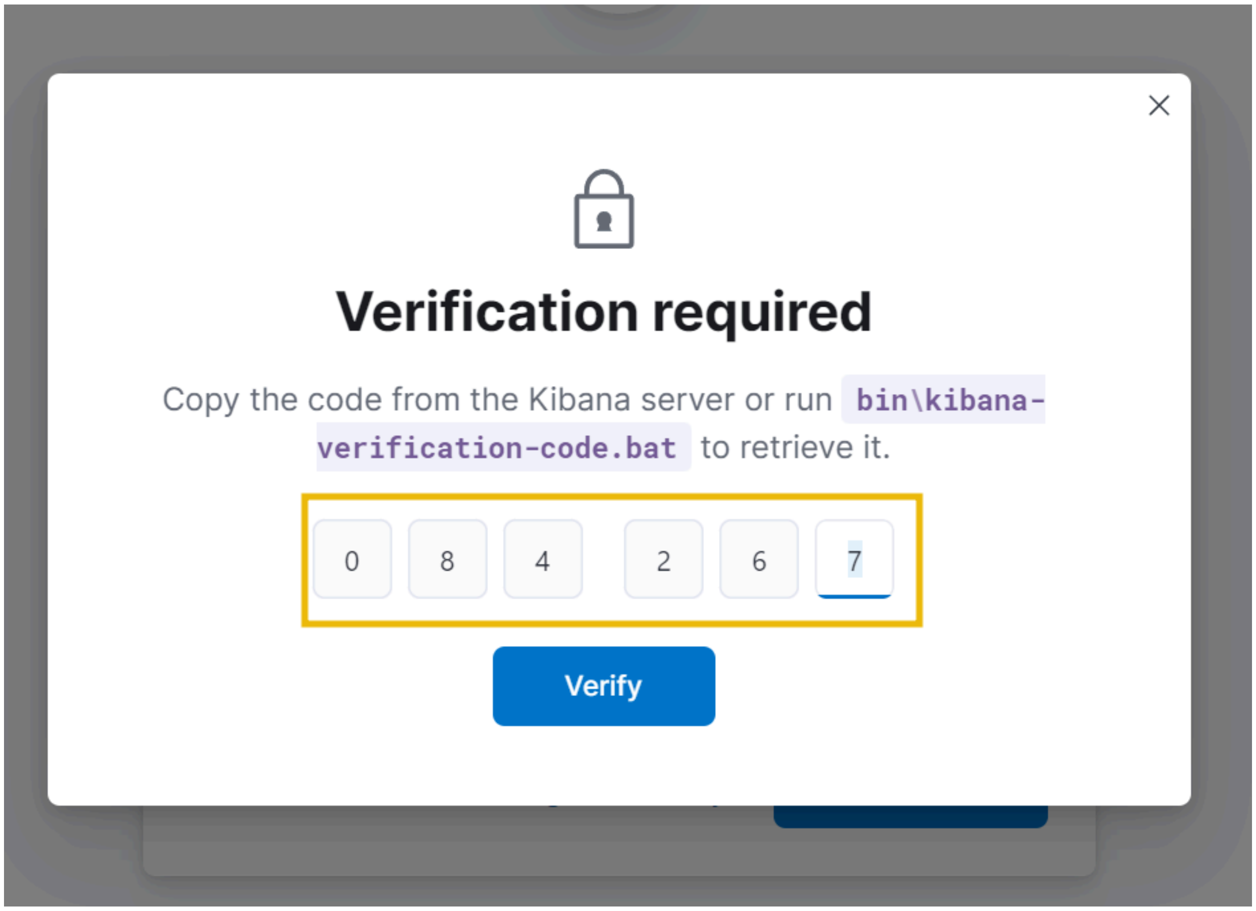


```
root@tryhackme:/etc/kibana# systemctl restart kibana.service
```

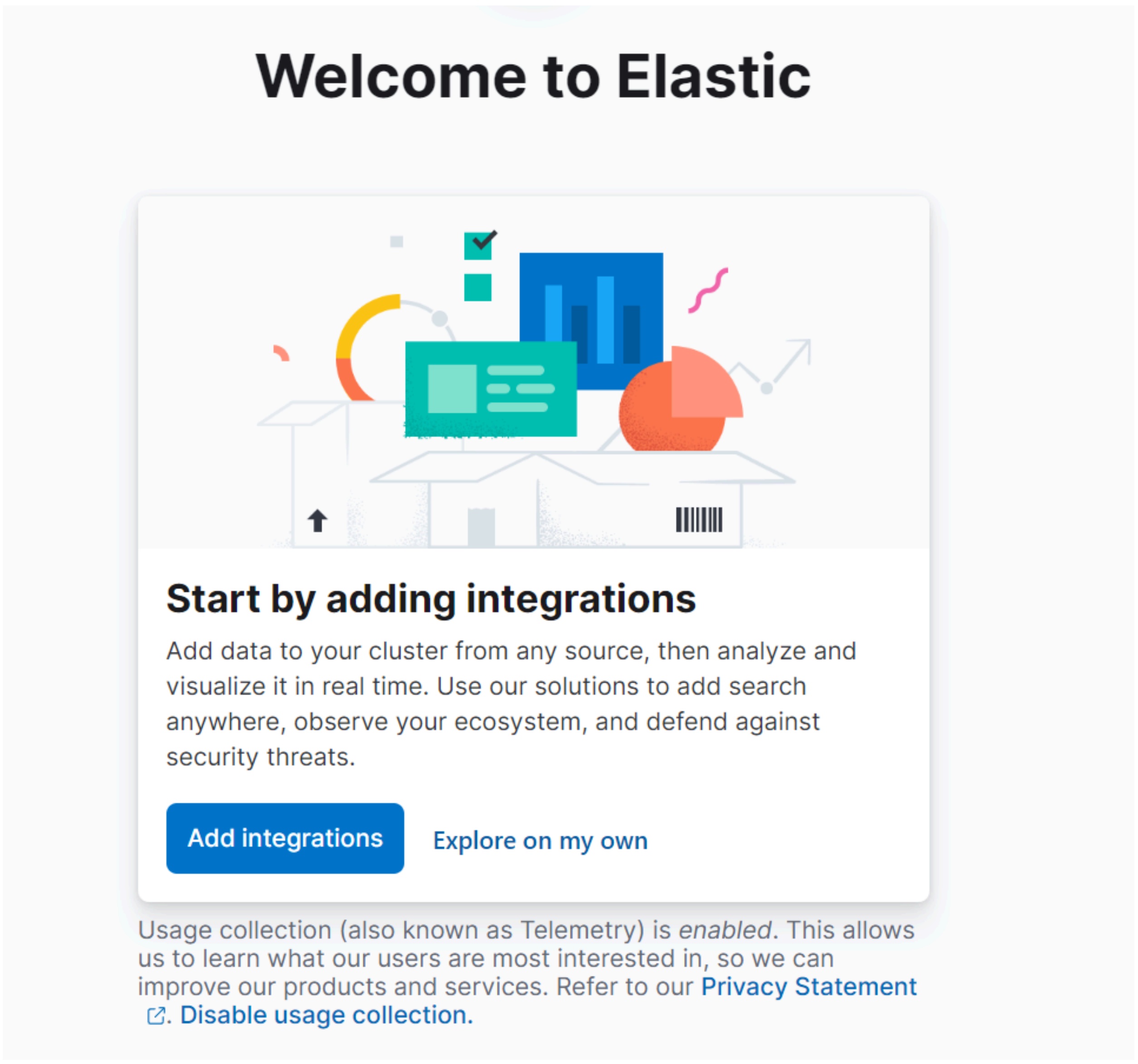
Open the browser, and go to MACHINE_IP:5601; we will see a Kibana Interface. We will need to generate an enrollment token for the Kibana instance using the following command */usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s kibana*. It will show a randomly generated token. Enter the token in the space and press Enter. It will ask for the elastic credentials. There are the same credentials that were generated during the installation of Elasticsearch:



Next, it will ask for the verification code, which can be obtained by running this command */usr/share/kibana/bin/kibana-verification-code*


Verification required
Copy the code from the Kibana server or run bin\kibana-verification-code.bat to retrieve it.
0 8 4 2 6 7
Verify

Congrats, ELK is installed properly:



Welcome to Elastic

Start by adding integrations

Add data to your cluster from any source, then analyze and visualize it in real time. Use our solutions to add search anywhere, observe your ecosystem, and defend against security threats.

Usage collection (also known as Telemetry) is *enabled*. This allows us to learn what our users are most interested in, so we can improve our products and services. Refer to our Privacy Statement. Disable usage collection.

Kibana can be used to visualize the logs that are parsed, and filtered from Logstash.

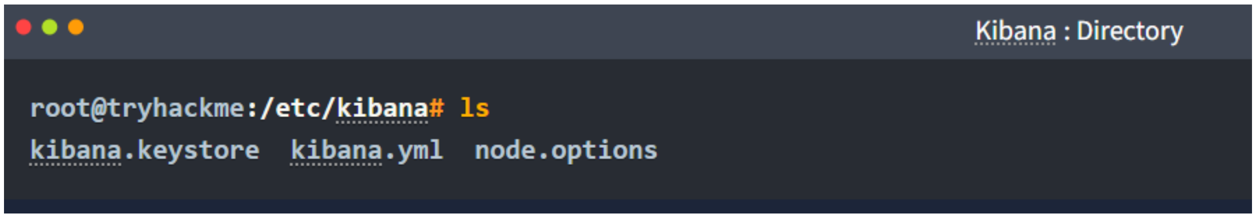
Note: As we are focused on Logstash in this room, please stop the Kibana service to avoid VM slowing down. Use the command systemctl stop kibana.service to stop Kibana.

****

Answer the questions below:

What is the default port Kibana runs on?
Answer: 5601

How many files are found in /etc/kibana/ directory?



```
root@tryhackme:/etc/kibana# ls
kibana.keystore  kibana.yml  node.options
```

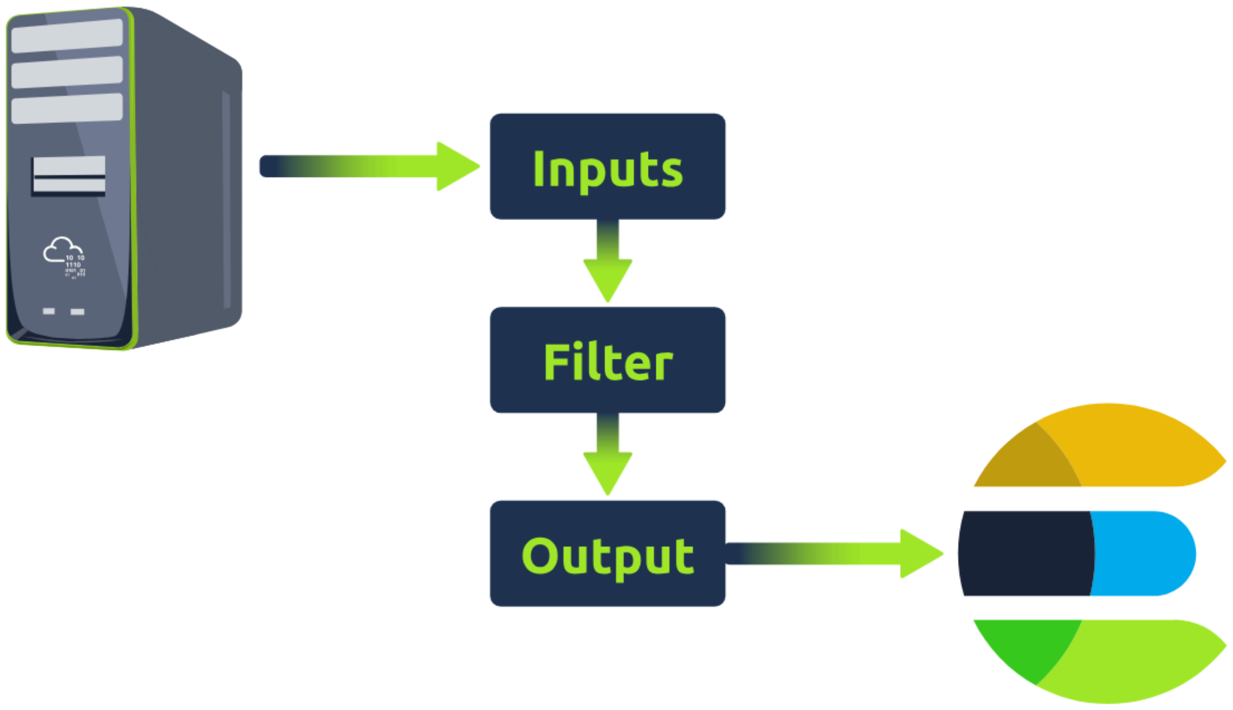
Answer: 3

Use the command systemctl stop kibana.service to stop Kibana.
No Answer Needed

Logstash: Overview

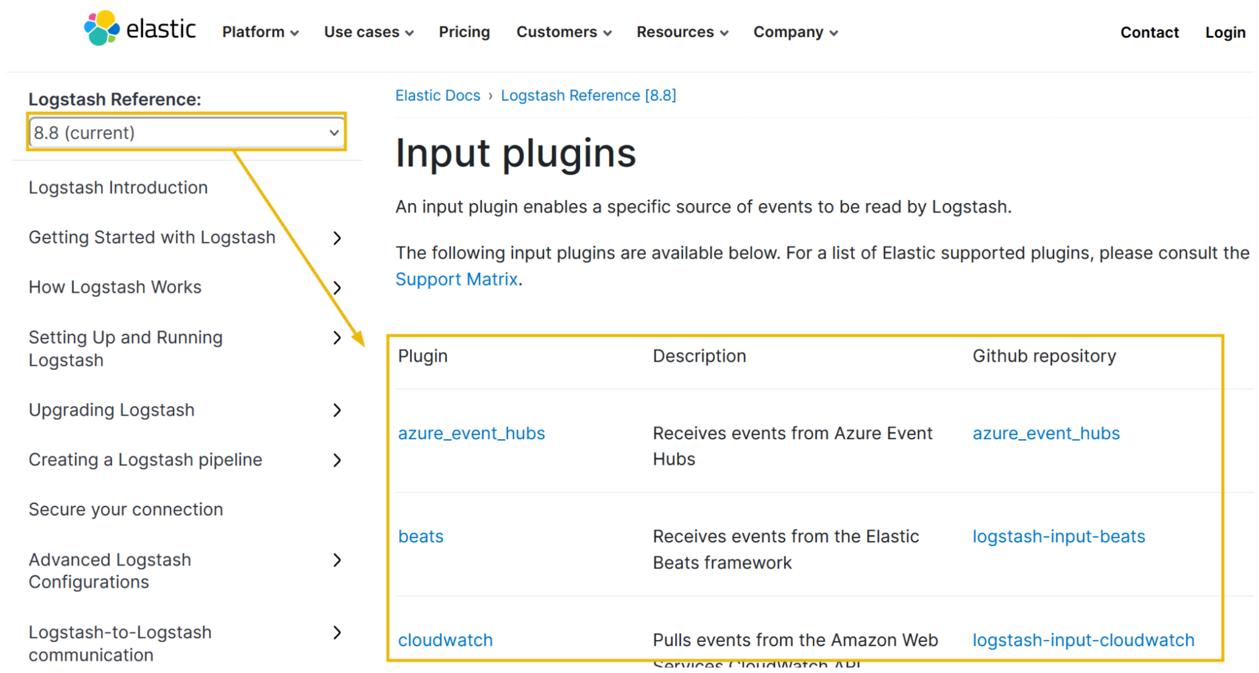
Let's dive deep into the details of Logstash.

Logstash is a data processing engine that takes data from different sources, applies the filter or normalizes it, and then sends it to the destination, Kibana, or a listening port. A Logstash configuration file is divided into three parts, as shown below.



Each part is explained in detail below:

Input
Logstash provides a wide range of input plugins that allow you to ingest data from diverse sources such as log files, system metrics, databases, message queues, APIs, and more. These input plugins facilitate data collection in various formats and protocols, as shown in the reference document.


elastic
Platform
Use cases
Pricing
Customers
Resources
Company
Contact
Login
Logstash Reference:
8.8 (current)
Logstash Introduction
Getting Started with Logstash
How Logstash Works
Setting Up and Running Logstash
Upgrading Logstash
Creating a Logstash pipeline
Secure your connection
Advanced Logstash Configurations
Logstash-to-Logstash communication
Elastic Docs › Logstash Reference [8.8]

Input plugins

An input plugin enables a specific source of events to be read by Logstash.

The following input plugins are available below. For a list of Elastic supported plugins, please consult the Support Matrix.

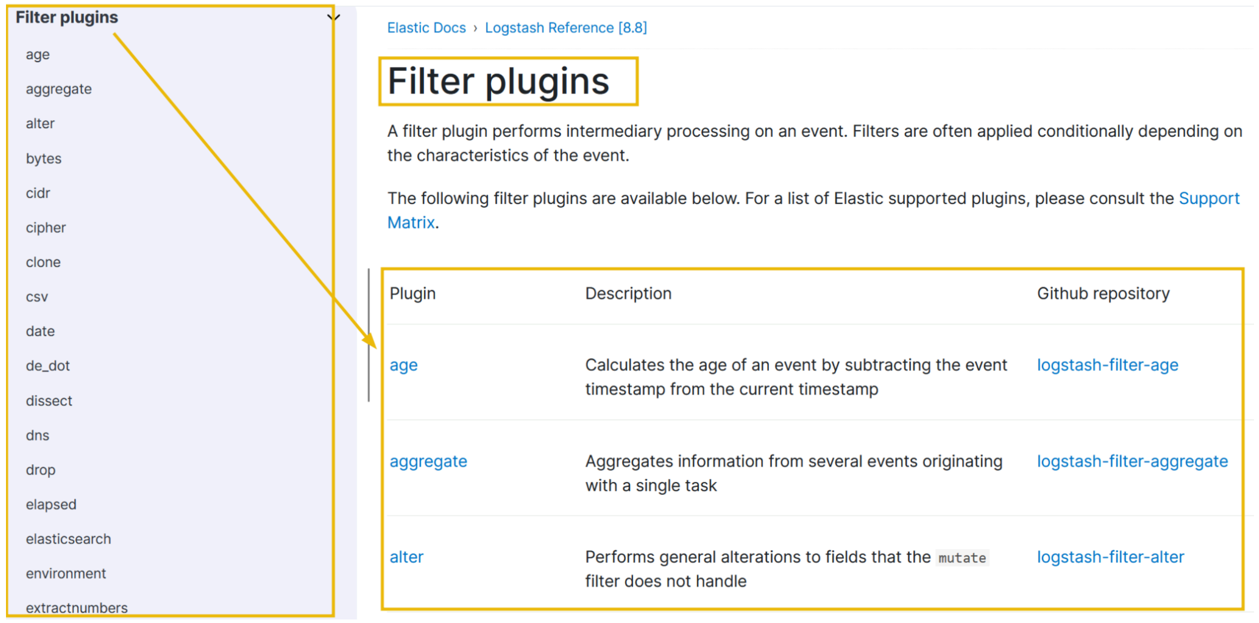
Plugin	Description	Github repository
azure_event_hubs	Receives events from Azure Event Hubs	azure_event_hubs
beats	Receives events from the Elastic Beats framework	logstash-input-beats
cloudwatch	Pulls events from the Amazon Web Services CloudWatch API	logstash-input-cloudwatch

Below is the list of the top 10 input plugins with a brief explanation:

Plugin Name	**Description**
File	Reads data from files in real-time or as a batch. Useful for ingesting log files or other structured data stored on the local file system.
Beats	Ingests data from Beats shippers (such as Filebeat or Metricbeat), lightweight agents designed to ship various data types to Logstash.
TCP	Listens for incoming TCP connections and reads data from them. Useful for receiving data from network devices or other systems over TCP.
UDP	Listens for incoming UDP packets and reads data from them. Ideal for receiving log data or other types of data sent over UDP.
Syslog	Collects logs sent in the Syslog format over UDP or TCP and is commonly used for gathering logs from various network devices or applications.
HTTP	Acts as a web server and reads data from HTTP requests. Useful

	for receiving data from webhooks, REST APIs, or other HTTP-based sources
JMX	Monitors Java applications by connecting to Java Management Extensions (JMX) and collecting metrics and other data.
SNMP Traps	Receives SNMP traps, which are notifications sent by network devices to alert management systems of specific events.
RabbitMQ	Consumes messages from a RabbitMQ message broker. Enables Logstash to receive data from other systems via RabbitMQ.
Amazon S3	Downloads objects from Amazon S3 buckets. Useful for ingesting data stored in S3, such as log files or other files in various formats.

Filter
Once the data is ingested, Logstash provides a rich set of filter plugins that enable you to manipulate and transform the data. Filters allow you to parse, enrich, modify, and filter incoming data. You can perform operations like extracting specific fields, converting data formats, applying regular expressions, adding timestamps, and enriching data with external sources. Logstash supports many filter plugins, as shown in the reference documentation.



Filter plugins

A filter plugin performs intermediary processing on an event. Filters are often applied conditionally depending on the characteristics of the event.

The following filter plugins are available below. For a list of Elastic supported plugins, please consult the Support Matrix.

Plugin	Description	Github repository
age	Calculates the age of an event by subtracting the event timestamp from the current timestamp	logstash-filter-age
aggregate	Aggregates information from several events originating with a single task	logstash-filter-aggregate
alter	Performs general alterations to fields that the `mutate` filter does not handle	logstash-filter-alter

It's important to note that the filter part is optional. Whether we want to apply filters or not depends on the use case. Below is the list of the top 10 input plugins with a brief explanation:

Plugin Name	**Description**
Grok	Parses unstructured log data using custom patterns and extracts structured fields from it.
Mutate	Performs various mutations on event fields, such as renaming, removing, converting data types, and more.
Date	Parses and manipulates dates and timestamps in event fields. Allows you to extract, format, or convert timestamps to a desired configuration.
JSON	Parses JSON-encoded strings in event fields and converts them into structured data. Useful for working with JSON log files or messages.
CSV	Parses comma-separated values (CSV) data in event fields and converts them into structured data.
GeoIP	Enriches IP addresses in event fields with geographical information, such as country, city, latitude, and longitude.
UserAgent	Parses User-Agent strings in event fields and extracts information about the client or device making the request.
Drop	Drops events that match specific conditions. Useful for filtering out unwanted events based on certain criteria.
Translate	Translates values in event fields based on defined mappings. It can be used for data normalization or mapping codes to meaningful values.
DNS	Performs DNS (Domain Name System) lookups on event fields containing IP addresses or hostnames and provides additional information.

Output
After the data is processed through filters, Logstash provides output plugins to send the transformed data to different destinations. These destinations can include Elasticsearch for indexing and search, various databases, message queues, cloud storage services, monitoring systems, and more. The reference documentation shows that Logstash supports various output plugins to accommodate different integration requirements.

We can use one or more output destinations depending on the requirement. Some of the common output plugins are explained below:

Plugin Name	**Description**
Elasticsearch	Sends events to Elasticsearch, a popular search and analytics engine. Allows you to index and store data for further analysis and search.
File	Writes events to files on the local file system or a network-mounted file system. Useful for storing data locally or archiving log files.
STDOUT	Prints events to the console (standard output). Useful for debugging or quick data inspection.
Kafka	Produces messages to an Apache Kafka message broker. Enables Logstash to send data to other systems via Kafka.
Redis	Pushes events to a Redis data structure server. Useful for publishing data to other systems or building real-time dashboards.
Amazon S3	Uploads events to Amazon S3 buckets. Useful for storing data in S3 for long-term storage or backup purposes.
Graphite	Sends events to a Graphite server, commonly used for real-time graphing and metrics monitoring.
StatsD	Sends metrics data to a StatsD server for aggregation and monitoring. They are commonly used in conjunction with Graphite or other monitoring tools.
Logz.io	Ships events to Logz.io, a cloud-based log management and analytics platform. Suitable for centralized log analysis and monitoring.
InfluxDB	Writes events to an InfluxDB time-series database. Useful for storing and analyzing timestamped data, such as metrics or sensor readings.

In the coming tasks, we will use most of these plugins to get the data, parse, and output to the destination of choice.

****

Answer the questions below:

Which plugin is used to perform mutation of event fields?
Answer: mutate

Which plugin can be used to drop events based on certain conditions?
Answer: drop

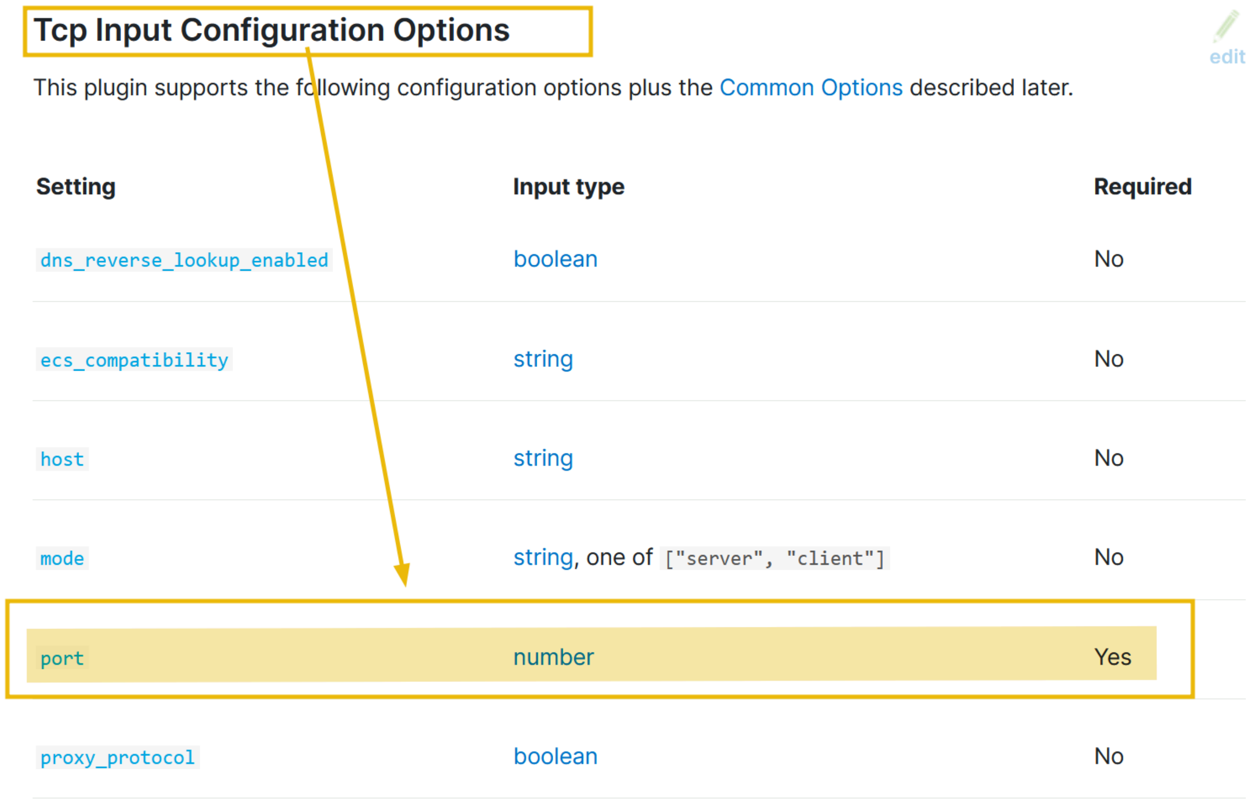
Which plugin parses unstructured log data using custom patterns?
Answer: Grok

Writing Configurations

While setting up Logstash, we made the changes in the Logstash configuration file logstash.yml to look at the configurations every 3 seconds to observe the changes to the files it monitors. The configuration files are located in the /etc/logstash/conf.d/ directory. Let's now go through the process of writing a simple configuration file.

Example:
Take the input from TCP port 5456, which sends JSON formatted logs, apply the appropriate filter, and send it to Elasticsearch.

1) Input: tcp
For the above example, we will take a look at the TCP plugin documentation. In the configuration options, only the port field is required, which means we can use the TCP plugin by only mentioning the TCP port, which we want the Logstash to listen to.



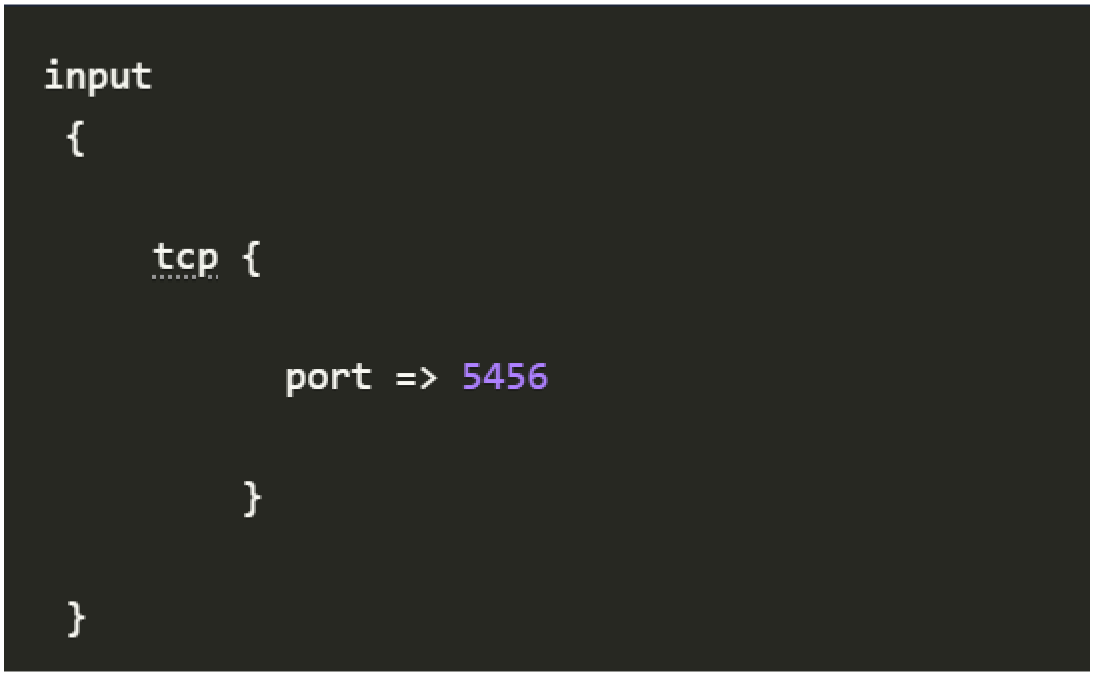
Tcp Input Configuration Options

edit

This plugin supports the following configuration options plus the Common Options described later.

Setting	Input type	Required
`dns_reverse_lookup_enabled`	boolean	No
`ecs_compatibility`	string	No
`host`	string	No
`mode`	string, one of `["server", "client"]`	No
`port`	number	Yes
`proxy_protocol`	boolean	No

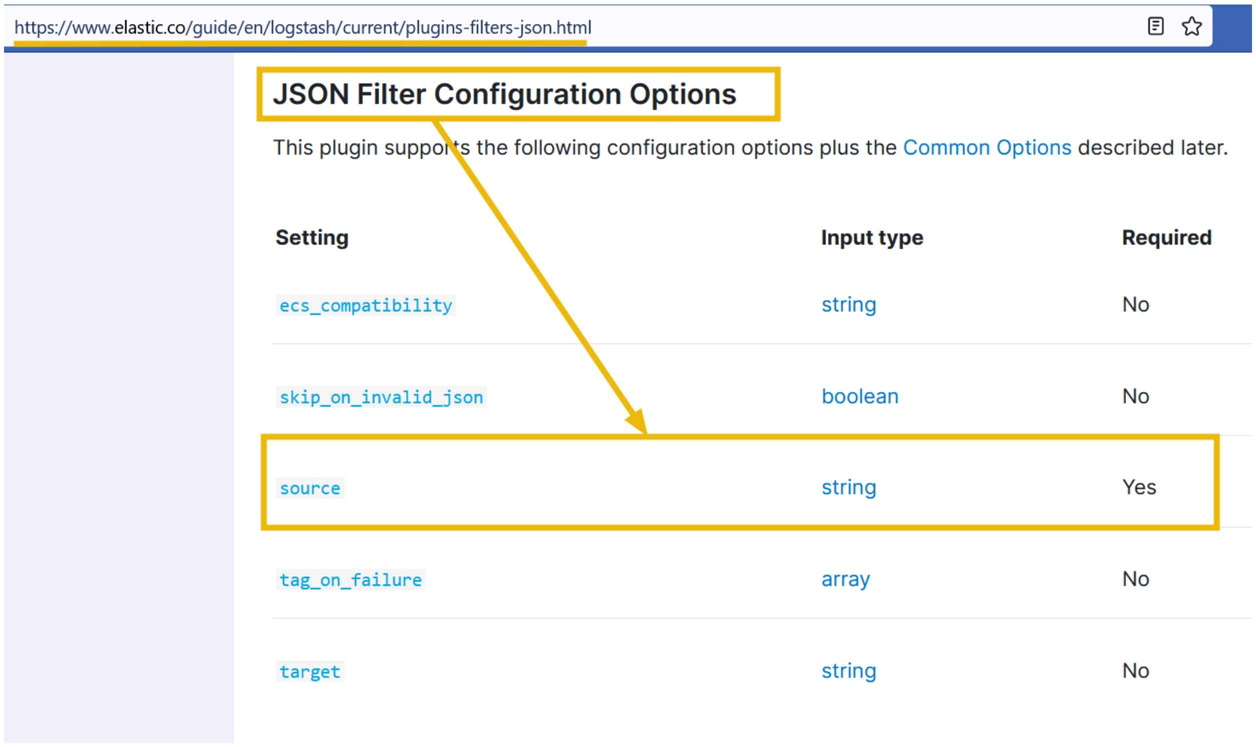
The input part of the configuration based on the above information would look like:



```
input
 {

     tcp {

           port => 5456

         }

 }
```

It is important to note that, in the configurations, we will use => the field value pairs, as shown in the above example.

2) Filter
The input stream is in JSON format; let's explore the JSON plugin documentation and see which fields are required.


https://www.elastic.co/guide/en/logstash/current/plugins-filters-json.html

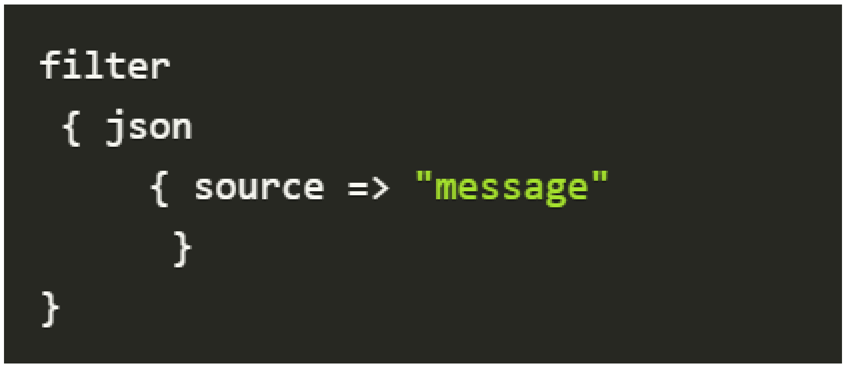
JSON Filter Configuration Options

This plugin supports the following configuration options plus the Common Options described later.

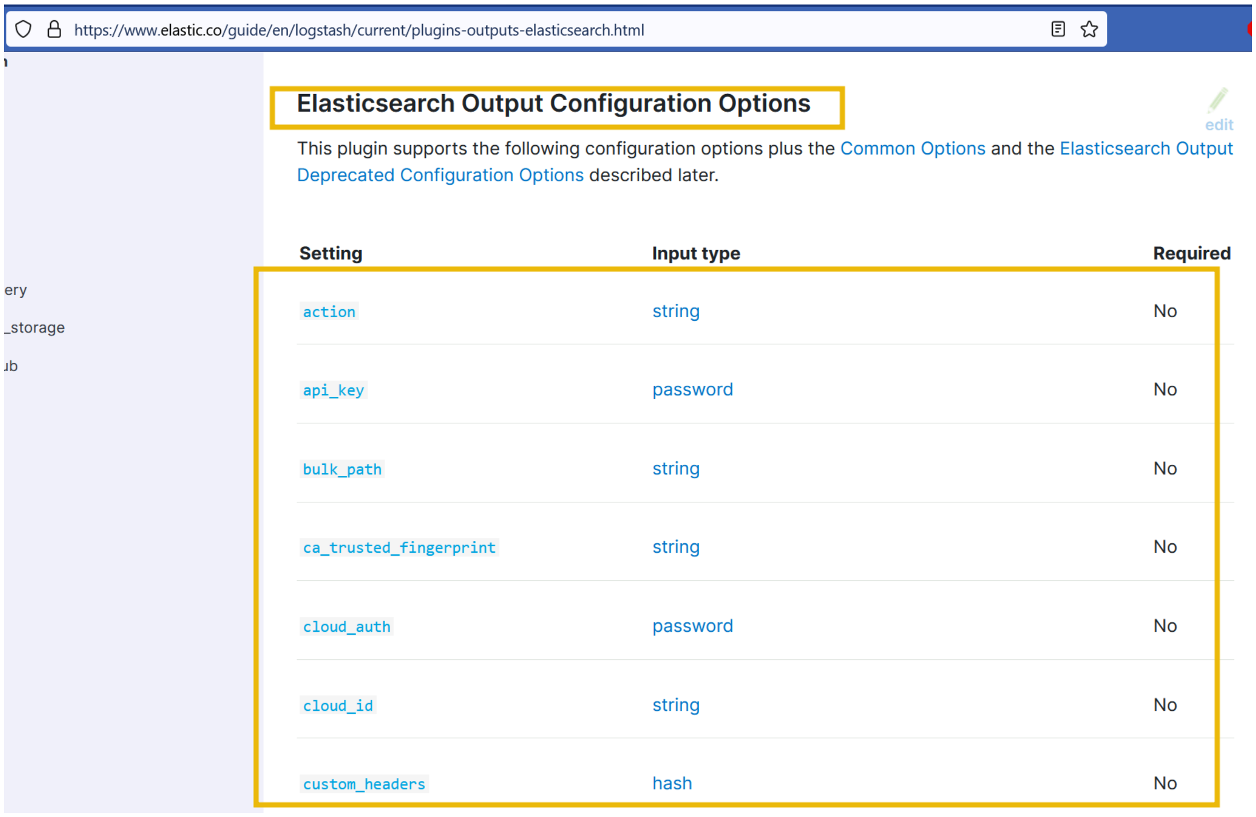
Setting	Input type	Required
`ecs_compatibility`	string	No
`skip_on_invalid_json`	boolean	No
`source`	string	Yes
`tag_on_failure`	array	No
`target`	string	No

The configuration shows we need to mention the source. Click on the source field, and it will show further details on how to use this option, along with the example.

Based on the information provided above, our filter configuration would be:


```
filter
 { json
     { source => "message"
      }
}
```

3) Output
As we want to send our output to Elasticsearch that we configured earlier, let's look at the [Elasticsearch output configuration](#).


https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html

ery

_storage

ub

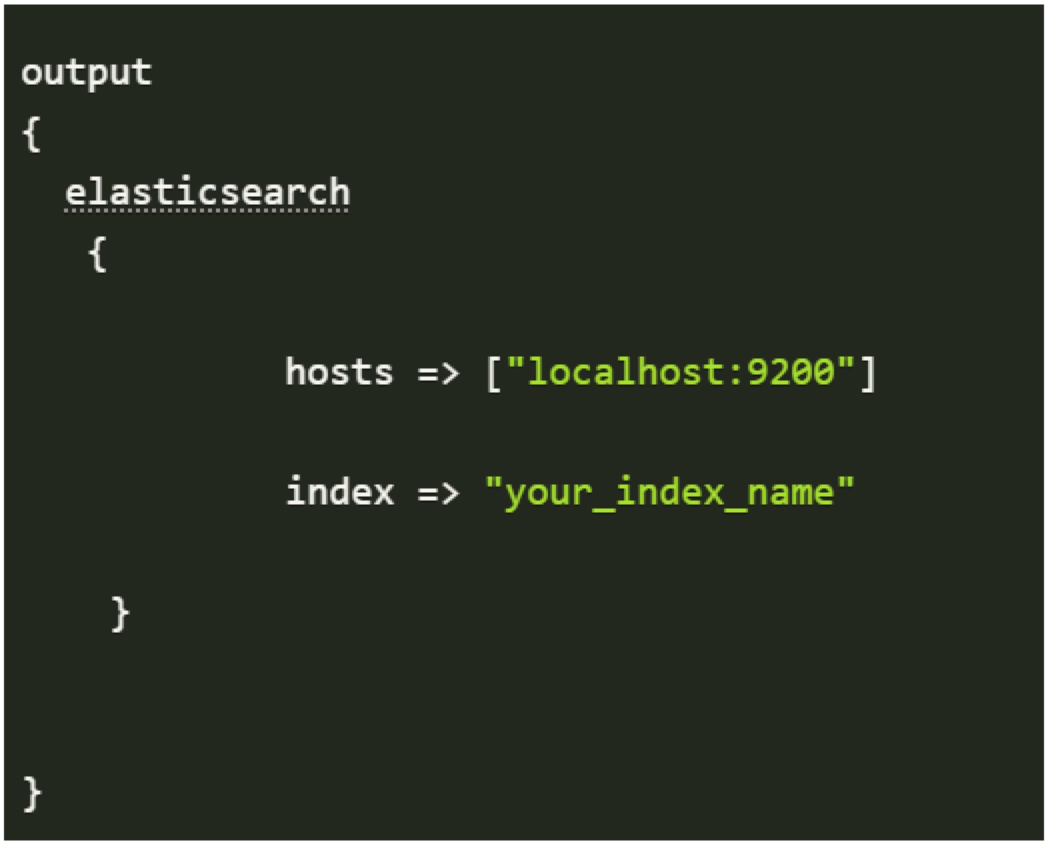
edit

Elasticsearch Output Configuration Options

This plugin supports the following configuration options plus the Common Options and the Elasticsearch Output Deprecated Configuration Options described later.

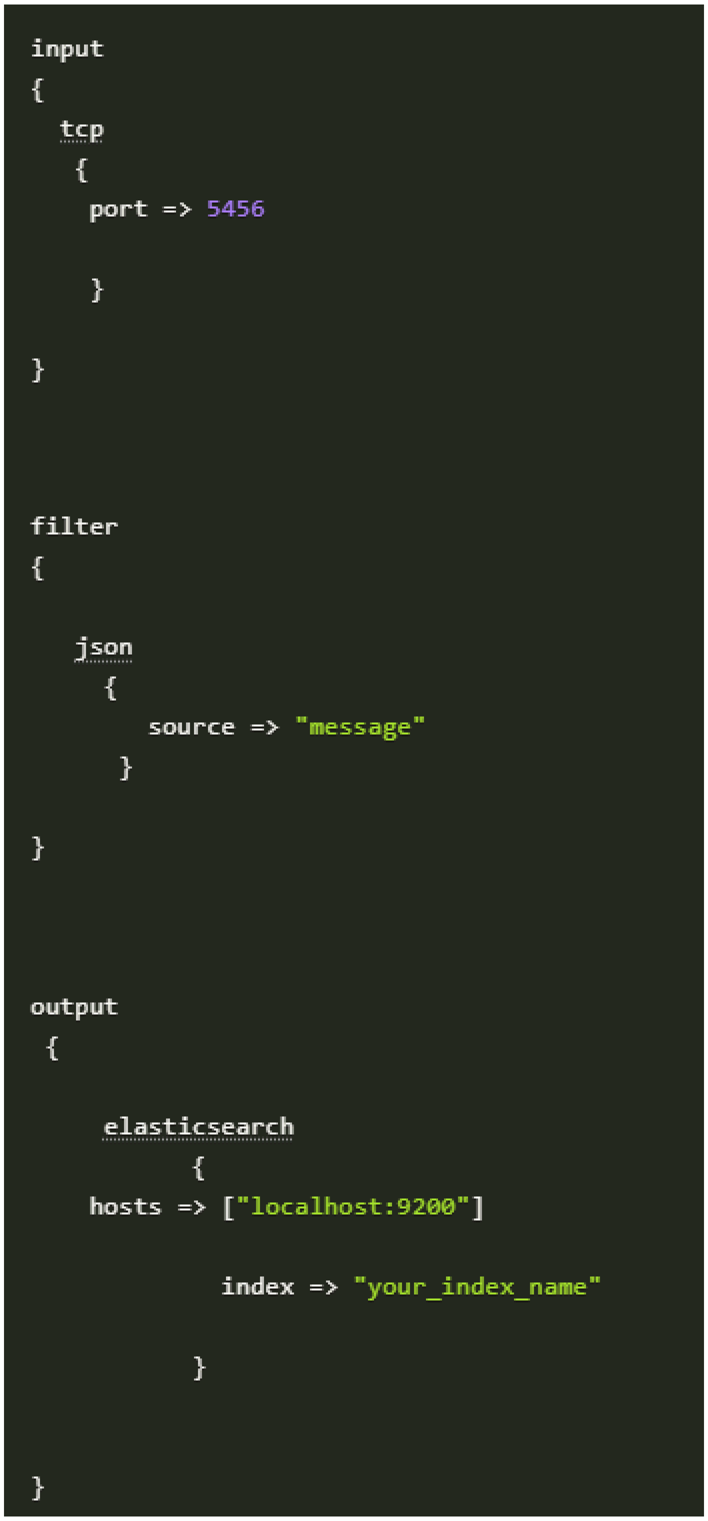
Setting	Input type	Required
`action`	string	No
`api_key`	password	No
`bulk_path`	string	No
`ca_trusted_fingerprint`	string	No
`cloud_auth`	password	No
`cloud_id`	string	No
`custom_headers`	hash	No

The configuration shows that no field is mandatory. But still, we will use the fields like host and index to show the host address and the index where we want the data to be indexed. The final output configuration would look like this:



```
output
{
  elasticsearch
   {

            hosts => ["localhost:9200"]

            index => "your_index_name"

    }


}
```

Final Configuration:
Now that we have written the configuration that takes the input from the TCP port 5456, applies the filter and sends it to Elasticsearch, our final configuration would look like this:



```
input
{
  tcp
   {
    port => 5456

    }

}



filter
{

   json
     {
        source => "message"
      }

}



output
 {

     elasticsearch
           {
    hosts => ["localhost:9200"]

             index => "your_index_name"

           }


}
```

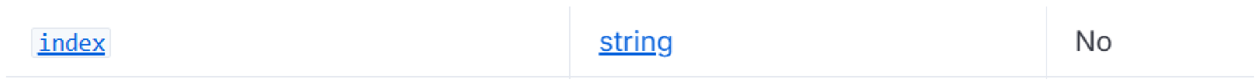
What’s next? We will save this configuration with the extension <any name>.conf into the path /etc/logstash/conf.d/ to make it work.

In the next task, we will create some configurations using different plugins.

****

Answer the questions below:

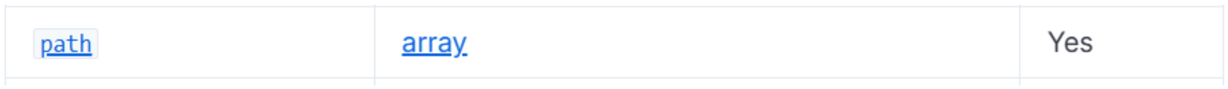
Is index a mandatory option in Elasticsearch plugin? (yay / nay)
From the documentation we see that index is not mandatory.



`index`	string	No

Answer: nay

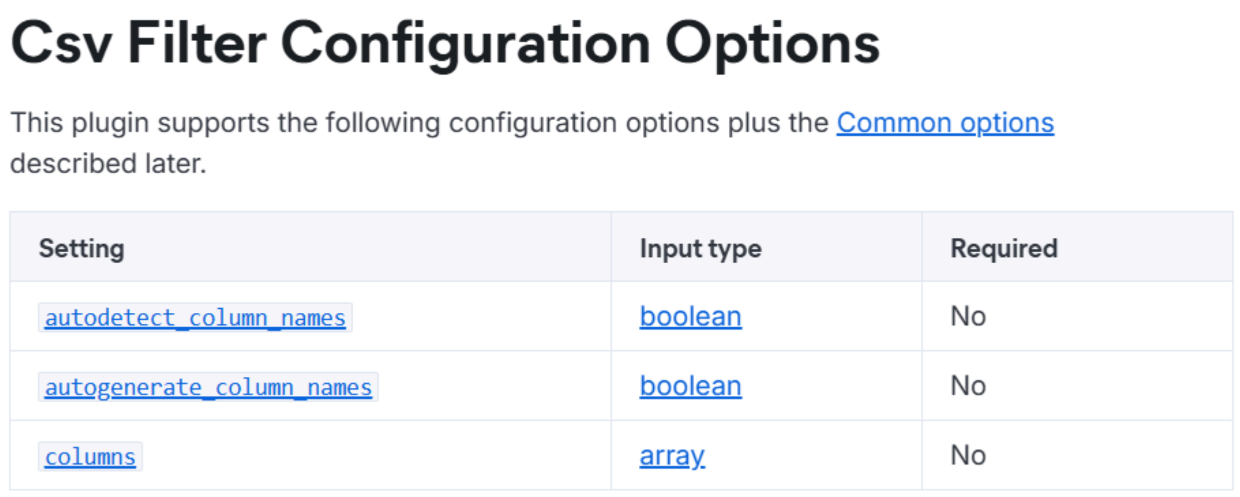
Look at the file input plugin documentation; which field is required in the configuration?
Looking at the documentation for the file input we see path is required in the configuration.



`path`	array	Yes

Answer: path

Look at the filter documentation for CSV; what is the third field option mentioned?



Csv Filter Configuration Options

This plugin supports the following configuration options plus the Common options described later.

Setting	Input type	Required
`autodetect_column_names`	boolean	No
`autogenerate_column_names`	boolean	No
`columns`	array	No

Answer: columns

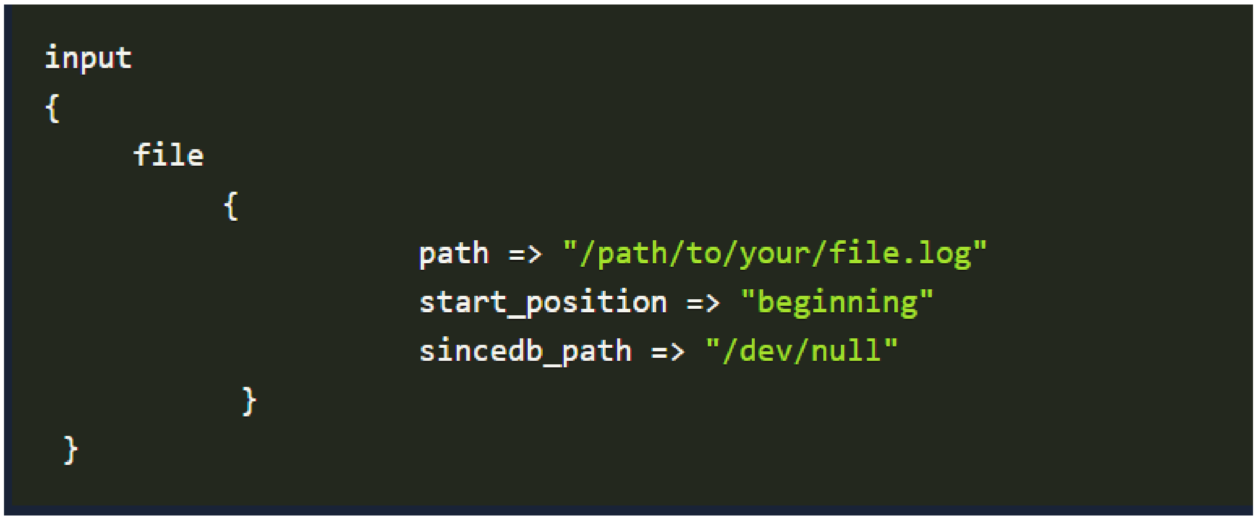
Which output plugin is used in the above example?
Answer: Elasticsearch

Logstash: Input Configurations

So far, we have explored Logstash, its components, and how different plugins are used with the help of official documentation. Let's explore a few common input plugins and their usages, along with the corresponding configurations:

File Input Plugin

This plugin reads data from files on the local system or network.

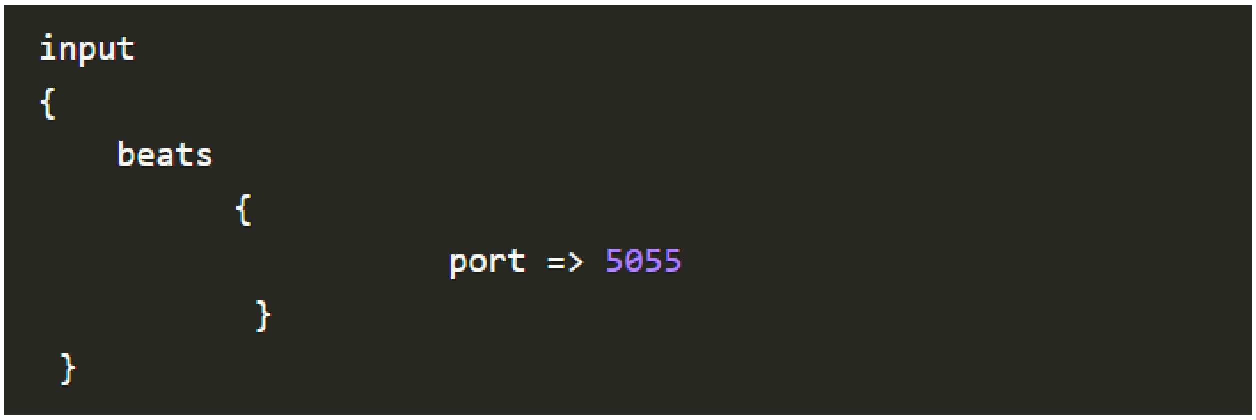


```
input
{
     file
          {
                    path => "/path/to/your/file.log"
                    start_position => "beginning"
                    sincedb_path => "/dev/null"
           }
 }
```

- path: Specifies the path to the file(s) to be read.
- start_position: Defines where Logstash should start reading the file. In this example, "beginning" means it will start reading from the beginning of the file.
- sincedb_path: Sets the path to the sincedb file, which keeps track of the current position of the file read.

Beats Input Plugin

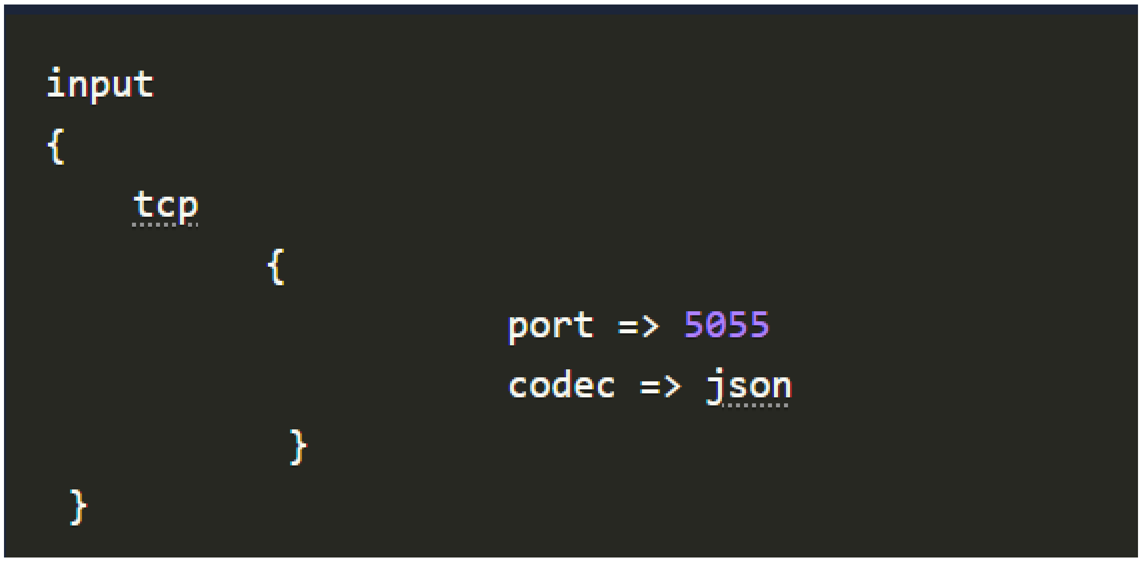
This plugin receives data from Beats, lightweight shippers that send data to Logstash.



```
input
{
    beats
          {
                    port => 5055
           }
 }
```

- port: Specifies the port number on which Logstash should listen for Beats connections.

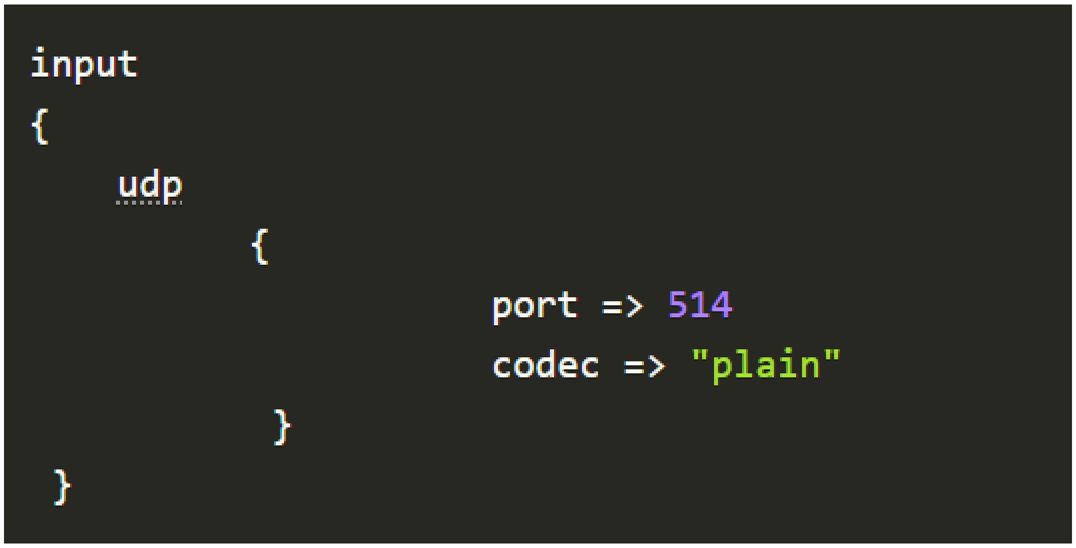
TCP Input Plugin
This plugin listens for data on a specified TCP port.



```
input
{
    tcp
          {
                      port => 5055
                      codec => json
           }
 }
```

- port: Sets the TCP port on which Logstash should listen for incoming data.
- codec: Defines the codec to be used for decoding the incoming data. In this example, the data is expected to be in JSON format.

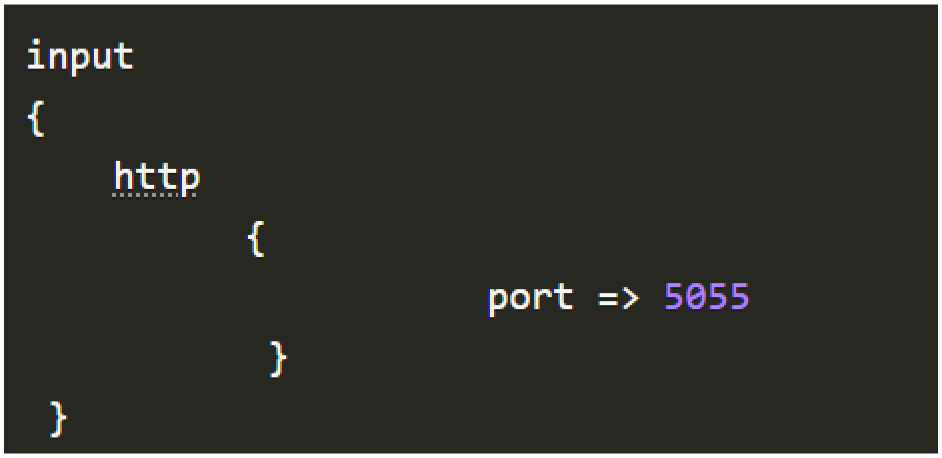
UDP Input Plugin
This plugin listens for data on a specified UDP port.



```
input
{
    udp
          {
                      port => 514
                      codec => "plain"
           }
 }
```

- port: Sets the UDP port on which Logstash should listen for incoming data.
- codec: Specifies the codec to be used for decoding the incoming data. In this example, the data is treated as plain text.

HTTP Input Plugin
This plugin sets up an HTTP endpoint to receive data via HTTP requests.



```
input
{
    http
          {
                     port => 5055
           }
 }
```

- port: Specifies the HTTP port on which Logstash should listen for incoming HTTP requests

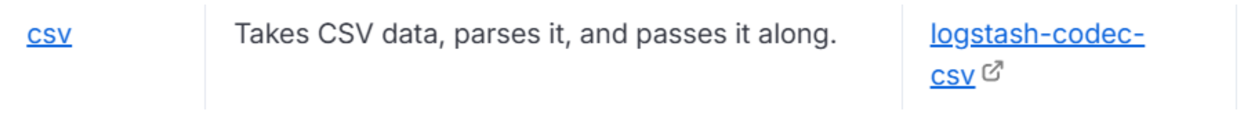
Each input plugin provides specific configuration options tailored to its functionality and requirements. The examples above demonstrate configuring the plugins by specifying the necessary settings, such as file paths, ports, and codecs. You can further customize these configurations based on your specific use case and the data sources you are working with. The Logstash documentation provides comprehensive information on each input plugin and its available configuration options.

****

Answer the questions below:

If we want to read a constant stream from a TCP port 5678, which input plugin will be used?
Answer: TCP

According to the Logstash documentation, the codec plugins are used to change the data representation. Which codec plugin is used for the CSV based data representation?



csv	Takes CSV data, parses it, and passes it along.	logstash-codec-csv

Answer: CSV

Logstash: Filter Configurations

Filters

Let’s now explore different filter plugins and how they are used in Logstash configuration with some examples:

Adding a field using the mutate filter

```
filter
{
    mutate
    {
                add_field => { "new_field" => "new_value" }
    }
 }
```

Explanation: This configuration utilizes the mutate filter to add a new field called new_field with the value "new_value" to each event. The mutate filter provides various operations for modifying fields.

Converting a field to lowercase using the mutate filter

```
filter
{
    mutate
    {
                lowercase => ["field_name"]   }
    }
 }
```

Explanation: This configuration employs the mutate filter to convert the value of the field_name field to lowercase. The original value will be replaced with its lowercase equivalent.

Extracting data using the grok filter

```
filter
{
    grok
    {
                match => { "message" => "%{PATTERN:field_name}" }
    }
 }
```

Explanation: This configuration uses the grok filter to extract data from the message field based on a predefined pattern (%{PATTERN}). The extracted data will be stored in a new field called field_name

Removing empty fields using the prune filter

```
filter
{
    prune
    {
                whitelist_names => ["field1", "field2"]   }
    }
 }
```

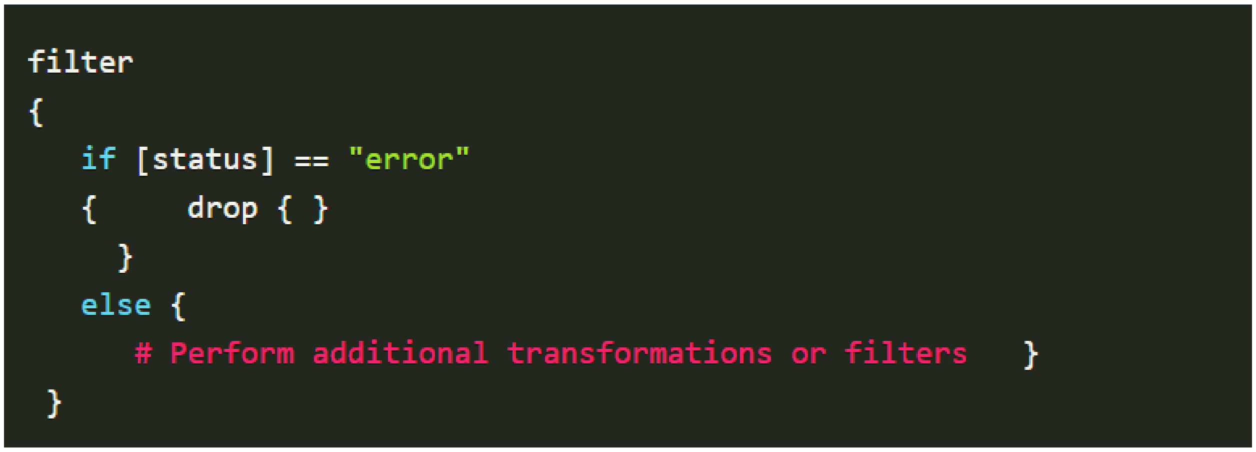
Explanation: This configuration utilizes the prune filter to remove empty fields from the event, except for those specified in the whitelist_names parameter. Only fields field1 and field2 will be retained in the event.

Replacing field values using the translate filter

```
filter
{
    translate
    {
                field => "country"
                destination => "country_name"
                dictionary => {      "US" => "United States"   "CA" => "Canada"   }
    }
 }
```

Explanation: This configuration utilizes the translate filter to replace the value of the country field with its corresponding value from the dictionary. For example, if the country field is "US", it will be replaced with "United States", and if it is "CA", it will be replaced with "Canada".

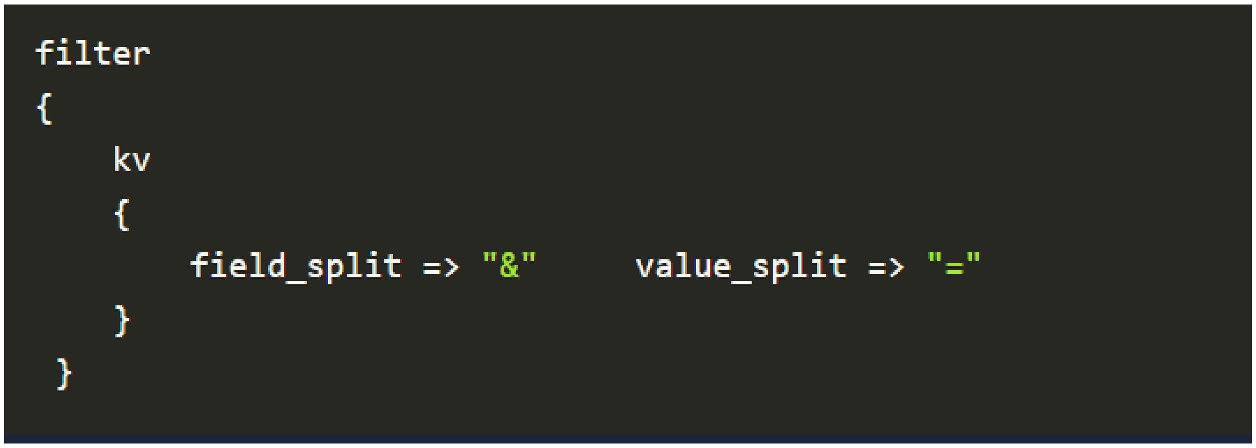
Dropping events based on a condition using the if-else statement



```
filter
{
   if [status] == "error"
   {     drop { }
     }
   else {
      # Perform additional transformations or filters    }
 }
```

Explanation: In this configuration, the if-else statement checks the value of the status field. If the value is "error", the event is dropped using the drop filter. Otherwise, if the condition is not met, the event proceeds for further transformations or filters.

Parsing key-value pairs using the kv filter



```
filter
{
    kv
    {
        field_split => "&"     value_split => "="
    }
 }
```

Explanation: This configuration uses the kv filter to parse key-value pairs in a field. The field_split parameter specifies the delimiter between key-value pairs (& in this case), and the value_split parameter specifies the delimiter between keys and values (= in this case).

Performing conditional field renaming using the rename filter

```
filter
{  mutate {
    rename
    {
                "old_field" => "new_field"
                add_field => { "another_field" => "value" }
    }
 }}
```

Explanation: This configuration employs the rename filter to rename the field old_field to new_field. Additionally, it adds a new field called another_field with the value "value". The rename filter allows for conditional field renaming.

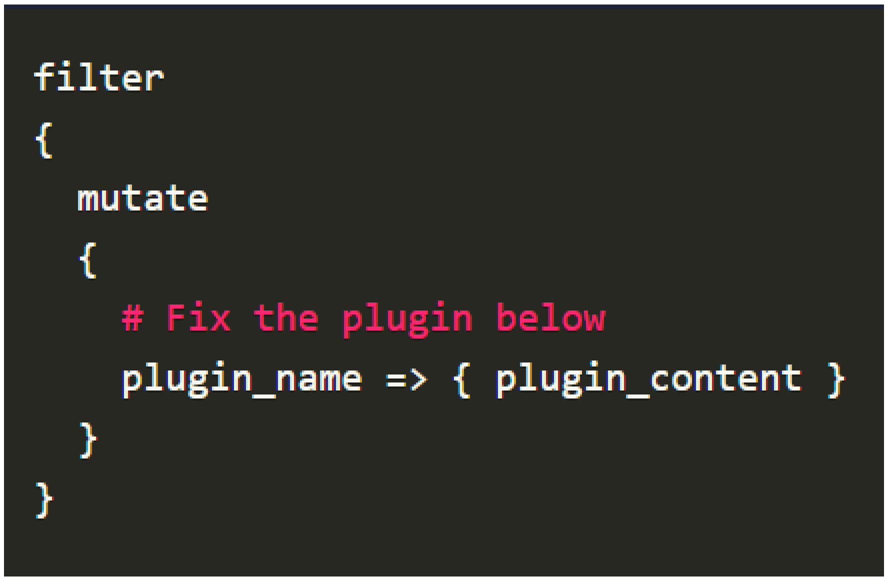
****

Answer the questions below:

Which filter plugin is used to remove empty fields from the events?
Answer: prune

Which filter plugin is used to rename, replace, and modify event fields?
Answer: mutate

Add a filter plugin to rename the field " src_ip " to " source_ip ".



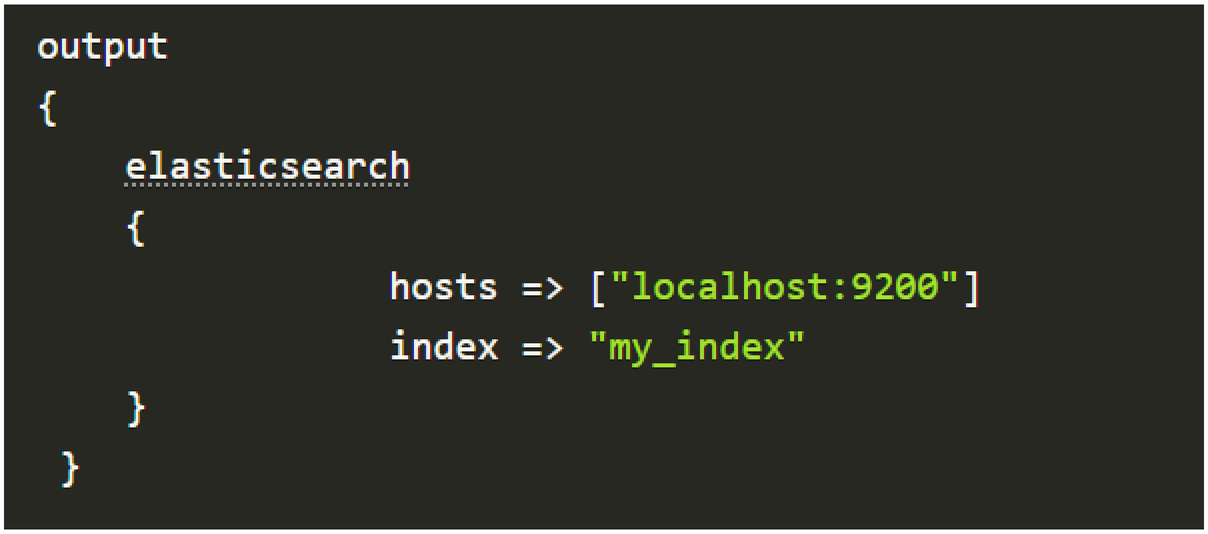
```
filter
{
  mutate
  {
    # Fix the plugin below
    plugin_name => { plugin_content }
  }
}
```

Provide the fixed line as an answer.
Answer: rename => { "src_ip" => "source_ip" }

Logstash: Output Plugins

Output
The output section of the Logstash configuration file is where you define the destination for processed events. It specifies where Logstash should send the transformed data. Some of the common examples of Logstash output plugins and their configurations:

Sending data to Elasticsearch using the elasticsearch output plugin



```
output
{
    elasticsearch
    {
                hosts => ["localhost:9200"]
                index => "my_index"
    }
 }
```

Explanation: This configuration uses the elasticsearch output plugin to send events to an Elasticsearch cluster. The hosts parameter specifies the Elasticsearch server's address and defines the index name where the data will be stored.

Writing data to a file using the file output plugin

```
output
{
    file
    {
                path => "/path/to/output.txt"
    }
 }
```

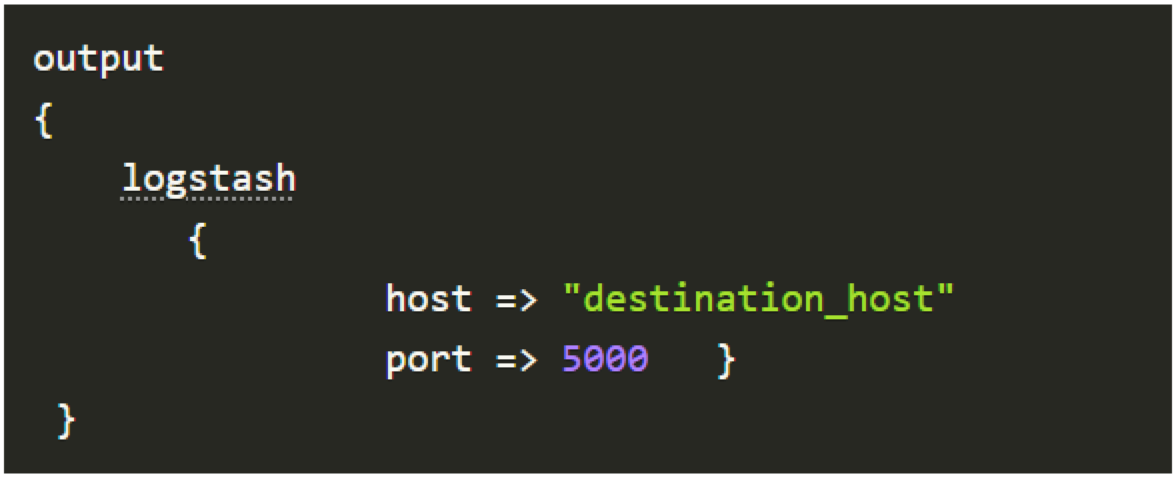
Explanation: This configuration utilizes the file output plugin to write events to a specified file (/path/to/output.txt). Each event will be appended to the file as a separate line.

Sending data to a message queue using the rabbitmq output plugin

```
output
{
   rabbitmq
   {
               host => "localhost"
               exchange => "my_exchange"
               routing_key => "my_routing_key"
    }
 }
```

Explanation: This configuration configures the rabbitmq output plugin to send events to a RabbitMQ server. The host parameter specifies the RabbitMQ server's address, while exchange and routing_key define the exchange and routing key for message routing within RabbitMQ.

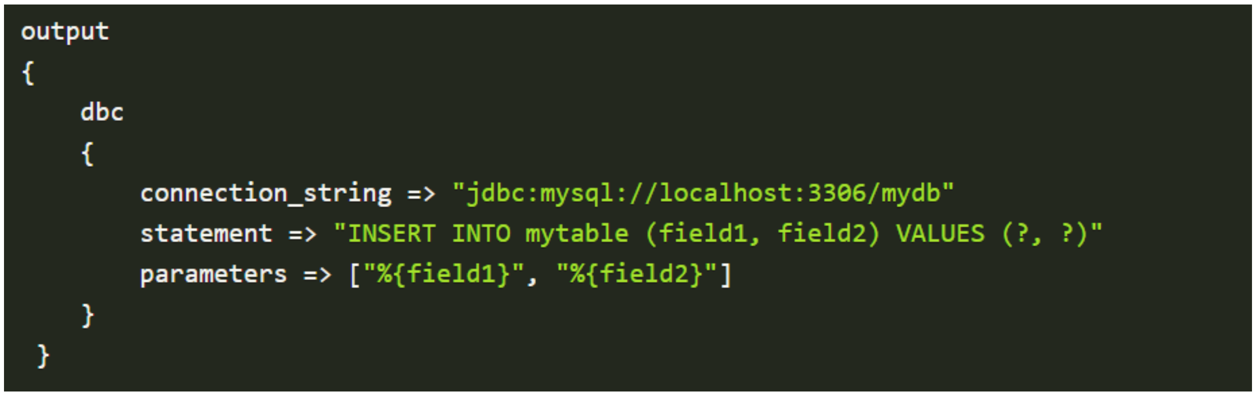
Forwarding data to another Logstash instance using the logstash output plugin



```
output
{
    logstash
       {
                host => "destination_host"
                port => 5000    }
 }
```

Explanation: This configuration sets up the logstash output plugin to forward events to another Logstash instance. The host parameter specifies the destination Logstash server’s address, and the port parameter defines the port to which the events will be sent.

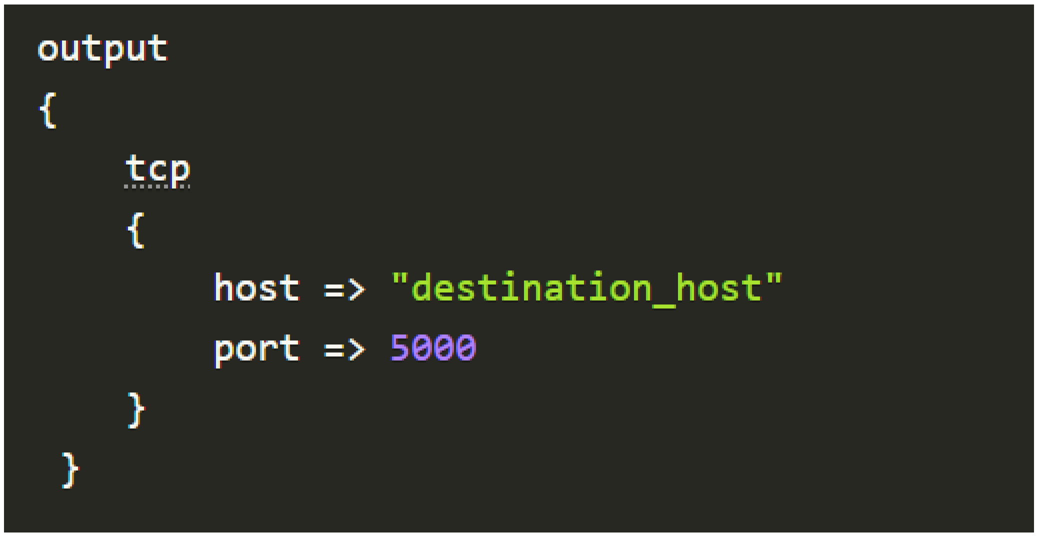
Sending data to a database using the jdbc output plugin



```
output
{
    dbc
    {
        connection_string => "jdbc:mysql://localhost:3306/mydb"
        statement => "INSERT INTO mytable (field1, field2) VALUES (?, ?)"
        parameters => ["%{field1}", "%{field2}"]
    }
 }
```

Explanation: This configuration utilizes the jdbc output plugin to insert events into a MySQL database. The connection_string parameter specifies the database connection details, and the statement parameter defines the SQL INSERT statement. The parameters parameter provides values for the placeholders in the statement.

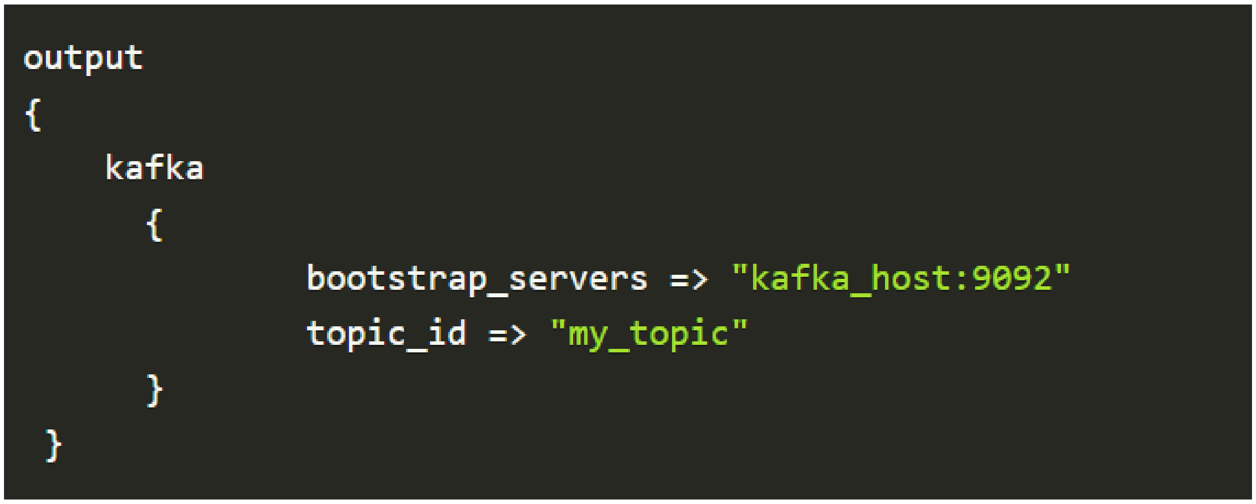
Forwarding data to a TCP server using the tcp output plugin



```
output
{
    tcp
    {
        host => "destination_host"
        port => 5000
    }
 }
```

Explanation: This configuration sets up the tcp output plugin to forward events to a remote TCP server. The host parameter specifies the destination server’s address, and the port parameter defines the port to which the events will be sent.

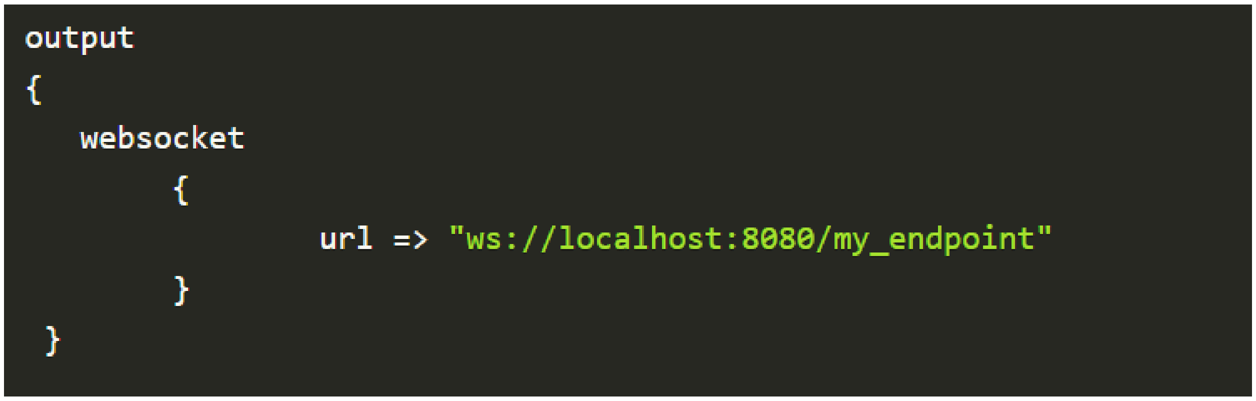
Sending data to a message broker using the kafka output plugin



```
output
{
    kafka
      {
              bootstrap_servers => "kafka_host:9092"
              topic_id => "my_topic"
      }
 }
```

Explanation: This configuration utilizes the kafka output plugin to publish events to an Apache Kafka topic. The bootstrap_servers parameter specifies the Kafka server’s address, and the topic_id parameter defines the topic to which the events will be sent.

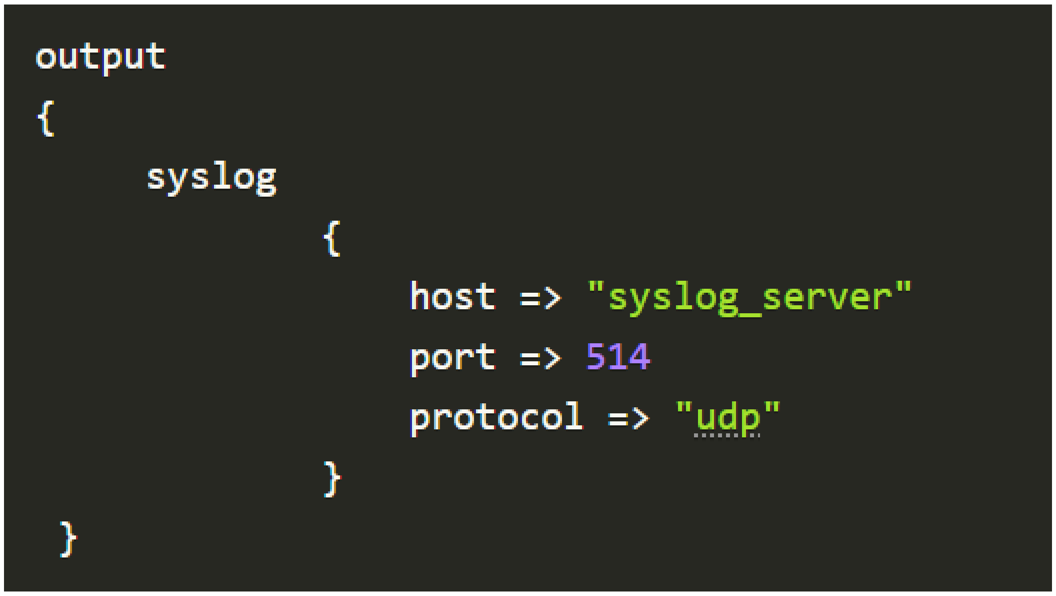
Forwarding data to a WebSocket endpoint using the websocket output plugin



```
output
{
   websocket
        {
                url => "ws://localhost:8080/my_endpoint"
        }
 }
```

Explanation: This configuration creates the websocket output plugin to send events to a WebSocket endpoint. The url parameter specifies the WebSocket server's URL and the specific endpoint (/my_endpoint) to which the events will be sent.

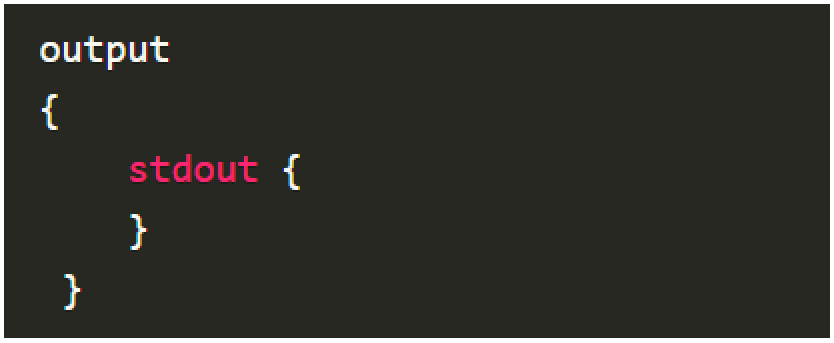
Sending data to a Syslog server using the syslog output plugin



```
output
{
      syslog
              {
                   host => "syslog_server"
                   port => 514
                   protocol => "udp"
              }
 }
```

Explanation: This configuration configures the syslog output plugin to send events to a Syslog server. The host parameter specifies the Syslog server's address, the port parameter defines the port to which the events will be sent, and the protocol parameter specifies the transport protocol (in this case, UDP).

Writing data to the console for debugging using the stdout output plugin



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

Explanation: This configuration uses the stdout output plugin to print events to the console. It is primarily used for debugging purposes to visually inspect the processed events and verify the Logstash pipeline’s transformations.

These examples demonstrate a variety of output plugins available in Logstash, each serving a specific purpose to send data to different destinations or systems. We can choose the appropriate output plugin based on our requirements and desired data flow.

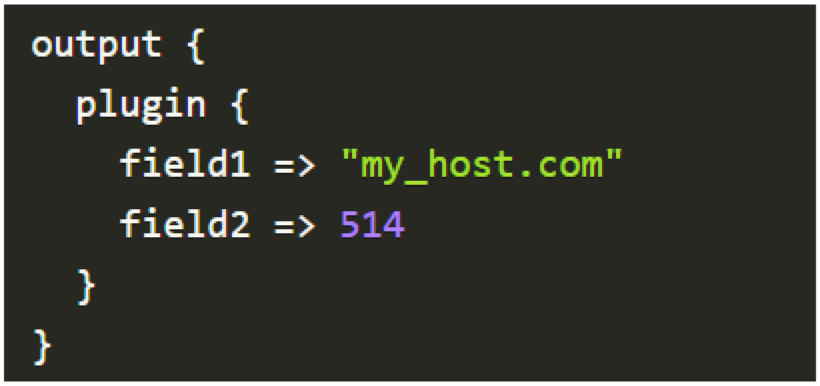
****

Answer the questions below:

Can we use multiple output plugins at the same time in order to send data to multiple destinations? (yay / nay)
Answer: yay

Which Logstash output plugin is used to print events to the console?
Answer: stdout

Update the following configuration to send the event to the Syslog server.

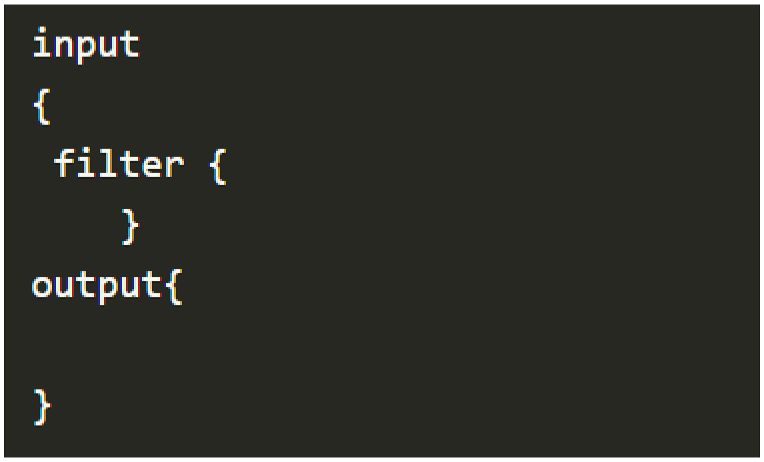


```
output {
  plugin {
    field1 => "my_host.com"
    field2 => 514
  }
}
```

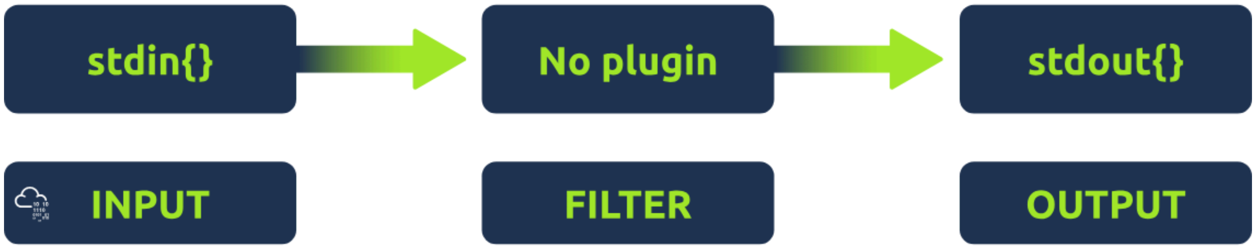
Answer: syslog,host,port

Logstash: Running Configurations

Let’s recall the Logstash components below. This image can be represented in configuration as follows:



```
input
{
 filter {
     }
output{


}
```



Logstash configurations are placed at /etc/logstash/conf.d/ for the logstash to read.

Configuration# 1: Stdin/Stdout

Let’s take a simple illustration below:

As we already know, Logtash provides various input plugins, which can be used to take input from different sources.

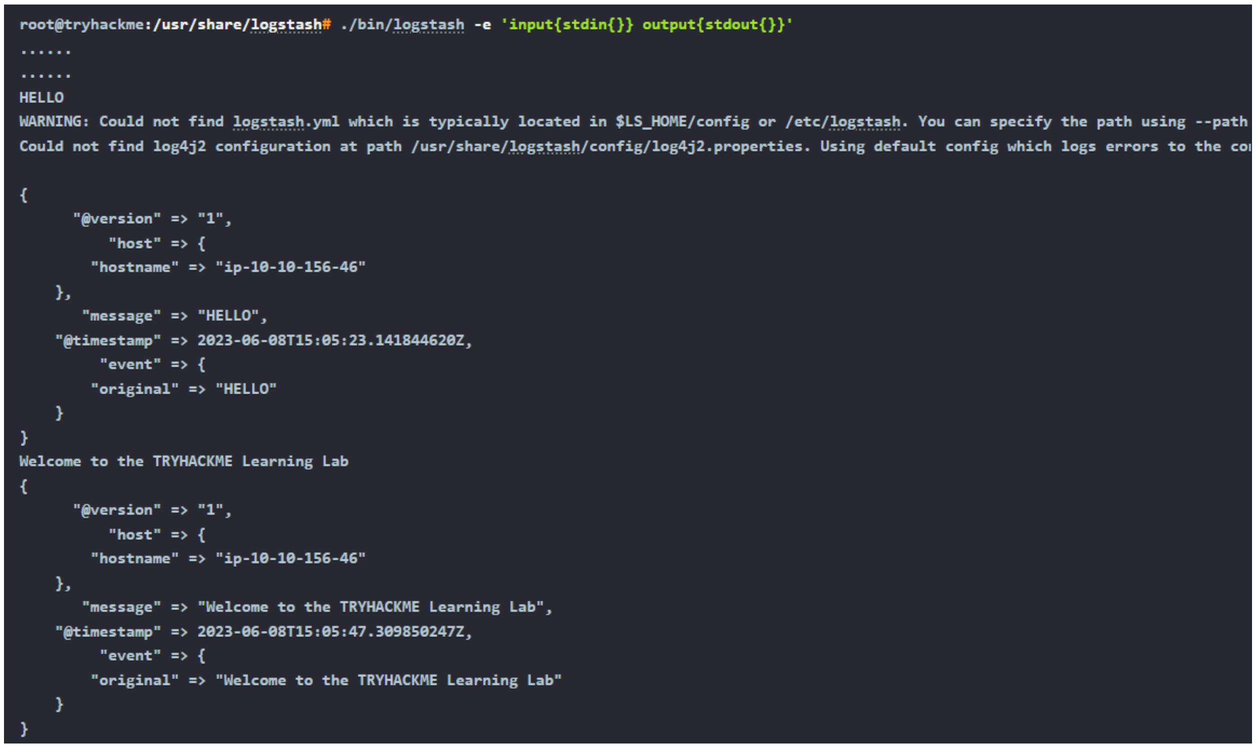
In our first example, we will take the input from the terminal using the stdin plugin and send the output back to the terminal using the output plugin stdout.

The configuration file would look like this:

```
input{

stdin{}

output{
    stdout{}
    }
```

We can skip that because we are not using any filters in this example. Let’s try this out using the following command:

```
/usr/share/logstash/bin/logstash -e 'input{ stdin{}} output{ stdout{} }'
```

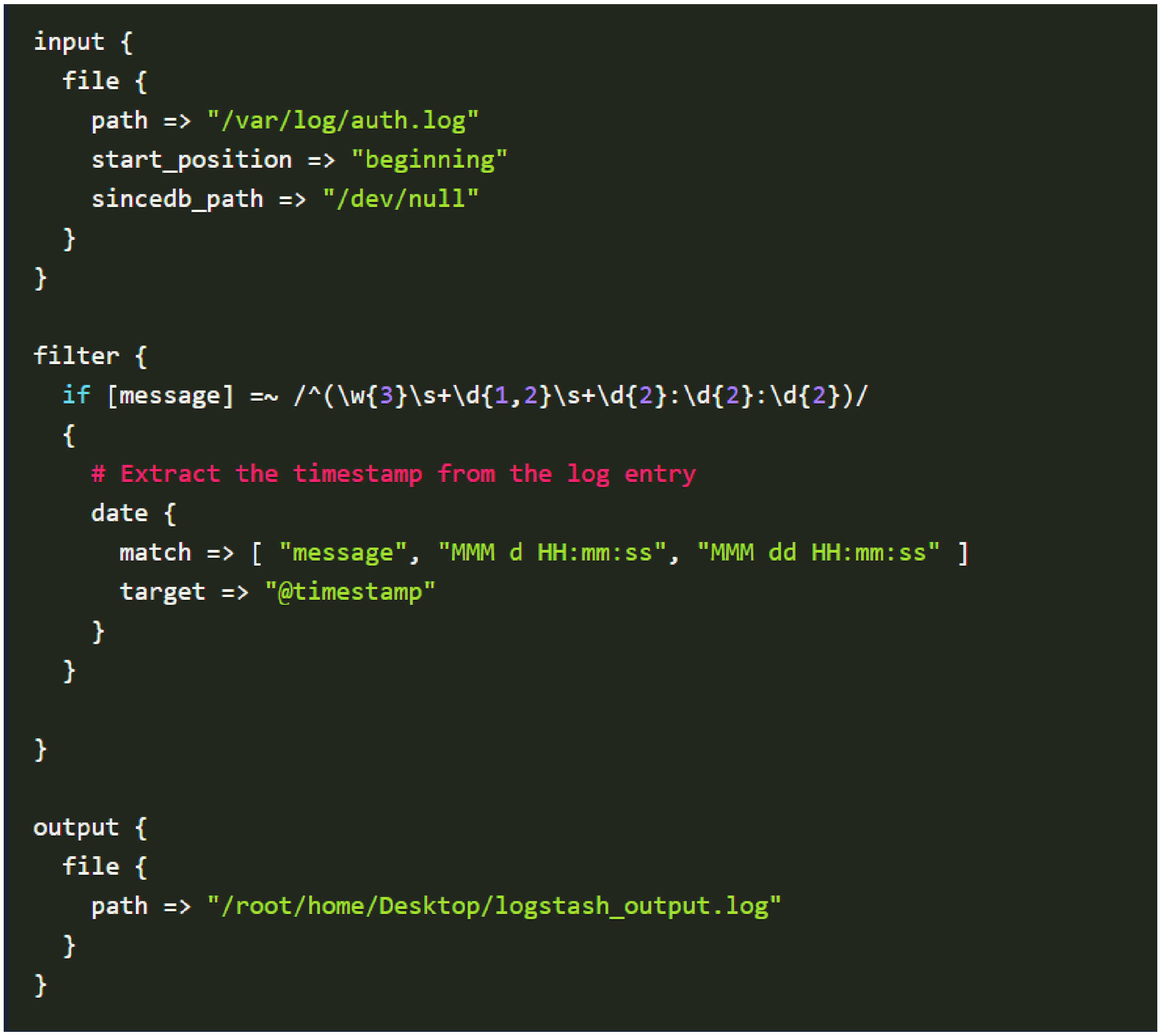


```
root@tryhackme:/usr/share/logstash# ./bin/logstash -e 'input{stdin{}} output{stdout{}}'
......
......
HELLO
WARNING: Could not find logstash.yml which is typically located in $LS_HOME/config or /etc/logstash. You can specify the path using --path
Could not find log4j2 configuration at path /usr/share/logstash/config/log4j2.properties. Using default config which logs errors to the co

{
      "@version" => "1",
          "host" => {
        "hostname" => "ip-10-10-156-46"
    },
       "message" => "HELLO",
    "@timestamp" => 2023-06-08T15:05:23.141844620Z,
         "event" => {
        "original" => "HELLO"
    }
}
Welcome to the TRYHACKME Learning Lab
{
      "@version" => "1",
          "host" => {
        "hostname" => "ip-10-10-156-46"
    },
       "message" => "Welcome to the TRYHACKME Learning Lab",
    "@timestamp" => 2023-06-08T15:05:47.309850247Z,
         "event" => {
        "original" => "Welcome to the TRYHACKME Learning Lab"
    }
}
```

In the above examples, we entered two inputs HELLO and Welcome to the TRYHACKME Learning Lab. Both were reflected back on the terminal, as we used stdout to output the data without applying any filters.

Configuration #2: Reading authentication logs

Now, we will see how we can get authentication logs located at /var/log/auth.log, apply some filters, and send them over to the destination of our choice.



```
input {
  file {
    path => "/var/log/auth.log"
    start_position => "beginning"
    sincedb_path => "/dev/null"
  }
}

filter {
  if [message] =~ /^(\w{3}\s+\d{1,2}\s+\d{2}:\d{2}:\d{2})/
  {
    # Extract the timestamp from the log entry
    date {
      match => [ "message", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
      target => "@timestamp"
    }
  }

}

output {
  file {
    path => "/root/home/Desktop/logstash_output.log"
  }
}
```

Save this configuration to a file called logstash.conf and run the following command from the location:

- */usr/share/bin/logstash*
- *logstash -f logstash.conf*

Logstash will start reading the auth logs, apply the defined filters, and write the filtered logs to the specified file on your Desktop.

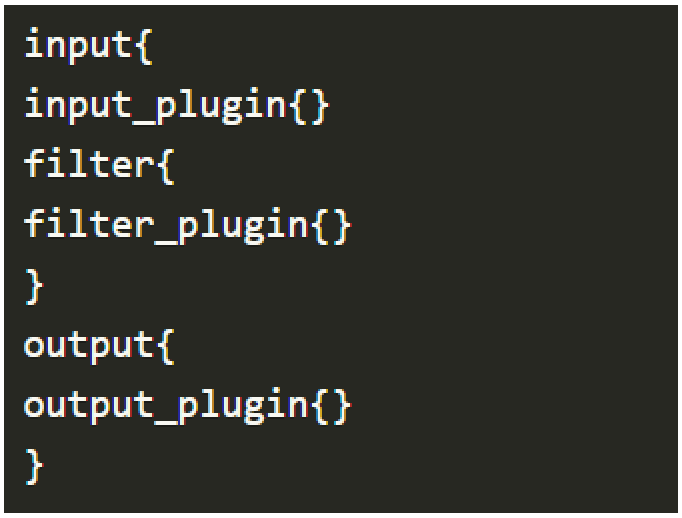
****

Answer the questions below:

Which command is used to run the logstash.conf configuration from the command line?

Answer: logstash -f logstash.conf

What will be the input, filter, and output plugins in the below-mentioned configuration if we want to get the comma-separated values from the command line, and apply the filter and output back to the terminal to test the result?

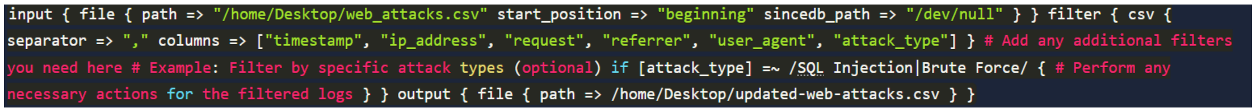


```
input{
input_plugin{}
filter{
filter_plugin{}
}
output{
output_plugin{}
}
```

Answer format: input_plugin,filter_plugin,output_plugin
Answer: stdin,csv,stdout

Conclusion

A simple example is to examine the web-attack.csv file. We will write a configuration, apply a filter to add corresponding columns, and send the output to another file on the Desktop. The configuration file below shows an example of how to do this:



```
input { file { path => "/home/Desktop/web_attacks.csv" start_position => "beginning" sincedb_path => "/dev/null" } } filter { csv {
separator => "," columns => ["timestamp", "ip_address", "request", "referrer", "user_agent", "attack_type"] } # Add any additional filters
you need here # Example: Filter by specific attack types (optional) if [attack_type] =~ /SQL Injection|Brute Force/ { # Perform any
necessary actions for the filtered logs } } output { file { path => /home/Desktop/updated-web-attacks.csv } }
```

Conclusion

Well, that's it from this room. It was an interesting topic to cover. Logstash helps us efficiently examine the logs after parsing and applying necessary filters, as we covered in the room. The topics we covered in this room are:

- How to install and configure ELK components.
- What are different input, filter, and output plugins available?
- Exploring the Logstash documentation.
- How to write simply to complex configurations.

There will be a next part of this room, where we will explore some realistic log sources and examine them in Kibana to visualize better and analyze them.