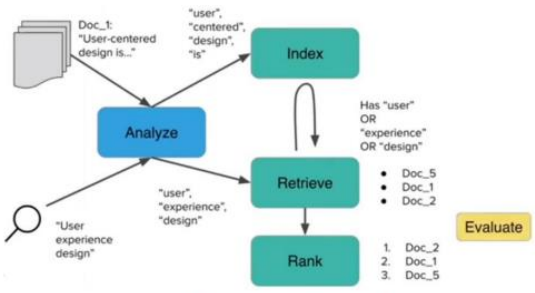
34. Databáze pro prohledávání a analýzu textu- princip, vlastnosti. Elasticsearch- architektura, prohledávání vs. analýza, invertovaný index.

Databáze pro prohledávání a analýzu textu- princip, vlastnosti.

- search-engine databáze
  - NoSQL dokumentové databáze zaměřené na vyhledání obsahu
  - využívají indexování
    - kategorizace podobných vlastností mezi daty
    - ➢ urychlení vyhledávání
  - optimalizované pro práci s daty
    - velké množství dat
    - strukturovaná i nestrukturovaná data
      - volné schéma
  - poskytují speciální funkce
    - full-textové vyhledávání
    - složité vyhledávací výrazy
    - řazení výsledků
    - distribuované vyhledávání



https://www.youtube.com/watch?v=dqRDyeFJUvk

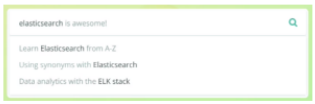
  - distribuované vyhledávání
    - škálování, sharding, replikace
  - příklady aplikace
    - textové vyhledávání
      - elektronický obchod
      - automatické dokončování hledaných výrazů, doporučení
      - setřídění výsledků na základě různých kritérií (jméno, cena, datum vydání, …)
    - logování a analýza
      - centralizace a zaindexování logů z různých zařízení

- komunikace s Elasticsearch REST API
  - využití HTTP žádostí pro přístup k datům
    - metody POST, GET, PUT a DELETE
    - odpovídá CRUD – vytvoření, čtení, update, smazání dat
    - libovolný HTTP klient
  - využití Kibany -> http://localhost:5601
    - vestavěná konzole
    - překládá uživatelské dotazy na HTTP žádosti
    - žádosti následně posílány do Elasticsearch
    - dotaz na stav clusteru

```
GET /_cluster/health
```

Elasticsearch- architektura, prohledávání vs. analýza, invertovaný index

- engine pro full-textové prohledávání a analýzu
  - open source
    - napsán v Javě
    - založený na Apache Lucene
  - dokumentová databáze
    - data ukládána do dokumentů s poli
    - využívá JSON
  - dotazování pomocí REST API
  - distribuovaný
    - vysoká škálovatelnost
    - ➢ rychlost prohledávání



https://www.udemy.com/course/elasticsearch-complete-guide/

elasticsearch

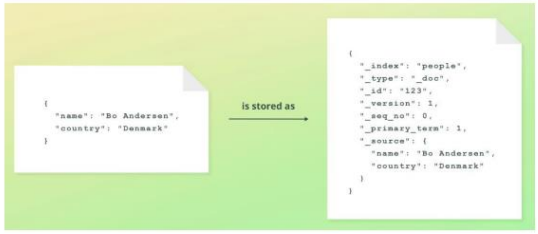
Architektura

ARCHITEKTURA ELASTICSEARCH

- základem je uzel (node)
  - automaticky založen při spuštění Elasticsearch
  - ➢ instance Elasticsearch obsahující data
  - uzlů může být samozřejmě více
    - každý uzel ukládá část dat
      - škálování
    - na jednom stroji může být více uzlů
    - fyzické, virtuální, docker kontejner, …
- každý uzel je součástí clusteru
  - automaticky založen při vytvoření prvního uzlu
    - uzel je buď přiřazen do existujícího clusteru
    - nebo uzel svým vznikem zakládá nový cluster
  - může jich být také víc
    - ale většinou pro logicky oddělené úlohy

https://www.udemy.com/course/elasticsearch-complete-guide/

- základní jednotkou pro ukládání dat je dokument
  - JSON objekty obsahující data
  - Elasticsearch ukládá společně s metadaty
    - původní dokument v poli _source



- data jsou organizována v indexech
  - každý dokument uložen v indexu
    - neomezený počet dokumentů
  - logicky shlukují dokumenty
    - kolekce dokumentů s podobnými vlastnostmi
    - poskytují možnosti pro škálování a dostupnost
  - dotazy jsou spouštěny nad indexy

https://www.udemy.com/course/elasticsearch-complete-guide/

- vytvoření indexu

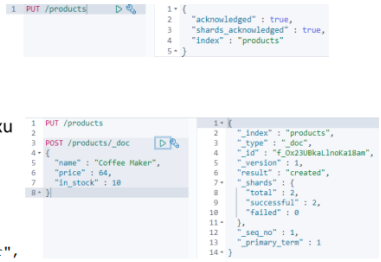
```
PUT /products
```

- smazání indexu

```
DELETE /products
```

- indexování dokumentů
  - vložení dokumentu do indexu
  - nutná specifikace indexu
  - dokument ve formátu JSON

```
POST /products/_doc
{
    "name" : "Coffee Maker",
    "price" : 64,
    "in_stock" : 10
}
```

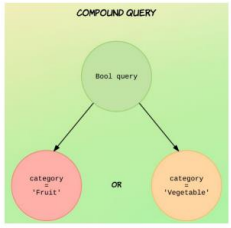


asd

Prohledávání vs analýza

Prohledávání

- dva způsoby psaní vyhledávácích dotazů
  - Query DSL
    - předání dotazovacího JSON objektu
    - hlavní způsob dotazování
    - dvě hlavní skupiny dotazů
    - leaf queries
      - vyhledávají hodnotu v konkrétních polích
      - např. kategorie Ovoce
    - compound queries
      - skládají se z leaf queries nebo dalších compound queries
      - rekurzivní
      - např. kategorie Ovoce nebo (boolean) Zelenina



https://www.udemy.com/course/elasticsearch-complete-guide/

```
GET /products/_search
{
  "query": {
    "match_all": {}
  }
}
```

```
GET /products/_search
{
  "query": {
    "match": {
      "description": "red wine"
    }
  }
}
```

- reprezentace výsledků
  - took
    - čas provedení dotazu [ms]
  - timed_out
    - boolean vypršení času na dotaz
  - _shards
  - hits
    - hlavní výsledek hledání
    - total
      - počet dokumentů odpovídajících vyhledávání
    - max_score
      - maximální skóre relevance
    - hits
      - pole samotných výsledků
      - _score
        - skóre relevance
        - vyjadřuje míru, s jakou výsledek odpovídá dotazu

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 8.577639,
    "hits" : [
      {
        "_index" : "products",
        "_type" : "_doc",
        "_id" : "65",
        "_score" : 8.577639,
        "_source" : {
          "name" : "Tuna - Bluefin",
          "price" : 27,
          "in_stock" : 26,
          "sold" : 378,
          "tags" : [
            "Meat"
          ],
          "description" : "Integer pede justo, lacinia
            eget, tincidunt eget, tempus vel, pede.
            Morbi porttitor lorem id ligula.
            Suspendisse ornare consequat lectus.",
          "is_active" : false,
          "created" : "2015/03/23"
        }
      }
    ]
  }
}
```

- skóre relevance (_score)
  - vyjadřuje, jakou měrou výsledek vyhledávání odpovídá dotazu
  - jsou podle něj řazeny výsledky vyhledávání
    - cílem není jen vrátit výsledky vyhledávání, ale vrátit relevantní výsledky
    - rozdíl oproti klasickým databázovým systémům
  - postup v Elasticsearch
    - boolean model
    - nejprve jsou nalezeny všechny dokumenty, které odpovídají dotazu
      - vyřadí ostatní dokumenty z počítání skóre relevance
    - pro vybrané dokumenty je následně počítáno skóre relevance

- jak je skóre relevance počítáno?
  - liší se podle typu vyhledávacího dotazu
  - možnost výběru algoritmu
    - i možná úprava skórování
  - donedávna využíván algoritmus TF/IDF
    - Term Frequency / Inverse Document Frequency
  - v současnosti algoritmus Okapi BM25
  - oba algoritmy jsou si velmi podobné
    - založené na stejném principu
    - Okapi BM25 řeší určitě nedostatky TF/IDF

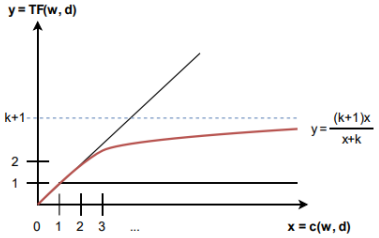
TF-IDF

- V praxi potřebujeme použít počet výskytů
- TF = term frequency
  - Kolikrát se výraz vyskytuje v dokumentu
- Problém s TF
  - Pokud se slovo vyskytuje často v každém dokumentu (spojky, předložky, ...)
- IDF = inverse document frequency
  - V jaké míře se slovo vyskytuje v celé kolekci
  - Využito k penalizaci častých slov

$$sim(q,d) = \sum_{i=1}^{N} x_i y_i = \sum_{w \in q \cap d} c(w,q)c(w,d)\log(\frac{M+1}{df(w)})$$

OKAPI BM25

- Problém TF-IDF
  - Jedno časté slovo z dotazu může dominovat nad ostatními
    - Např. často v nějakém nerelevantním dokumentu, ale méně často v celé kolekci
- Parametr $k \geq 0$
  - k = 0: binární TF
  - Příliš velké k: y=x (TF)



$$sim(q,d) = \sum_{i=1}^{N} x_i y_i = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k} log(\frac{M+1}{df(w)})$$

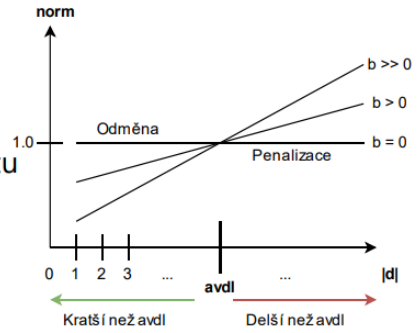
Kromě TF transformace se používá normalizace délky dokumentu

- Dlouhý dokument bude mít větší šanci vyhledání uživatelskými dotazy

- Penalizace s pomocí parametru $b \in [0,1]$

$$norm = 1 - b + b\frac{|d|}{avdl}$$

$|d|$ = délka dokumentu
$avdl$ = průměrná délka dokumentu



- Výsledná rovnice BM25 při počítání relevance pomocí skalárního součinu:

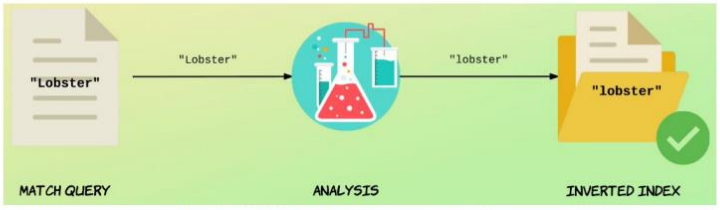
$$sim(q,d) = \sum_{i=1}^{N} x_i y_i = \sum_{w \in q \cap d} c(w,q) \frac{(k+1)c(w,d)}{c(w,d)+k(1-b+b\frac{|d|}{avdl})} log(\frac{M+1}{df(w)})$$

- kontext dotazů
  - dotaz může být položen ve dvou různých kontextech
  - dotazovací kontext (query context)
    - ptáme se: „Jak moc dokumenty odpovídají dotazu?“
    - nejprve je zjištěno, jestli dokument odpovídá dotazu
    - následně je vyhodnoceno skóre relevance
    - články věnované vyhledávanému termínu, …
  - filtrovací kontext (filter context)
    - ptáme se: „Vyhovují dokumenty dotazu?“
    - binární – ano / ne
    - odpovídá prvnímu kroku z dotazovacího kontextu
    - není počítáno skóre relevance
    - mohou být cachované
    - datumy, statusy, rozsahy, …

- dotazovací typy
  - dotazy na úrovni termínů (term level queries)
    - vyhledávají přesnou hodnotu vůči invertovanému indexu (po analýze)
      - ne vůči přímo cílovému dokumentu
    - vyhledávaná hodnota není analyzována
    - vhodné např. pro datumy, čísla, …
      - ne pro řetězce

https://www.udemy.com/course/elasticsearch-complete-guide/

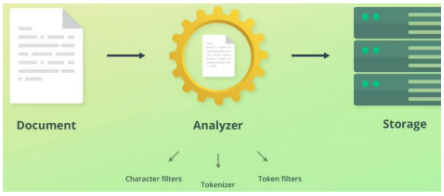
- dotazovací typy
  - fulltextové dotazy (full-text queries)
    - vyhledávaná hodnota je analyzována stejným analyzérem jako invertovaný index
    - je možné najít jen hodnoty v invertovaném indexu
    - vhodné pro fulltextové vyhledávání
      - řetězce



https://www.udemy.com/course/elasticsearch-complete-guide/

Analýza

- při indexaci jsou textová pole analyzována
  - o analýzu se stará analyzér skládající se ze tří komponent
    - znakový filtr (character filter)
    - tokenizér (tokenizer)
    - token filtr (token filter)
  - výsledky analýzy uloženy v polích efektivních pro prohledávání



https://www.udemy.com/course/elasticsearch-complete-guide/

Znakový filtr

- transformují vstupní text úpravou znaků
  - přidání, změna, odstranění
- nemusí být žádný, ale i několik
- případně aplikovány v předem definovaném pořadí
- např. filtr html_strip
  - filtr odstraňující HTML elementy a transformující HTML entity

Input: `"I&apos;m in a <em>good</em> mood - and I <strong>love</strong> açaí!"`

Output: `"I'm in a good mood - and I love açaí!"`

https://www.udemy.com/course/elasticsearch-complete-guide/

- např. filtr odstraňující diakritiku

Tokenizer:

- tokenizér (tokenizer)
  - přesně jeden v analyzéru
  - stará se o tokenizaci
    - rozdělení řetězce na tokeny (token = nejmenší jednotka textu; většinou grafické slovo)
  - může odstraňovat znaky
    - interpunkce
  - ukládá také offset znaků
  - např. rozdělení řetězce podle bílého znaku
    - výsledkem jsou 4 tokeny

Input: `"I REALLY like beer!"`

Output: `["I", "REALLY", "like", "beer"]`

Token filtry:

- obdržují výstup z tokenizéru jako vstup
- mohou přidávat, modifikovat nebo mazat tokeny
- nemusí být žádný, ale i několik
- případně aplikovány v předem definovaném pořadí
- např. lowercase filtr
  - převedení všech tokenů na malá písmena

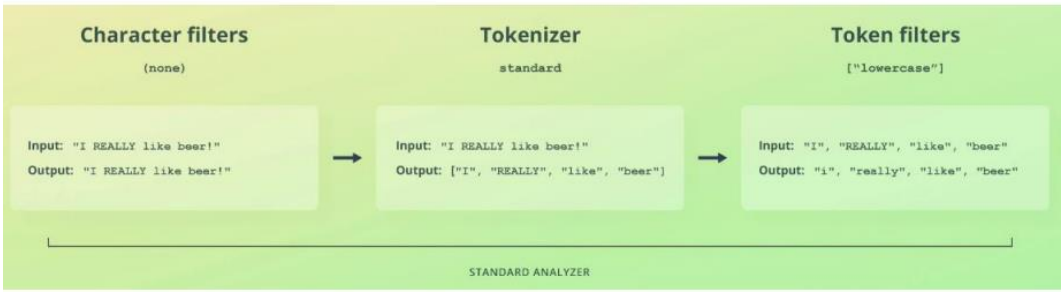
Input: `["I", "REALLY", "like", "beer"]`

Output: `["i", "really", "like", "beer"]`

https://www.udemy.com/course/elasticsearch-complete-guide/

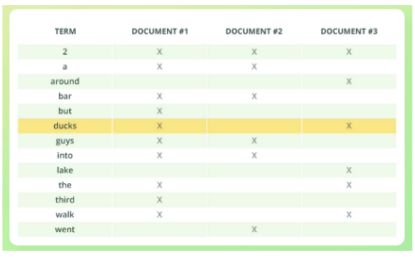
- co se stane s textem v základním nastavení?
  - žádný znakový filtr není použit
  - tokenizer dělí řetězec na tokeny podle Unicode Segmentation algoritmu
    - víceméně rozděluje řetězec podle bílých znaků, pomlček a podobně
    - odstraňuje také interpunkci
  - lowercase token filter převádí tokeny na malá písmena
  - ➢ standardní analyzér
    - použit na všechna textová pole, pokud není specifikováno jinak
    - ➢ použit ve většině případů

Standartní analyzér:



➢ tokeny jsou uloženy v datové struktuře invertovaný index

- ➢ efektivní vyhledání výrazu a dokumentů, kde se výraz vyskytuje
- proč invertovaný?
  - obrácené mapování logičtější
    - dokument odkazuje na výrazy, které obsahuje
    - to ale neumožňuje efektivní vyhledávání výrazů
    - ➢ invertovaný index
- v praxi obsahují invertované indexy další informace
  - např. informace pro skóre relevance



TERM	DOCUMENT #1	DOCUMENT #2	DOCUMENT #3
2	X	X	X
a	X	X	
around			X
bar	X	X	
but	X		
ducks	X		X
guys	X	X	
into	X	X	
lake			X
the	X		X
third	X		
walk	X		X
went		X	

https://www.udemy.com/course/elasticsearch-complete-guide/

- vytvářeny pro každé textové pole
  - jeden invertovaný index = jedno textové pole
  - ➢ definovány na úrovní polí
  - pro netextová pole používány jiné datové struktury
    - např. BKD stromy pro číselné hodnoty a data

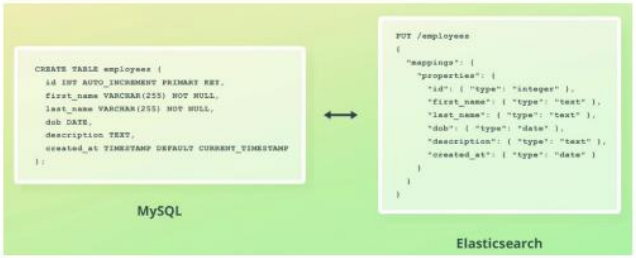
TERM	DOCUMENT #1	DOCUMENT #2
coffee	X	
maker	X	
toaster		X

TERM	DOCUMENT #1	DOCUMENT #2
coffee		X
delicious	X	
fast		X
makes	X	X
super		X
toasts	X	

Mapování

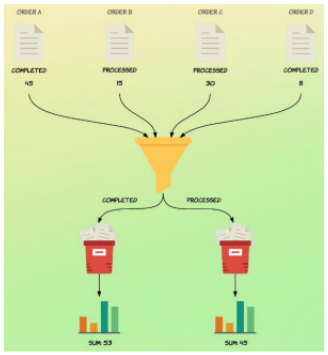
MAPOVÁNÍ

- definuje strukturu dokumentů, a jak jsou indexovány a ukládány
  - pole a jejich datové typy
- zjednodušeně se dá přirovnat k relaci v relačních databázích



Agregace

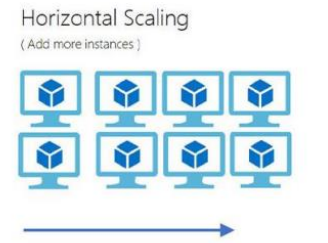
- způsob jak seskupit data a jak z nich získat statistiky a závěry
- pracuje nad dokumenty definovanými prováděcím kontextem
- dva základní typy
  - agregace metrik (metric aggregation)
    - počítají statistiky přes data
    - min, max, avg, sum, …
  - bucket agregace
    - vytváří skupiny (buckets) dokumentů
    - každý bucket má kritéria, která rozhodují, jestli do nich dokument spadá
    - umožňuje vnořené agregace
    - agregace termínů (term), filter agregace agregace rozsahů (range), …



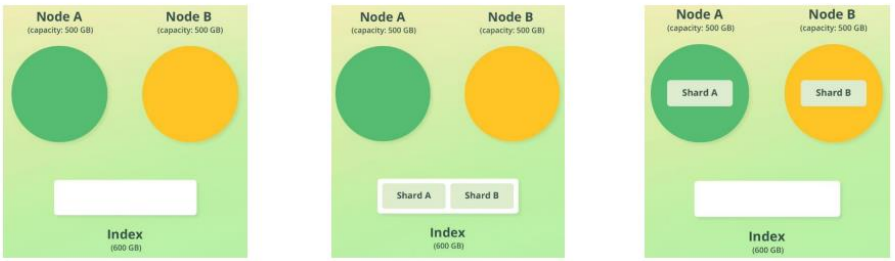
https://www.udemy.com/course/elasticsearch-complete-guide/

Sharding

- rozdělení indexů na menší části
  - každá část je nazývána shard
  - prováděno na úrovni indexů
    - ne uzlů nebo clusterů
- horizontální škálování
  - škálování datového úložiště

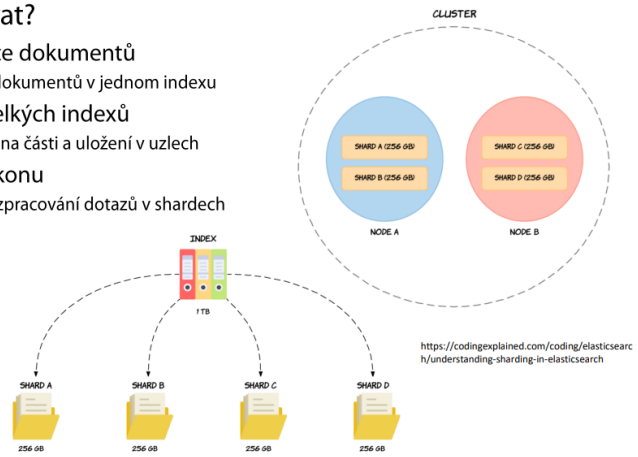


https://www.webairy.com/horizontal-and-vertical-scaling/



- každý shard
  - slouží víceméně jako samostatný index
    - Apache Lucene index
    - ➢ každý index je složen z alespoň jednoho Lucene indexu
  - nemá předdefinovanou velikost
    - roste s vkládanými dokumenty
  - může obsahovat až 2 miliardy dokumentů
- v základu každý index má jeden shard
  - možné konfigurovat
    - zvýšení počtu shardů pomocí Split API
    - snížení Shrink API
  - ➢ vhodné rozmyslet dopředu
    - neexistuje optimální počet shardů
    - závisí na konkrétní aplikaci

- proč shardovat?
  - ukládání více dokumentů
    - miliardy dokumentů v jednom indexu
  - rozdělení velkých indexů
    - rozdělení na části a uložení v uzlech
  - zlepšení výkonu
    - paralelní zpracování dotazů v shardech



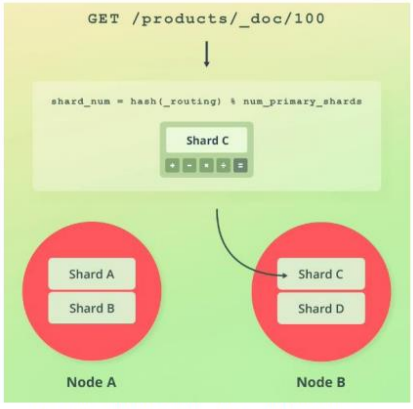
Routing:

proces přiřazující dokumentu shard

- určuje, kde bude dokument uložen
  - na jakém shardu
- udává, jak jsou dokumenty nalezeny po zaindexování
  - pro operace čtení, aktualizace nebo smazání
- ➢ Elasticsearch využívá jednoduchý vzorec

```
shard_num = hash(_routing) %
            num_primary_shards
```

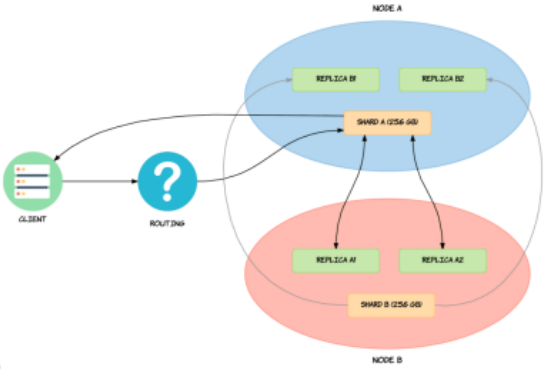
  - _routing – odpovídá _id dokumentu
    - možnost vlastního nastavení
  - num_primary_shards – celkový počet shardů
- zajišťuje rovnoměrné rozmístění dokumentů mezi shardy



https://www.udemy.com/course/elasticsearch-complete-guide/

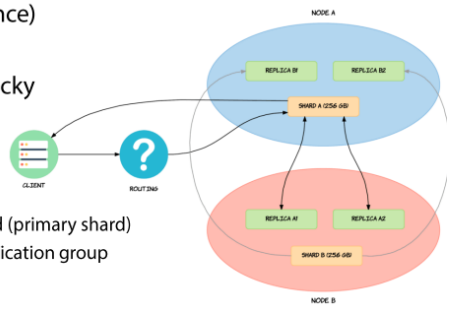
Replikace

- replikace v Elasticsearch
  - pro synchronizaci využíván model primary-backup
  - primární shard je vstupem pro indexovací operace
    - přidávání, update, mazání, …
  - primární shard operaci validuje a následně lokálně aplikuje na svá data
  - po dokončení je operace předána všem replikám
  - operace je paralelně provedena na všech replikách



https://codingexplained.com/coding/elasticsearch/understanding-replication-in-elasticsearch

  - po potvrzení od všech replik informuje primární shard klienta
- replikace
  - automatická distribuce změn v originále do jeho kopií
  - ➢ ochrana proti chybám (fault tolerance)
    - selhání hardware
  - nativní podpora, zapnuta automaticky
  - definována na úrovni indexů
  - vytváří úplné kopie shardů
    - repliky, repliky shardů (replica shards)
    - původní shard se nazývá primární shard (primary shard)
    - primary shard a replica shards tvoří replication group
    - ➢ mohou plně obsluhovat dotazy

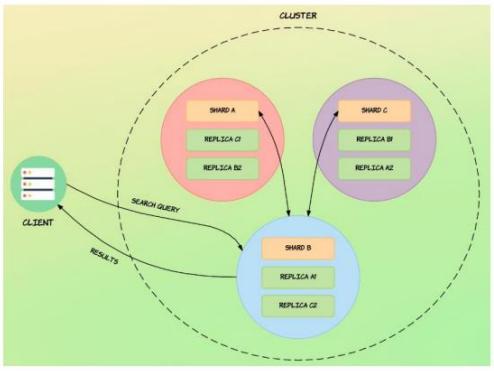


https://codingexplained.com/coding/elasticsearch/understanding-replication-in-elasticsearch

  - pro synchronizaci využíván model primary-backup
  - může také zvyšovat propustnost dotazování

Zpět k prohledávání:

- princip
  - dotaz dorazí od klienta na jeden z uzlů (modrý)
  - ten se stává koordinačním uzlem
  - broadcastem přeposílá dotaz ostatním shardům
    - primární i repliky
  - shardy odpovídají
  - koordinační uzel sloučí výsledky a vrací je klientovi
  - ➢ liší se výrazně od čtení podle _id



https://www.udemy.com/course/elasticsearch-complete-guide/