BAB IV

IMPLEMENTASI DAN PEMBAHASAN

4.1 Konfigurasi ELK-Stack

Dalam penelitian ini versi ELK-*stack* yang digunakan adalah 7.7.1 yang dijalankan terkontainerisasi menggunakan docker. Oleh karena itu sebelum membuat serta melakukan konfigurasi ELK-*stack* perlu disiapkan docker-*stack* pada *environtment* yang akan digunakan.

Direktori kerja penelitian ini memiliki 3 buah sub-direktori yaitu Elasticsearch, Logstash, dan Kibana. Hal ini dilakukan supaya berkas-berkas docker-*stack* maupun berkas-berkas konfigurasi ELK-*stack* terorganisir dengan baik.

4.1.1 Berkas Dockerfile

Untuk implementasi ELK-*stack* sesuai dengan kebutuhan perlu dibuat tiga berkas *Dockerfile* untuk masing-masing Elasticsearch, Logstash, dan Kibana. Tiap-tiap *Dockerfile* menerima *argument* ELK_VERSION dari *environtment variable* yang kemudian digunakan untuk menentukan versi *base image* resmi dari *docker.elastic.co* yang akan digunakan.

elasticsearch > Dockerfile

```
1  ARG ELK_VERSION
2
3  # https://www.docker.elastic.co/
4  FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
5
6  # Add your elasticsearch plugins setup here
7  # Example: RUN elasticsearch-plugin install analysis-icu
8
```

Gambar 4.1 - Dockerfile Elasticsearch

logstash > Dockerfile

```
1  ARG ELK_VERSION
2
3  # https://www.docker.elastic.co/
4  FROM docker.elastic.co/logstash/logstash:${ELK_VERSION}
5
6  # Add your logstash plugins setup here
7  # Example: RUN logstash-plugin install logstash-filter-json
8
```

Gambar 4.2 - Dockerfile Logstash

kibana > Dockerfile

```
1  ARG ELK_VERSION
2
3  # https://www.docker.elastic.co/
4  FROM docker.elastic.co/kibana/kibana:${ELK_VERSION}
5
6  # Add your kibana plugins setup here
7  # Example: RUN kibana-plugin install <name|url>
8
```

Gambar 4.3 - Dockerfile Kibana

4.1.2 Berkas Docker Compose

Untuk menjalankan ketiga container dari ELK-stack sekaligus dengan baik perlu didefinisikan berkas *docker-compose.yml*. Definisi yang berada di dalam berkas ini sebagai berikut:

a. Daftar *container* yang dijalankan

b. Konfigurasi *build* yang akan digunakan untuk menjalankan tiap *container*

c. *Volume* yang akan digunakan tiap *container*

d. *Port* yang akan diekspose tiap *container*

e. *Environtment variable* yang akan diteruskan ke tiap *container*

f. Jaringan virtual yang akan digunakan tiap *container*

g. Dependensi dari tiap *container* yang dijalankan

docker-compose.yml

```yaml
version: '3.2'

services:
  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELK_VERSION: $ELK_VERSION
    volumes:
      - type: bind
        source: ./elasticsearch/config/elasticsearch.yml
        target: /usr/share/elasticsearch/config/elasticsearch.yml
        read_only: true
      - type: bind
        source: $ELASTIC_DATA
        target: /usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xmx256m -Xms256m"
      ELASTIC_PASSWORD:
      # Use single node discovery in order to disable production mode and avoid bootstrap checks
      # see https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap-checks.html
      discovery.type: single-node
    networks:
      - elk

```

Gambar 4.4 - Docker Compose Bagian 1

docker-compose.yml

```yaml
  logstash:
    build:
      context: logstash/
      args:
        ELK_VERSION: $ELK_VERSION
    volumes:
      - type: bind
        source: ./logstash/config/logstash.yml
        target: /usr/share/logstash/config/logstash.yml
        read_only: true
      - type: bind
        source: ./logstash/config/pipelines.yml
        target: /usr/share/logstash/config/pipelines.yml
        read_only: true
      - type: bind
        source: ./logstash/pipeline
        target: /usr/share/logstash/pipeline
        read_only: true
      - type: bind
        source: ./logstash/other
        target: /usr/share/logstash/other
        read_only: true
      - type: bind
        source: $LOGSTASH_LASTRUN
        target: /usr/share/logstash/lastrun
    ports:
      - "5000:5000/tcp"
      - "5000:5000/udp"
      - "9600:9600"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
      ELASTIC_PASSWORD:
      MYSQL_HOST:
```

Gambar 4.5 - Docker Compose Bagian 2

```
docker-compose.yml
61          MYSQL_HOST:
62          MYSQL_PORT:
63          MYSQL_DATABASE:
64          MYSQL_TIMEZONE:
65          MYSQL_USER:
66          MYSQL_PASSWORD:
67        networks:
68          - elk
69        depends_on:
70          - elasticsearch
71
72      kibana:
73        build:
74          context: kibana/
75          args:
76            ELK_VERSION: $ELK_VERSION
77        volumes:
78          - type: bind
79            source: ./kibana/config/kibana.yml
80            target: /usr/share/kibana/config/kibana.yml
81            read_only: true
82        ports:
83          - "5601:5601"
84        environment:
85          ELASTIC_PASSWORD:
86        networks:
87          - elk
88        depends_on:
89          - elasticsearch
90
91    networks:
92      elk:
93        driver: bridge
```

Gambar 4.6 - Docker Compose Bagian 3

4.1.3 Berkas-berkas ELK-Stack

Konfigurasi khusus Elasticsearch terdapat pada sub-direktori "elasticsearch" direktori kerja, khususnya di dalam direktori "config" yaitu berkas *elasticsearch.yml*. Pada berkas ini diatur nama kluster, konfigurasi keamanan, serta monitoring elasticsearch sendiri.

```
elasticsearch > config > ! elasticsearch.yml
1   ---
2   ## Default Elasticsearch configuration from Elasticsearch base image.
3   ## https://github.com/elastic/elasticsearch/blob/master/distribution/docker/src/docker/
    config/elasticsearch.yml
4   #
5   cluster.name: "docker-cluster"
6   network.host: 0.0.0.0
7
8   ## X-Pack settings
9   ## see https://www.elastic.co/guide/en/elasticsearch/reference/current/setup-xpack.html
10  #
11  xpack.license.self_generated.type: basic
12  xpack.security.enabled: true
13  xpack.monitoring.collection.enabled: true
14
```

Gambar 4.7 - Berkas elasticsearh.yml

Konfigurasi khusus Logstash terdapat pada sub-direktori “logstash” direktori kerja, khususnya di dalam direktori “config” yaitu berkas *logstash.yml*. Pada berkas ini diatur *host* logstash, *host* elasticsearch yang dihubungkan, monitoring logstash sendiri, serta *username* dan *password* dari *host* elasticsearch yang dihubungkan.

logstash > config > ! logstash.yml

```
1  ---
2  ## Default Logstash configuration from Logstash base image.
3  ## https://github.com/elastic/logstash/blob/master/docker/data/logstash/config/logstash-full.
   yml
4  #
5  http.host: "0.0.0.0"
6  xpack.monitoring.elasticsearch.hosts: [ "http://elasticsearch:9200" ]
7
8  ## X-Pack security credentials
9  #
10 xpack.monitoring.enabled: true
11 xpack.monitoring.elasticsearch.username: elastic
12 xpack.monitoring.elasticsearch.password: "${ELASTIC_PASSWORD}"
13
```

Gambar 4.8 - Berkas logstash.yml

Konfigurasi khusus Kibana terdapat pada sub-direktori “kibana” direktori kerja, khususnya di dalam direktori “config” yaitu berkas *kibana.yml*. Pada berkas ini diatur *host* kibana, *host* elasticsearch yang dihubungkan, monitoring kibana sendiri, serta *username* dan *password* dari *host* elasticsearch yang dihubungkan.

kibana > config > ! kibana.yml

```
1  ---
2  ## Default Kibana configuration from Kibana base image.
3  ## https://github.com/elastic/kibana/blob/master/src/dev/build/tasks/os_packages/
   docker_generator/templates/kibana_yml.template.js
4  #
5  server.name: kibana
6  server.host: 0.0.0.0
7  elasticsearch.hosts: [ "http://elasticsearch:9200" ]
8  monitoring.ui.container.elasticsearch.enabled: true
9
10 ## X-Pack security credentials
11 #
12 elasticsearch.username: elastic
13 elasticsearch.password: "${ELASTIC_PASSWORD}"
14
```

Gambar 4.9 - Berkas kibana.yml

4.2 Pra-Processing Data

Pra-*Processing* data dilakukan pada *pipeline* logstash dengan memanfaatkan logstash filter. *Pipeline* logstash didefinisikan di sub-direktori "logstash" direktori kerja, khususnya di dalam direktori "config" yaitu berkas *logstash.yml*. *Pipeline* yang dibutuhkan adalah berjumlah 3 buah, sesuai dengan rancangan prosedur pada Gambar 3.5, Gambar 3.6 dan Gambar 3.7.

logstash > config > ! pipelines.yml

```
1  - pipeline.id: speeda-operator
2    path.config: "${HOME}/pipeline/speeda-operator"
3  - pipeline.id: speeda-stasiun
4    path.config: "${HOME}/pipeline/speeda-stasiun"
5  - pipeline.id: speeda-lokasi
6    path.config: "${HOME}/pipeline/speeda-lokasi"
7
```

Gambar 4.10 - Berkas pipelines.yml

4.2.1 Pra-processing Data Operator

Pra-*processing* Data Operator dilakukan dengan berkas *pipeline* speeda-operator.conf. Secara umum berkas *pipeline* terbagi menjadi 3 bagian, yaitu bagian *input*, *filter*, dan *output*.

logstash > pipeline > speeda-operator > speeda.operator.conf

```
1  input {
2    jdbc {
3      jdbc_driver_library => "${HOME}/other/lib/mysql-connector-java-8.0.21.jar"
4      jdbc_driver_class => "com.mysql.jdbc.Driver"
5      jdbc_connection_string => "jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DATABASE}?
       serverTimezone=${MYSQL_TIMEZONE}"
6      jdbc_user => "${MYSQL_USER}"
7      jdbc_password => "${MYSQL_PASSWORD}"
8      tracking_column => "loc_id"
9      use_column_value => true
10     last_run_metadata_path => "${HOME}/lastrun/operator.txt"
11     statement => "SELECT * from location WHERE loc_id > :sql_last_value"
12     schedule => "* * * * *"
13   }
14 }
15
```

Gambar 4.11 - Pipeline Input Pra-processing Data Operator

Pada potongan kode *pipeline* dalam Gambar 4.11 digunakan jbdc *plugin* sebagai penghubung ke sumber data basisdata Mysql. Detail koneksi basisdata mysql diatur melalui *environtment variable*. Untuk data operator, logstash jbdc *plugin* menggunakan field “loc_id” sebagai kolom pelacak. Penggunaan “loc_id” dimaksudkan karena data operator akan selalu bertambah baru sedangkan data nama serta kode operator tidak akan berubah. Nilai dari kolom pelacak terakhir disimpan di lokasi “${HOME}/lastrun/operator.txt”, nilai ini akan digunakan pada *query* setiap kali dijalankan. Diamana *query* dijadwalkan untuk berjalan setiap menit.

logstash > pipeline > speeda-operator > speeda.operator.conf

```
16  filter {
17    ruby {
18      path => "${HOME}/other/filter/speeda.operator.rb"
19    }
20  }
21
```

Gambar 4.12 - Pipeline Filter Pra-processing Data Operator

Pada potongan kode *pipeline* dalam Gambar 4.12 digunakan ruby filter *plugin* sebagai pemroses data. *Plugin* ini akan menjalankan kode program berbahasa ruby yang berlokasi di “${HOME}/other/filter/speeda.operator.rb” untuk memproses setiap data operator yang diterima.

logstash > other > filter > speeda.operator.rb

```
5   def filter(event)
6   id = event.get("loc_id")
7   code = event.get("loc_code")
8   name = event.get("loc_name")
9   operator = {
10    "id" => id,
11    "code" => code,
12    "name" => name
13  }
14  event.set("operator", operator)
15
16  event.remove("loc_desc")
```

Gambar 4.13 - Filter speeda.operator.rb

Potongan kode pada Gambar 4.13 merupakan kode program yang digunakan untuk *cleansing* dan transformasi data operator.

logstash > pipeline > speeda-operator > speeda.operator.conf

```
22  output {
23    # stdout {
24    #   codec => rubydebug
25    # }
26    elasticsearch {
27      hosts => "elasticsearch:9200"
28      user => "elastic"
29      password => "${ELASTIC_PASSWORD}"
30      index => "operator"
31      template => "${HOME}/other/mapping/speeda.operator.json"
32      template_name => "operator"
33      document_id => "%{[operator][id]}"
34    }
35  }
```

Gambar 4.14 - Pipeline Output Pra-processing Data Operator

Potongan kode *pipeline* dalam Gambar 4.13 mengatur bagaimana data operator yang telah di proses disimpan dalam indeks elasticsearch. Hal yang menjadi perhatian dalam kode ini adalah penentuan "[operator][id]" sebagai *document_id* serta template yang merupakan mapping dari indeks. Berkas "speeda.operator.json" yang merupakan *mapping* dari data operator disesuaikan dengan rancangan pada Tabel 3.7.

logstash > other > mapping > {} speeda.operator.json > ...

```
1   {
2     "template": "operator",
3     "mappings": {
4       "properties": {
5         "operator": {
6           "properties": {
7             "code": {
8               "type": "text",
9               "fields": {
10                "keyword": { "type": "keyword", "ignore_above": 256 }
11              }
12            },
13            "id": { "type": "long" },
14            "name": {
15              "type": "text",
16              "fields": {
17                "keyword": { "type": "keyword", "ignore_above": 256 }
18              }
```

Gambar 4.15 - Mapping speeda.operator.json

4.2.2 Pra-processing Data Stasiun

Pra-*processing* Data Stasiun dilakukan dengan berkas *pipeline* speeda-stasiun.conf. Secara umum berkas *pipeline* terbagi menjadi 3 bagian, yaitu bagian *input*, *filter*, dan *output*.

logstash > pipeline > speeda-stasiun > speeda.stasiun.conf

```
input {
  jdbc {
    jdbc_driver_library => "${HOME}/other/lib/mysql-connector-java-8.0.21.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DATABASE}?
    serverTimezone=${MYSQL_TIMEZONE}"
    jdbc_user => "${MYSQL_USER}"
    jdbc_password => "${MYSQL_PASSWORD}"
    tracking_column => "sta_add_date"
    tracking_column_type => "timestamp"
    use_column_value => true
    last_run_metadata_path => "${HOME}/lastrun/stasiun.txt"
    statement => "SELECT * FROM stations JOIN location ON loc_id = sta_loc WHERE
    sta_add_date > :sql_last_value OR sta_update_date > :sql_last_value ORDER BY
    sta_add_date ASC"
    schedule => "* * * * *"
  }
}
```

Gambar 4.16 - Pipeline Input Pra-processing Data Stasiun

Pada potongan kode *pipeline* dalam Gambar 4.16 digunakan jbdc *plugin* sebagai penghubung ke sumber data basisdata Mysql. Detail koneksi basisdata mysql diatur melalui *environtment variable*. Untuk data stasiun, logstash jbdc *plugin* menggunakan field “sta_add_date” sebagai kolom pelacak. Penggunaan “sta_add_date” dimaksudkan karena data stasiun akan selalu bertambah atau berubah seiring berjalannya waktu, oleh sabab itu perlu dilakukan pembaharuan dengan mengambil data yang tanggal ditambahkan atau tanggal dirubahnya lebih baru dari tanggal ditambahkan yang terakhir. Nilai dari kolom pelacak terakhir berupa *timestamp* disimpan di lokasi “${HOME}/lastrun/stasiun.txt”, nilai ini akan digunakan pada *query* setiap kali dijalankan. Diamana *query* dijadwalkan untuk berjalan setiap menit.

logstash > pipeline > speeda-stasiun > speeda.stasiun.conf

```
filter {
  mutate {
    convert => [ "sta_lat", "float" ]
    convert => [ "sta_long", "float" ]
  }
  ruby {
    path => "${HOME}/other/filter/speeda.stasiun.rb"
  }
}
```

Gambar 4.17 - Pipeline Filter Pra-processing Data Stasiun

Pada potongan kode *pipeline* dalam Gambar 4.17 digunakan mutate filter *plugin* untuk mengkonversi field “sta_lat” dan “sta_long” menjadi bertipe data *float*. Digunakan juga ruby filter *plugin* sebagai pemroses data. *Plugin* ini akan menjalankan kode program berbahasa ruby yang berlokasi di “${HOME}/other/filter/speeda.stasiun.rb” untuk memproses setiap data stasiun yang diterima.

logstash > other > filter > speeda.stasiun.rb

```
def filter(event)
  id = event.get("sta_id")
  name = event.get("sta_name")
  status = event.get("sta_status")
  lat = (event.get("sta_lat")).round(5)
  lon = (event.get("sta_long")).round(5)
  location = {
    "lat" => lat,
    "lon" => lon
  }
  operator_id = event.get("loc_id")
  operator_code = event.get("loc_code")
  operator_name = event.get("loc_name")
  operator ={
    "id" => operator_id,
    "code" => operator_code,
    "name" => operator_name
  }
  warehouse = event.get("is_warehouse")

  event.set("id", id)
  event.set("name", name)
  event.set("status", status)
  event.set("location", location)
  event.set("operator", operator)
  event.set("warehouse", warehouse)

  event.remove("sta_id")
```

Gambar 4.18 - Filter speeda.stasiun.rb

Potongan kode pada Gambar 4.18 merupakan kode program yang digunakan untuk *cleansing* dan transformasi data stasiun. Dalam potongan kode tersebut terdapat proses penting *cleansing* untuk menyamakan data "sta_lat" dan "sta_long" dengan tingkat presisi 5 angka di belakang koma.

Gambar 4.19 - Pipeline Output Pra-processing Data Stasiun

Potongan kode *pipeline* dalam Gambar 4.19 mengatur bagaimana data stasiun yang telah di proses disimpan dalam indeks elasticsearch. Dalam potongan kode ini terdapat penentuan "id" sebagai *document_id* serta template yang merupakan mapping dari indeks. Berkas "speeda.stasiun.json" yang merupakan *mapping* dari data stasiun disesuaikan dengan rancangan pada Tabel 3.6.

Gambar 4.20 - Mapping speeda.stasiun.json

4.2.3 Pra-processing Data Lokasi

Pra-*processing* Data Lokasi dilakukan dengan berkas *pipeline* speeda-lokasi.conf. Secara umum berkas *pipeline* terbagi menjadi 3 bagian, yaitu bagian *input*, *filter*, dan *output*.

logstash > pipeline > speeda-lokasi > speeda.lokasi.conf

```
input {
  jdbc {
    jdbc_driver_library => "${HOME}/other/lib/mysql-connector-java-8.0.21.jar"
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DATABASE}?
    serverTimezone=${MYSQL_TIMEZONE}"
    jdbc_user => "${MYSQL_USER}"
    jdbc_password => "${MYSQL_PASSWORD}"
    tracking_column => "pos_id"
    use_column_value => true
    last_run_metadata_path => "${HOME}/lastrun/lokasi.txt"
    statement => "SELECT * FROM bike_positions JOIN bikes ON bike_id = pos_bike JOIN
    location ON loc_id = bike_loc WHERE pos_id > :sql_last_value LIMIT 20000"
    schedule => "* * * * *"
  }
}
```

Gambar 4.21 - Pipeline Input Pra-processing Data

Pada potongan kode *pipeline* dalam Gambar 4.21 digunakan jbdc *plugin* sebagai penghubung ke sumber data basisdata Mysql. Detail koneksi basisdata mysql diatur melalui *environtment variable*. Untuk data lokasi, logstash jbdc *plugin* menggunakan field “pos_id” sebagai kolom pelacak. Penggunaan “pos_id” dimaksudkan karena data lokasi akan selalu bertambah baru dan tidak akan pernah berubah. Nilai dari kolom pelacak terakhir disimpan di lokasi “${HOME}/lastrun/lokasi.txt”, nilai ini akan digunakan pada *query* setiap kali dijalankan. Diamana *query* dijadwalkan untuk berjalan setiap menit.

logstash > pipeline > speeda-lokasi > speeda.lokasi.conf

```
filter {
  mutate {
    convert => [ "pos_lat", "float" ]
    convert => [ "pos_long", "float" ]
    convert => [ "pos_utc_time", "integer" ]
    convert => [ "pos_src", "integer" ]
  }
  ruby {
    path => "${HOME}/other/filter/speeda.lokasi.rb"
  }
  mutate {
    convert => [ "dayofweek", "integer" ]
    convert => [ "hourofday", "integer" ]
```

Gambar 4.22 - Pipeline Filter Pra-processing Data Lokasi

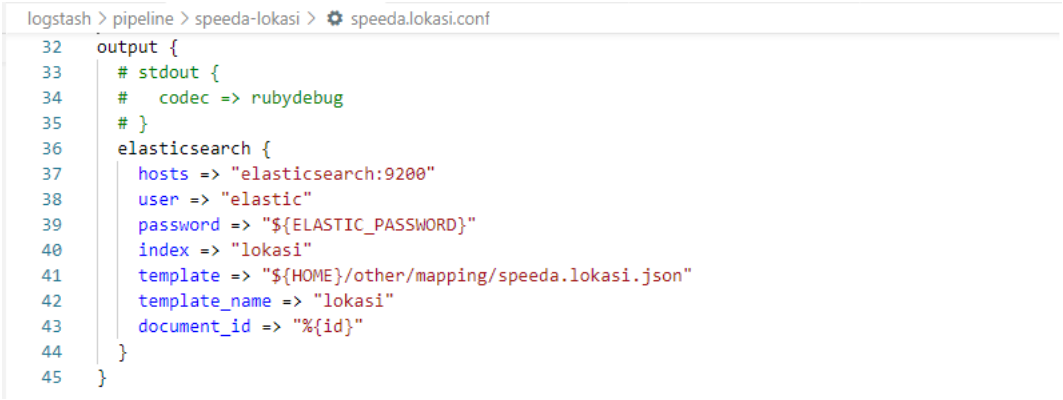
Pada potongan kode *pipeline* dalam Gambar 4.22 digunakan mutate filter *plugin* untuk mengkonversi field "sta_lat" dan "sta_long" menjadi bertipe data *float* serta mengkonversi field "pos_utc_time" dan "pos_src" menjadi bertipe data *integer*. Digunakan juga ruby filter *plugin* sebagai pemroses data. *Plugin* ini akan menjalankan kode program berbahasa ruby yang berlokasi di "${HOME}/other/filter/speeda.lokasi.rb" untuk memproses setiap data stasiun yang diterima. Mutate filter *plugin* digunakan kembali untuk mengkonversi field "dayofweek" dan "hourofday" yang dihasilkan oleh ruby filter menjadi bertipe data *integer*.

logstash > other > filter > speeda.lokasi.rb

```
def filter(event)
  id = event.get("pos_id")
  lat = (event.get("pos_lat")).round(5)
  lon = (event.get("pos_long")).round(5)
  location = {
    "lat" => lat,
    "lon" => lon
  }
  stamp = event.get("pos_utc_time")
  unless stamp
    stamp = event.get("pos_insert")
  end
  t = Time.at(stamp.to_i)
  d = DateTime.parse(t.to_s)
  l = d.new_offset("+07:00")
  datetime = l.strftime("%FT%T.%L%:z")
  dayofweek = l.strftime("%u")
  hourofday = l.strftime("%H")
```

Gambar 4.23 - Filter speeda.lokasi.rb

Potongan kode pada Gambar 4.23 merupakan kode program yang digunakan untuk *cleansing* dan transformasi data lokasi. Dalam potongan kode tersebut terdapat proses penting *cleansing* untuk menyamakan data “pos_lat” dan “pos_long” dengan tingkat presisi 5 angka di belakang koma. Terdapat juga kondisi untuk memilih nilai dari “*timestamp*” data, serta mengkonversinya sesuai dengan zona waktu GMT+7 kemudian di transformasi dengan format tanggal standar, format hari, dan format jam.



Gambar 4.24 - Pipeline Output Pra-processing Data Lokasi

Potongan kode *pipeline* dalam Gambar 4.24 mengatur bagaimana data lokasi yang telah di proses disimpan dalam indeks elasticsearch. Dalam potongan kode ini terdapat penentuan “id” sebagai *document_id* serta template yang merupakan mapping dari indeks. Berkas “speeda.lokasi.json” yang merupakan *mapping* dari data lokasi disesuaikan dengan rancangan pada Tabel 3.5.

logstash > other > mapping > {} speeda.lokasi.json > ...

```
{
  "template": "lokasi",
  "mappings": {
    "properties": {
      "id": { "type": "long" },
      "bike": {
        "properties": {···
        }
      },
      "operator": {
        "properties": {···
        }
      },
      "location": {
        "type": "geo_point"
      },
      "datetime": {
        "type": "date"
      },
      "dayofweek": {
        "type": "long"
      },
      "hourofday": {
        "type": "long"
      },
      "src": { "type": "long" },
      "trx": { "type": "long" }
    }
  }
}
```

Gambar 4.25 - Mapping speeda.lokasi.json

4.3 Penyimpanan Data

Data disimpan pada elasticsearch oleh logstash dengan menjalankan *pipeline* yang telah dikonfigurasi dan didefinisikan. Untuk dapat menjalankan elasticsearch, logstash dan kibana (ELK-*stack*) pada docker, diperlukan definisi *environtemnt variabel* beserta perintah-perintah docker.

Environtment variable didefinisikan dengan membuat berkas ".env" pada direktori kerja. Kemudian menyesuaikan nilai dari variabel-variabel yang digunakan pada ELK-*stack* secara menyeluruh.

```
.env
1   ELK_VERSION=7.7.1
2   ELASTIC_DATA=/home/hidaru/docker/elasticsearch/data
3   ELASTIC_PASSWORD=r@h4asia
4   LOGSTASH_LASTRUN=/home/hidaru/docker/elasticsearch/lastrun
5   MYSQL_HOST=172.17.0.1
6   MYSQL_PORT=3306
7   MYSQL_DATABASE=akakom_speeda
8   MYSQL_TIMEZONE=Asia/Jakarta
9   MYSQL_USER=wahid
10  MYSQL_PASSWORD=akakom
11
```

Gambar 4.26 - Environtment Variable

ELK-*stack* pada docker dijalankan dengan perintah “docker-compose up -d”. Setelah berjalan, status *container-container* ELK-*stack* dapat dicek menggunakan perintah “docker ps”.

```
hidaru at elucidator in /mnt/hid/dev/akakom/skripsi/docker-elk on skripsi!
 ± docker-compose up -d
Creating network "docker-elk_elk" with driver "bridge"
Creating docker-elk_elasticsearch_1 ... done
Creating docker-elk_kibana_1        ... done
Creating docker-elk_logstash_1      ... done
```

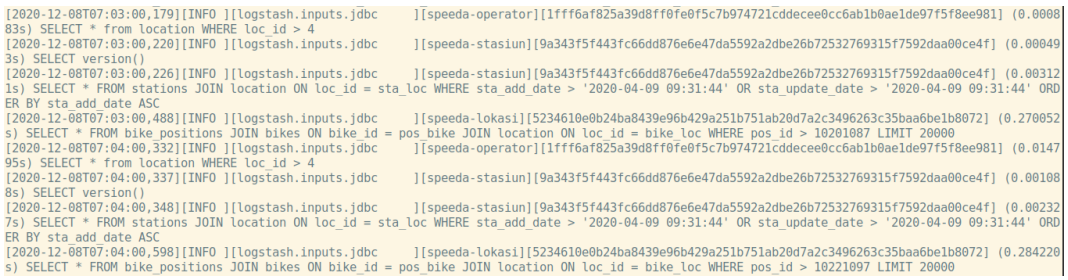
Gambar 4.27 - Menjalankan Docker

```
hidaru at elucidator in /mnt/hid/dev/akakom/skripsi/docker-elk on skripsi!
 ± docker ps
CONTAINER ID        IMAGE                     COMMAND                  CREATED
 STATUS              PORTS
             NAMES
5166774fbf21        docker-elk_logstash       "/usr/local/bin/dock…"   2 minutes ago
 Up 2 minutes        0.0.0.0:5000->5000/tcp, 0.0.0.0:9600->9600/tcp, 0.0.0.0:5000->5000/udp
, 5044/tcp   docker-elk_logstash_1
5ad08a31f740        docker-elk_kibana         "/usr/local/bin/dumb…"   2 minutes ago
 Up 2 minutes        0.0.0.0:5601->5601/tcp
             docker-elk_kibana_1
49efbf1df764        docker-elk_elasticsearch  "/tini -- /usr/local…"   2 minutes ago
 Up 2 minutes        0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp
             docker-elk_elasticsearch_1
```

Gambar 4.28 - Mengecek Docker

Untuk memastikan *pipeline* logstash berjalan dengan baik, sehingga proses *pra-processing* dapat terlaksana, dapat dilakukan pengecekan pada log logstash menggunakan perintah “docker logs docker-elk_logstash_1”. Pada Gambar 4.29 *pipeline* operator, stasiun, dan lokasi telah berjalan dengan baik. Nilai *tracking*

column juga telah sesuai dengan nilai kolom pelacak terakhir dari masing-masing *pipeline* yang tersimpan. Hal ini dapat diperhatikan dari nilai kolom pelacak *pipeline* lokasi yang bertambah dari 10.201.087 menjadi 10.221.097 sesuai dengan nilai limit pada *query* kurang/lebih 20000 data.



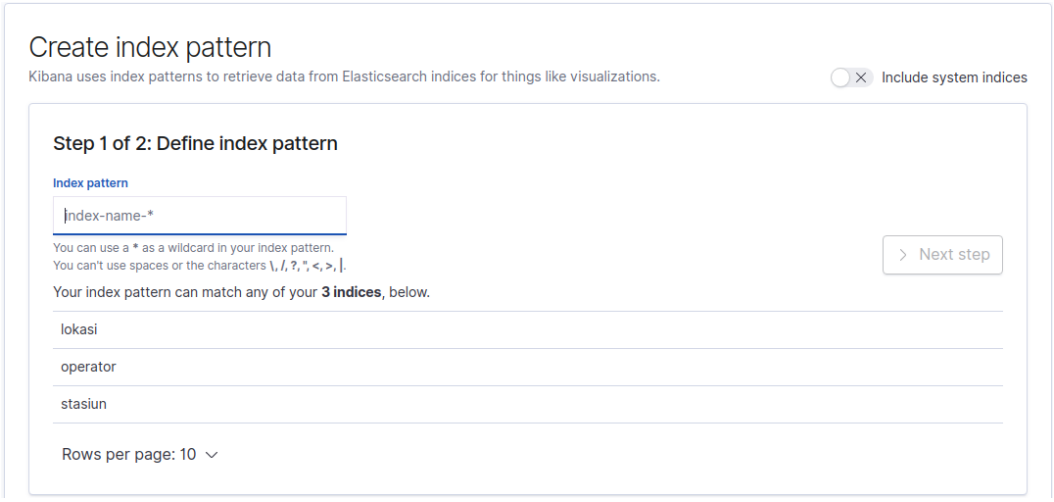
Gambar 4.29 - Mengecek pipeline Logstash

4.4 Visualisasi Data

Visualisasi data dilakukan menggunakan Kibana. Kibana dapat diakses menggunakan browser dengan url “http://localhost:5601”. Pembuatan visualisasi dapat dilakukan secara GUI. Hasil visualisasi yang telah dirancang dapat diekspor dan kemudian dapat diimpor untuk diimplementasikan di *environtment* ELK-*stack* lain.

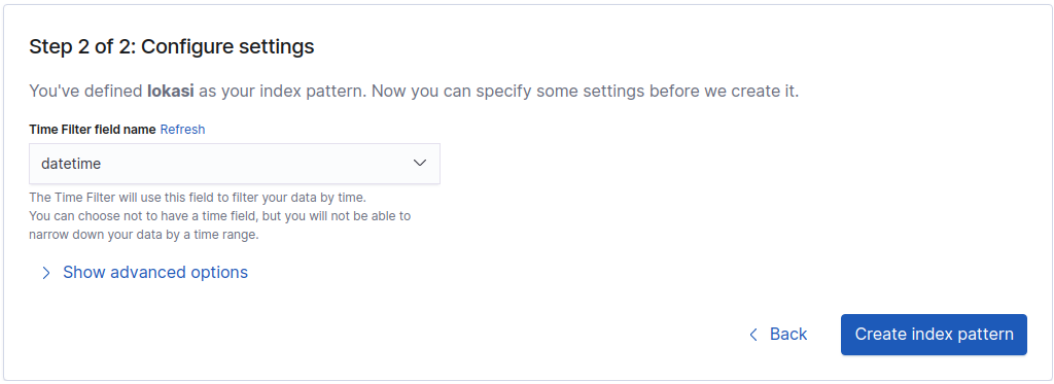
4.4.1 Index Pattern

Untuk melakukan visualisasi data, perlu disiapkan terlebih dahulu *Index Pattern* untuk masing-masing indeks elasticsearch. *Index Pattern* disiapkan melalui menu *Management* > *Index Pattern*. Nama yang dimasukkan tepat sesuai dengan nama indeks tanpa wildcard, yaitu “operator”, “stasiun”, dan “lokasi”. Hal ini dilakukan karena *index pattern* yang dibuat memang bukan merupakan gabungan dari beberapa indeks.

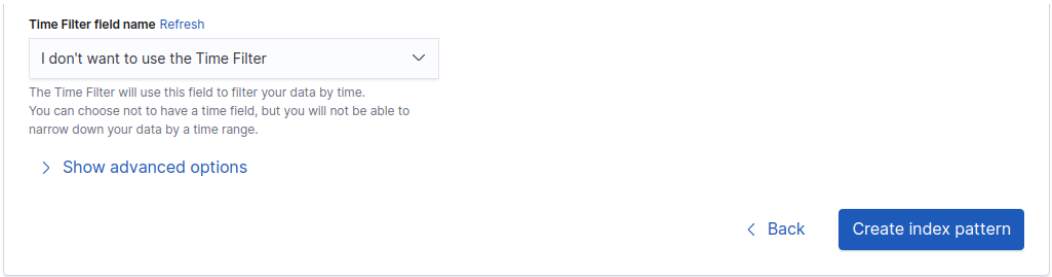


Gambar 4.30 - Membuat Index Pattern

Untuk *Index Pattern* lokasi, *field* yang digunakan untuk *time filter* adalah *field* "datetime". Sedangkan untuk operator dan stasiun tidak menggunakan *field* apapun untuk *time filter*.



Gambar 4.31 - Index Pattern Time Filter "lokasi"



Gambar 4.32 - Index Pattern Time Filter "operator"/"stasiun"

4.4.2 Heatmap Intensitas

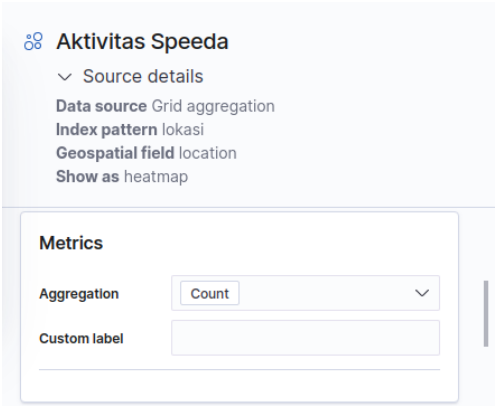
Heatmap Intensitas dibuat menggunakan menu “Map”, pada map yang dibuat ditambahkan 4 buah *layer*. Satu *layer* yang merupakan intensitas aktivitas sepeda, sedangkan 3 lainnya merupakan *layer* untuk titik-titik stasiun. Tiga *layer* tersebut dibutuhkan untuk membedakan stasiun gudang dan stasiun biasa yang tutup serta yang aktif.

Layer intensitas sepeda merupakan sebuah *grid aggregation* berupa *count* (cacah) dari *field* geospasial “location” yang ada di *index pattern* lokasi. Agregasi ini kemudian ditampilkan sebagai *heatmap*.

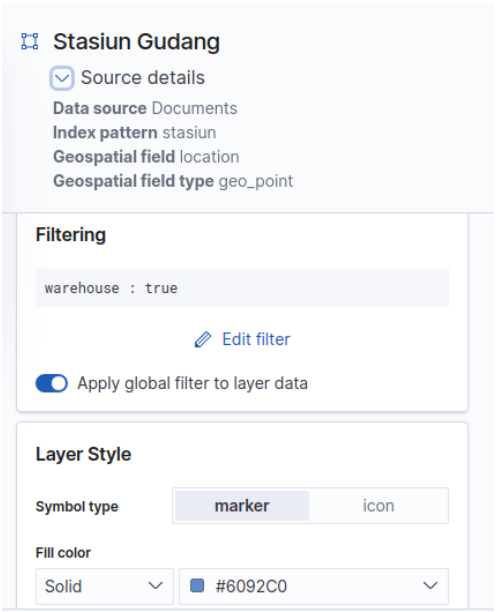
Layer stasiun gudang merupakan titik dengan kode warna #6092C0 (biru) dari *field* geospasial “location” yang ada di *index pattern* stasiun, dengan kondisi *filed* “warehouse” pada data bernilai sama dengan “true”.

Layer stasiun tutup merupakan titik dengan kode warna #E7664C (merah) dari *field* geospasial “location” yang ada di *index pattern* stasiun, dengan kondisi *filed* “warehouse” pada data bernilai sama dengan “false” serta *field* “status.keyword” bernilai sama dengan “CLOSE”.

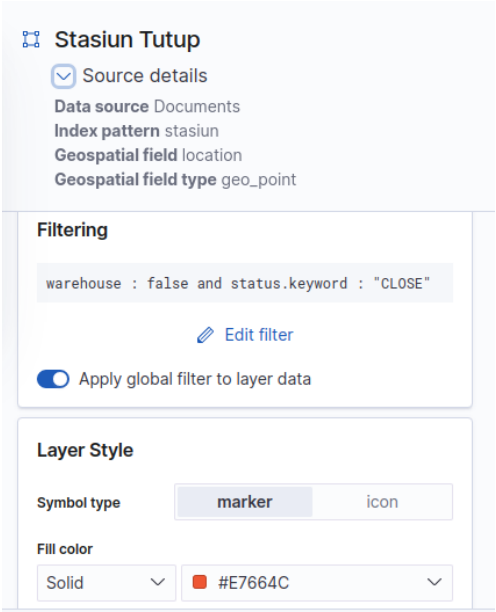
Layer stasiun merupakan titik dengan kode warna #54B399 (hijau) dari *field* geospasial “location” yang ada di *index pattern* stasiun, dengan kondisi *filed* “warehouse” pada data bernilai sama dengan “false” serta *field* “status.keyword” bernilai sama dengan “OPEN”.



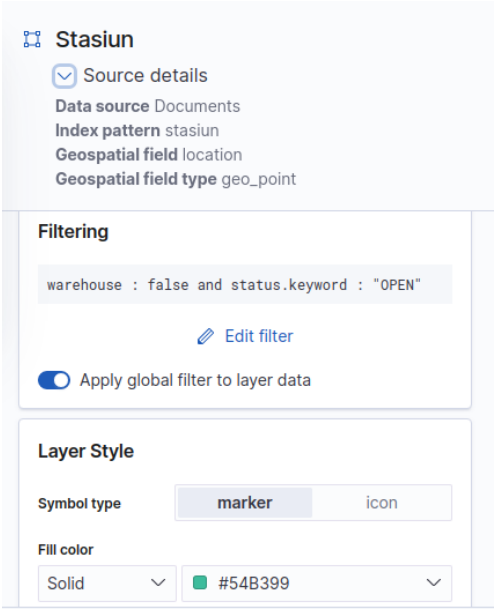
Gambar 4.33 - Layer Intensitas Sepeda



Gambar 4.34 - Layer Stasiun Gudang



Gambar 4.35 - Layer Stasiun Tutup

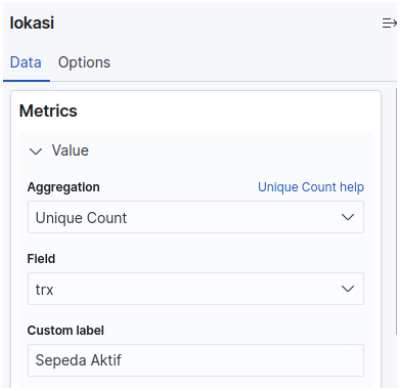


Gambar 4.36 - Layer Stasiun Aktif

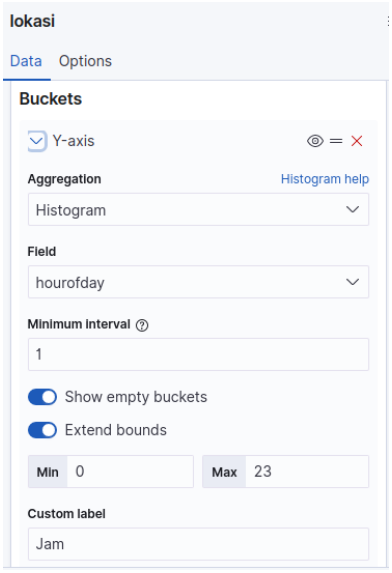
4.4.3 Heatmap Hari/Jam

Heatmap Hari/Jam dibuat menggunakan menu “Visualization”, visualisasi yang dibuat bertipe “Heatmap”. Untuk nilai ukur (*metric*) digunakan agregat cacah

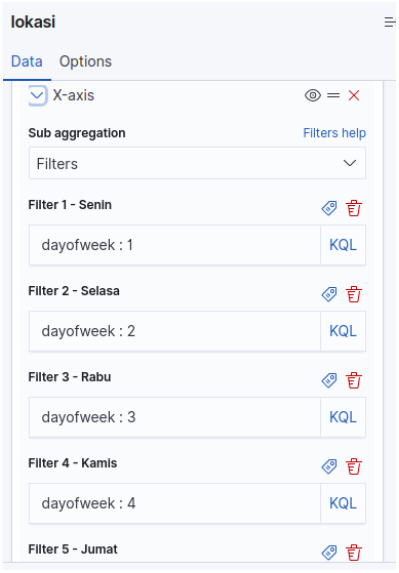
unik dari *field* “trx” pada *index pattern* lokasi. Nilai sumbu y merupakan jam, didefinisikan sebagai agregasi histogram dari *field* “hourofday” pada *index pattern* lokasi yang memiliki rentang 0 sampai 23 dengan interval 1 yang dapat menampilkan nilai kosong. Nilai sumbu x merupakan nama hari, didefinisikan sebagai 7 buah filter dari *field* “dayofweek” pada *index pattern* lokasi yang memiliki nilai mewakili hari tertentu (Nilai 1 mewakili hari senin).



Gambar 4.37 – Metric Heatmap Hari/Jam



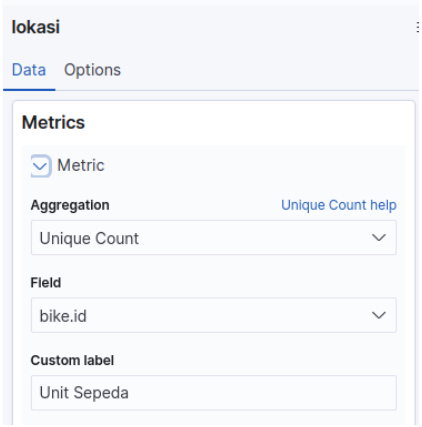
Gambar 4.38 - Sumbu Y Heatmap Hari/Jam



Gambar 4.39 - Sumbu X Heatmap Hari/Jam

4.4.4 Penghitung Unit Sepeda

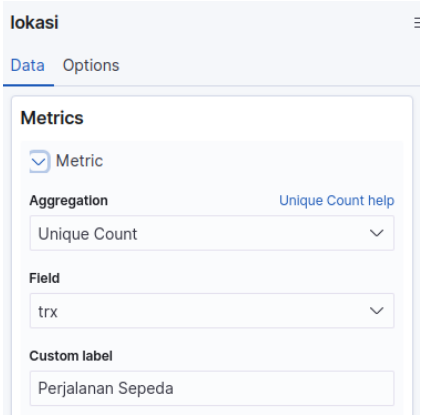
Penghitung Unit Sepeda dibuat menggunakan menu “Visualization”, visualisasi yang dibuat bertipe “Metric”. Untuk nilai ukur (*metric*) digunakan agregat cacah unik dari *field* “bike.id” pada *index pattern* lokasi.



Gambar 4.40 - Metric Unit Sepeda

4.4.5 Penghitung Perjalanan Peminjaman Sepeda

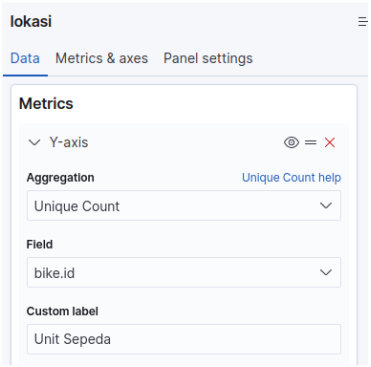
Penghitung Perjalanan Peminjaman Sepeda dibuat menggunakan menu “Visualization”, visualisasi yang dibuat bertipe “Metric”. Untuk nilai ukur (*metric*) digunakan agregat cacah unik dari *field* “trx” pada *index pattern* lokasi.



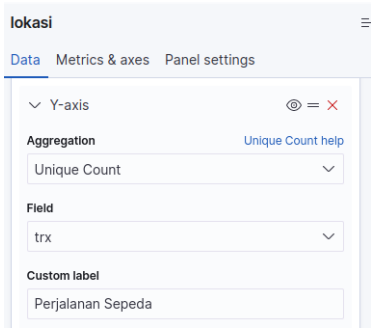
Gambar 4.41 - Metric Perjalanan Peminjaman Sepeda

4.4.6 Grafik Unit Sepeda dan Perjalanan Peminjaman Sepeda

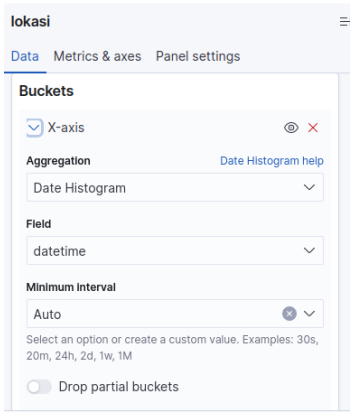
Grafik Unit Sepeda dan Perjalanan Peminjaman Sepeda dibuat menggunakan menu "Visualization", visualisasi yang dibuat bertipe "Area". Terdapat dua sumbu y grafik area, yaitu sumbu y untuk unit sepeda dan sumbu y untuk perjalanan peminjaman sepeda. Untuk sumbu y unit sepeda nilai ukur (*metric*) yang digunakan adalah agregat cacah unik dari *field* "bike.id" pada *index pattern* lokasi. Sedangkan untuk sumbu y perjalanan peminjaman sepeda nilai ukur (*metric*) yang digunakan adalah agregat cacah unik dari *field* "trx" pada *index pattern* lokasi. Sebagai sumbu x digunakan agregat histogram tanggal dari *field* "datetime" dengan interval yang otomatis.



Gambar 4.42 - Sumbu Y Unit Sepeda



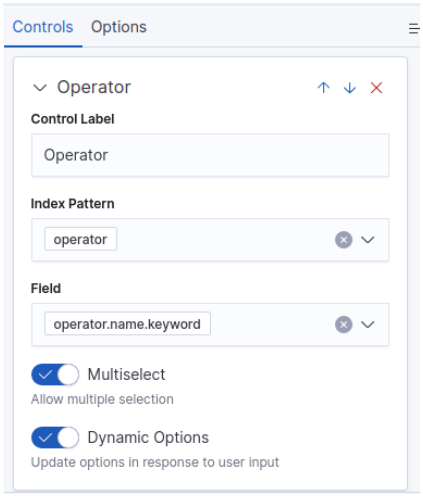
Gambar 4.43 - Sumbu Y Perjalanan Peminjaman Sepeda



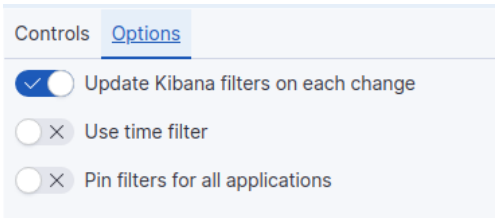
Gambar 4.44 - Sumbu X Histogram Tanggal

4.4.7 Dropdown Filter Data Operator

Dropdown Filter Data Operator dibuat menggunakan menu "Visualization", visualisasi yang dibuat bertipe "Controls". Untuk data *dropdown* digunakan *field* "operator.name.keyword" dari *index pattern* operator. *Dropdown* ini memungkinkan untuk dilakukan pilihan jamak dan bersifat dinamis. Setiap perubahan nilai dari *dropdown* juga diatur untuk dapat merubah *filter* kibana dalam menampilkan data.



Gambar 4.45 – Konfigurasi Dropdown Filter Data Operator

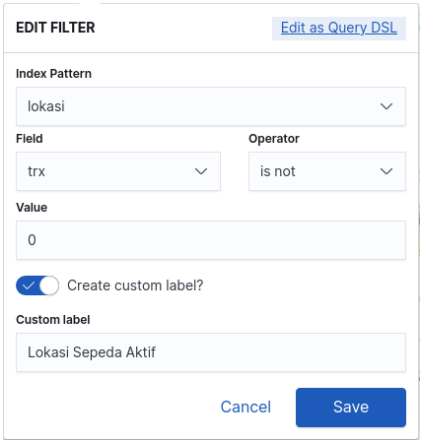


Gambar 4.46 - Konfigurasi Dropdown Filter Data Operator (2)

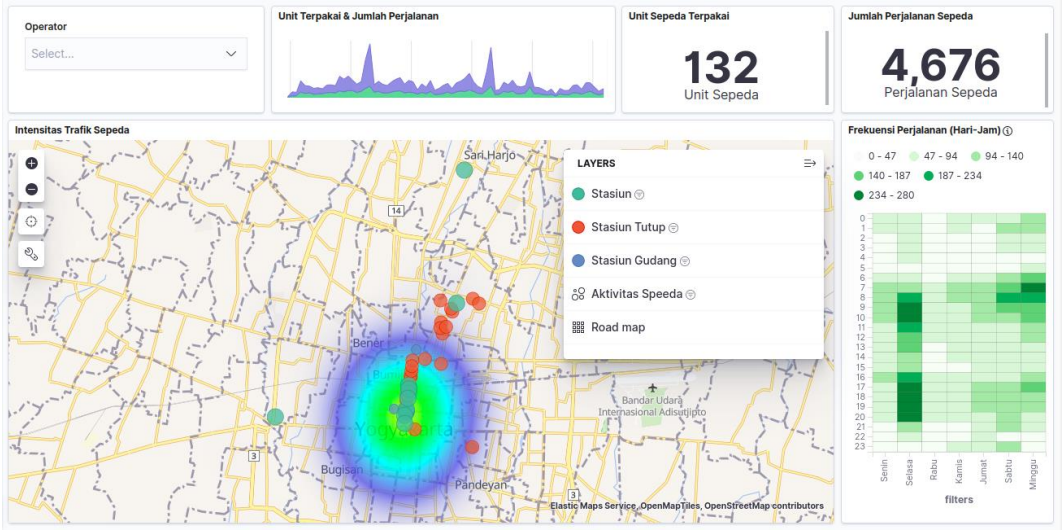
4.4.8 Dasbor Terintegrasi

Dasbor Terintegrasi dibuat menggunakan menu "Dashboard", yang berisi kumpulan dari seluruh peta dan visualisasi yang dibuat dengan data yang

terintegrasi. Pada dasbor dilakukan *filter* dari *index pattern* lokasi dengan kondisi *field* “trx” pada data tidak memiliki nilai “0”. Tampilan antarmuka dan tata letak dasbor dibuat berdasarkan *mockup* pada Gambar 3.10 yang diimplementasikan menjadi sesuai dengan tangkapan layar pada Gambar 4.48. Secara baku, kibana melengkapi dasbor dengan kontrol filter dengan *query* dan filter rentang tanggal.



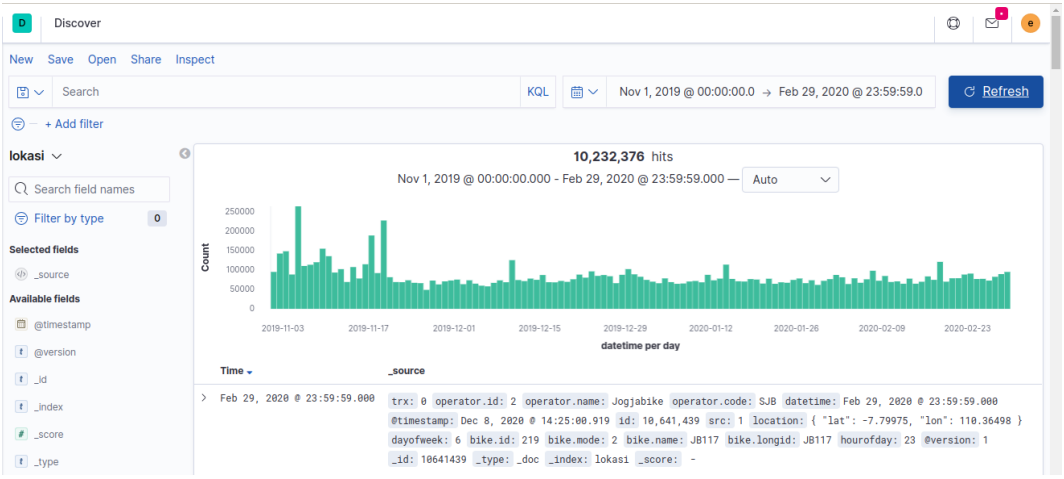
Gambar 4.47 - Filter Sepeda Aktif Dasbor Terintegrasi



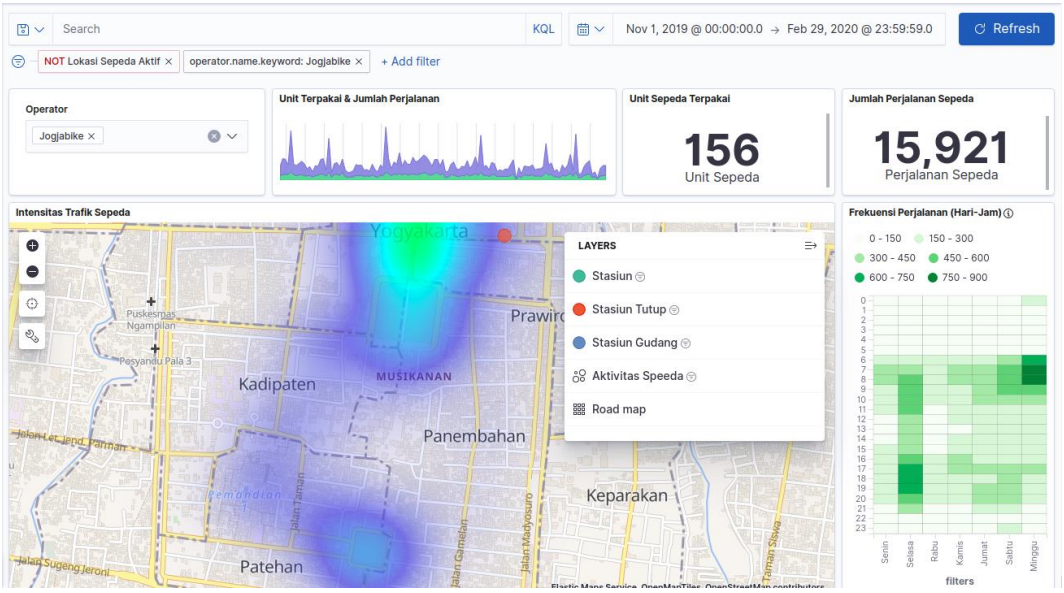
Gambar 4.48 - Tampilan Antarmuka dan Tata Letak Dasbor Terintegrasi

4.5 Membaca Dasbor

Dasbor yang dibuat digunakan untuk memvisualisasikan data dengan rentang tangal 1 November 2019 sampai dengan 29 Februari 2020. Pada rentang tanggal tersebut diketahui terdapat 10.232.376 data lokasi tersimpan pada indeks lokasi elasticsearch.



Gambar 4.49 - Data Pada Indeks Lokasi Elasticsearch



Gambar 4.50 - Dasbor 1 November 2019 s/d 29 Februari 2020

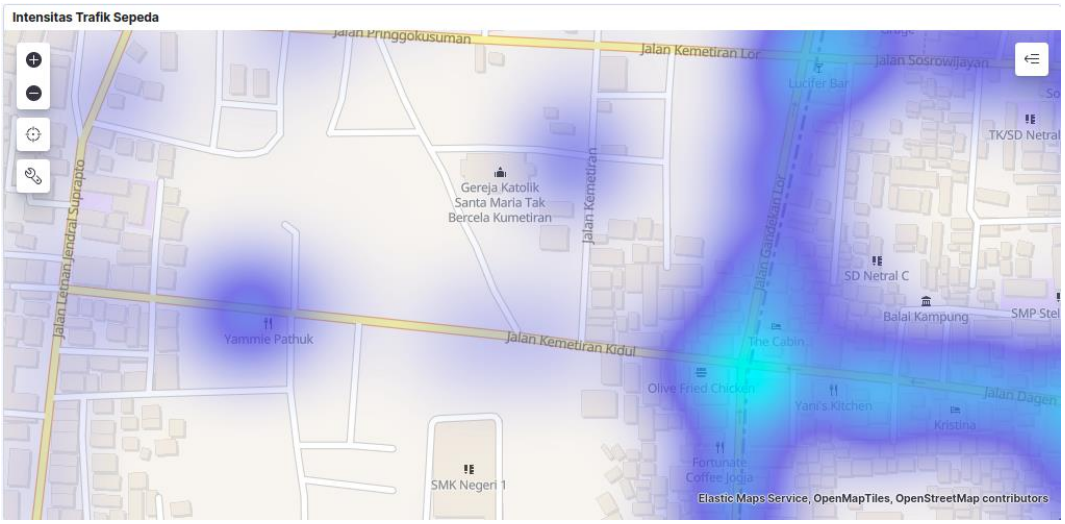
Dapat diketahui dari Gambar 4.50 bahwa dalam rentang tanggal 1 November 2019 sampai dengan 29 Februari 2020 operator Jogjabike memiliki jumlah perjalanan peminjaman sepeda yang fluktuatif. Pada rentang tanggal tersebut sebanyak 156 unit sepeda berbeda telah digunakan dalam 15.921 perjalanan peminjaman sepeda pelanggan. Dari perjalanan peminjaman tersebut lebih banyak terjadi pada hari selasa pagi serta sore, sabtu pagi, dan minggu pagi. Pada *Heatmap* Intensitas yang tertampil, area alun-alun utara dan alun-alun selatan banyak dilalui/dikunjunggi pelanggan meski tidak terdapat stasiun sepeda ataupun dekat dengan stasiun sepeda.

Hasil visualisasi Gambar 4.50 dapat memberikan *insight* untuk mendukung operasional dan pengembangan bisnis operator Jogjabike sebagai berikut:

1. Pada hari rabu dapat dipertembangkan untuk dikenakan tarif khusus yang lebih rendah dengan tujuan menarik animo pelanggan karena trafik penggunaan yang masih rendah.
2. Pada hari sabtu-minggu siang dan/atau sore dapat dipertimbangkan untuk diadakan promo tertentu dengan tujuan mempertahankan trafik penggunaan yang relatif tinggi pada pagi hari.
3. Dapat dipertimbangkan untuk mendirikan stasiun baru di sekitar alun-alun utara, alun-alun sealatan, dan taman sari, sebagai tempat yang banyak dilalui atau dikunjungi pelanggan.

Untuk menambah *insight* kepada operator Jogjabike, *heatmap* intensitas trafik dapat di-*eksplore* lebih jauh untuk mengetahui lokasi lainnya. Selain

menemukan area yang banyak dilalui ataupun dikunjungi, dapat juga menemukan tempat-tempat yang bisa diajak kerjasama sebagai partner atau sponsor.



Gambar 4.51 - Eksplorasi Peta Intensitas Di Kemetiran

Hasil eksplorasi pada Gambar 4.51 dapat memberikan *insight* tambahan untuk operator Jogjabike. Dapat dipertimbangkan untuk mengadakan partnership/sponsorship dengan Lucifer Bar, Yamie Pathuk, Olive Fried Chicken, Fortunate Coffee Jogja, dan Hotel The Cabin sebagai tempat yang memiliki intensitas trafik yang tinggi.