**Elasticsearch by JavaInUse:**

Elasticsearch is a search engine based on Lucene. It stores the data in the form of Json. We can easily query and search for the data using http call. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License.

Features of Elasticsearch -

* **Real-Time Analysis**- By integrating rapid, high-powered search mechanism with strong analytics features, users are able to have a much better grasp of the nature of data. By finding out more about data, we can build a better business.
* **Scalability**- Elasticsearch is built to be always available, and to scale with your needs. Scale can come from buying bigger servers (vertical scale, or scaling up) or from buying more servers (horizontal scale, or scaling out).
* **Resiliency**- If any node in Elasticsearch becomes non functional due to any reason, there will not be any issue since there are backup nodes.
* **Documents**- Users can store sophisticated business information as a structured JSON document within Elasticsearch. Everything automatically gets integrated into the index. User can check all indices with one request, so that you can quickly answer complicated questions.
* **No Schema**- Getting started with Elasticsearch is very easy. The indexing of JSON document is very easy as program knows to identify the structure and format of the data.
* **RESTful API**- This API is very important part of Elasticsearch. Any task can be done making use of REST API.
* **Open Source License**- Elasticsearch uses the open source Apache 2 license, which allows users to install it, work with it, and customize it completely for free. Apache 2 is one of the most user-friendly licenses available for open source apps.
* **Apache Lucene**- Elasticsearch is built on top of Apache Lucene.Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java.

# Perform basic operations with Elasticsearch:

# In this post we perform the basic CRUD operations using Elasticsearch. For this we make use of the Sense plugin for google chrome.

# How To Use Sense And Head ElasticSearch Chrome Extensions : <https://www.youtube.com/watch?v=RvK8RK6wNdw>

Lets Begin

# We will be performing the following operations- a. INDEXING A DOCUMENT USING ELASTICSEARCH. b. FETCHING DOCUMENTS USING ELASTICSEARCH. c. UPDATING DOCUMENTS USING ELASTICSEARCH. d. DELETING DOCUMENTS USING ELASTICSEARCH.

# 

# 

# 

# 

# 

# 

# 

# 

# Installing the Head Plugin for Elasticsearch.

# 

# 

# Understanding Elasticsearch Cluster, Node, Index and Document using example

# 

# 

# 

# 

# 

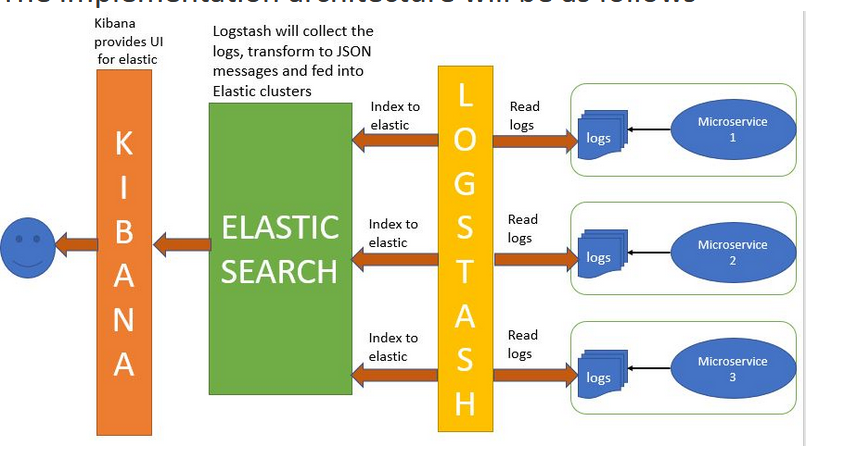
# By default the elastic search creates 5 primary shards and 5 replica shards. But a single node can by default hold only 5 shards so the 5 replica shards could not be created and so they are shown as unassigned. Due to this the cluster health is shown as yellow. The cluster health rule is as follows-

# 

# 

# 

**ELK:**



## What is ELK? Need for it?

The ELK Stack consists of three open-source products - Elasticsearch, Logstash, and Kibana from Elastic.

* Elasticsearch is a NoSQL database that is based on the Lucene search engine.
* Logstash is a log pipeline tool that accepts inputs from various sources, executes different transformations, and exports the data to various targets. It is a dynamic data collection pipeline with an extensible plugin ecosystem and strong Elasticsearch synergy
* Kibana is a visualization UI layer that works on top of Elasticsearch.

These three projects are used together for log analysis in various environments. So Logstash collects and parses logs, Elastic search indexes and store this information while Kibana provides a UI layer that provide actionable insights.

**Use Cases-**

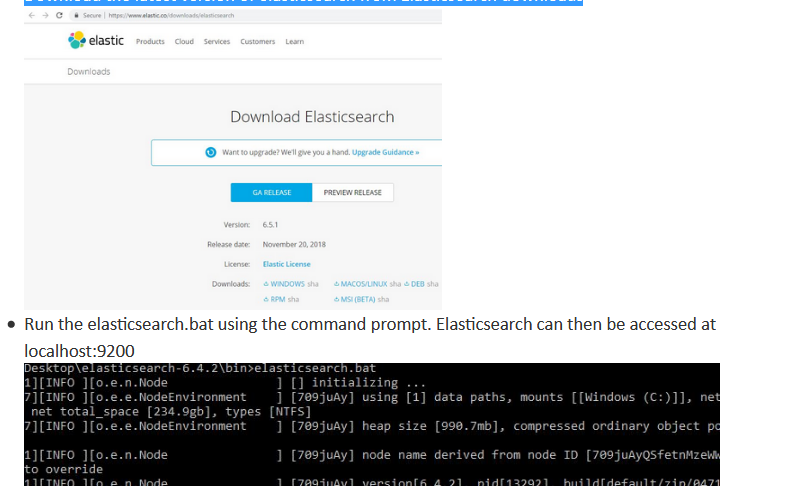
* Consider you have a single application running and it produces logs. Now suppose you want analyze the logs generated. One option is to manually analyze them. But suppose these logs are large, then manually analyzing them is not feasible.
* Suppose we have multiple Application running and all these applications produce logs. If we have to analyze the logs manually we will need to go through all the log files. These may run into hundreds.

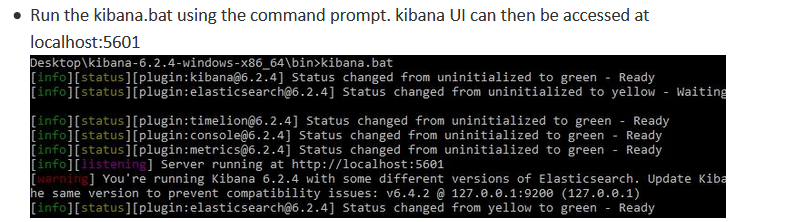
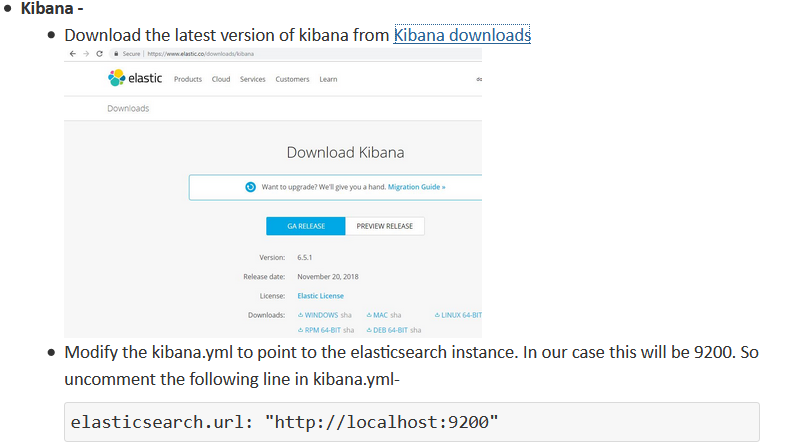
We can use ELK here to analyze the logs more efficiently and also using more complex search criterias. It provides log aggregation and efficient searching.

## Lets Begin

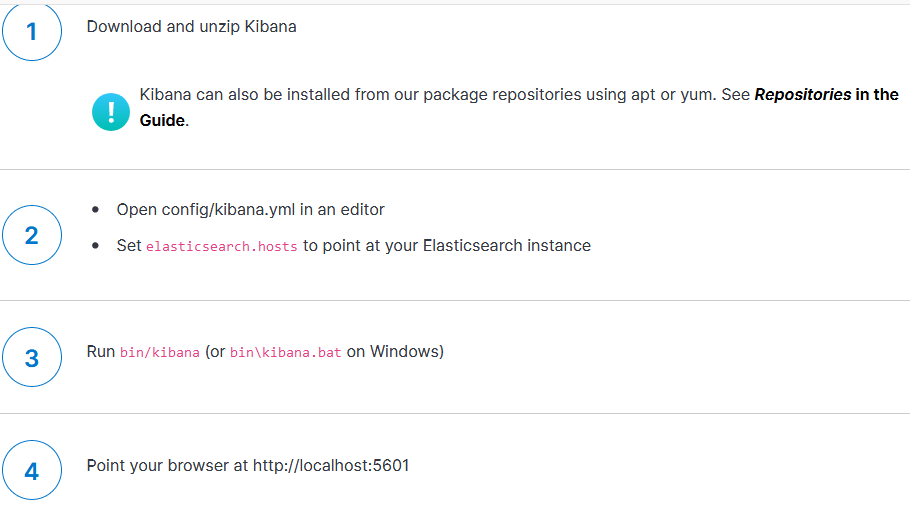
We will first download the required stack.

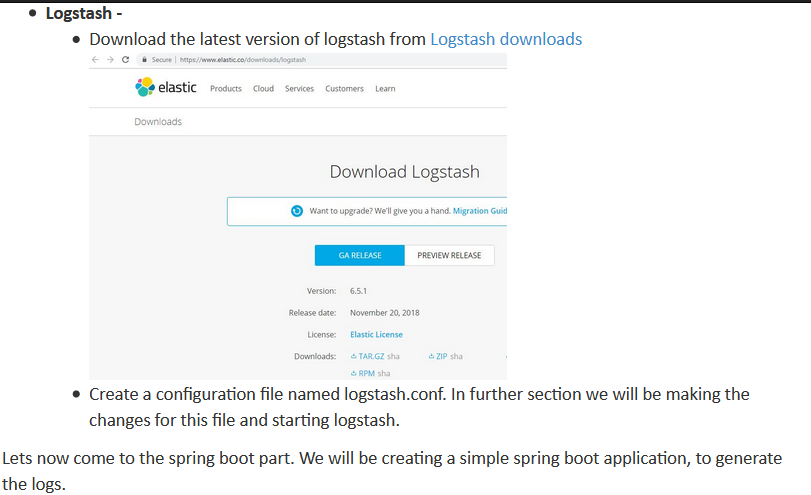
* **Elasticsearch -** 
  + Download the latest version of elasticsearch from [Elasticsearch downloads](https://www.elastic.co/downloads/elasticsearch)



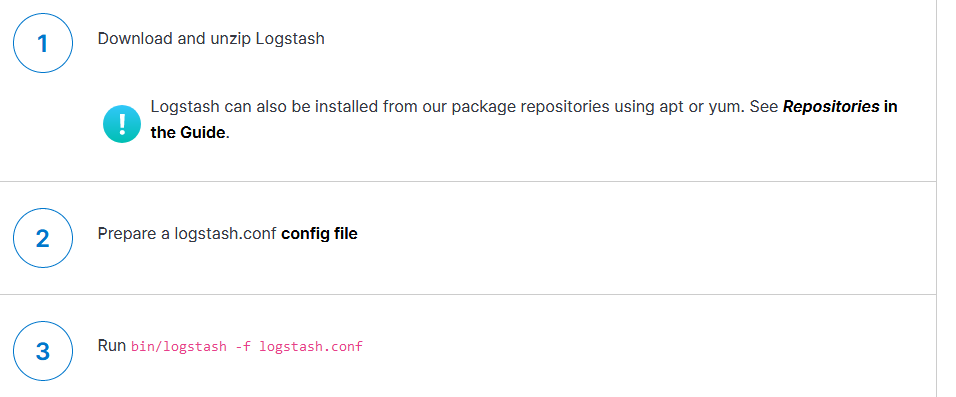


**How to install Kibana :**





**How To install Logstash :**



**Define the controller to expose REST API. We will be making use of these calls to write content to the log file :**

@RestController

class ELKController {

private static final Logger LOG = Logger.getLogger(ELKController.class.getName());

@Autowired

RestTemplate restTemplete;

@Bean

RestTemplate restTemplate() {

return new RestTemplate();

}

@RequestMapping(value = "/elk")

public String helloWorld() {

String response = "Welcome to JavaInUse" + new Date();

LOG.log(Level.INFO, response);

return response;

}

@RequestMapping(value = "/exception")

public String exception() {

String response = "";

try {

throw new Exception("Exception has occured....");

} catch (Exception e) {

e.printStackTrace();

LOG.error(e);

StringWriter sw = new StringWriter();

PrintWriter pw = new PrintWriter(sw);

e.printStackTrace(pw);

String stackTrace = sw.toString();

LOG.error("Exception - " + stackTrace);

response = stackTrace;

}

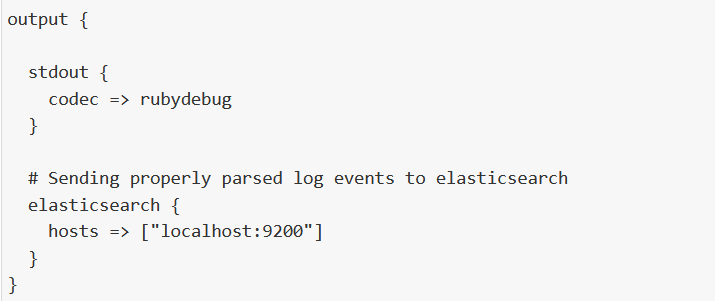
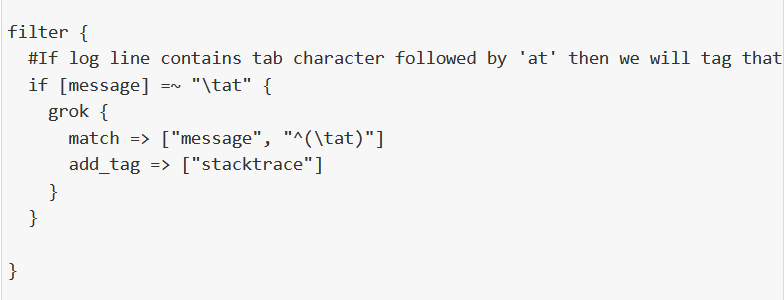
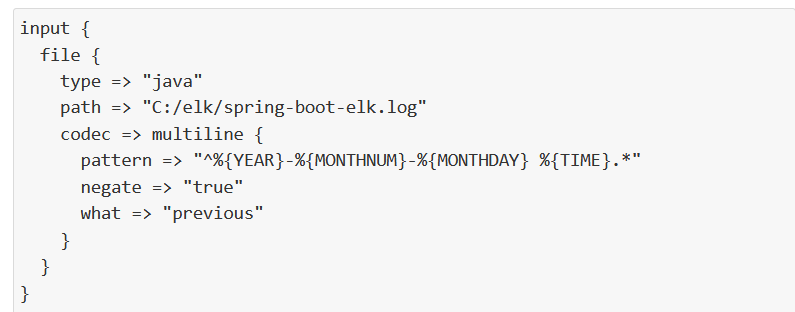
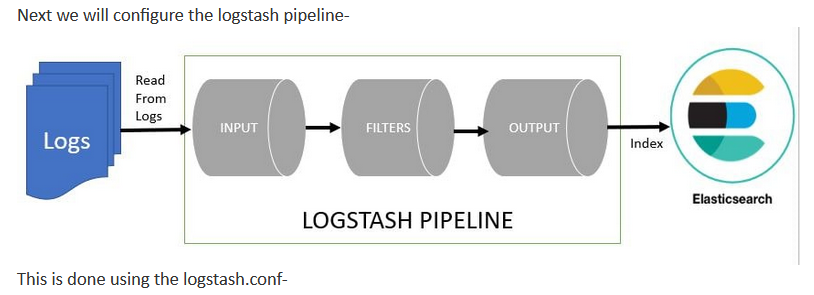
return response;

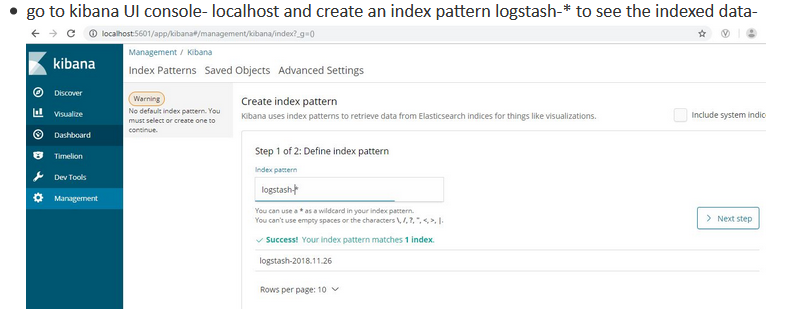
}

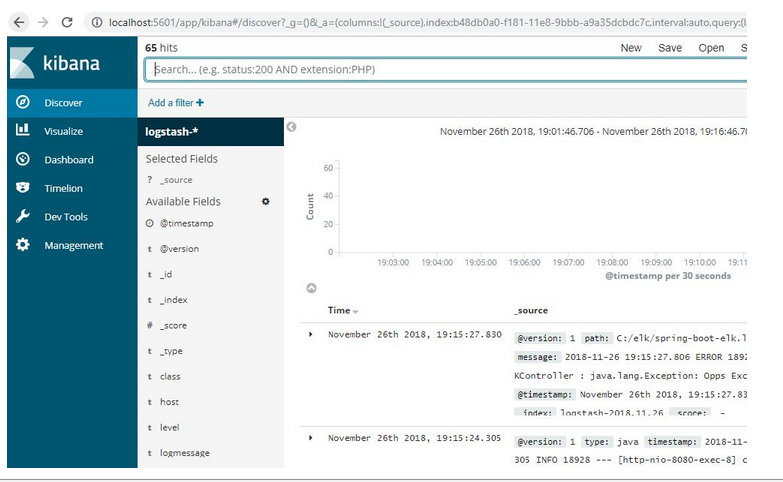
}

Finally specify the name and location of the log file to be created in the application.properties file.

**logging.file=C:/elk/spring-boot-elk.log**







# File Beat + ELK(Elastic, Logstash and Kibana) Stack to index logs to Elasticsearch - Hello World Example :

# In the last section we saw Logstash was reading log files using the logstash filereader. Suppose we have to read data from multiple server log files and index it to elasticsearch. One option is to install logstash on all the servers and then index it to the elasticsearch server.

# 

# Logstash consumes a lot of resources so it is not an optimum solution to have logstash installed on all fileservers. Instead we can use Beats in such scenarios.

# 

# 

# Configure ElasticSerach & Kibana as previous example.

# Logstash :

# Similar to how we did in the [Spring Boot + ELK tutorial](https://www.javainuse.com/spring/springboot-microservice-elk), create a configuration file named logstash.conf. Here Logstash is configured to listen for incoming Beats connections on port 5044. Also on getting some input, Logstash will filter the input and index it to elasticsearch.

# Read input from filebeat by listening to port 5044 on which filebeat will send the data

**input** {

beats {

type => "test"

port => "5044"

}

}

**filter** {

#If log line contains tab character followed by 'at' then we will tag that entry as stacktrace

if [message] =~ "\tat" {

grok {

match => ["message", "^(\tat)"]

add\_tag => ["stacktrace"]

}

}

}

**output** {

stdout {

codec => rubydebug

}

# Sending properly parsed log events to elasticsearch

elasticsearch {

hosts => ["localhost:9200"]

}

}

# 

**FileBeat-**

* Download filebeat from [FileBeat Download](https://www.elastic.co/downloads/beats/filebeat)
* Unzip the contents. Open filebeat.yml and add the following content. We are specifying the logs location for the filebeat to read from. The hosts specifies the Logstash server and the port on which Logstash is configured to listen for incoming Beats connections.

filebeat:

prospectors:

-

paths:

- C:/elk/\*.log

input\_type: log

multiline.pattern: '^[0-9]{4}-[0-9]{2}-[0-9]{2}'

multiline.negate: true

multiline.match: after

output:

logstash:

hosts: ["localhost:5044"]

Start filebeat as follows-

**filebeat.exe -c filebeat.yml**

