# PDT protokol 5

Jakub Povinec

https://github.com/kuko6/tweets-elastic

---

(tento protokol som písal ako markdown a teda export do pdf nie je ideálny. Odporúčam radšej originál, ktorý je v `docs/protokol.md` a taktiež aj na githube)

## Úloha 1

Elasticsearch som spúšťal pomocou dockeru, resp. `docker compose`, kde konfigurácia jednotlivých inštancií/kontajnerov elasticu je obsiahnutá v súbore `docker-compose.yml`. Inštancie elasticu sa dajú spustiť pomocou príkazu `docker compose up` a vypnúť pomocou `docker compose down` alebo postupne, napr. pomocou `docker stop [názov kontajnera]`

Stav jednotlivých inštancií sa dá overiť pomocou `GET http://localhost:9200/_cat/nodes?v`

```
ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role   master name
172.18.0.3           56         100  14    1.28    0.62     0.23 cdfhilmrstw -      es02
172.18.0.2           25         100  14    1.28    0.62     0.23 cdfhilmrstw *      es01
172.18.0.4           21          99  14    1.28    0.62     0.23 cdfhilmrstw -      es03
```

*Fig. 1 Stav jednotlivých elastic inštancií*

## Úloha 2

Optimálny počet shardov v elasticu záleží na počte nodov, veľkosti indexovaných dát a taktiež dostupných zdrojov (CPU a pamäte). Shardy umožňujú paralelizáciu dopytov s tým, že rozdelia indexované dáta medzi jednotlivé nody. Najideálnejšie je teda, aby bol počet shardov deliteľný počtom nodov, kedy by sa shardy vedeli rovnomerne rozdeliť medzi jednotlivé nody. Podľa tohto článku (How many shards should I have in my Elasticsearch cluster?), by mal mať shard velkosť okolo 20 až 40GB. V našom prípade majú dáta zhruba 70-80GB a teda ideálny počet shardov je buď 3 alebo 6. Aj keď je väčší počet shardov efektívnejší z hľadiska rýchlosti dopytov, každý shard si musí uchovávať ešte dodatočné informácie o uložených dátach, kde pri menšom počte shardov môže byť tento "overhead" výrazne menší. Keďže je pri tomto zadaní môj najväčší problém hlavne nedostatok voľného miesta, zvolil som radšej **menší počet shardov 3**.

Repliky zas predstavujú kópie jednotlivých shardov. Repliky sú vždy na inom node ako primárne shardy a slúžia na nahradenie daného shardu v prípade, ak node na ktorom sa nachádza vypadne. Ideálne je mať vždy aspoň jednu repliku pre každý shard.

Počet shardov a nodov sa zadáva pri vytváraní indexu pomocou `PUT http://localhost:9200/tweets`

```
{
    "settings": {
        "index": {
            "number_of_shards": 3,
            "number_of_replicas": 1
        },
}
```

*Fig. 2 Špecifikovanie počtu shardov a replík pri vytváraní indexu*

V našom prípade bolo finálne rozdelenie shardov a replík nasledovné:

```
tweets 0 p STARTED 0 225b 172.18.0.2 es01
tweets 0 r STARTED 0 225b 172.18.0.4 es03
tweets 1 r STARTED 0 225b 172.18.0.3 es02
tweets 1 p STARTED 0 225b 172.18.0.4 es03
tweets 2 r STARTED 0 225b 172.18.0.2 es01
tweets 2 p STARTED 0 225b 172.18.0.3 es02
```

*Fig. 3 Rozdelenie shardov a replík*

## Úloha 3

Táto úloha je rozdelená na viac častí: vytvorenie mappingu a denormalizácia dát z postgresql databázy.

### Mapping

Pri vytváraní mappingu som sa snažil zachovať podobnú štruktúru dát a tabuliek aká je v postgres databáze s tým, že som vynechal stĺpce s id z tabuliek: `context_annotations`, `conversation_hashtags` a `hashtags`, `annotations` a `links` pretože sa nenachádzajú v twitter dokumentácií pre objekt tweet a teda mi prišli zbytočné. Celé mapovanie sa nachádza v `src/config/mapping.json`.

```
{
    "id": { "type": "long" },
    "content": { "type": "text", "analyzer": "englando" },
    "possibly_sensitive": { "type": "boolean" },
    "language": { "type": "keyword" },
    "source": { "type": "keyword" },
    "retweet_count": { "type": "integer" },
    "reply_count": { "type": "integer" },
    "like_count": { "type": "integer" },
    "quote_count": { "type": "integer" },
    "created_at": { "type": "date", "format": "yyyy-MM-dd'T'HH:mm:ssZZZZZ" }
}
```

*Fig. 4 Mapping pre dáta z tabuľky conversations*

V mapovaní dát pre tabuľku `conversations` som pre pole `content` špecifikoval vlastný analyzér (definovaný v 4. úlohe). Polia `language` a `source` som zaindexoval ako `keyword`, keďže sa zvyčajne jedná len o samotné slová alebo frázy, ktoré nemusia byť analyzované. Taktiež som definoval formát dátumu pre pole `created_at`, ktoré zodpovedá formátu v akom sú dáta v pôvodnej databáze.

```
{
    "author": {
        "properties": {
            "id": { "type": "long" },
            "name": {
                "type": "text",
                "fields": {
                    "ngram": { "type": "text", "analyzer": "custom_ngram" },
                    "shingles": { "type": "text", "analyzer": "custom_shingles" }
                }
            },
            "username": {
                "type": "text",
                "fields": {
                    "ngram": { "type": "text", "analyzer": "custom_ngram" }
                }
            },
            "description": {
                "type": "text",
                "analyzer": "englando",
                "fields": {
                    "shingles": { "type": "text", "analyzer": "custom_shingles" }
                }
            },
            "followers_count": { "type": "integer" },
            "following_count": { "type": "integer" },
            "tweet_count": { "type": "integer" },
            "listed_count": { "type": "integer" }
        }
    }
}
```

*Fig. 5 Mapping pre dáta z tabuľky authors*

Informácie o autorovi sú v dokumente uložené ako samostatný objekt `author`. Pre atribúty `name`, `username` a `description` som pridal aj dodatočné mapovanie pre vytvorené analyzéry `custom_ngram` a `custom_shingles`.

```
{
    "context_annotations": {
        "type": "nested",
        "properties": {
            "entity": {
                "properties": {
                    "id": { "type": "long" },
                    "name": { "type": "keyword" },
                    "description": { "type": "text", "analyzer": "englando" }
                }
            },
            "domain": {
                "properties": {
                    "id": { "type": "long" },
                    "name": { "type": "keyword" },
                    "description": { "type": "text", "analyzer": "englando" }
                }
            }
        }
    }
}
```

*Fig. 6 Mapping pre dáta z tabuľky context_annotations*

Dáta z `context_annotations` sú reprezentované ako pole objektov `entity` a `domain`. Aby sa dalo dopytovať na jednotlivé objekty v poli, je potrebné `context_annotations` indexovať ako `nested`.

```
{
    "conversation_hashtags": {
        "properties": {
            "tag": { "type": "text", "analyzer":"keyword_lowercase" }
        }
    }
}
```

*Fig. 7 Mapping pre dáta z tabuľky conversation_hashtags*

Hashtagy sú indexované ako pole objektov, ktoré majú atribút `tag`, reprezentujúci samotný hashtag. `Tag` je indexovaný ako `text` s vlastným analyzátorom, ktorý ich indexuje v lowercase. V tomto prípade netreba špecifikovať typ `nested`, keďže objekty obsahujú len jeden atribút.

```
{
    "annotations": {
        "type": "nested",
        "properties": {
            "value": { "type": "keyword" },
            "type": { "type": "keyword" },
            "probability": { "type": "half_float" }
        }
    },
    "links": {
        "type": "nested",
        "properties": {
            "url": { "type": "keyword" },
            "title": { "type": "keyword" },
            "description": { "type": "keyword" }
        }
    }
}
```

*Fig. 8 Mapping pre dáta z tabuliek annotations a links*

Mapovanie tabuliek `annotations` a `links` je viacmenej rovnaké, kde obe tabuľky sú reprezentované ako pole objektov. V oboch prípadoch mi prišlo najlepšie indexovať textové polia ako `keyword`.

```
{
    "conversation_references": {
        "type": "nested",
        "properties": {
            "id": { "type": "long" },
            "type": { "type": "keyword" },
            "author": {
                "properties": {
                    "id": { "type": "long" },
                    "name": {
                        "type": "text",
                        "fields": {
                            "ngram": { "type": "text", "analyzer": "custom_ngram" },
                            "shingles": { "type": "text", "analyzer": "custom_shingles" }
                        }
                    },
                    "username": {
                        "type": "text",
                        "fields": {
                            "ngram": { "type": "text", "analyzer": "custom_ngram" }
                        }
                    }
                }
            },
            "content": { "type": "text", "analyzer": "englando" },
            "hashtags": {
                "properties":{
                    "tag": { "type": "text", "analyzer": "keyword_lowercase" }
                }
            }
        }
    }
}
```

*Fig. 9 Mapping pre dáta z tabuľky conversation_references*

Referencie sú reprezentované ako pole objetkov (tweetov), na ktoré sa odkazujú. Referencované tweety majú vybrané atribúty normálnych tweetov (aj s rovnakými typmi a indexovaním) s tým, že ešte obsahujú typ referencie, ktorý je indexovaný ako `keyword`.

## Denormalizácia

Keďže je pri denormalizácií potrebné joinovať veľké množstvo tabuliek, vytvoril som si pre zrýchlenie danej query dodatočné indexy pre všetky cudzie kľúče.

```
CREATE INDEX ca_conversation_id ON context_annotations(conversation_id);
CREATE INDEX ca_domain_id ON context_annotations(context_domain_id);
CREATE INDEX ca_entity_id ON context_annotations(context_entity_id);
CREATE INDEX h_conversation_id ON conversation_hashtags(conversation_id);
CREATE INDEX h_hashtag_id ON conversation_hashtags(hashtag_id);
CREATE INDEX an_conversation_id ON annotations(conversation_id);
CREATE INDEX l_conversation_id ON links(conversation_id);
CREATE INDEX cr_conversation_id ON conversation_references(conversation_id);
CREATE INDEX cr_parent_id ON conversation_references(parent_id);
```

*Fig. 10 SQL pre vytvorenie indexov pre cudzie kľúče*

Pri denormalizácií dát som postupne joinoval jednotlivé tabuľky s tabuľkou `conversations`. Kvôli lepšej prehladnosti, ako aj následnému importu dát do elasticu som sa rozhodol rovno uložiť dáta z ostatných tabuliek ako json, na čo v postgrese slúžia funkcie `to_json()` a `json_build_object()`.

Tabuľky, ktoré pre daný tweet mohli vrátiť viac riadkov (všetky okrem `authors`) boli kvôli jednoduchšej agregácií výsledkov ešte "zabalené" v subquery. Na agregáciu som použil funkciu `json_agg()`, ktorá tieto záznamy (json objekty) spojí do listu. Zaujímavou časťou je subquery pre získanie referencií z tabuľky `conversation_references`, kde sa na získanie autora ako aj hashtagov parent tweetu použijú samostatné subquery. Samotné query pre denormalizáciu je na obrázku (Fig. 11)

```sql
SELECT
    c.id, c."content", c.possibly_sensitive, c."language", c."source", c.retweet_count, c.reply_count,
c.like_count, c.quote_count, c.created_at,
    to_json(a.*) author,
    COALESCE(ca.jsons, '[]') context_annotations,
    COALESCE(ch.jsons, '[]') conversation_hashtags,
    COALESCE(an.jsons, '[]') annotations,
    COALESCE(l.jsons, '[]') links,
    COALESCE(cr.jsons, '[]') conversation_references
FROM conversations c
JOIN authors a ON c.author_id = a.id
LEFT JOIN (
    SELECT ca.conversation_id, json_agg(json_build_object('entity', ce.*, 'domain', cd.*)) jsons
    FROM context_annotations ca
    JOIN context_entities ce ON ca.context_entity_id = ce.id
    JOIN context_domains cd ON ca.context_domain_id = cd.id
    GROUP BY ca.conversation_id
) ca ON ca.conversation_id = c.id
LEFT JOIN (
    SELECT ch.conversation_id, json_agg(json_build_object('tag', h.tag)) jsons
    FROM conversation_hashtags ch
    JOIN hashtags h ON ch.hashtag_id = h.id
    GROUP BY ch.conversation_id
) ch ON ch.conversation_id = c.id
LEFT JOIN (
    SELECT an.conversation_id, json_agg(json_build_object('value', an."value", 'probability', an.probability,
'type', an."type")) jsons
    FROM annotations an
    GROUP BY an.conversation_id
) an ON an.conversation_id = c.id
LEFT JOIN (
    SELECT l.conversation_id, json_agg(json_build_object('url', l.url, 'title', l.title, 'description',
l.description)) jsons
    FROM links l
    GROUP BY l.conversation_id
) l ON l.conversation_id = c.id
LEFT JOIN (
    SELECT
        cr.conversation_id,
        json_agg(json_build_object(
            'id', p.id, 'type', cr."type", 'content', p."content",
            'author', (
                SELECT json_build_object('id', pa.id, 'name', pa."name", 'username', pa.username)
                FROM authors pa
                WHERE pa.id = p.author_id
            ),
            'hashtags', (
                SELECT json_agg(json_build_object('tag', h.tag))
                FROM conversation_hashtags ch
                JOIN hashtags h ON ch.hashtag_id = h.id
                WHERE ch.conversation_id = p.id
            )
        )) jsons
    FROM conversation_references cr
    JOIN conversations p ON cr.parent_id = p.id
    GROUP BY cr.conversation_id
) cr ON cr.conversation_id = c.id;
```

*Fig. 11 SQL pre denormalizovanie tabuľky*

Ďalším krokom by ešte mohlo byť vytvorenie novej tabuľky a tieto získané dáta do nej pridať, čo by následne zrýchlilo samotný import dát do elasticu, keďže by sa jednotlivé riadky len vyberali z databázy. Ja som, ale nemohol urobiť tento krok kvôli nedostatočnému voľnému miestu.

## Úloha 4

Vlastné analyzéry a filtre sa dajú špecifikovať pri vytváraní indexu. Analyzéry sú vlastne definované presne podľa zadania okrem `keyword_lowercase`, ktorý slúži na indexovanie hashtagov v lowercase a ako `tokenizer` využíva `keyword`, keďže hashtagy sú aj tak zvyčajne len jeden výraz. Nastavenia indexu spolu s analyzátormi a filtrami sa nachádza v `src/config/settings.json`.

```
{
    "analysis": {
        "analyzer": {
            "englando": {
                "type": "custom",
                "tokenizer": "standard",
                "char_filter": ["html_strip"],
                "filter": [
                    "english_possessive_stemmer",
                    "lowercase",
                    "english_stop",
                    "english_stemmer"
                ]
            },
            "custom_ngram": {
                "type": "custom",
                "tokenizer": "standard",
                "char_filter": ["html_strip"],
                "filter": [
                    "lowercase",
                    "asciifolding",
                    "filter_ngrams"
                ]
            },
            "custom_shingles": {
                "type": "custom",
                "tokenizer": "standard",
                "char_filter": ["html_strip"],
                "filter": [
                    "lowercase",
                    "asciifolding",
                    "filter_shingles"
                ]
            },
            "keyword_lowercase": {
                "type": "custom",
                "tokenizer": "keyword",
                "filter": ["lowercase"]
            }
        },
    }
}
```

*Fig. 12 Definovanie vlastných analyzérov*

Pri vytváraní analyzérov bolo taktiež potrebné ešte dodefinovať filtre: `english_possessive_stemmer`, `english_stop`, `english_stemmer`, `filter_shingles` a `filter_ngrams`, ktoré sú na obrázku (Fig. 13).

```
{
       "filter": {
            "filter_ngrams": {
                "type": "ngram",
                "min_gram": 1,
                "max_gram": 10
            },
            "filter_shingles": {
                "type": "shingle",
                "token_separator": ""
            },
            "english_possessive_stemmer": {
                "type": "stemmer",
                "language": "possessive_english"
            },
            "english_stop": {
                "type": "stop",
                "stopwords": "_english_"
            },
            "english_stemmer": {
                "type": "stemmer",
                "language": "english"
           }
       }
}
```

*Fig. 13 Definovanie vlastných filtrov*

Pri `filter_ngrams` bolo potrebné ešte dodatočne zmeniť nastavenie indexu pre `max_ngram_diff` na hodnotu `9`.

## Úloha 5

Pred samotným importom je najskôr potrebné vytvoriť daný index a mapovanie, kde na vytvorenie a nastavenie indexu slúži napr. dopyt `PUT http://localhost:9200/tweets`, ktorý má v body json obsahujúci nastavenia a špecifikované analyzéry pre daný index. Mapping sa vytvára podobne pomocou `PUT http://localhost:9200/tweets/_mapping`, kde je rovnako v body špecifikovaný daný mapping. Ďalšia možnosť je využiť elastic klient a mapping vytvoriť rovno v skripte pomocou funkcie `es.indices.create(index, settings, mapping)`, do ktorej idú ako parametre rovno objekty pre nastavenia a mapping. Ja som mapping vytváral rovno v skripte vo funkcií `create_mapping(es)`, zároveň sa json súbory pre nastavenia a mapping nachádzajú v `src/config/settings.json` resp. `src/config/mapping.json`.

```
es.indices.create(
    index=INDEX_NAME,
    settings=settings,
    mappings=mapping
)
```

*Fig. 14 Vytvorenie indexu pomocou python klienta*

Dáta som do elasticu importoval pomocou python scriptu, ktorého hlavná časť sa nachádza v `src/main.py`. Import pozostáva hlavne z vyberania dát z postgresql a ich následnému importovaniu do elasticu.

Pre samotné importovanie slúži funkcia `import_data(conn, es, data_size)`, ktorá sa skladá z dvoch hlavných cyklov, kde prvý cyklus slúži na samotné vyberanie dát z databázy.

```
while True:
    cur = query_data(conn, last_id, limit)
    data = []

    while True:
        rows = cur.fetchmany(batch_size)
        if len(rows) == 0:  break

        for row in rows:
            header = {'index': {'_index': INDEX_NAME, '_id': row['id']}}
            data.extend([header, row])
                processed_rows += 1
        last_id = header['index']['_id']

        es.bulk(index=INDEX_NAME, operations=data)
        data.clear()

        total_processed_rows += processed_rows
    cur.close()

    if processed_rows == 0 or total_processed_rows >= data_size:
        break
```

*Fig. 15 Hlavná časť importovania dát z funkcie import_data()*

Keďže tabuľka `conversations` obsahuje približne 32 miliónov riadkov, nie je veľmi efektívne ich vybrať všetky naraz, preto sa dáta vyberajú vo viacerých skupinách podľa určeného limitu, ktorý bol nastavený na 8 miliónov. Na začiatku cyklu sa vždy zavolá funkcia `query_data(conn, last_id, limit)`, ktorá v podstate len vytvorí server cursor a vykoná query z Fig. 11 spolu s daným limitom a podmienkou `WHERE c.id > {last_id}`, ktorá zaisťuje, že ďalšia dávka, alebo skupina, bude obsahovať iba riadky, ktoré sú v tabuľke ďalej ako posledný importovaný riadok, ktorého id = `last_id`.

Druhý cyklus už slúži na importovanie dát do elasticu po určených dávkach (v mojom prípade po 200 riadkoch). V cykle sa najskôr pomocou `cur.fetchmany(batch_size)` vyberie dávka dokumentov, cez ktorú je potrebné prejsť a pripraviť dáta pre bulk import. V elasticu musí byť pri bulk importe pred samotným dokumentom ešte jeden riadok (`header`), ktorý opisuje aká akcia sa má vykonať (`'index'`), v ktorom indexe (`'_index'`) a id daného dokumentu (`_id`), ktoré je v našom prípade rovnaké ako id tweetu v databáze. Nakoniec sa vykoná samotný import pomocou `es.bulk(index_name, data)` a cyklus sa zopakuje.

Moja metóda, ale nie je veľmi efektívna. Určite by sa dala zlepšiť napr. pomocou paralelizácie dopytov do postgresu a následného importovania do elasticu. Kedy by mohlo niekoľko threadov súčastne vyberať iné dávky dokumentov, čím by sa skrátilo čakanie na vybratie ďalšej skupiny (`WHERE` podmienka), ktoré sa časom ešte predlžuje.

Ukážka za-indexovaného dokumentu sa nachádza aj v `docs/sample_document.json`.

```
{
    "_index": "tweets",
    "_id": "1497032529894805509",
    "_score": null,
    "_source": {
        "id": 1497032529894805509,
        "content": "RT @one_sorrow: SPREAD AND SHARE, YOU CAN HELP UKRAINE #Ukraine #Russia https://t.co/
rp2IFCKMi3",
        "possibly_sensitive": false,
        "language": "en",
        "source": "Twitter for Android",
        "retweet_count": 6085,
        "reply_count": 0,
        "like_count": 0,
        "quote_count": 0,
        "created_at": "2022-02-25T03:15:43+01:00",
        "author": {
            "id": 1409764954845159428,
            "name": "ted | wil | alex",
            "username": "michaelkinnie",
            "description": "⋆they/he/it/xey⋆\n⋆queer and nonbinary⋆\n⋆ted nivison, wilbur soot, & alex
kralie irl (srs)⋆",
            "followers_count": 9,
            "following_count": 126,
            "tweet_count": 1181,
            "listed_count": 0
        },
        "context_annotations": [
            {
                "entity": {
                    "id": 1484601166080081920,
                    "name": "Russo-Ukrainian conflict",
                    "description": null
                },
                "domain": {
                    "id": 123,
                    "name": "Ongoing News Story",
                    "description": "Ongoing News Stories like 'Brexit'"
                }
            },
            {
                "entity": {
                    "id": 1484601166080081920,
                    "name": "Russo-Ukrainian conflict",
                    "description": null
                },
                "domain": {
                    "id": 123,
                    "name": "Ongoing News Story",
                    "description": "Ongoing News Stories like 'Brexit'"
                }
            }
        ],
        "conversation_hashtags": [
            {
                "tag": "Ukraine"
            },
            {
                "tag": "Russia"
            }
        ],
        "annotations": [
            {
                "value": "UKRAINE",
                "probability": 0.954,
                "type": "Place"
            }
        ],
        "links": [
            {
                "url": "https://twitter.com/one_sorrow/status/1496727690157588483/photo/1",
                "title": null,
                "description": null
```

*Fig. 16 Ukážka zaindexovaného dokumentu*

## Úloha 6

Na importovanie prvých 5000 záznamov do elasticu stačí len zavolať funkciu `import_data(conn, es, data_size)` s `data_size=5000`. Celý import trval okolo 2s a finálny počet dokumentov sa dá napr. overiť pomocou jednoduchého vyhľadávania, ktoré vráti všetky zaindexované dokumenty.

```
{
    "query": {
        "match_all": {}
    }
}
```

*Fig. 17 Query pre nájdenie všetkých dokumentov*

Táto query naozaj nájde 5000 dokumentov:

```
  "hits": {
    "total": {
      "value": 5000,
      "relation": "eq"
    },
```

*Fig. 18 Výsledok z vyhľadávania všetkých dokumentov*

## Úloha 7

```
ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role   master name
172.23.0.3           21          99  17    1.66    0.79     0.39 cdfhilmrstw -      es03
172.23.0.4           34          98  24    1.66    0.79     0.39 cdfhilmrstw -      es01
172.23.0.2           19          98  27    1.66    0.79     0.39 cdfhilmrstw *      es02
```

*Fig. 19 Pôvodne usporiadanie nodov*

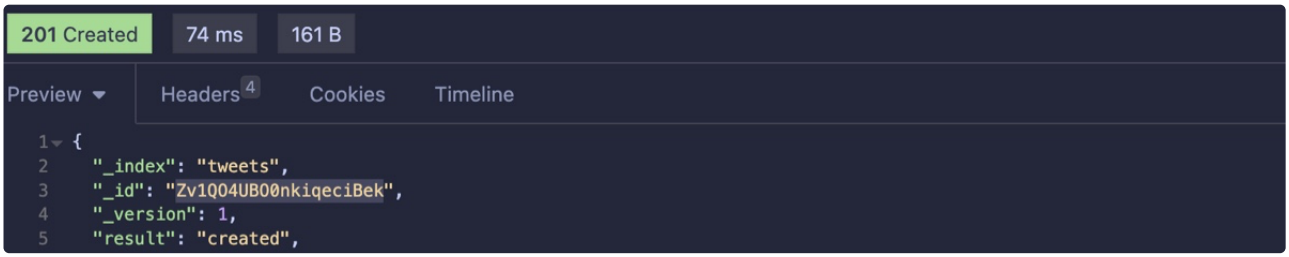
Na začiatku je node `es02` nastavený ako master. Ak ho zastavím pomocou `docker stop elastic-es02-1`, zvolí sa nový master (v tomto prípade `es01`) a všetky shardy sa prerozdelia na zvyšné nody.

```
ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role   master name
172.23.0.4           23         100   0    0.10    0.37     0.32 cdfhilmrstw *      es01
172.23.0.3           59         100   0    0.10    0.37     0.32 cdfhilmrstw -      es03
```

```
tweets          0 p STARTED 6265  3.3mb 172.23.0.4 es01
tweets          0 r STARTED 6265  3.3mb 172.23.0.3 es03
tweets          1 r STARTED 6259  3.2mb 172.23.0.4 es01
tweets          1 p STARTED 6259  3.2mb 172.23.0.3 es03
tweets          2 p STARTED 6259  3.4mb 172.23.0.4 es01
tweets          2 r STARTED 6259  3.4mb 172.23.0.3 es03
```

*Fig. 20 Nové usporiadanie nodov a shardov*

V tomto rozdelení je taktiež ešte možné pridávať, prehľadávať a mazať dokumenty.



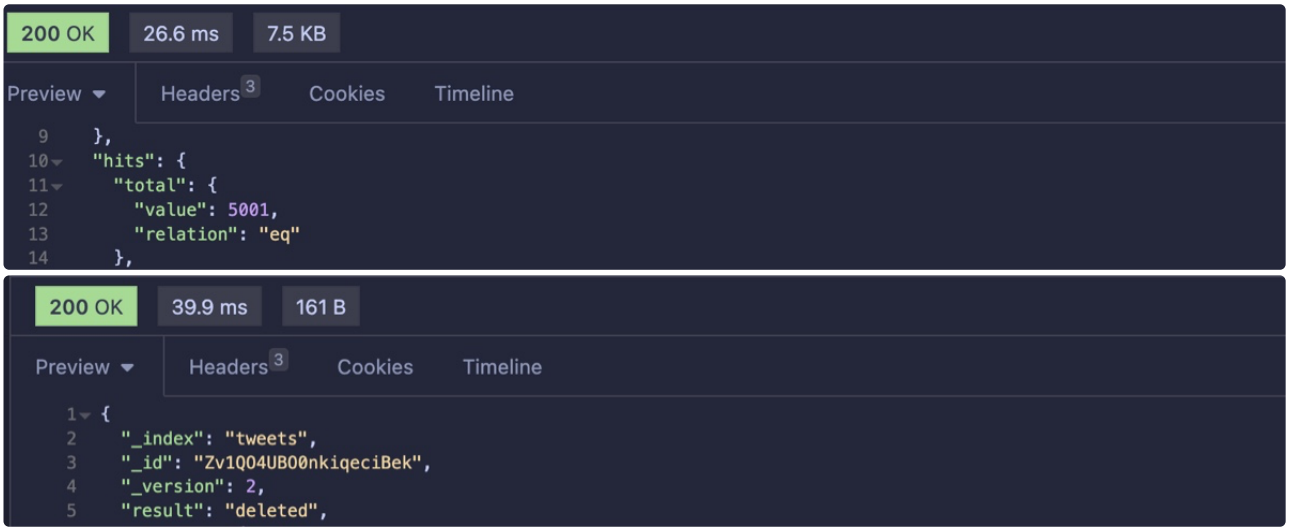



*Fig. 21 Výsledok pridávania, prehľadávania a mazania po vypnutí jedného nodu*

Ak ale vypnem aj node `es03`, kluster prestane fungovať. Toto je spôsobené tým, že klaster už neobsahuje dostatočné množstvo nodov (mala by byť dostupná nadpolovičná väčšina "master-eligible" nodov), ktoré sa môžu stať masterom a nedokáže sa zvoliť nový master.

Elasticsearch sa dá nastaviť aj tak aby tvoril klaster len jeden node, na čo slúži configurácia `discovery.type=single-node`. Toto nastavenie, ale nie je odporúčané pre komerčné aplikácie. Takýto klaster s jedným nodom nie je odolný, keďže ak tento node zlyhá, celá aplikácia prestane fungovať - čo je aj jedným z dôvodov prečo takéto kvórum existuje.

## Úloha 8

Na začiatku má dokument `"_seq_no": 1687` a `"_primary_term": 1`.


*Fig. 22 Pôvodný stav dokument*

Ak pomocou `POST http://localhost:9200/tweets/_update/1082538033130414080` upravím hodnotu `"retweet_count"`, zmení sa len `"_seq_no": 1687`, ktoré odzrkadluje počet zmien samotného dokumentu. Teda, ak pomocou nasledovného skriptu:

```
{
  "script": {
    "source": "ctx._source.retweet_count += params.number_of_retweets",
    "params": {
      "number_of_retweets": 2
    }
  }
}
```

*Fig. 23 Skript na úpravu dokumentu*

štyrikrát zmením počet retweetov, `_seq_no` sa zvýši o 4.

```
  "_index": "tweets",
  "_id": "1082538033130414080",
  "_version": 5,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 1691,
  "_primary_term": 1
}
```


*Fig. 24 Dokument po zvýšení retweetov*

Ak zrušim node `es03`, na ktorom sa nachádza shard s daným dokumentom (kde je uložený daný dokument sa dá zistiť pomocou `"analyse": true`), zvýši sa aj `"_primary_term"`, ktorý vlastne hovorí koľkokrát sa zmenil primárny shard daného dokumentu. Teda v tomto prípade sa z repliky shardu, na ktorom bol daný dokument stal nový primárny shard.


```
{
  "_index": "tweets",
  "_id": "1082538033130414080",
  "_version": 6,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 1692,
  "_primary_term": 2
}
```


*Fig. 25 Dokument po vypnutí nodu `es03`*

Ak by som spustil node `es03` a zas zrušil node `es01`, na ktorom sa dokument nachádza tentokrát, hodnota `"_primary_term"` by sa znovu zvýšila.


```
{
  "_index": "tweets",
  "_id": "1082538033130414080",
  "_version": 8,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "_seq_no": 1694,
  "_primary_term": 3
}
```


*Fig. 26 Dokument po vypnutí nodu `es01`*

Tieto hodnoty slúžia na sledovanie zmien dokumentov, kedy napr. v prípade výpadku jedného zo shardov zabezpečujú, že shard a jeho repliky majú rovnakú (najaktuálnejšiu) verziu daného dokumentu a zabezpečujú rýchlejšie zotavenie shardu po jeho výpatku.

## Úloha 9

V tejto úlohe je najskôr potrebné vymazať index na čo slúži dopyt `DELETE http://localhost:9200/tweets` alebo pomocou elastic klienta pre python a funkcie `es.indices.delete(index='tweets')`.

Pre odstránenie replík stačí len prepísať v nastaveniach indexu (Fig. 2) `"number_of_replicas"` na `0`. Oproti importovaniu 5000 záznamov je v tomto prípade potrebné v skripte nastaviť veľkosť dát (`data_size`) na `-1`, kedy sa naiportuje celá databáza:

```
import_data(conn, es, data_size)
```

Pre prvých 8 miliónov dokumentov bol skript pomerne rýchly a dokázal importovať 10000 dokumentov za približne 4s. Problém, ale nastal po prekročení 2 miliónov, kedy elastic začal hlásiť chybu s voľným miestom. Tento problém sa mi nepodarilo vyriešiť, ale najskôr bude chyba s nastavením elastic klastera alebo samotného dockera, pretože v tomto momente ešte bolo voľné miesto na SSD, a taktiež nemohol byť problém v nedostatku RAM.

Nakoniec sa mi podarilo naimportovať len 2 691 600 dokumentov. Počet dokumentov sa dá pozrieť pomocou `GET http://localhost:9200/tweets/_count`.


```
{
  "count": 2691600,
  "_shards": {
    "total": 3,
    "successful": 3,
    "skipped": 0,
    "failed": 0
  }
}
```


*Fig. 27 Počet naimportovaných dokumentov*

## Úloha 10

Túto úlohu som nestihol 😕