Exercise 1

Step 1: See a simple LogStash-Pipeline in Action

1.
   ```
   docker compose -f docker-compose.yml up -d
   ```

2.
   ```
   docker exec -it elk-stack-dock-logstash-1 /bin/bash
   ```

3. If you start a shell with Docker Desktop enter `bash` to use a more comfortable shell.

4. Find out where logstash is installed

   `which logstash`

   /usr/share/logstash/bin/logstash

5. `cd` to bin directory of logstash

   `cd /usr/share/logstash/`

6. Test a simple pipeline with input from and output to commandline.

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

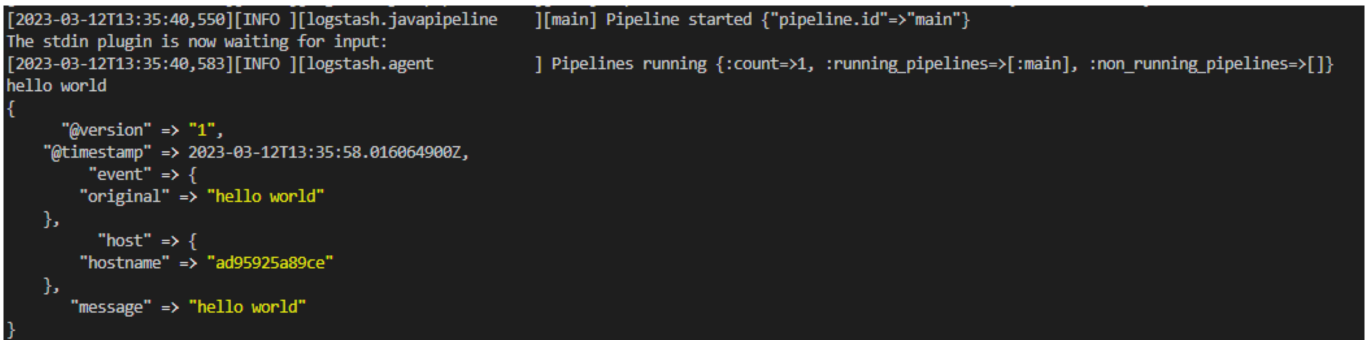
7. When using a docker container with an running Instance of logstash you have to use another `data` directory.

8. You have to start the second logstash process with the flag `--path.data data2`

   `bin/logstash -e 'input { stdin { } } output { stdout {} }' --path.data data2`

9. Wait until pipeline is running

   `[INFO ] 2022-02-14 07:56:05.167 [Agent thread] agent - Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}` Then type `hello world`

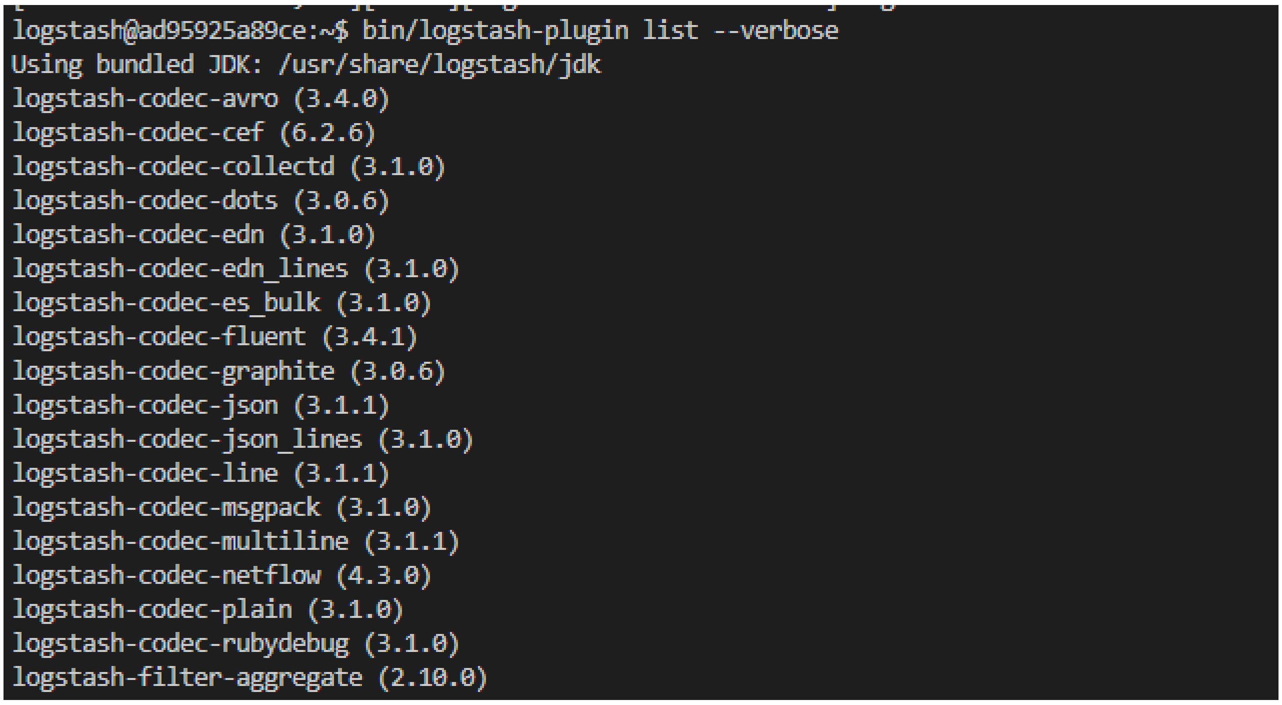


```
[2023-03-12T13:35:40,550][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[2023-03-12T13:35:40,583][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
hello world
{
      "@version" => "1",
    "@timestamp" => 2023-03-12T13:35:58.016064900Z,
         "event" => {
        "original" => "hello world"
    },
          "host" => {
        "hostname" => "ad95925a89ce"
    },
       "message" => "hello world"
}
```

10. Stop the pipeline with `STRG-D`

11. List logstash plugins with its version

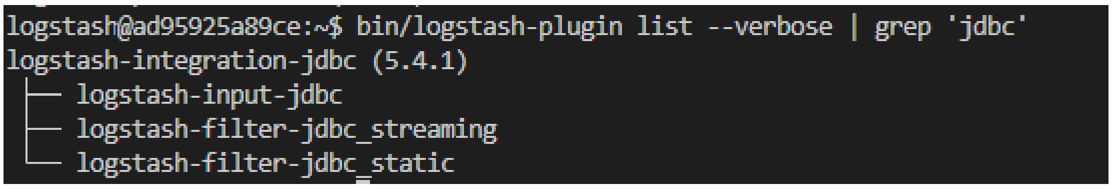
    `bin/logstash-plugin list --verbose`



```
logstash@ad95925a89ce:~$ bin/logstash-plugin list --verbose
Using bundled JDK: /usr/share/logstash/jdk
logstash-codec-avro (3.4.0)
logstash-codec-cef (6.2.6)
logstash-codec-collectd (3.1.0)
logstash-codec-dots (3.0.6)
logstash-codec-edn (3.1.0)
logstash-codec-edn_lines (3.1.0)
logstash-codec-es_bulk (3.1.0)
logstash-codec-fluent (3.4.1)
logstash-codec-graphite (3.0.6)
logstash-codec-json (3.1.1)
logstash-codec-json_lines (3.1.0)
logstash-codec-line (3.1.1)
logstash-codec-msgpack (3.1.0)
logstash-codec-multiline (3.1.1)
logstash-codec-netflow (4.3.0)
logstash-codec-plain (3.1.0)
logstash-codec-rubydebug (3.1.0)
logstash-filter-aggregate (2.10.0)
```

12. See if jdbc-integration-plugin is installed

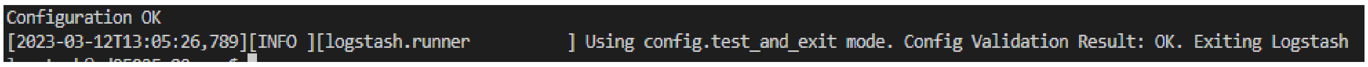
    `bin/logstash-plugin list --verbose | grep 'jdbc'`



```
logstash@ad95925a89ce:~$ bin/logstash-plugin list --verbose | grep 'jdbc'
logstash-integration-jdbc (5.4.1)
 ├── logstash-input-jdbc
 ├── logstash-filter-jdbc_streaming
 └── logstash-filter-jdbc_static
```

13. Test the logstash configuration with

    ```
    /usr/share/logstash/bin/logstash --path.settings /usr/share/logstash/config
    -t
    ```



```
Configuration OK
[2023-03-12T13:05:26,789][INFO ][logstash.runner          ] Using config.test_and_exit mode. Config Validation Result: OK. Exiting Logstash
```

Step 2: Working with LogStach jdbc and PostgreSQL

Work with logstash as shown in Step 1

The next steps are useful for testing a jdbc-connection to a Postgres container

- replace ip with ip or name of your host
- host.docker.internal:postgresport if you use a docker container with a separate network
- use postgres connector instead of mariadb in the following example

```
bin/logstash --path.data data2 -e 'input {
      jdbc {
      jdbc_connection_string => "jdbc:postgresql://10.3.105.88:5432/onlineshop"
      jdbc_user => "postgres"
      jdbc_password => "geheim"
      jdbc_driver_class => "org.postgresql.Driver"
      statement => "SELECT * FROM visit"
      }
      } output { stdout {} }'
```

- Open a commandline
- See this medium blog for how to setup a pipeline with jdbc.

`curl https://jdbc.postgresql.org/download/postgresql-42.5.4.jar -o /usr/share/logstash/logstash-core/lib/jars/postgresql-jdbc.jar`

1. Create a table in postgres. For Example: orders, visits, order_items

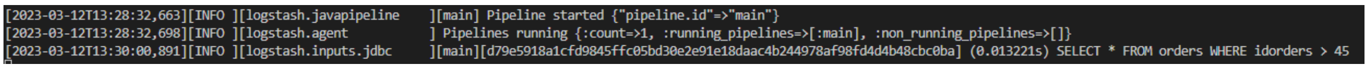
```
psql -U postgres

CREATE DATABASE onlineshop;
\connect onlineshop
```

2. See Examples.
3. create a configuration file to get data from postgres public.orders|visits|order_items in /usr/share/logstash/pipeline/logstash.conf
4. See jdbc-input Doku
5. Add tracking for PK column with :sql_last_value
6. You have to add a line for scheduling to make logstath run without interruption: `schedule => "*/5 * * * *"`
   - This file is mapped to `<path-to-your-docker>/elk-stack-dock/logstash/pipeline/logstash.conf`
   - optional: configure a pipeline in `/usr/share/logstash/config/pipelines.yml` (a default is there). This file is not mapped.

Complete file:

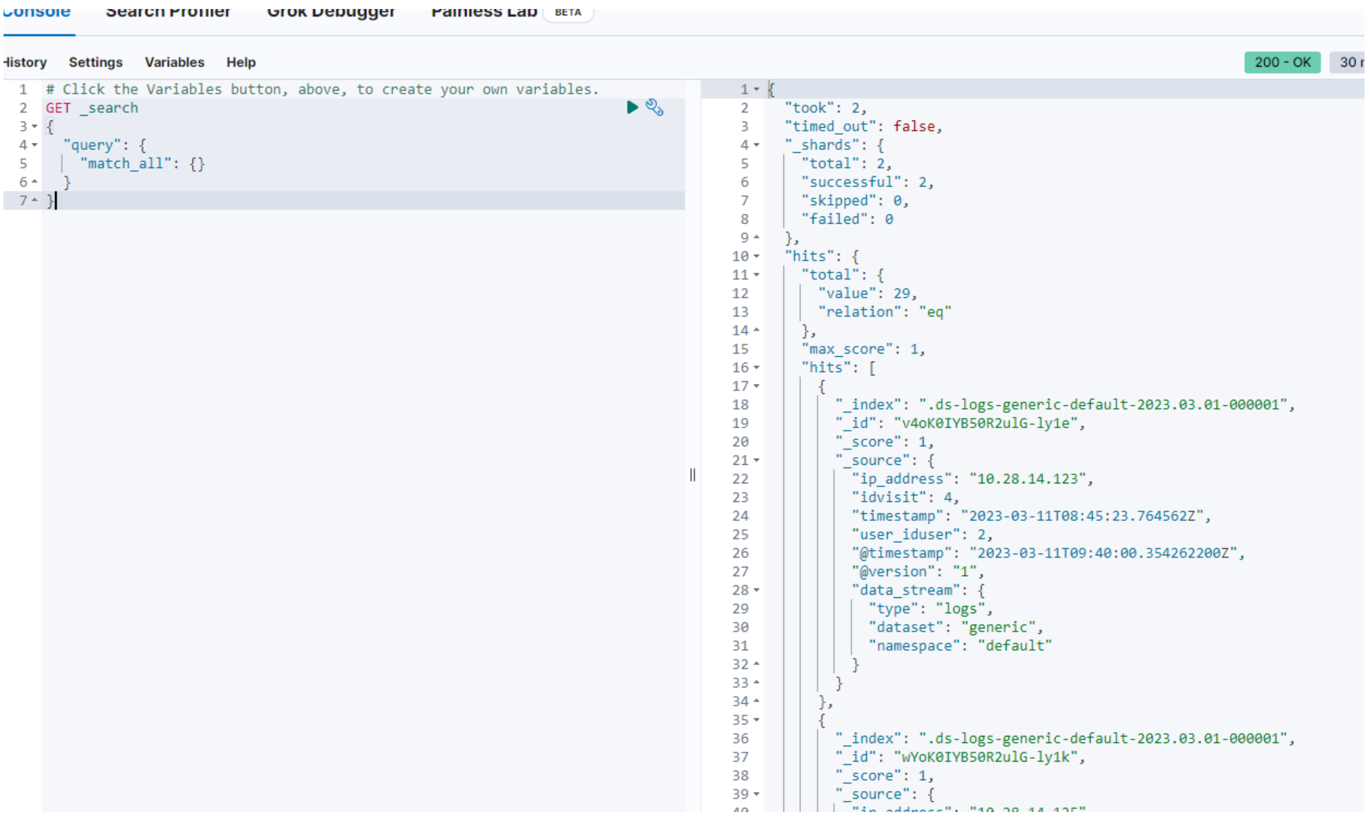
```
input {
    jdbc {
        jdbc_connection_string => "jdbc:postgresql://10.3.105.88:5432/onlineshop"
        jdbc_user => "postgres"
        jdbc_password => "geheim"
        jdbc_driver_class => "org.postgresql.Driver"
        statement => "SELECT * FROM orders WHERE idorders > :sql_last_value"
        use_column_value => true
        tracking_column => "idorders"
        schedule => "*/5 * * * *"
    }
}

output {
    elasticsearch {
        hosts => "elasticsearch:9200"
        user => "logstash_internal"
        password => "${LOGSTASH_INTERNAL_PASSWORD}"
        index => "orders"
        document_id => " orders_%{idorders}"
        doc_as_upsert => true
    }
}
```

Execute pipeline with `bin/logstash -f pipeline/logstash.conf --path.data data2`



```
[2023-03-12T13:28:32,663][INFO ][logstash.javapipeline    ][main] Pipeline started {"pipeline.id"=>"main"}
[2023-03-12T13:28:32,698][INFO ][logstash.agent           ] Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
[2023-03-12T13:30:00,891][INFO ][logstash.inputs.jdbc     ][main][d79e5918a1cfd9845ffc05bd30e2e91e18daac4b244978af98fd4d4b48cbc0ba] (0.013221s) SELECT * FROM orders WHERE idorders > 45
```

7. Go to Kibana -> Management -> Dev Tools and query logstash index, to see the data rows stored in ES by the pipeline built.



```
# Click the Variables button, above, to create your own variables.
GET _search
{
  "query": {
    "match_all": {}
  }
}
```

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 2,
    "successful": 2,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 29,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": ".ds-logs-generic-default-2023.03.01-000001",
        "_id": "v4oK0IYB50R2ulG-ly1e",
        "_score": 1,
        "_source": {
          "ip_address": "10.28.14.123",
          "idvisit": 4,
          "timestamp": "2023-03-11T08:45:23.764562Z",
          "user_iduser": 2,
          "@timestamp": "2023-03-11T09:40:00.354262200Z",
          "@version": "1",
          "data_stream": {
            "type": "logs",
            "dataset": "generic",
            "namespace": "default"
          }
        }
      },
      {
        "_index": ".ds-logs-generic-default-2023.03.01-000001",
        "_id": "wYoK0IYB50R2ulG-ly1k",
        "_score": 1,
        "_source": {
```

8. Compare with MongoDB Pipeline

- See https://www.mongodb.com/developer/products/mongodb/mongoimport-guide/ to get a glue how MongoDB supports you in this case.
- See for a video-description: https://www.mongodb.com/developer/videos/import-data-into-mongodb-3-ways---bonus--export-data-from-postgres/
- MongoDB Compass can be installed locally, but Docker project was last updated in 2021.

9. Write a short summary, what is different to ELK and give a statement, what you think is easier to automate.

Both are tools for managing and analyzing data, but they have different approaches.

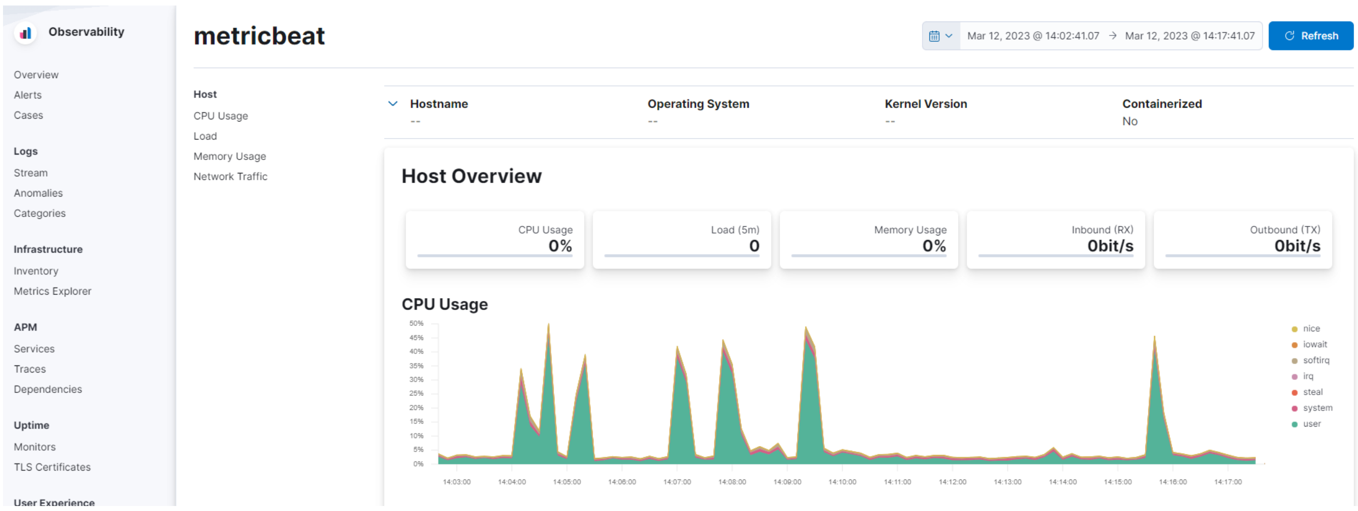
THe elk stack is designed for real-time analysis of log data. MongoDB pipeline is a tool for manipulating and processing data in MongoDB, which is a NoSQL document based database. It is designed for more general data processing. For complex data MongoDB would be easier because it provides a wide range of functions and operations with the Aggregation framework or MapReduce.

Step 3: Adding a Metricbeat pipeline and build dashboards in Kibana

- See Metricbeat README for this.
- First only start the ELK-Stack. See logs of Kibana, to see if all services are already running.
- In a browser connect to `http://localhost:5601`. username/pwd: elastic/changeme
- Then use the command to start metricbeat container

```
docker-compose -f docker-compose.yml -f extensions/metricbeat/metricbeat-compose.yml up
```

1. activate requested passwords for metricbeat in `<path-to-your-docker>/elk-stack-dock/.env`.
2. rerun setup container
3. restart metricbeat



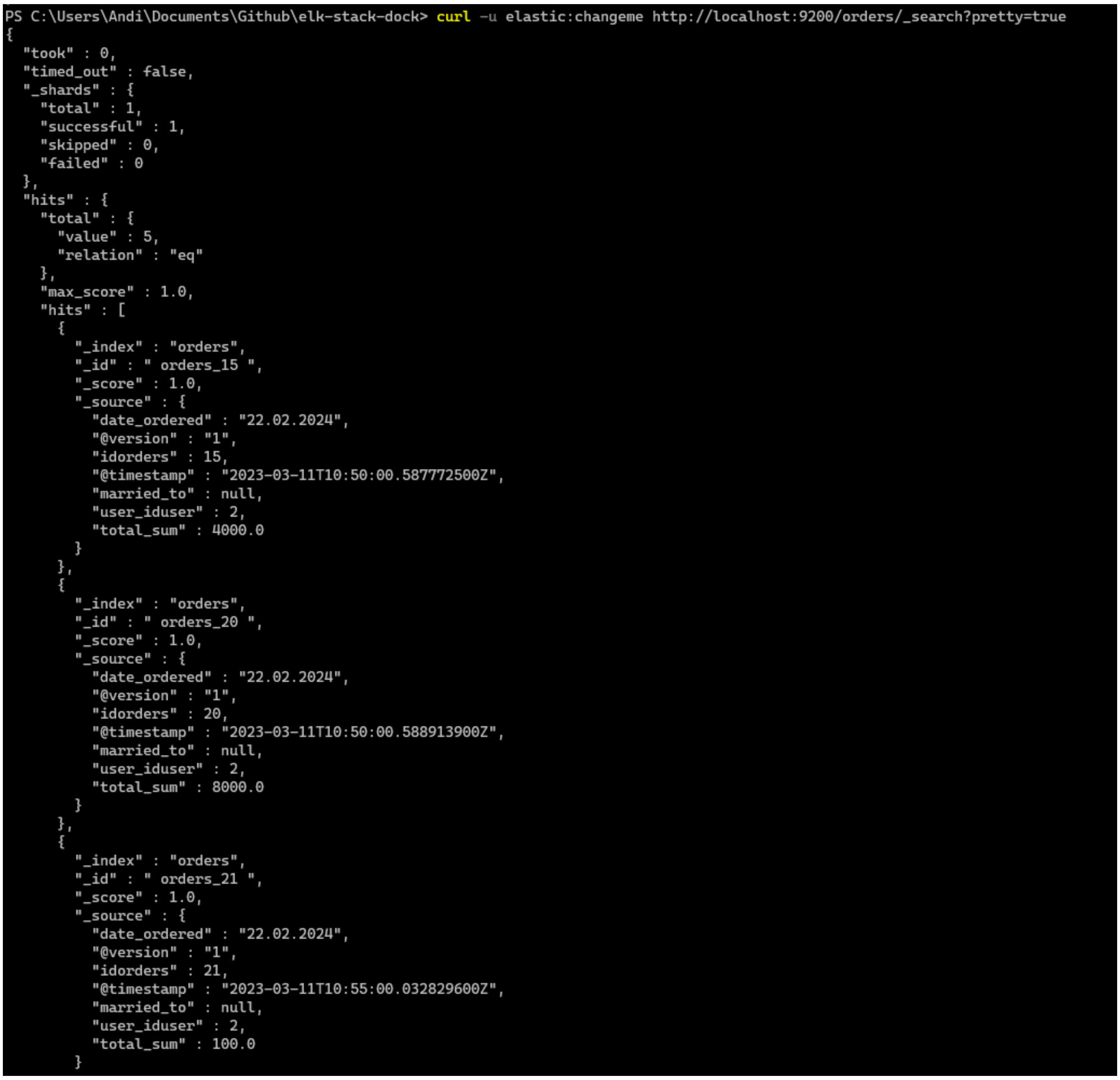
metricbeat

Hostname	Operating System	Kernel Version	Containerized
--	--	--	No

Host Overview

CPU Usage	Load (5m)	Memory Usage	Inbound (RX)	Outbound (TX)
0%	0	0%	0bit/s	0bit/s

4. Test, if data are retrieved by elasticsearch
   - `curl -u elastic:changeme http://localhost:9200/_search`
   - `curl -u elastic:changeme http://localhost:9200/logstash/_search`
   - `curl -u elastic:changeme http://localhost:9200/logstash/_search?pretty=true`

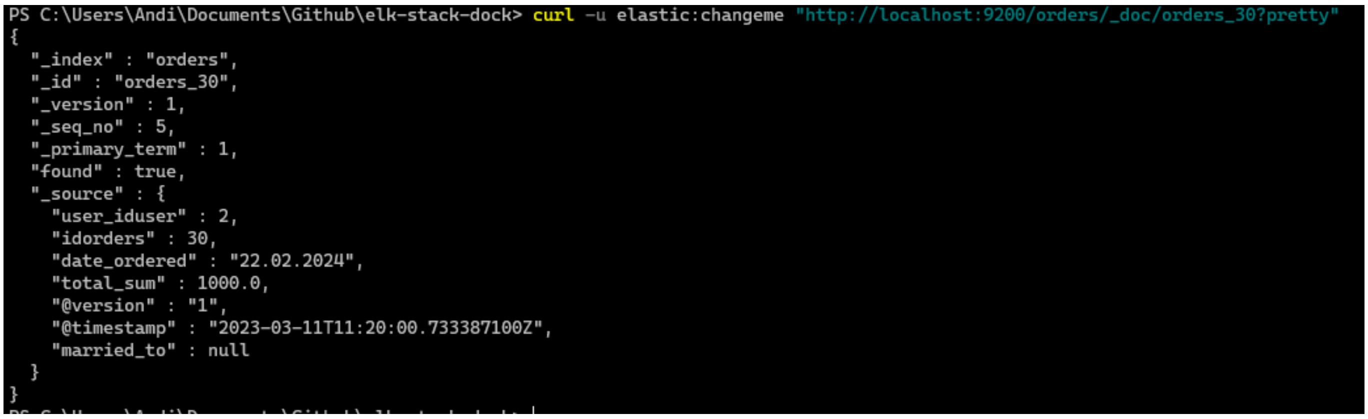


```
PS C:\Users\Andi\Documents\Github\elk-stack-dock> curl -u elastic:changeme http://localhost:9200/orders/_search?pretty=true
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "orders",
        "_id" : " orders_15 ",
        "_score" : 1.0,
        "_source" : {
          "date_ordered" : "22.02.2024",
          "@version" : "1",
          "idorders" : 15,
          "@timestamp" : "2023-03-11T10:50:00.587772500Z",
          "married_to" : null,
          "user_iduser" : 2,
          "total_sum" : 4000.0
        }
      },
      {
        "_index" : "orders",
        "_id" : " orders_20 ",
        "_score" : 1.0,
        "_source" : {
          "date_ordered" : "22.02.2024",
          "@version" : "1",
          "idorders" : 20,
          "@timestamp" : "2023-03-11T10:50:00.588913900Z",
          "married_to" : null,
          "user_iduser" : 2,
          "total_sum" : 8000.0
        }
      },
      {
        "_index" : "orders",
        "_id" : " orders_21 ",
        "_score" : 1.0,
        "_source" : {
          "date_ordered" : "22.02.2024",
          "@version" : "1",
          "idorders" : 21,
          "@timestamp" : "2023-03-11T10:55:00.032829600Z",
          "married_to" : null,
          "user_iduser" : 2,
          "total_sum" : 100.0
        }
```

```
See [Query DSL Doku]
(https://www.elastic.co/guide/en/elasticsearch/reference/8.6/query-dsl.html) for
the next steps.
```

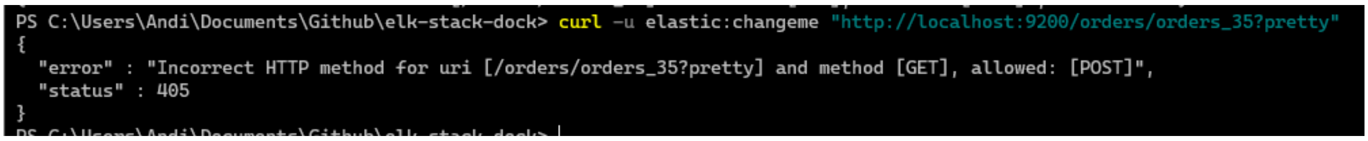
5. either find the curl command to query for a specific entry in a table with a match

```
curl -u elastic:changeme "http://localhost:9200/orders/_doc/orders_30?pretty"
```



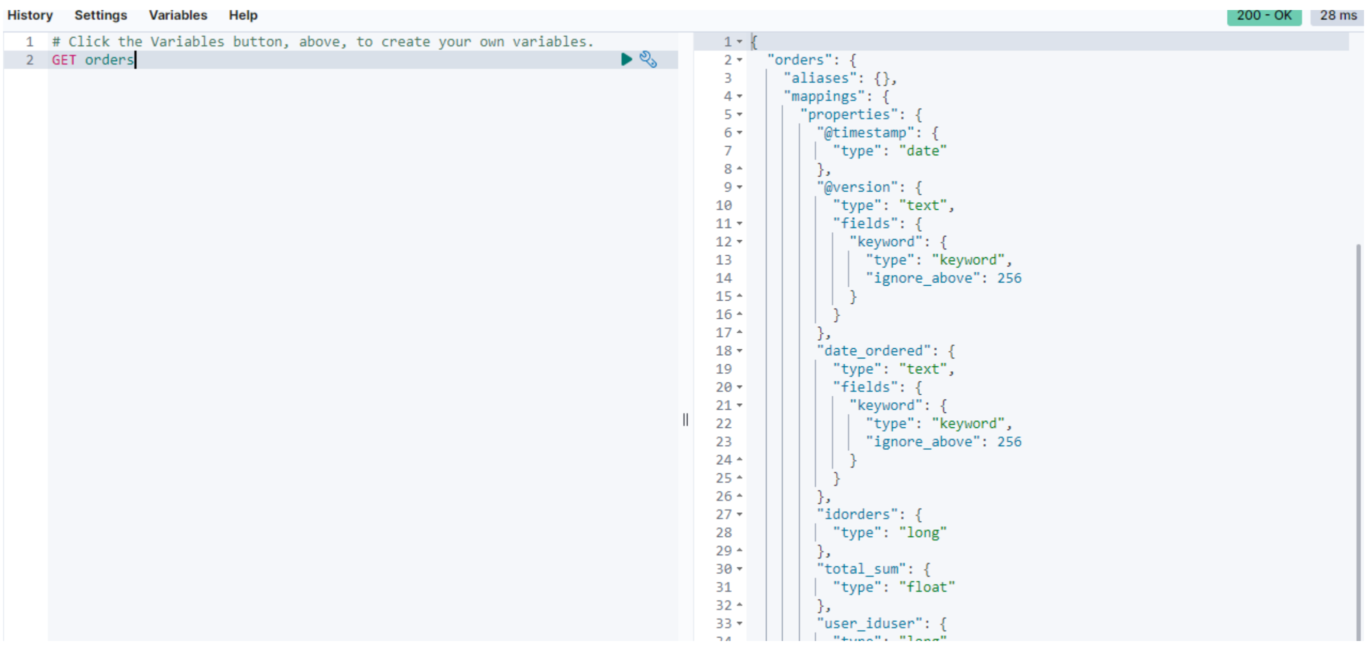
```
PS C:\Users\Andi\Documents\Github\elk-stack-dock> curl -u elastic:changeme "http://localhost:9200/orders/_doc/orders_30?pretty"
{
  "_index" : "orders",
  "_id" : "orders_30",
  "_version" : 1,
  "_seq_no" : 5,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "user_iduser" : 2,
    "idorders" : 30,
    "date_ordered" : "22.02.2024",
    "total_sum" : 1000.0,
    "@version" : "1",
    "@timestamp" : "2023-03-11T11:20:00.733387100Z",
    "married_to" : null
  }
}
```

6. and one, that doesn't match any database entry



```
PS C:\Users\Andi\Documents\Github\elk-stack-dock> curl -u elastic:changeme "http://localhost:9200/orders/orders_35?pretty"
{
  "error" : "Incorrect HTTP method for uri [/orders/orders_35?pretty] and method [GET], allowed: [POST]",
  "status" : 405
}
```

7. Connect to Kibana in Browser
8. Goto Burger Menu -> Management -> Dev Tools
9. Query for all entries in logstash index.

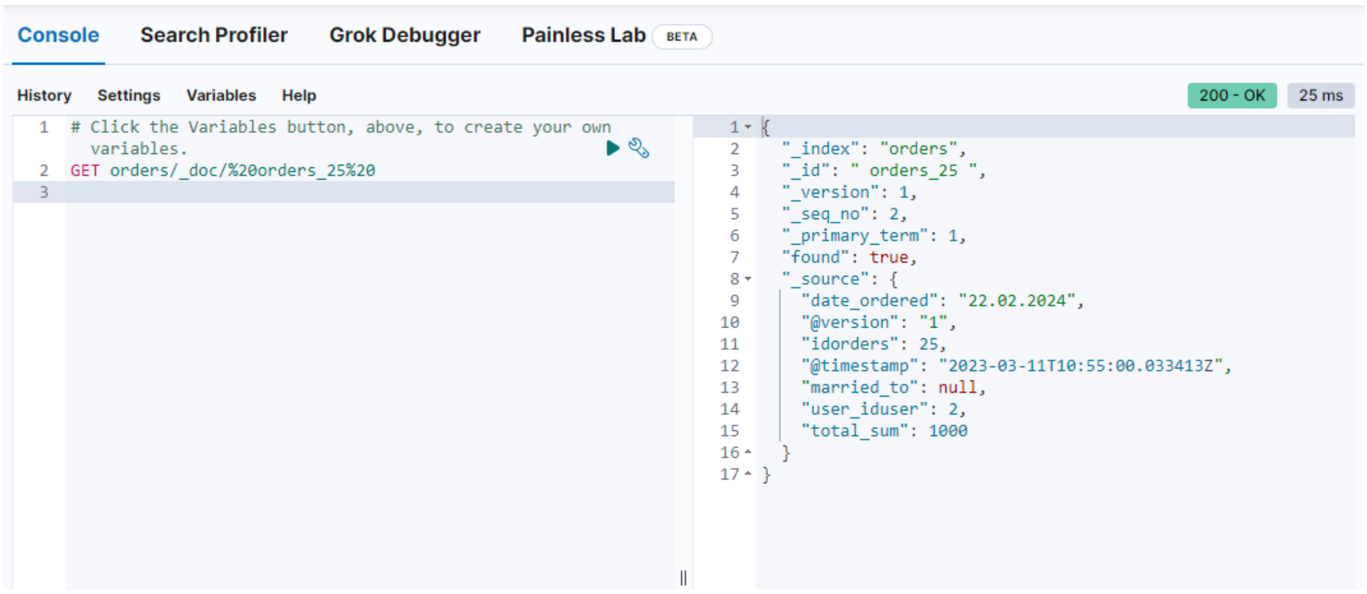


```
# Click the Variables button, above, to create your own variables.
GET orders
```

```
{
  "orders": {
    "aliases": {},
    "mappings": {
      "properties": {
        "@timestamp": {
          "type": "date"
        },
        "@version": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "date_ordered": {
          "type": "text",
          "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
            }
          }
        },
        "idorders": {
          "type": "long"
        },
        "total_sum": {
          "type": "float"
        },
        "user_iduser": {
```

11. Query for a specific entry, like you did with curl

```
GET orders/_doc/orders_25
```



```
# Click the Variables button, above, to create your own
  variables.
GET orders/_doc/%20orders_25%20
```

```
{
  "_index": "orders",
  "_id": " orders_25 ",
  "_version": 1,
  "_seq_no": 2,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "date_ordered": "22.02.2024",
    "@version": "1",
    "idorders": 25,
    "@timestamp": "2023-03-11T10:55:00.033413Z",
    "married_to": null,
    "user_iduser": 2,
    "total_sum": 1000
  }
}
```

12. Sort the entries in asc and desc order

Query all and sort with total_sum

```
GET orders/_search
{
  "sort" : [
    { "total_sum": {"order": "desc"}}
  ],
  "query": {
    "match_all": {}
  }
}
```