ELK:

Elastic search

- Elasticsearch is a NoSQL search engine.
- It provides a distributed, full-text search engine with an HTTP web interface and of JSON documents.
- Elasticsearch is developed in Java and is released as open source under the terms of the Apache License.
- Official clients are available in Java, .NET (C#), PHP, Python, Apache Groovy, Ruby and many other languages.
- According to the DB-Engines ranking, Elasticsearch is the most popular enterprise search engine.
- Elasticsearch does not have tables, and a schema is not required.
- Elasticsearch stores data documents that consist of JSON strings inside an index.
- Elasticsearch is a distributed, **RESTful** search and analytics engine.
- Elasticsearch is a NRT (near real time) search platform.
- What this means is there is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.

Terminology

- **Cluster**
    - A cluster is a collection of one or more nodes (servers) that together holds your entire data and provides federated indexing and search capabilities across all nodes.
- **Node**
    - A node is a single server that is part of your cluster, stores your data, and participates in the cluster's indexing and search capabilities.
- **Index**
    - An index is a collection of documents that have somewhat similar characteristics.
    - For example, you can have an index for customer data, another index for a product catalog, and yet another index for order data.
    - An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it.
    - In a single cluster, you can define as many indexes as you want.

- **Document**
  - A document is a basic unit of information that can be indexed.
  - For example, you can have a document for a single customer, another document for a single product, and yet another for a single order.
  - This document is expressed in JSON (JavaScript Object Notation).
- **Shards and replicas**
  - Every time you index a document elasticsearch will decide which primary shard is supposed to hold that document and will index it there.
  - Having multiple shards helps taking advantage of parallel processing on a single machine, but the whole point is that if we start another elasticsearch instance on the same cluster, the shards will be distributed in an even way over the cluster.
  - Every elasticsearch index is composed of at least one primary shard since that's where the data is stored.
  - Another type of shard is a replica. The default is 1, meaning that every primary shard will be copied to another shard that will contain the same data.
  - Replicas are used to increase search performance and for fail-over.

Setup:

- Elastic download is available in: https://www.elastic.co/downloads/elasticsearch
- Download the zip file

- Once download is done, extract the zip file to a known location
- Open the bin folder (e.g. “cd <Elastic search unzipped folder>\bin”) and run **elasticsearch** file (or **elasticsearch.bat** on Windows) and wait until you see: started
- Open any browser and navigate to your local elastic search engine at http://localhost:9200/
- The engine information will show.
- Currently we have no data stored.

CRUD operations (create, read, update, delete):

- **Create**
  - To create a new Document, we use an POST request method.
  - We can post to the following server address: **http://localhost:9200/my_app/users/1**
  - The path can be anything, but we will need to remember it for getting this data later..

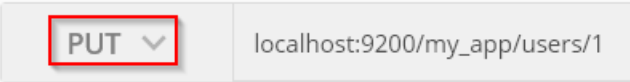
- **Read**
  - To query data we can go to a specific index as we did earlier: **localhost:9200/my_app/users/1**
  - Another way to retrieve all relevant data will be using the **_search** which will return all available data from a specific point.
  - For example the following URL will return all users: **localhost:9200/my_app/users/_search**
  - We can also use query parameters using "?q=:" syntax in Query to filter the records:

    localhost:9200/my_app/users/_search**?q=_id:2**

OR

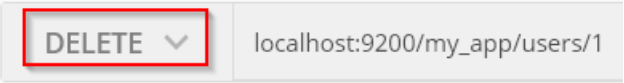
localhost:9200/my_app/users/_search**?q=name:John**

- **Update**
  - To update an existing document data, we will use **PUT** method:
  - The address will be the same as the “posting address” (localhost:9200/my_app/users/1) except this time we will be using **PUT** method:



  - Inside the body, simply change anything you want to update and press **send**
  - To simply test result, run search again (localhost:9200/my_app/users/_search)

- **Delete**
  - To delete an existing document data, we will use **DELETE** method:
  - The address will be the same as the “posting address” (localhost:9200/my_app/users/1) except this time we will be using **DELETE** method:



  - In delete, there no reason to add request body..
  - In the example above we changed a user name
  - To simply test result, run search again (localhost:9200/my_app/users/_search)

Kibana

- Kibana is an open source data visualization plugin for Elasticsearch.
- It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.
- Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.
- The combination of Elasticsearch, Logstash, and Kibana, referred to as the "Elastic Stack" (formerly the "ELK stack"), is available as a product or service.
- Logstash provides an input stream to Elastic for storage and search, and Kibana accesses the data for visualizations such as dashboards.

Setup:

- Kibana download is available in: https://www.elastic.co/downloads/kibana
- Choose yorur OS

- Once download is done, extract the zip file to a known location
- Open the config folder (e.g. “cd <Kibana unzipped folder>\config”) and open **config/kibana.yml** in notepad/++
- Find the line contains **elasticsearch.url** and remove the hash(#) from the beginning of the line and save the file.

```
# The URL of the Elasticsearch instance to use for all your queries.
#elasticsearch.url: "http://localhost:9200"
```

- This will point Kibana to our Elastic search instance.
- Now go to bin folder and run **bin/kibana** (or **bin\kibana.bat** on Windows)
- Open any browser and navigate to your local elastic search engine at http://localhost:5601/

```
[info][listening] Server running at http://localhost:5601
```

- Once you are inside Kibana management tab, you will need to define what the index pattern is, you want Kibana to “listen” to.
- Simply use the one from earlier like the following: my_app/*
- You should then see success message

- You can now press **next step** and then **create index pattern** and wait until indexing is done.
- Now, we can go to visualize tab → Create new visualization for the example we will choose Metric.
- Choose your index and give it a name, and press save button
- To see the results immediately, we can go to Postman and create more records and then press refresh.

  ** We can create as many dashboards / visualization as we need

LogStash

- Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite "stash." (Ours is Elasticsearch, naturally.)
- Logstash is used to gather logging messages, convert them into json documents and store them in an ElasticSearch cluster.
- Of course those can be viewed in Kibana, which makes the ELK stack so powerful

Setup:

- Logstash download is available in: https://www.elastic.co/downloads/logstash
- Download the zip file

- Once download is done, extract the zip file to a known location
- Open the bin folder (e.g. "cd <Logstash unzipped folder>\bin") and create a file **logstash.conf** the following configuration should be inside:

```
input {
  file {
    path => "C:/LogStash/myLogs/log_file.txt"
  }}
output {
    stdout {
        codec => rubydebug
    }
  elasticsearch {
    hosts => ["localhost:9200"]
    action => "index"
    index => "log"
}}
```

Logs input source

Using rubbydebug to see logs in cosole

Elastic search path and index for the data to be stored

Grok

- **Grok** is a Logstash plugin used to parse arbitrary text and structure it.
- **Grok** is a great way to parse unstructured log data into something structured and queryable.
- This tool is perfect for syslog logs, apache and other webserver logs, mysql logs, and in general, any log format that is generally written for humans and not computer consumption.
- Logstash ships with about 120 patterns by default. You can find them here: https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns
- Let's take an easy server access log which holds the IP address and the method of the request:

  192.168.99.100 GET

- To filter this type of messages, we can add a filter using grok inside **Logstash.conf** which will look like that:

```
filter {
grok {
match => { "message" => "%{IP:client} %{WORD:method}" }
    }
}
```

Grafana

- Grafana is an open-source, general purpose dashboard and graph composer, which runs as a web application.
- Grafana allows you to query, visualize, alert on
  and understand your metrics no matter where they are stored.
- Create, explore, and share dashboards with your team and foster a data driven culture.
- Our data source will be elastic search

Setup:

- Download **Grafana** at:
  http://docs.grafana.org/installation/windows/
- Extract the zip to a known location
- Double click **grafana-server.exe**
- Enter **Grafana** default address at **localhost:3000**
- Default user name and password are **admin**
- Once entered, you will be asked to change password it for security reasons.

- Connect data source by pressing add data source
- Then, add the below configurations.
- We will use **log** index name to see the Logstash index we created earlier.
- Press save and test button.
- If everything is OK, you should see “Index OK”

- Like Kibana, Grafana is a visualization tool, so we will need to setup our dashboard.
- To create a new dashboard/s, press **Dashboard → Home → New dashboard → Graph → Panel options → Edit and add a new graph.**

- Grafana has an alert system, so you can configure alerts to be sent in real time
- To configure the alert system go to **/conf** folder and open **defaults.ini** file to edit.
- Once email is setup, go to test it by adding a recipient here:
- This will only work if "**Less secured apps**" is configured in your Gmail account: https://myaccount.google.com/u/1/security