Predlog arhitekture sistema i konfiguracije propratnih servisa

Upravljanje digitalnim dokumentima 25-26

Nikola Simić E2 17/2024

1. Arhitektura sistema

Monolitna Java Spring Boot aplikacija će biti jezgro sistema koje će komunicirati sa ELK stack servisima, RustFs skladištem za objekte i PostgreSql bazom podataka. Eksterni servisi će raditi unutar Docker okruženja i biće konfigurisani tako da se nalaze u okviru iste virtuelne mreže kako bi mogli međusobno da komuniciraju.

Aplikacija će biti organizovana u jednostavnu n-tier arhitekturu koja razdvaja:

1. Prezentacioni/API sloj koji se sastoji od HTTP endpointa i mehanizama za validaciju inputa.
2. Logički sloj u kojem se nalazi glavna logika aplikacije - obrada zahteva,
3. Data/Eksterni sloj koji služi za persistovanje podataka i komunikaciju sa eksternim servisima kao što su baze podataka, ElasticSearch i RustFs.

Aplikacija će koristiti sledeće biblioteke za komunikaciju sa eksternim servisima:

1. Spring Data Elasticsearch – komunikacija sa ElasticSearchom
2. AWS Java SDK for S3 – komunikacija sa RustFsom
3. Spring Data JPA – komunikacija sa PostgreSql bazom podataka
4. Spring Starter logging + Logstash Logback encoder – preusmeravanja struktuiranih JSON logova ka LogStashu

2. Konfiguracija eksternih servisa i integracija sa aplikacijom

2.1 Konfiguracija i integracija ElasticSearch-a

ElasticSearch kontejner će biti pokrenut korišćenjem official slike sa sajta docker.elastic.co. (isto važi za Logstash i Kibana kontejnere u nastavku teksta). Početna konfiguracija podrazumeva da će se ElasticSearch pokretatit u single-node modu sa postavljenim ograničenjem za veličinu heap-a kako bi se obavio početni setup aplikacije. Po završetku aplikacije biće odvojeno vreme za optimizaciju konfiguracije. Komunikacija sa Kibanom će se izvršavati preko HTTP-a, a ova

konfiguracija se nalazi na samom Kibana kontejneru u environment promenljivoj ELASTICSEARCH_HOSTS.

Integracija sa aplikacijom će se svoditi na definisanje Java modela za ES indeks kao što je sledeći:

```java
@Entity
@Table(name = "security_documents")
@Document( indexName = "security_documents")
public class SecurityDocument {

    @Id
    @Field(name = "id")
    private long id;

    @Column(name = "full_name")
    @Field(type = FieldType.Text, name = "full_name", store = true)
    private String fullName;

    @Column(name = "org_name")
    @Field(type = FieldType.Text, name = "org_name", store = true)
    private String orgName;

    @Column(name = "org_address")
    @Field(type = FieldType.Text, name = "org_address", store = true)
    private String orgAddress;

    @Column(name = "threat_name")
    @Field(type = FieldType.Text, name = "threat_name", store = true)
    private String threatName;

    @Column(name = "threat_level")
    @Enumerated(EnumType.STRING)
    @Field(type = FieldType.Text, name = "threat_level", store = true)
    private ThreatLevel threatLevel;

    @Column(name = "threatSampleHash")
    @Field(type = FieldType.Text, name = "threatSampleHash", store = true)
    private String threatSampleHash;

    @Transient
    @Field(type = FieldType.Text, analyzer = "serbian", searchAnalyzer = "serbian")
    private String documentContent;

    @Column(name = "document_key")
    private String documentKey;

    @Transient
    @GeoPointField
    @Field(name = "geoPoint", store = true)
    private GeoPoint geoPoint;
}
```

Proširivanjem *ElasticsearchRepository* interfejsa obezbedićemo mogućnost pretraživanja indeksa slično kao i sa klasičnim JpaRepository interfejsom. Za predprocesiranje srpskog teksta biće iskorišćen ugrađeni ES analyzer.

2.2 Konfiguracija i integracija RustFs-a

Nakon skorašnjih promena MinIO projekta, koje uključuju prestajanje rada na open-source verziji (maintaincance mod), odlučeno je da se kao alternativa na ovom projektu koristi *the new kid on the block* - RustFs. S obzirom da RustFs takođe koristi S3 kompatibilan HTTP API integracija sa aplikacijom se neće razlikovati od one koja bi postojala sa MinIO servisom.

Glavna konfiguracija koja je potrebna prilikom komunikacije AWS Java SDK za S3 je overridovanje endopointa tako da usmerava ka RustFs kontejneru. Kao dodatna mera bezbednosti umesto kredencijala je moguće koristiti Access Keys, koji se generišu u RustFs konzoli. Dokumenti koji se uploaduju kroz aplikaciju će nakon parsiranja biti sačuvani u RustFs bucket, a ključ do dokumenta će biti sačuvan u PostgreSql bazi podataka. Na ovaj način je obezbeđeno da se do dokumenta može doći samo kroz aplikaciju koristeći identifikator iz PostgresSql baze.

2.3 Konfiguracija i integracija Logstash-a

Logstash će biti konfigurisan da obrađuje logove iz PostgreSql i RustFs Docker kontejnera korišćenjem *GELF* drajvera i *GELF* inputa u logstash konfiguraciji. Ovim logovima će se korišćenjem filtera dodeliti odgovarajući metapodaci koji će služiti da se logovi raspodele u odgovarajuće *ES* indekse. Logovi iz Java aplikacije biće rutirani direktno ka logstashu preko standardne tcp konekcije. Korišćenjem *GROK* filtera će biti izvršeno *on-the-fly* struktuiranje logova. Na primer: dat je jedan default spring boot log entry:

```
2026-01-25 13:01:23.456  INFO 12345 --- [nio-8080-exec-1] c.e.app.service.DocumentService    : Received document upload request for filename: invoice.pdf, size: 2.5 MB
```

Na osnovu poznate strukture (timestamp, log level, thread...) biće napisan GROK filter koji će od iz ovog loga izvući značajna polja i sačuvati ih u svoj event i nakon toga outputovati u ES indeks. Primer filtera:

```
grok {
  match => {
    "message" => "%{TIMESTAMP_ISO8601:log_timestamp} %{LOGLEVEL:log_level} %{NUMBER:process_id} --- \[%{DATA:thread_name}\] %{JAVACLASS:logger_name} %{GREEDYDATA:log_message}"
  }
}
```

Ovaj filter je moguće proširiti i custom regex šablonima ako želimo da izvučemo dodatne informacije iz log poruke. Na primer: sledećim šablonom možemo da extractujemo informaciju o id-u korisnika koji je invoke-ovao API:

```
pattern_definitions => { "user_pattern" => "User \\{%{NUMBER:user_id}\\} " }
```

Bitno je da je ovaj pattern potrebno pozvati pre GREEDYDATA šablona:

```
grok {
  match => {
    "message" => "%{TIMESTAMP_ISO8601:log_timestamp} %{LOGLEVEL:log_level} %{NUMBER:process_id} --- \[%{DATA:thread_name}\] %{JAVACLASS:logger_name} %{DATA:user_pattern} %{GREEDYDATA:log_message}"
  }
}
```

2.4 Konfiguracija i integracija Kibana-e

Kibana kontejner je konfigurisan tako da ima konekciju ka elasticsearch kontejneru. Iz Kibane će biti moguće vršiti vizualizacije podataka kao i pretraživanje logova sačuvanih u ES indeksima, a agregiranih od strane Logstasha.

3. Pretrage

3.1 Approximate KNN pretraga

Da bi bilo moguće izvršiti KNN pretragu u ES-u indeksirani podaci moraju biti "vektorisani", odnosno konvertovati tekst u vektore brojeva koji se matematički mogu procesirati. Razmatraće se dve opcije za ovaj zadatak:

1. Korišćenjem eland – python ES klijenta na osnovu uputstva iz ES dokumentacije. Ovo rešenje deluje robusno i oslanja se na ES ekosistem.
2. Korišćenjem custom python skripte, specifično korišćenjem biblioteke *sentance transformers* sa nekim predefinisanim modelom kao što je *paraphrase-multilingual-MiniLM-L12-v2*. Skripta će biti pozvana kao eksterni proces od strane Java servera ili će biti implementirana kao HTTP mikroservis.

Nakon vektorizacije teksta koja će se izvršiti tokom parsiranja dokumenta, rezultanti vektor će biti sačuvan u ES indeks nakon čega KNN pretraga postaje trivijalna.

3.2 Boolean semi-structured + Phraze Query pretraga

Parsiranjem inputa iz free search teksta potrebno je prepoznati boolean tokene: & (AND), | (OR) i ! (NOT). Reći ćemo da se svi tokeni čuvaju u Token klasi koja ima polje tip (boolean, term, phraze). Kroz ovako parsirane tokene se može iterirati i na osnovu tipa tokena imati određenu proceduru. Na primer: ukoliko je trenutni token ! (NOT), preuzmi sledeći token i na nativni ES query dodati "*field.notMatches(value)*" (pseudokod). Drugi način bi bio pretvoriti ovaj niz tokena u postfixnu notaciju i nakon toga kreirati ES upit. Prioritet za rešenje je jednostavna implementacija.

3.3 Geolokaciona pretraga

Na osnovu adrese parsirane iz dokumenta moguće je odrediti koordinate koje će kasnije biti sačuvane u ES indeksu kao tip polja GeoPoint. Proces dobijanja koordinata na osnovu adrese naziva se geokodiranje i postoji veliki broj online servisa koji pružaju ovu uslugu. https://geocode.maps.co/ je servis sa besplatnom pretplatom za koji se smatra da može da izvrši ovaj zadatak. Ukoliko se ovo ispostavi pogrešno biće iskorišćen drugi servis, ali suština ostaje ista. Na osnovu unete adrese ili grada će se geokodiranjem dobiti koordinate, i uz unetu distancu moguće je izvršiti *geo_distance* upit nad ES indeksom.