ELATSIC/ ELK STACK

- **E – ElasticSearch**
- **L- Logstash**
- **K- Kibana**

ElasticSearch:

- It is open-source **Search & Analytics Engine** and can also serve as a **NOSQL Database** which will store data in the form of **Json** and uses **RESTFUL API** to store and retrieve data.
- It works based on Apache Lucene which is also known as "**Heart of Elasticsearch"**

Logstash:

It is used to **read, write, filter and modify data** from various sources and store it in Elasticsearch.

Kibana:

- It is a web-interface which is used to **Discover, Analyze, Monitor and visualize** the data from Elasticsearch.
- It also used to apply **Machine Learning** Algorithms on the data from Elasticsearch to get insights of **data Anomaly** and **future trends**.

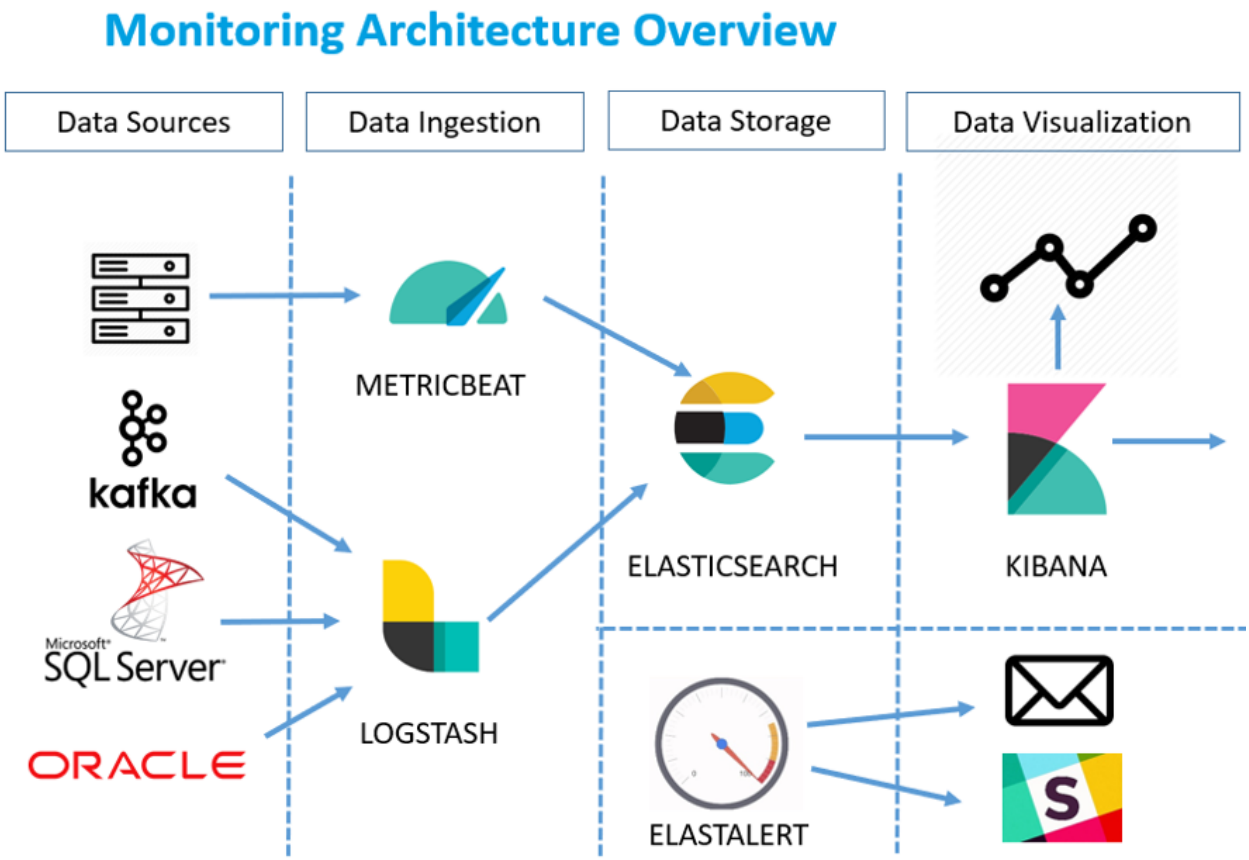
Beats:

They are Light-weight data shippers which is used to ship data from data source to ElasticSearch.

Types:

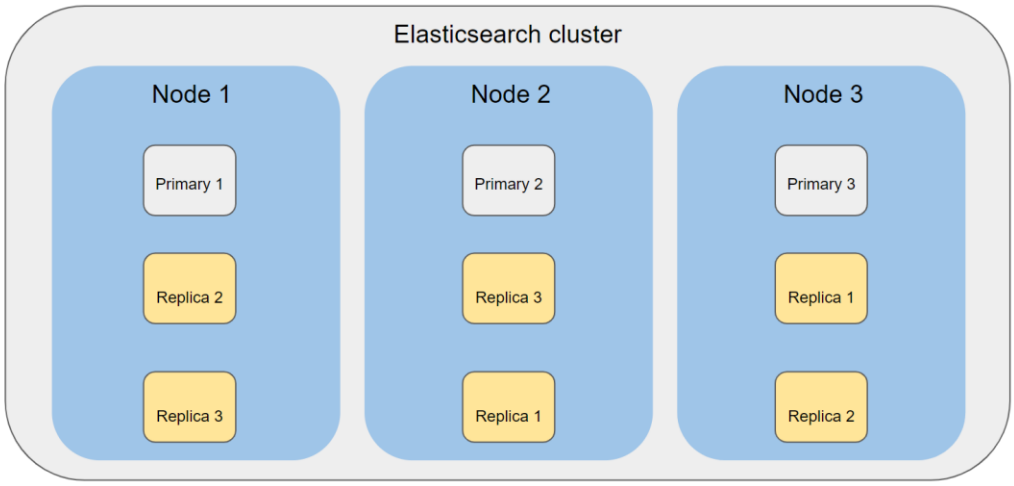
File Beats	Log files
Metric Beats	Metrics(CPU,Memory)
Packet Beats	Network Data
Win log Beats	Windows Event Logs
Audit Beats	Audit (OS files)
Heart Beats	Uptime Monitor
Function Beats	Serverless Shipper

Architecture Of Elastic Stack:



Terminology in Elastic Stack:

Elastic terms	RDBMS terms	Refers to
Fields	Columns	Key-value Pair (JSON Objects)
Documents	Rows	Collection of Fields
Index	Table	Collection of Documents
Cluster	Database	Collection of Index
Shards	-	Horizontal partitioning of Index
Replica	-	Copy of Shards



Primary Shards and its Replicas in Elasticsearch cluster

Types of Nodes in Cluster:

Master Node	• Responsible for creation or Deletion of Index. • Tracks the other nodes. • Determines the location of shards.
Data Node	Responsible for performing CRUD, Search and Aggregation functions.
Ingest Node	Responsible for processing a document before indexing them.
Co-Ordinating Node	• Performs Routing • Aids for Search Reduction Phase • Responsible for Distributing the works via BULK Indexing.

Installation and Set Up Procedure:

Server:

- Install Oracle VM.
- Install Ubuntu. (Server)
- Download and install:
    - wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
    - sudo apt-get install apt-transport-https
    - echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-8.x.list
    - sudo apt-get update && sudo apt-get install elasticsearch

- sudo nano /etc/elasticsearch/elasticsearch.yml
  After this enable the following fields in the .yml file:
  node.name: node-1
  network.host: 0.0.0.0
  discovery.seed.hosts: ["127.0.0.1"]
  xpack.security.enabled: false
  cluster.initial_master_nodes: ["node-1"]
- sudo chmod 755 -R /var/log/elasticsearch/

- Configure Elasticsearch to start automatically when the system boots up
  - sudo /bin/systemctl daemon-reload
  - sudo /bin/systemctl enable elasticsearch.service
- Elasticsearch can be started as follows
  - sudo /bin/systemctl start elasticsearch.service
  - sudo /bin/systemctl status elasticsearch.service
- Install curl
  - sudo apt-get install curl
- Go to Settings ->Networks ->Port Forwarding
  and add the following Network Configuration:
  - **Elasticsearch-127.0.0.1-9200**
  - **Kibana 127.0.0.1 5601**
  - **SSL 127.0.0.1 22**
- Open Terminal and Type the following Commands:
  - Sudo apt-get install openssh-server
  - Sudo systemctl enable ssh
  - Sudo adduser username
  - Sudo usermod -aG sudo username

Client:

- Download Putty. (Client)
- Configure the ElasticSearch in 127.0.0.1 in port 9200 and load it and open it.
- Login as username and enter the password.
- Then load the dataset into Elasticsearch using the Command:
  - Dataset: http://media.sundog-soft.com/es8/movies.json
- Create the index and post the data using the command:
  - curl -X PUT "localhost:9200/movies?pretty"
  - curl -XPOST "localhost:9200/movies/_bulk?pretty" --data-binary @movies.json

CRUD OPERATIONS IN INDEX:

- Command to know the mappings of the index movies:
  - curl -XGET “127.0.0.1:9200/movies/_mappings?pretty” (Note: To reduce the mapping area of any index we can define the datatype as “Flattened”)
- Command to add document to the index movies:

- curl -XPUT 127.0.0.1:9200/movies/_doc/

- Command to delete document from the index movies:
  - curl -XDELETE 127.0.0.1:9200/movies/_doc/1234567 //1234567-id
- Command to Update a document to the index movies:
  - Curl -XPUT 127.0.0.1:9200/movies/_doc/1234567
- Command to do OPTIMISTIC CONCURRENCY CONTROL to update a document is:
  - Curl -XPUT 127.0.0.1:9200/movies/_doc/1234567?if seq_no=7&if_primary_term=1

REALTIONSHIP BETWEEN DOCUMENTS IN A INDEX:

Command to establish a parent child relationship between franchise and film:

```
Curl -XPUT 127.0.0.1:9200/Series -d ' {
"mappings" : {
        "properties" : {
                "film_to_franchise" : {
                        "type" : "join" ,
                        "relations": { "franchise" , "flim"}        //franchise -parent
                        }                                           //flim-child
                }
        }
}'
```

Command to find the child who has "franchise" as parent:

```
Curl -XGET 127.0.0.1:9200/series/_search?pretty -d' {
"query" : {
        "has_parent" : {
                "parent_type": "franchise",
                "query" : { "match" : { "title" : "Star Wars"} }
        }
        }
}'
```

Command to find the parent who has “The Force Awakens” as child:

Curl -XGET 127.0.0.1:9200/series/_search?pretty -d’ {

“query” : {

“has_child” : {

“type”: “film”,

“query” : { “match” : { “title” : “The Force Awakens”} }

}

}

}’

```
student@harsini-VirtualBox:~$ curl -XGET 127.0.0.1:9200/series/_search?pretty -d' {
        "query" : {
                "has_child" : {
                        "type": "film",
                        "query" : { "match" : { "title" : "The Force Awakens"} }
                }
        }
}'
{
  "took" : 104,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "series",
        "_id" : "1",
        "_score" : 1.0,
        "_routing" : "1",
        "_source" : {
          "id" : "1",
          "film_to_franchise" : {
            "name" : "franchise"
          },
          "title" : "Star Wars"
        }
      }
    ]
  }
}
```

SEARCH IN ELASTICSEARCH:

Query Line Search

The query is given directly as a parameter

Example: Query to get the details of the movie which is released after the year 2010 and has the word “trek” in the title.

- Curl -XGET ‘http://localhost:9200/movies/_search?q=+year:>2010+title:trek’

Request Body Search

The query is given as a request body

Example: Query to get the details of the movie which is released after the year 2010 and has the word “trek” in the title.

- Curl -XGET ‘http://localhost:9200/movies/_search?pretty’ -d ‘{

  “query” : {

  “bool” : {

  “must” : { “term” : { “title” : “trek”}},

  “filter” : { “range” : { “year” : { “gte” : 2010}}}

}

}

}'

Difference between match and match_phrase

Term Matching:

- **match:** Breaks down the input text into individual terms and matches any of them.
- **match_phrase:** Searches for the exact sequence of terms as a phrase.

Order and Proximity:

- **match:** Ignores the order of terms and proximity.
- **match_phrase:** Considers the order and ensures the terms appear close to each other as specified.

Match_Phrase Search

Search for exach phrases given in the query

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

"query": {

"match_phrase": {

"title": {

"query": "Star Wars",

"slop": 2

}

}

}}'


```
"timed_out" : false,
"_shards" : {
  "total" : 1,
  "successful" : 1,
  "skipped" : 0,
  "failed" : 0
},
"hits" : {
  "total" : {
    "value" : 13,
    "relation" : "eq"
  },
  "max_score" : 9.028755,
  "hits" : [
    {
      "_index" : "movies",
      "_id" : "61160",
      "_score" : 9.028755,
      "_source" : {
        "id" : "61160",
        "title" : "Star Wars: The Clone Wars",
        "year" : 2008,
        "genre" : [
          "Action",
          "Adventure",
          "Animation",
          "Sci-Fi"
        ]
      }
    },
    {
      "_index" : "movies",
      "_id" : "135216",
      "_score" : 9.028755,
      "_source" : {
        "id" : "135216",
        "title" : "The Star Wars Holiday Special",
        "year" : 1978,
        "genre" : [
          "Adventure",
          "Children",
          "Comedy",
          "Sci-Fi"
        ]
      }
    },
```

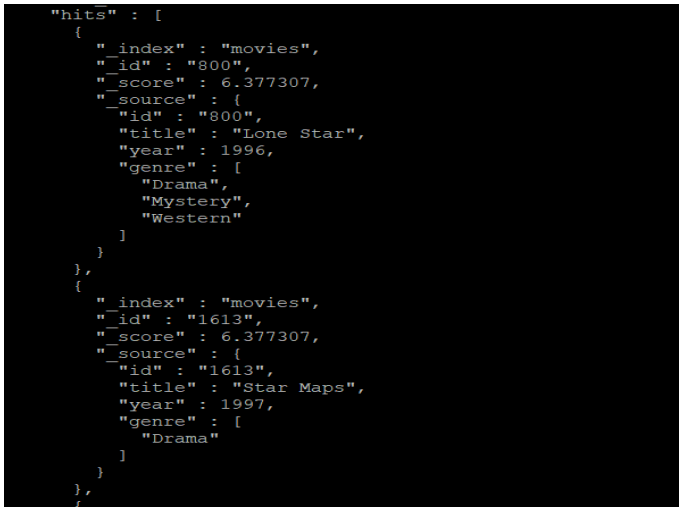
Pagination

While searching for the query we can do pagination by defining from and size keywords.

- From – specifies the starting point
- Size – specifies the number of results to be retrieved

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

```
"query": {
        "match": {
                "title": "Star"
        }
},
"from": 0,
"size": 10
}'
```



Sorting

It is used to sort the result which is fetched using search query

- Curl -XGET 127.0.0.1:9200/movies/_search?sort=year&pretty'

In order to sort based on text value we need to define them as keyword in the raw data format

- Curl -XPUT 127.0.0.1:9200/movies/ -d ‘ {

```
“mappings” : {
    “properties” : {
        “title” : {
            “type” : “text” ,
            “fields” : { “raw” : { “type” : “keyword” } }
        }
    }
}
}’
```

Difference between Text and Keyword fields

Text Fields

- **Purpose:** Text fields are used for full-text search. They are analyzed, meaning the text is processed and broken down into individual terms (tokens) using an analyzer.

Keyword Fields

- **Purpose:** Keyword fields are used for exact matching, sorting, and aggregations. They are not analyzed, meaning the text is indexed as a single token.

Why Have a Keyword Field for a Text Field?

1. **Exact Matching:** When you need to perform exact match queries on a field, such as finding all documents where the title is exactly "Star Wars." Analyzing the text would break it down into individual terms, making it unsuitable for exact matches.
2. **Sorting:** Sorting requires the exact values of the field. Analyzed text fields cannot be sorted properly because they are broken down into multiple terms.
3. **Aggregations:** Aggregations, like counting unique values, require the exact terms. Text fields, which are analyzed, cannot be used for accurate aggregations.

Fuzzy Queries

Fuzzy queries are designed to handle search terms that may contain misspellings or typographical errors. They can identify similar terms within a certain edit distance, allowing for more flexible searches. Fuzzy queries support:

- Substitution Eg: Apple -> Appla
- Insertion Eg: Apple -> Applea
- Deletion Eg: Apple->Aple

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

"query": {

"fuzzy": {

"title": {“value”:"Ster, “fuzziness”:1}

}

}

}’

Partial Matching

It is used to perform partial matching for search

Prefix query

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

```
        "query": {
            "prefix": {
                “year”:"201”
            }
        }
    }’
```

Wildcard query

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

```
        "query": {
            "wildcard": {
                “year”:"19*”
            }
        }
    }’
```

Search-as-you-type

As the name suggests, it will perform search as you type

- curl -XGET 'http://localhost:9200/movies/_search?pretty' -d '{

```
    "query": {
        "match_phrase_prefix": {
            "title": {
                "query": "Star Wars",
                "slop": 10
            }
        }
    }
}'
```

```
"hits" : [
  {
    "_index" : "movies",
    "_id" : "61160",
    "_score" : 15.817083,
    "_source" : {
      "id" : "61160",
      "title" : "Star Wars: The Clone Wars",
      "year" : 2008,
      "genre" : [
        "Action",
        "Adventure",
        "Animation",
        "Sci-Fi"
      ]
    }
  },
  {
    "_index" : "movies",
    "_id" : "135216",
    "_score" : 15.817083,
    "_source" : {
      "id" : "135216",
      "title" : "The Star Wars Holiday Special",
      "year" : 1978,
      "genre" : [
        "Adventure",
        "Children",
        "Comedy",
        "Sci-Fi"
      ]
    }
  },
```


EXCEPTION HANDLING FOR SEARCH:

To ignore the exception which is throw by datatype:

Example: When we try to perform search operation using the keyword datatype:

Curl –location –XPUT 127.0.0.1:9200/microservice-logs/_settings \

--data-raw ‘ {

“index.mapping.ignore_malformed” : true

}

To ignore the exception which is throw because we are exceeding the default limit(1000):

Curl –location –XPUT 127.0.0.1:9200/big_Objects/_settings \

--data-raw ‘ {

“index.mapping.total.fields.limit” : 1005

}

Importing Data from different sources to Elasticsearch

- Java- Elastic.co
- Python – Elasticsearch.package
- Ruby-Elasticsearch.ruby
- Perl-Elasticsearch.pm

Importing Data using Python:

Create and Run IndexRatings.py file

IndexRatings.py file:

```
import csv
from collections import deque
import elasticsearch
from elasticsearch import helpers

def readMovies():
    csvfile = open('ml-latest-small/movies.csv', 'r', encoding="utf8")

    reader = csv.DictReader( csvfile )

    titleLookup = {}

    for movie in reader:
            titleLookup[movie['movieId']] = movie['title']

    return titleLookup

def readRatings():
    csvfile = open('ml-latest-small/ratings.csv', 'r', encoding="utf8")

    titleLookup = readMovies()

    reader = csv.DictReader( csvfile )
    for line in reader:
        rating = {}
        rating['user_id'] = int(line['userId'])
        rating['movie_id'] = int(line['movieId'])
        rating['title'] = titleLookup[line['movieId']]
        rating['rating'] = float(line['rating'])
        rating['timestamp'] = int(line['timestamp'])
        yield rating


es = elasticsearch.Elasticsearch(["http://127.0.0.1:9200"])

#es.indices.delete(index="ratings",ignore=404)
deque(helpers.parallel_bulk(es,readRatings(),index="ratings", request_timeout=300), maxlen=0)
es.indices.refresh()
```

Run the command

curl -XGET 127.0.0.1:9200/ratings/_search?pretty

Importing Data from MySql:

Install Mysql Connector:

- sudo apt-get install mysql-server
- wget http://files.grouplens.org/datasets/movielens/m1-100k.zip
- unzip ml-100k.zip
- sudo mysql --local-infile=1 -u root -p
- CREATE DATABASE movielens;CREATE TABLE movielens.movies (

  movieID IN PRIMARY KEY NOT NULL,

  title TEXT,releaseDate DATE

);

Update the mysql.conf file

sudo cat /etc/logstash/conf.d/mysql.conf

Mysql.conf

```
input{
  jdbc{
   jdbc_connection_string => "jdbc:mysql://localhost:3306/movielens"
   jdbc_user => "student"
   jdbc_password => " *****"
   jdbc_driver_library => "home/student/usr/share/logstash/mysql-connector-java-8.0.16/mysql-connector-java-8.0.16.jar"
   jdbc_driver_class => "com.mysql.jdbc.Driver"
   statement => "SELECT * from movies"
}
}
output{
 stdout { codec => json_lines }
elasticsearch{
 hosts => ["localhost:9200"]
 index => "movielens-sql"
```

```
}
}
```

Run the commands:

```
sudo /usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/mysql.conf
curl -XGET 127.0.0.1:9200/movielens-sql/_search?pretty
```

Importing Data from .Csv:

```
sudo cat /etc/logstash/conf.d/csv-read-drop.conf
```

csv-read-drop.conf:

```
input {
  file {
    path => "/home/student/csv-data/csv-schema-short-numerical.csv"
    start_position => "beginning”
  }
}
filter {
  csv {
      separator => ","
      skip_header => "true"
      columns =>
["id","timestamp","paymentType","name","gender","ip_address","purpose","country","age"]
  }
}
output {
   elasticsearch {
     hosts => "http://localhost:9200"
     index => "demo-csv"
  }
```

```
stdout {}
```

Run the Commands:

```
sudo /usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/demo-csv.conf
curl -XGET 127.0.0.1:9200/demo-csv/_search?pretty
```

Importing Data from .Json/.Log:

- cd /etc/logstash/conf.d/
- sudo vi json-read.conf

json-read.conf: (using filter)

```
input {
        file {
                start_position => "beginning"
                path=> "/home/student/json-data/sample-json.log"
        }
}
filter {
        json {
                source => "message"
        }
}
output {
        elasticsearch {
                hosts => "http://localhost:9200"
                index=>"demo-json"
        }
        stdout {}
}
```

Run the commands:

```
sudo /usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/demo-json.conf
```

curl -XGET 127.0.0.1/demo-json/_search?pretty

demo-json-drop.conf: (using mutate and removing unwanted fields)

```
input {
        file {
                start_position => "beginning"
                path=> "/home/student/json-data/sample-json.log"
        }
}
filter {
        json {
                source => "message"
        }
        if [paymentType]  == "Mastercard" {
                drop{}
        }
        mutate{
                remove_field =>["message","@timestamp","path","host","@version"]
        }
}
output {
        elasticsearch {
                hosts => "http://localhost:9200"
                index=>"demo-json-drop"
        }
        stdout {}
}
```

Run the Commands:

sudo /usr/share/logstash/bin/logstash -f /etc/logstash/conf.d/json-drop.conf

curl -XGET 'http://localhost:9200/demo-json-drop/_search?pretty'

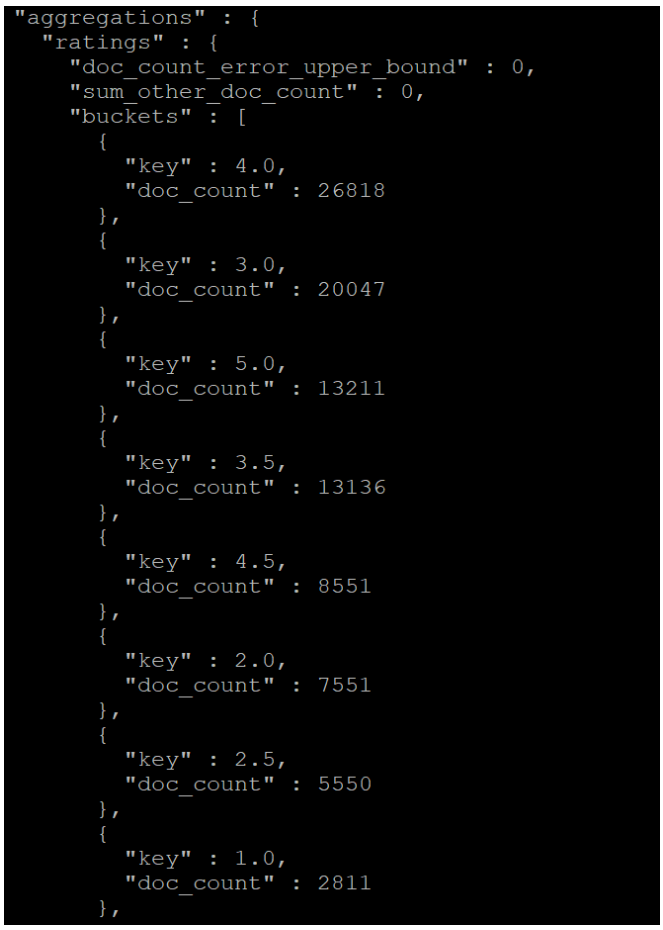
AGGREGATION

- Metrics - Avg,Min,Max
- Buckets - Histogram,Piechart

Metrics:

1. Aggregation on 'Ratings' index

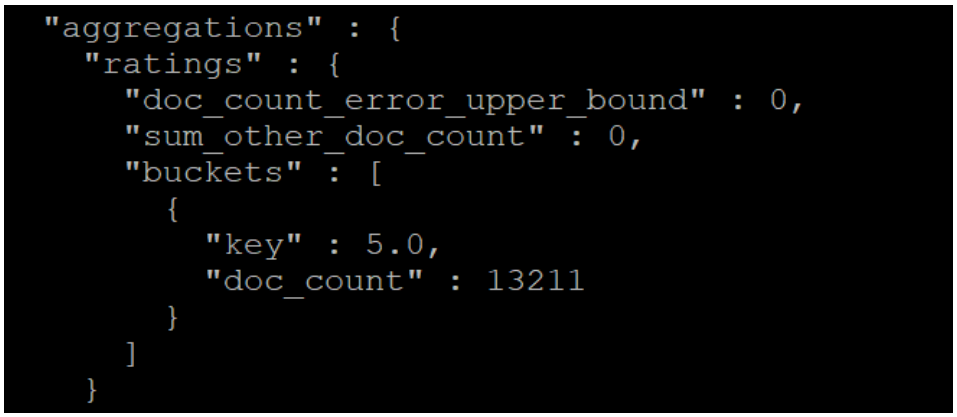
```
curl -XGET '127.0.0.1:9200/ratings/_search?pretty' -d '
{ "aggs" : {
        "ratings" : {
                "terms" : {
                        "field":"rating"
}}}}'
```




```
"aggregations" : {
  "ratings" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : 4.0,
        "doc_count" : 26818
      },
      {
        "key" : 3.0,
        "doc_count" : 20047
      },
      {
        "key" : 5.0,
        "doc_count" : 13211
      },
      {
        "key" : 3.5,
        "doc_count" : 13136
      },
      {
        "key" : 4.5,
        "doc_count" : 8551
      },
      {
        "key" : 2.0,
        "doc_count" : 7551
      },
      {
        "key" : 2.5,
        "doc_count" : 5550
      },
      {
        "key" : 1.0,
        "doc_count" : 2811
      },
```

2. **Using match and aggregation**

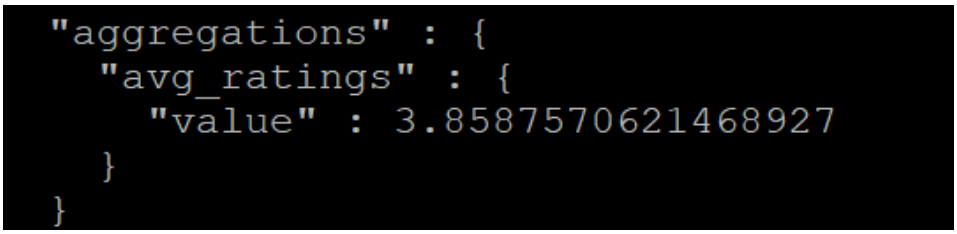
```
curl -XGET '127.0.0.1:9200/ratings/_search?pretty' -d '
{
        "query":
                {"match":{"rating":5.0}},
        "aggs" : {
                "ratings" : {
                        "terms" : {
                                "field":"rating"
}}}}'
```



```
"aggregations" : {
  "ratings" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : 5.0,
        "doc_count" : 13211
      }
    ]
  }
```

3. **Using match_phrase and avg_aggregation**

```
curl -XGET '127.0.0.1:9200/ratings/_search?pretty' -d '
{
        "query":{
                "match_phrase":{"title":"Star Wars"}},
        "aggs" : {
                "avg_ratings" : {
                        "avg" : {
                                "field":"rating"
}}}}'
```



```
"aggregations" : {
  "avg_ratings" : {
    "value" : 3.858757062146892
  }
}
```

Buckets-Histogram:

1. **Histogram on the field ratings at interval 1.0**

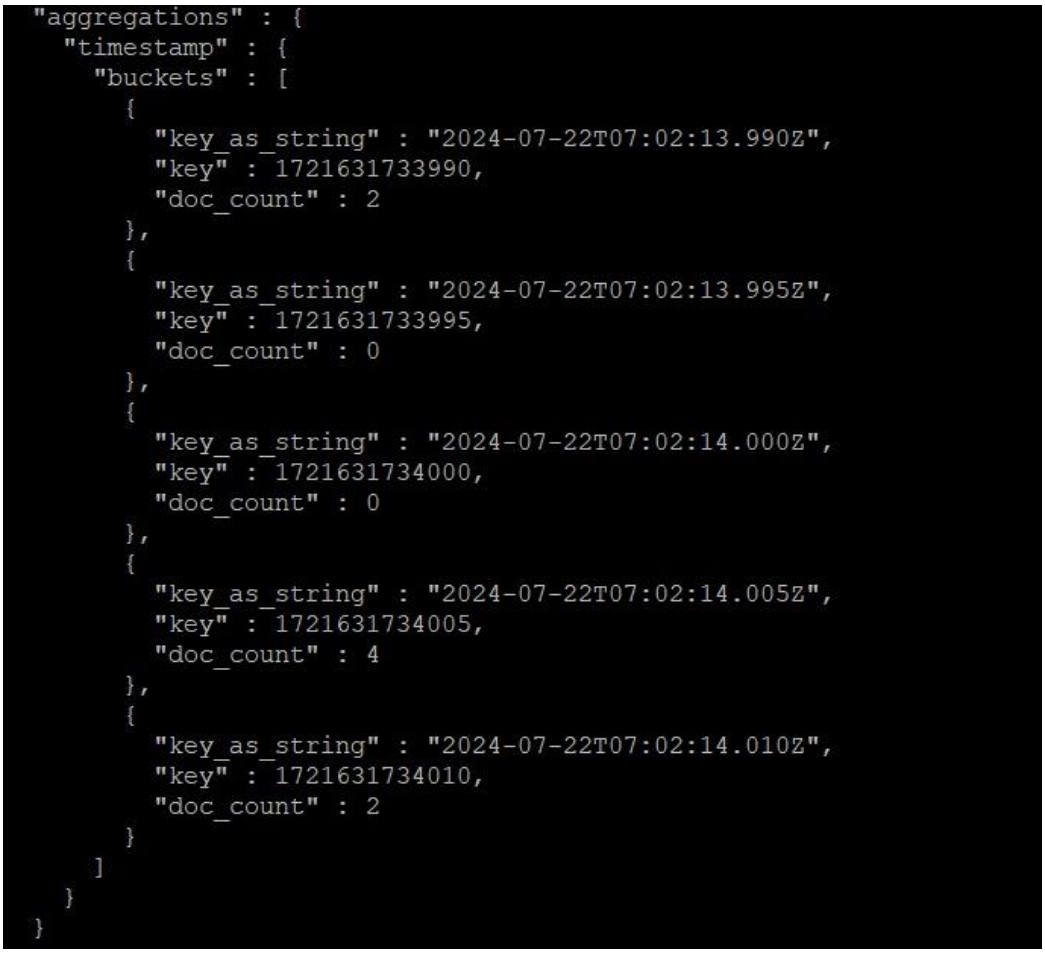
```
curl -XGET '127.0.0.1:9200/ratings/_search?pretty' -d '
{
        "aggs" : {
                "whole_ratings" : {
                        "histogram" : {
                                "field":"rating", "interval" : 1.0
}}}}'
```


```
"aggregations" : {
  "whole_ratings" : {
    "buckets" : [
      {
        "key" : 0.0,
        "doc_count" : 1370
      },
      {
        "key" : 1.0,
        "doc_count" : 4602
      },
      {
        "key" : 2.0,
        "doc_count" : 13101
      },
      {
        "key" : 3.0,
        "doc_count" : 33183
      },
      {
        "key" : 4.0,
        "doc_count" : 35369
      },
      {
        "key" : 5.0,
        "doc_count" : 13211
      }
    ]
  }
```

Buckets-Time Series

1. Histogram for time series data(calendar interval or fixed interval)

```
curl -XGET '127.0.0.1:9200/demo-grok/_search?pretty' -d '
{
        "aggs" : {
                "timestamp" : {
                "date_histogram" : {
                "field":"@timestamp", "fixed_interval" : "5ms"
}}}}'
```




```
"aggregations" : {
  "timestamp" : {
    "buckets" : [
      {
        "key_as_string" : "2024-07-22T07:02:13.990Z",
        "key" : 1721631733990,
        "doc_count" : 2
      },
      {
        "key_as_string" : "2024-07-22T07:02:13.995Z",
        "key" : 1721631733995,
        "doc_count" : 0
      },
      {
        "key_as_string" : "2024-07-22T07:02:14.000Z",
        "key" : 1721631734000,
        "doc_count" : 0
      },
      {
        "key_as_string" : "2024-07-22T07:02:14.005Z",
        "key" : 1721631734005,
        "doc_count" : 4
      },
      {
        "key_as_string" : "2024-07-22T07:02:14.010Z",
        "key" : 1721631734010,
        "doc_count" : 2
      }
    ]
  }
}
```

NESTED AGGREGATION:

Average movie ratings that contain word ‘Star’ in its title

```
curl -XGET '127.0.0.1:9200/ratings/_search?pretty' -d '
{
        "query":
                {
                "match_phrase" : { "title" : "Star Wars" }},
                        "aggs" :
                        { "titles":
                                {
                                "terms" : { "field" : "title.raw" },
                                "aggs" :
                                        { "avg_ratings" :
                                                { "avg" :
                                                        { "field" : "rating" }
}}}}}'
```

KIBANA

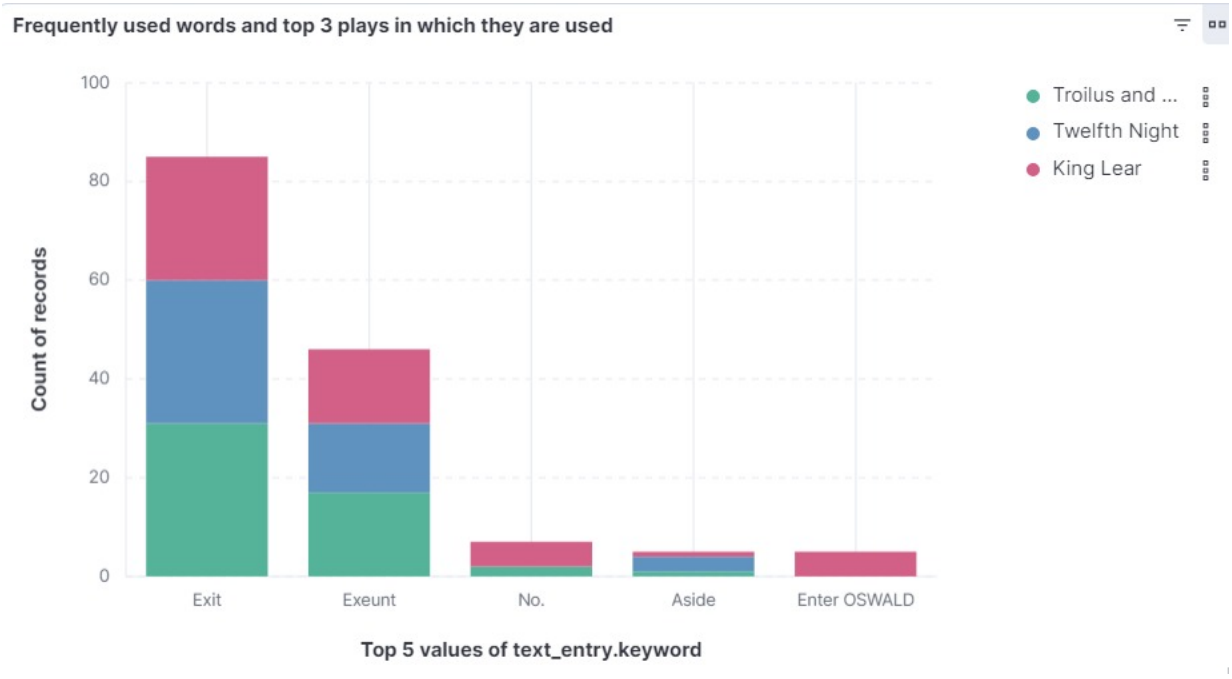
Install and enable Kibana:

- sudo apt list kibana
- sudo apt-get install kibana=8.14.3
- sudo nano /etc/kibana/kibana.yml
- sudo /bin/systemctl enable kibana.service
- sudo /bin/systemctl start kibana.service
- Open kibana service from port localhost:5601

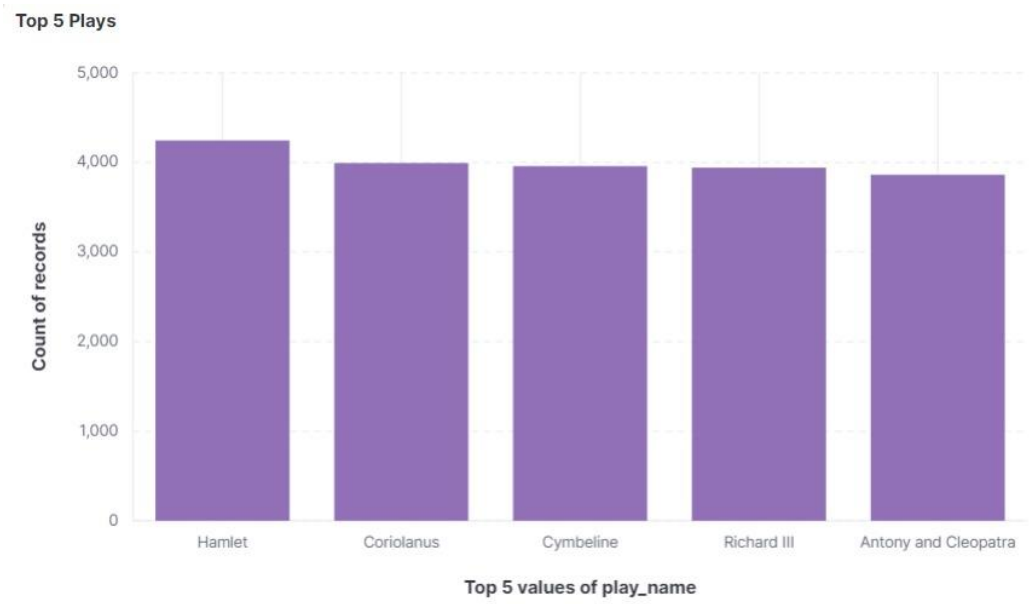
Workouts and Dashboard:

Displaying the most frequently used words in the Shakespear's works using Tag Cloud in Aggregation

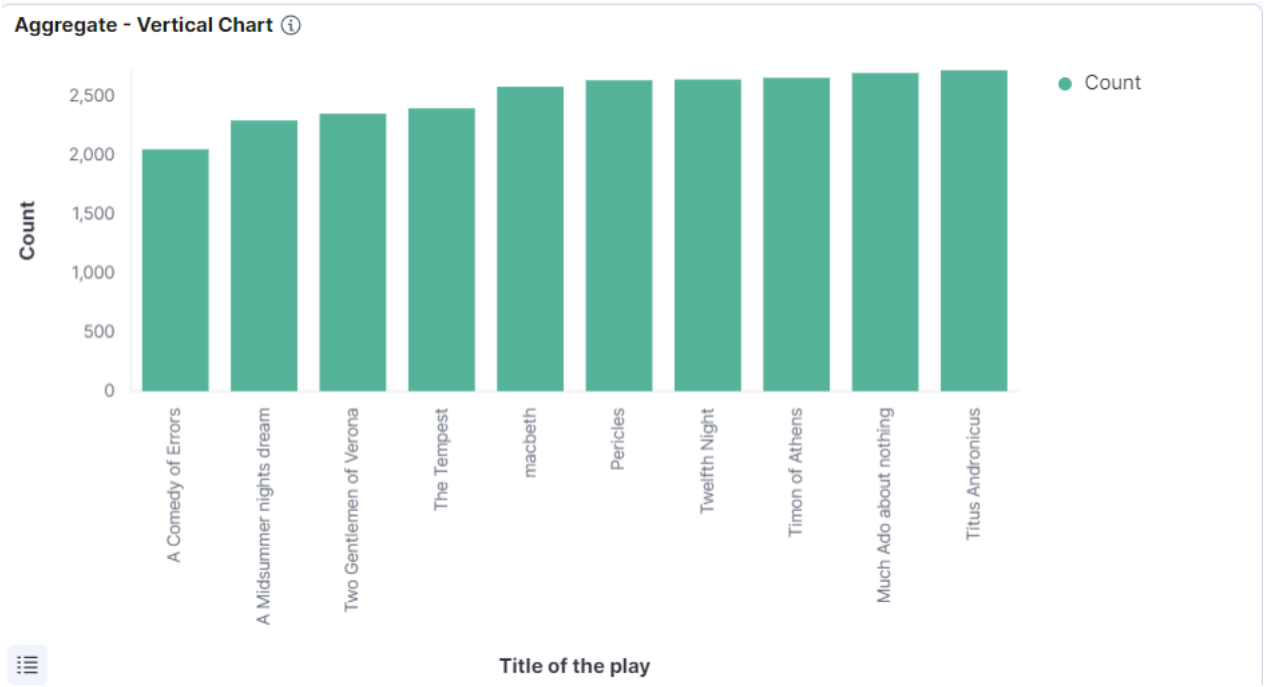
Top 5 frequently used words and the plays in which they are used.



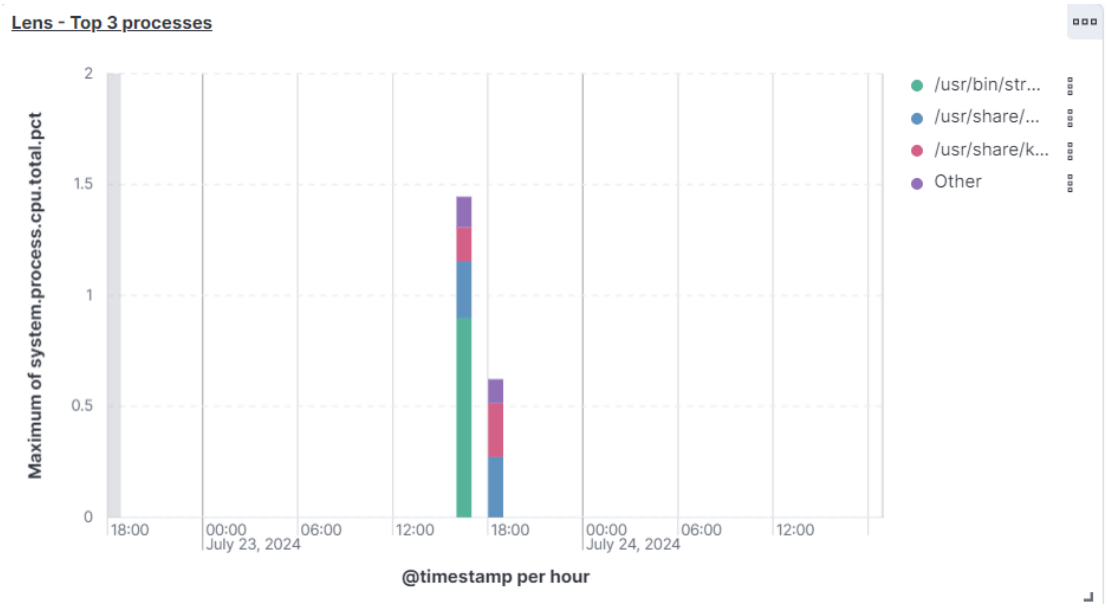
Vertical Bar chart to display the top 5 plays (having highest documents).



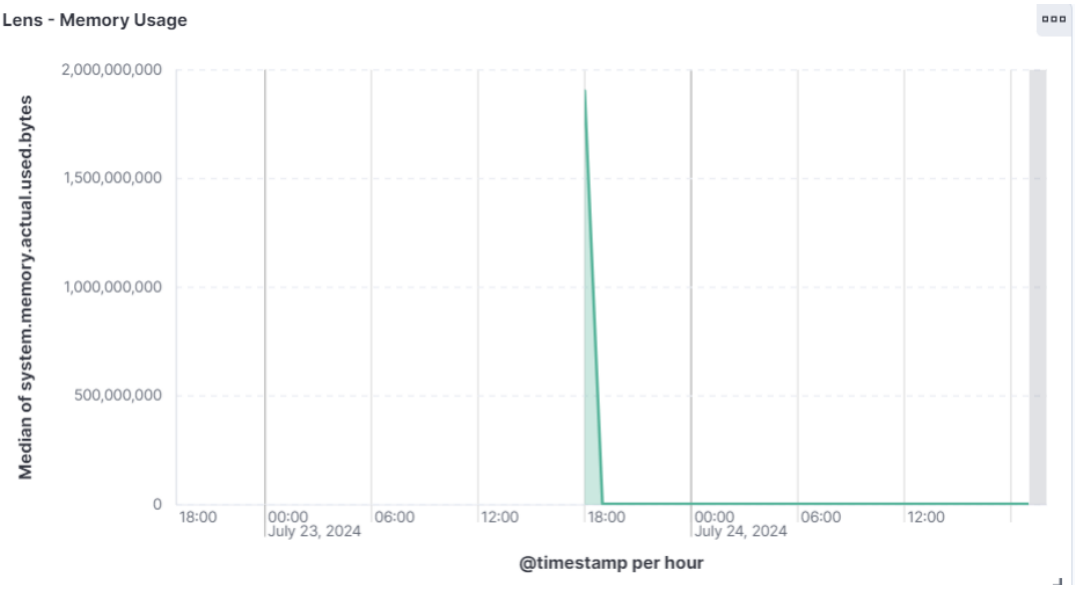
Vertical Bar chart displaying number of documents in last 10 plays.



Top 3 processes

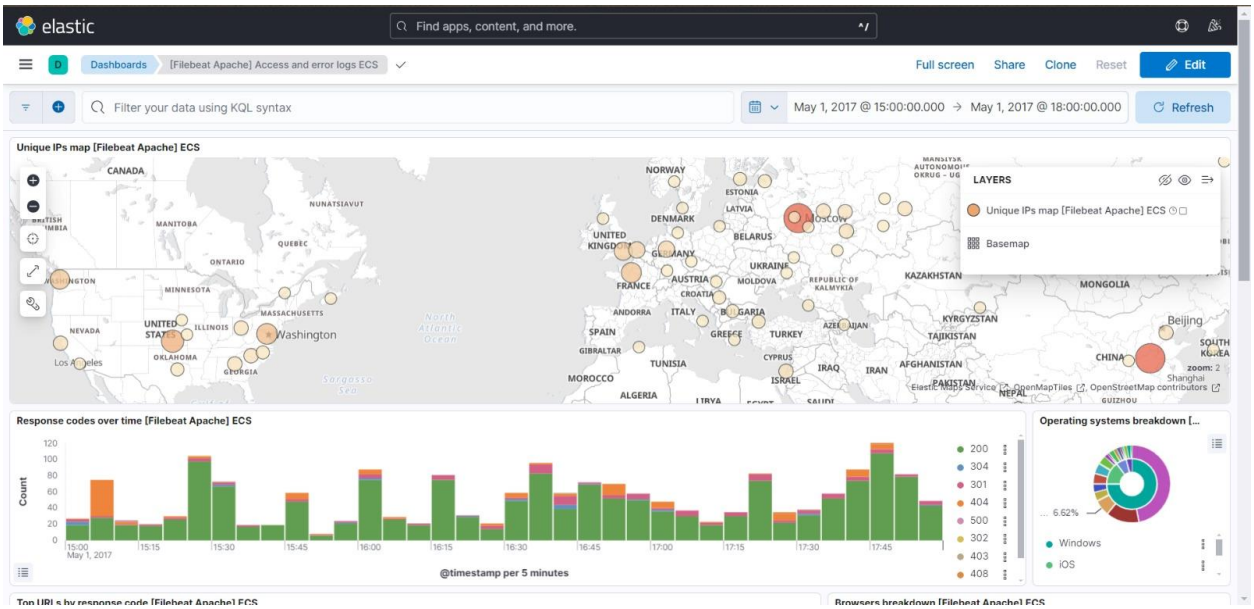


Memory Usage



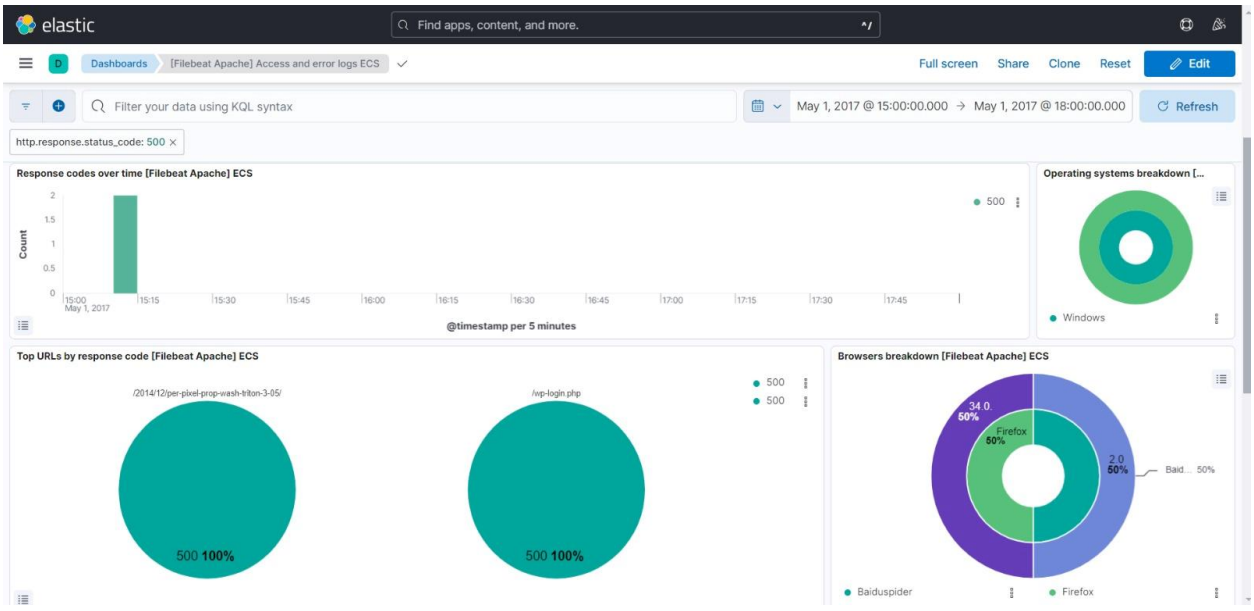
LOG ANALYSIS IN KIBANA

The following dashboard displays the entire logs details of specific web



The following dashboard displays the logs details of specific web where it faced internal server error

STATUS CODE: 500



Documents (4) Field statistics

Columns 3 Sort fields 1

Get the best look at your search results

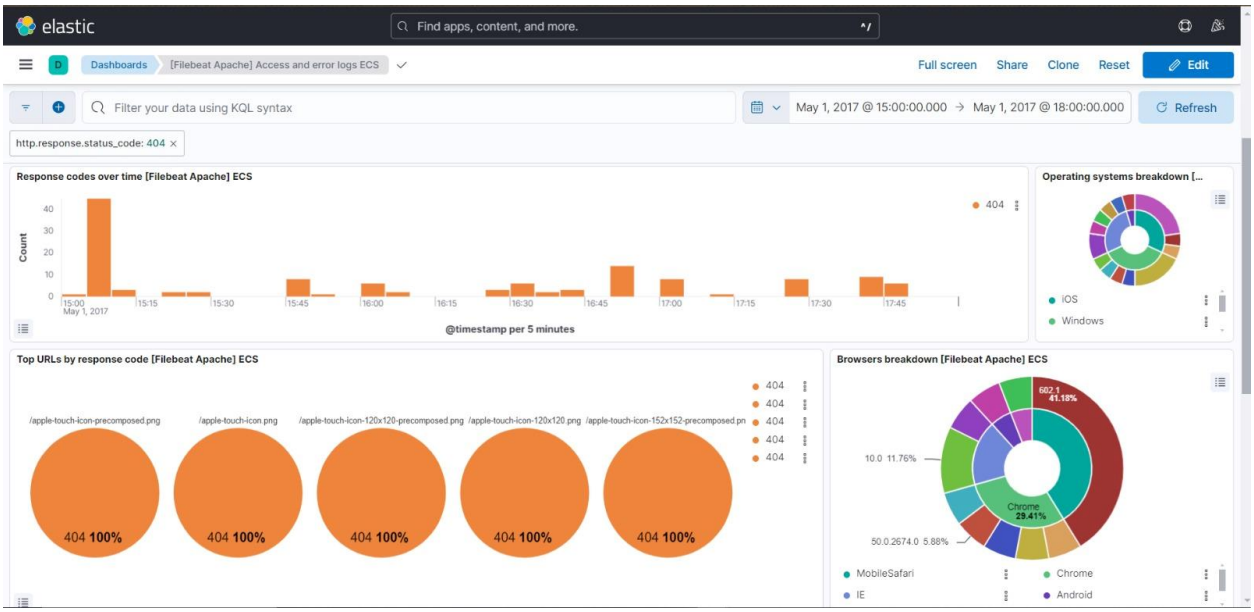
Add relevant fields, reorder and sort columns, resize rows, and more in the document table.

Take the tour Dismiss

@timestamp	source.geo.location	user_agent.name
May 4, 2017 @ 09:43:32.000	POINT (-97.822 37.751)	Googlebot
May 4, 2017 @ 09:43:31.000	POINT (-97.822 37.751)	Googlebot
May 4, 2017 @ 09:43:31.000	POINT (-97.822 37.751)	Googlebot

The following dashboard displays the logs details of specific web where it faced resource not found error

STATUS CODE: 404



ELASTICSEARCH and SQL

1. **To get the type mappings**

➢ curl -XPOST 127.0.0.1:9200/_sql? Format=txt -d '

{"query": "DESCRIBE movies”}'

```
    column      |     type      |    mapping
----------------+---------------+---------------
genre           |VARCHAR        |text
genre.keyword   |VARCHAR        |keyword
id              |VARCHAR        |text
id.keyword      |VARCHAR        |keyword
title           |VARCHAR        |text
title.keyword   |VARCHAR        |keyword
year            |BIGINT         |long
```

2. **To get the movies with year field less than 2000 and limit results to 10**

➢ curl -XPOST 127.0.0.1:9200/_sql?format=txt -d '

{"query": "SELECT title, year from movies where year < 2000 limit 10" }'

```
          title           |     year
--------------------------+---------------
Toy Story                 |1995
Jumanji                   |1995
Grumpier Old Men          |1995
Waiting to Exhale         |1995
Father of the Bride Part II|1995
Heat                      |1995
Sabrina                   |1995
Tom and Huck              |1995
Sudden Death              |1995
GoldenEye                 |1995
```

CANVAS AND SQL

- Firstly, we must create a work pad which can consist of single or multiple pages.
- Each page can consist of elements like charts, graphs, maps, etc.…

Four elements:

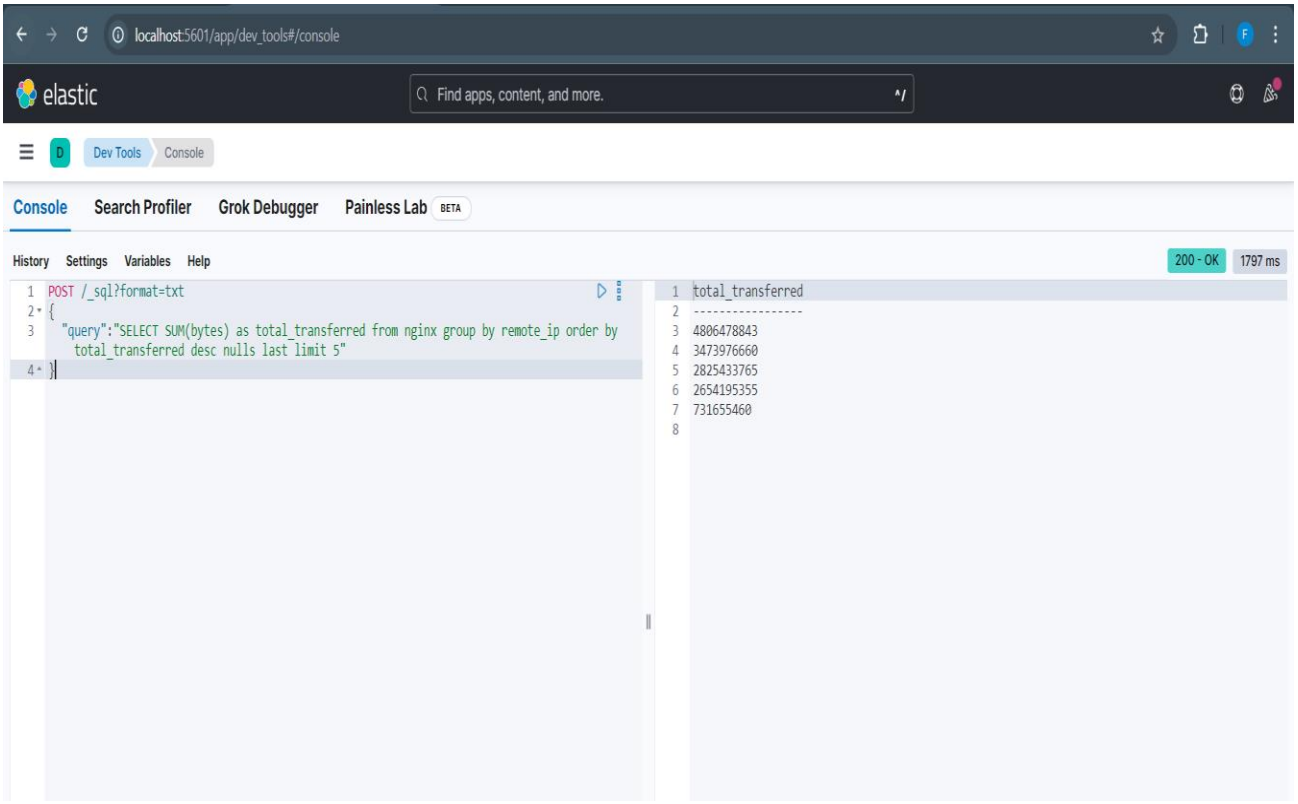
- Charts – bar chart, pie chart, doughnut, area, line, etc.…
- Shapes – text boxes
- Images – can have no. of images varied based on the live data from elastic search
- Supporting elements – dropdown, filter options

Canvas Data Sources

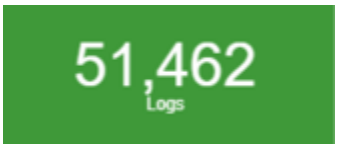
Elasticsearch SQL queries

Steps to create a canvas in Kibana

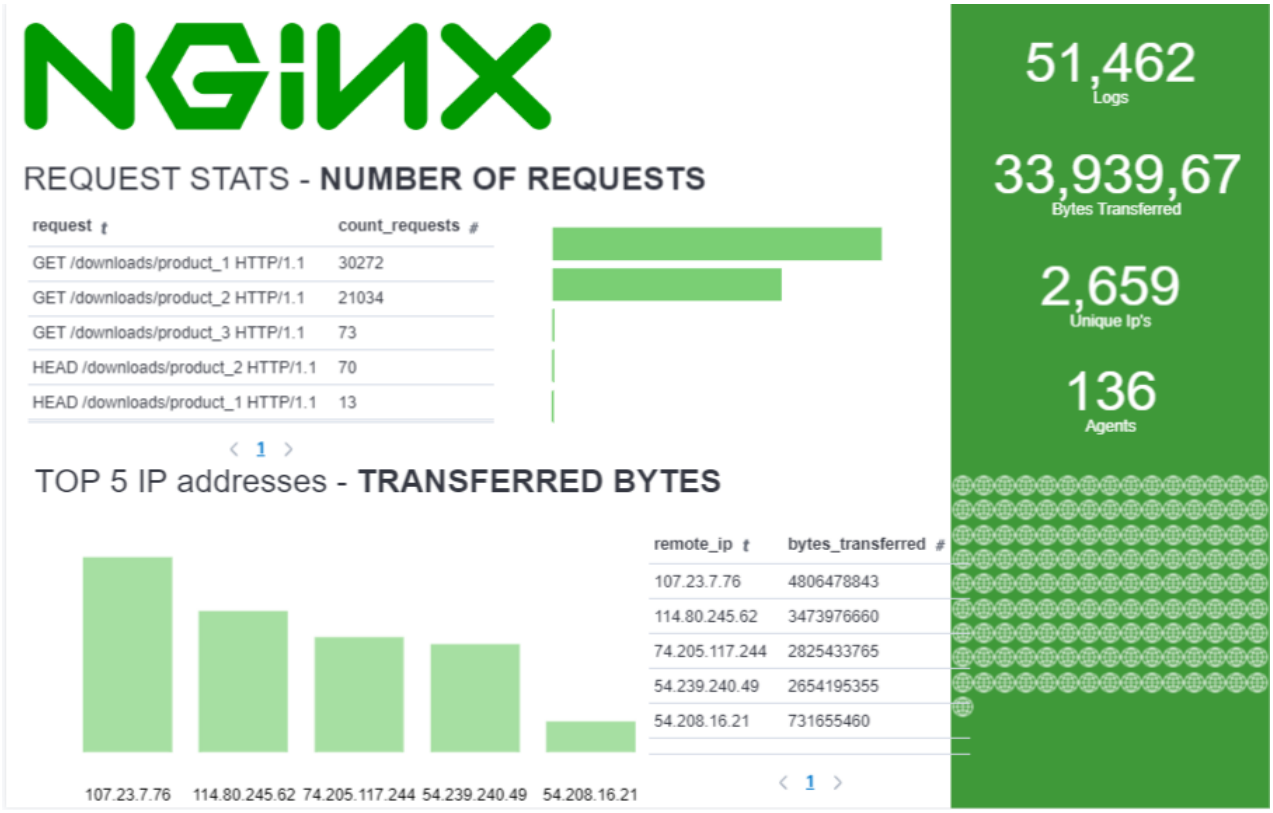
- Inject Log data into Elastic search index - For examples: **nginx** in our canvas
- Select the Kibana space in which we want to work
- Click on Kibana dev tools and check if the SQL queries work fine on the index we will be using to create canvas metrics



1. Navigate to Analytics -> Canvas -> Work pad
2. After creating new work pad, start adding elements, for example, metric element inside chart



3. In display tab, we can change the font properties of the metric and in data source using Elasticsearch SQL
   - **SELECT Count (*) AS count_document FROM nginx.**
4. In display we will use the value **count_document** to display the total logs in nginx index.
5. For inserting tables, insert the table element from charts and will get the data from elasticsearch SQL
   - **SELECT request, count (*) as count_requests FROM ngix GROUP BY request ORDER BY count_requests DESC**
6. Insert chart -> Bar chart element. Change the data source (will be same as table)
   In display, we will change the **x-axis to count_requests** and **y-axis to request.**



BACKUP AND TROUBLE SHOOTING

Categories

- Node setup
- Discovery and cluster formation
- Indexing data and sharding
- Searching
- Backing up data

Steps to perform Back up in Elasticsearch:

- sudo nano /etc/elasticsearch/elasticsearch.yml
- Now add path.repo : ["/home/student/backups"] after the path.logs in elasticsearch.yml
- sudo cp /etc/elasticsearch/elasticsearch.yml ~/
- sudo mkdir -p /home/student/backups
- sudo chgrp elasticsearch /home/student/backups
- sudo chmod g+w /home/student/backups/

- sudo /bin/systemctl stop elasticsearch.service
- sudo /bin/systemctl start elasticsearch.service
- curl --request PUT localhost:9200/_snapshot/backup-repo \

  --data-raw ' {

  "type" : "fs",

  "settings": {

  "location":"/home/student/backups/backup-repo"

  }

  }'

- curl --request PUT localhost:9200/_snapshot/backup-repo/snapshot-1
- curl --request GET localhost:9200/_snapshot/backup-repo/snapshot-1? Pretty

Potential Issues and Trouble Shooting in Elasticsearch:

Open a new PUTTY window – Terminal 2

sudo visudo

In the bottom of the file add:username ALL=(elasticsearch) NOPASSWD: ALL

sudo -su elasticsearch

cd /var/log/elasticsearch/

tail -n 500 elastisearch.log | grep ERROR

cat Elasticsearch.log | grep Bootstrap --context=3

MEMORY LOCK ISSUE:

In Terminal-2:

- sudo nano /etc/elasticsearch/elasticsearch.yml
- Uncomment the line bootstrap.memory_lock:true
- sudo systemctl stop elasticsearch.service
- sudo systemctl start elasticsearch.service

**It will throw you an error: It will show memory is locked error

Go back to Terminal-1

- sudo systemctl edit elasticsearch.service
- Add the following in the file to resolve the above error:
  [Service]
  LimitMEMLOCK=infinity
- sudo systemctl start elasticsearch.service

HEAP MEMORY ALLOCATION ISSUE:

In Terminal-2:

- sudo nano /etc/elasticsearch/jvm.options
- Comment out both -Xmslg and Xmxlg in the jvm file
- And add:
  -Xms500m
  -Xmslg
- sudo systemctl stop elasticsearch.service
- sudo systemctl start elasticsearch.service

**It will throw error initial heap size not equal to the initial allocation error

- sudo nano /etc/elasticsearch/jvm.options
- Uncomment out both -Xmslg and Xmxlg
- And remove:
  -Xms500m
  -Xmslg
- sudo systemctl stop elasticsearch.service
- sudo systemctl start elasticsearch.service

NODE SETUP ISSUES:

In Terminal-2

- sudo cat /usr/lib/system/system/elasticsearch.service
- sudo nano /etc/elasticsearch/elasticsearch.yml
- Comment out:
  discovery.seed_hosts: ["127.0.0.1"] and
  cluster.initial_master_nodes: ["node-1"]

**It will throw master not found exception

- sudo systemctl stop elasticsearch.service

In Terminal-1

- rm -rf /var/lib/elasticsearch/*
- Go back to terminal 2 and do
- sudo vim /etc/elasticsearch/elasticsearch.yml
- Uncomment and change:
  cluster-name:lecture-cluster
  discovery.seed_hosts: ["127.0.0.1:9301"]
  sudo systemctl start elasticsearch.service

**The cluster_uuid will be na which means the cluster is not formed.

sudo systemctl stop elasticsearch.service

Reasons for not forming cluster:

It may be due to network issues where nodes within cluster might be unable to communicated with each other.

Creating index with 1 shard and 1 replica:

- Curl –request PUT localhost:9200/test \
  --data-raw '{
  "settings":{
  "number_of_shards":1,
  "number_of_replicas":1}}'

- To check about the shard's status using:
  1.Curl localhost:9200/_cat/shards? V
  2. It will return that the status as started or unassigned.

- Cluster allocation API to explain why shards aren't allocated
  1.Curl localhost:9200/_cluster/allocation/explain? Pretty
  2.Reason – replica to the same node is not allowed.
  3.How to overcome? Add a new node and take replica to the new node.

Steps to setup 2nd node:

- Sudo nano /etc/elasticsearch-node2/elasticsearch.yml
  Node.name : node-2
  Master.nodes will be node-1 and node-2

- Start the 2nd node on the same VM
  Sudo systemctl start elasticsearch-node2
- To simulate red status
  1.Create index with 2 shards and no. of replicas to 0 (no backups)

  2.Curl –request PUT localhost:9200/test \

  --data-raw '{
  "settings":{
  "number_of_shards":2,
  "number_of_replicas":0
  }}'

  3.Two shards will be in 2nd nodes in the cluster

- We will stop node-2 and cut communication between 2 nodes in the cluster
- Now if we check status of cluster, it will be red
  Curl –silent localhost:9200/_cluster/health?pretty | grep status
- To overcome this, we can allocate a new shard
  Curl –request POST "localhost:9200/_cluster/reroute?pretty" \
  --data-raw '{
  "commands": {
  {
  "allocate_empty_primary":{
  "index":"test",
  "shard":1,
  "node":"node-1"
  "accept_data_loss":"true"}}]}'

- Since out data is loss, we will be using backup data
- Move configuration file to previous place (default setting)

INDEX DESIGN CHANGES (SPLITTING, SHRINKING)

Index settings:

➔ Dynamic – can be changed after index creation
- Number_of_replicaas
- Refresh intervals
- Blocks – disabling readability/writability of index
- Pipeline – preprocessing pipeline for every documents

➔ Static – can't be changed after index creation

Number_of_shards

Sharding goals:

High availability - working uninterrupted for a long time

High resiliency – resist errors

Increase/decrease shards

To decrease

- POST /{source_index}/_shrink/{target_index-name}

To increase

- POST /{source_index}/_split/{target_index-name}

SNAPSHOTS

- Backups to NAS, Amason S3, Azure
- Store only changes
- Elasticsearch.yml
- Path.repo : ["/home/<user>/backups"]

Setup Steps:

1. Configuration file
   - Path.repo : ["/home/student/backups"]
2. Mkdir backups
3. Restart elasticsearch.service
4. PUT _snapshot/backup-repo

   {

   "type":fs,

   "settings":{

   "location":"/home/student/backups/backup-repo"}}

SPRINGBOOT AND ELASTIC STACK

Spring boot-Elastic Stack

Spring Boot application sending logs to "Elastic Search Logstash Kibana" demo

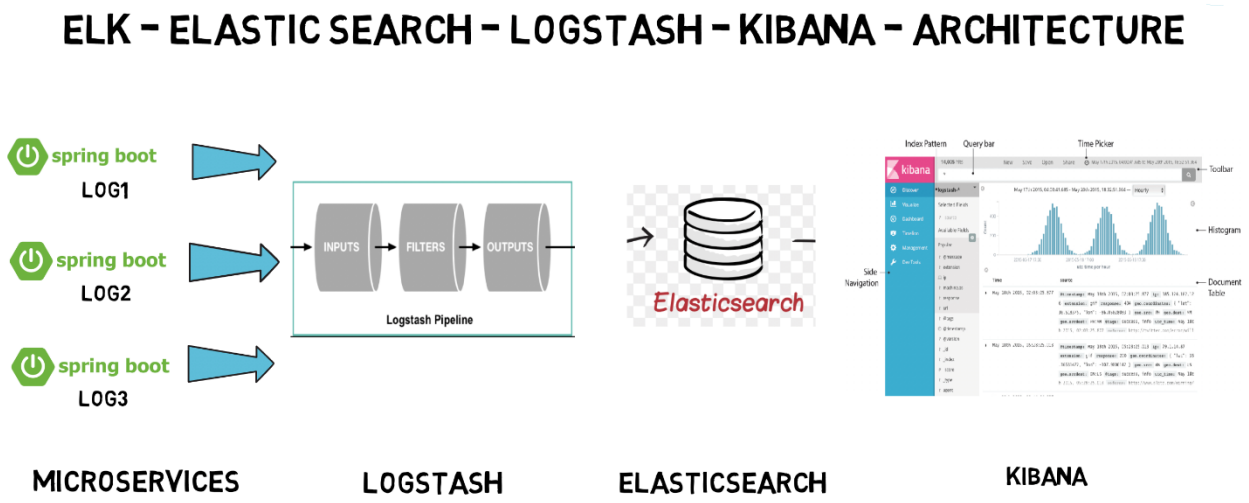
Why Elastic Stack?

With use of microservices, we have been able to overcome many traditional problems and it allow us to create stable distributed applications with desired control on the code, team size, maintenance, release cycle, cloud ennoblement, automation, etc. But it has also introduced few challenges in other areas e.g. "distributed log management" and ability to view logs of full transaction distributed among many services and distributed debugging in general.

Actually, the challenge is that microservices are isolated among themselves and they do not share common database and log files. As the number of microservice increases and we enable cloud deployment with automated continuous integration tools, it is very much necessary to have some provision of debugging the components when we have any problem.

Thanks to the open source. We already have bundle of tools which can do the magic if used properly together. One such popular set of tools are Elastic Search, Logstash and Kibana – together referred as ELK stack. They are used for searching, analyzing, and visualizing log data in a real time.

ELK Stack with SPRINGBOOT & MICROSERVICES:



Logstash processes the application log files based on the filter criteria we set and sends those logs to Elasticsearch. Through Kibana, we view and analyze those logs when required. Elasticsearch is a distributed, JSON-based search and analytics engine designed for horizontal scalability, maximum reliability, and easy management.

Logstash is a dynamic data collection pipeline with an extensible plugin ecosystem and strong Elasticsearch synergy.
Kibana gives the visualization of data through a UI.

Elastic stack configuration
All these three tools are based on JVM and before start installing them, please verify that JDK has been properly configured. Check that standard JDK 1.8 installation, JAVA_HOME and PATH set up is already done.

Elasticsearch
Download latest version of Elasticsearch from this download page https://www.elastic.co/downloads/elasticsearch and unzip it any folder.
Run bin\elasticsearch.bat from command prompt.
By default, it would start at http://localhost:9200

Kibana
Download the latest distribution from download page https://www.elastic.co/de/downloads/kibana and unzip into any folder.
Open config/kibana.yml in an editor and set elasticsearch.url to point at your Elasticsearch instance. In our case as we will use the local instance just uncomment elasticsearch.hosts: "http://localhost:9200"
Run bin\kibana.bat from command prompt.
Once started successfully, Kibana will start on default port 5601 and Kibana UI will be available at http://localhost:5601

Logstash
Download the latest distribution from download page https://www.elastic.co/downloads/logstash and unzip into any folder.
Create one file logstash.conf as per configuration instructions.
Now run bin/logstash -f logstash.conf to start logstash

Logstash configuration

We need to create a logstash configuration file so that it listen to the log file and push log messages to elastic search.

```
input {
  file {
    type => "java"
    path => "<PLEASE_UPDATE_YOUR_LOG_FILE_PATH_HERE>"
    codec => multiline {
      pattern => "^%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME}.*"
      negate => "true"
      what => "previous"
    }
```

```
  }
}

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"]
    }
  }

 grok {
    match => [ "message",
              "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- \[(?<thread>[A-Za-z0-9-]+)\] [A-Za-
z0-9.]*\.(?<class>[A-Za-z0-9#_]+)\s*:\s+(?<logmessage>.*)",
              "message",
              "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- .+? :\s+(?<logmessage>.*)"
            ]
  }


  date {
    match => [ "timestamp" , "yyyy-MM-dd HH:mm:ss.SSS" ]
  }
}

output {

  stdout {
    codec => rubydebug
  }

  # Sending properly parsed log events to elasticsearch
  elasticsearch {
    hosts => ["localhost:9200"]
  }
}
```

Once ELK stack is up and running. You could start this demo project and call the REST endpoints, view the logs in KIBANA.

Do a final maven build using mvn clean install and start the application and test by browsing http://localhost:8080/elk or

http://localhost:8080/elkdemo or http://localhost:8080/exception

Don’t be afraid by seeing the big stack trace in the screen as it has been done intentionally to see how ELK handles exception message.

Go to the application root directory and verify that the log file i.e. elk-example.log has been created and do a couple of visits to the endpoints and verify that logs are getting added in the log file.

Test The Demo Application with ELK

Now when all components are up and running, let’s verify the whole ecosystem.

Go to application and test the end points couple of times so that logs got generated and then go to Kibana console and see that logs are properly stacked in the Kibana with lots of extra features like we can filter, see different graphs etc. in built.

Here is the view of generated logs in Kibana.