Fakultet tehničkih nauka
Univerzitet u Novom Sadu

Arhitekture sistema velikih skupova podataka

Elastic Stack

Student
Stefan Aleksić

Broj indeksa
E2 42/2022

13. januar 2023.

Sadržaj

Spisak izvornih kodova

Spisak slika

Glava 1

Uvod

Obrada velikih skupova podataka je grupa tehnika ili modela programiranja za pristup podacima velikih razmera kako bi se izdvojile korisne informacije za podršku procesu donošenja odluka [1].

Veliki skupovi podataka se obično čuvaju na hiljadama servera, tako da tradicionalni modeli programiranja kao što je interfejs za prosleđivanje poruka *(eng. message passing interface - MPI)* [2] ne mogu efikasno da rukovode njima. Stoga se novi modeli paralelnog programiranja koriste za poboljšanje performansi NoSQL baza podataka u centrima podataka. *MapReduce* [3] je jedan od najpopularnijih modela programiranja za obradu velikih skupova podataka korišćenjem velikih klastera. Glavna prednost ovog modela programiranja je jednostavnost, tako da korisnici mogu lako da ga iskoriste za obradu velikih podataka.

Hadoop [4] je implementacija *MapReduce* otvorenog koda i široko se koristi za obradu velikih podataka. *Hadoop* usvaja *HDFS* sistem datoteka [4]. Korišćenjem ovog sistema datoteka, podaci će biti locirani blizu čvora za obradu da bi se minimizirali troškovi komunikacije.

Spark [5], razvijen na Univerzitetu Kalifornije u Berkliju, predstavlja alternativu *Hadoop*-u, koji je dizajniran da prevaziđe ograničenja diska I/O i poboljša performanse ranijih sistema. Glavna karakteristika *Spark*-a koja ga čini jedinstvenim je njegova sposobnost da izvrši proračune u memoriji. Omogućava da se podaci keširaju u memoriji, čime se eliminišu *Hadoop*-ovo ograničenje diska za iterativne zadatke.

Pomenute tehnologije se uglavnom koriste za *batch* obradu podataka. Međutim, u

današnje vreme, ogromna količina podataka je generisana svake sekunde. Generisani podaci su većinom nestrukturne prirode i neophodno ih je obraditi i prilagoditi određenoj strukturi, pridodati im semantiku, obogatiti ih i slično, kako bi bilo lakše izvlačiti korisne informacije, odnosno kako bi se lakše vršila njihova pretraga. Baš u te svrhe se koristi alat koji je tema ovog rada, odnosno *Elastic stack* [6].

U drugom poglavlju rada biće opisane osnove *Elastic stack*-a, zajedno sa njegovim komponentama i njihovim specifikacijama. U okviru trećeg poglavlja, biće prikazan i objašnjen primer korišćenja jednog sistema koji se zasniva na alatu *Elastic stack*. Na kraju rada, u okviru poslednjeg poglavlja, biće izneti zaključci samog rada.

Glava 2

Elastic stack

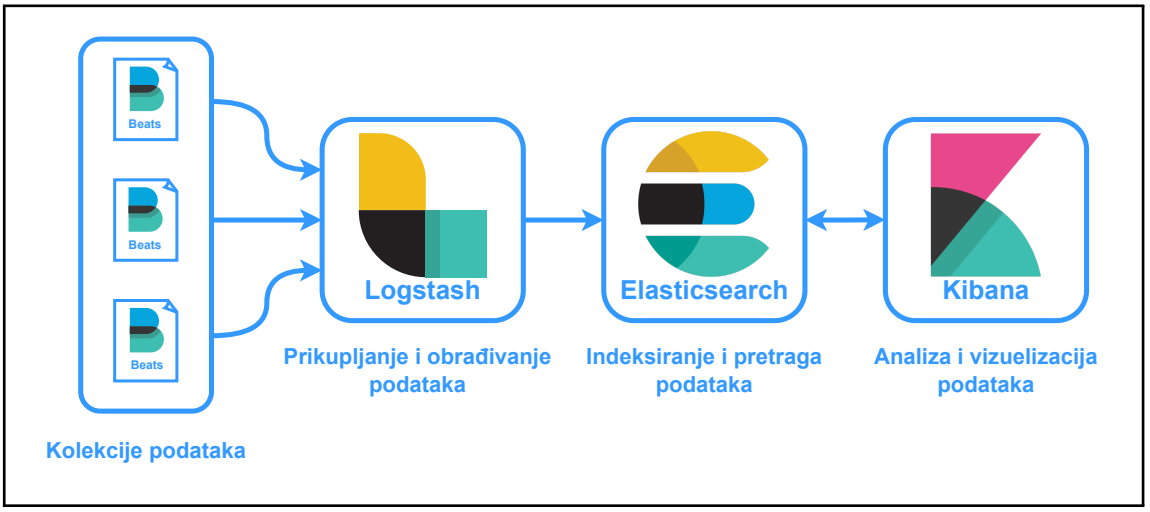
U ovom poglavlju biće opisan ***Elastic stack***, kao primer alata za obradu velikih skupova podataka u realnom vremenu. Bitno je napomenuti da se dalji tekst oslanja na dokumentaciju trenutno najnovije javno dostupne verzije alata 8.6 [6], odnosno najnovije verzije komponenti koje čine alat, a koje će biti istaknute pri obrađivanju svake od komponenti. U ovom poglavlju će, između ostalog, biti opisani delovi alata neophodni za implementaciju sistema koja će biti data u sledećem poglavlju kao primer korišćenja alata.

2.1 Osnove *Elastic stack*-a

Elastic stack, takođe poznat kao ***ELK stack***, predstavlja alat, koji postoji kao otvoreno rešenje *(eng. open source)* kompanije ***Elastic*** [7], a čije funkcionalnosti podrazumevaju: prikupljanje podataka u bilo kom formatu, njihovu obradu, nadogradnju, skladištenje, pretraživanje, analizu i vizuelizaciju u realnom vremenu. Akronim *ELK* identifikuje ključne komponente ovog steka, a to su: ***Elasticsearch*** [8], ***Logstash*** [9] i ***Kibana*** [10].

Apstrakcija toka podataka je prikazana na slici 2.1. Podaci mogu doći iz bilo kog izvora, što je na slici generalizovano komponentom *Beats* [11], koja predstavlja opcioni dodatak ***ELK stack***-a i služi za slanje operativnih podataka na ***Logstash***. ***Logstash*** komponenta predstavlja uređivač *(eng. formatter)* sistema, njena glavna funkcionalnost jeste prikupljanje ulaznih podataka bilo kog formata, parsiranje

podataka i slanje istih u željenom obliku na ***Elasticsearch***. ***Elasticsearch*** uz pomoć indeksa vrši skladištenje podataka i olakšava njihovo pretraživanje, takozvano pretraživanje celog teksta *(eng. full-text search)*. Tako indeksirani podaci se uz pomoć ***Kibana*** mogu vizualno prikazivati i ujedno se vršiti njihova analiza. Neke od najčešćih vrsta podataka koji se obrađuju uz pomoć ***ELK stack***-a podrazumevaju: evidentirane *(eng. logged)*, metričke, bezbednosne i podatke biznis logike.



Slika 2.1: *Apstrakcija toka podataka kroz* ***ELK stack***

2.2 *Elasticsearch*

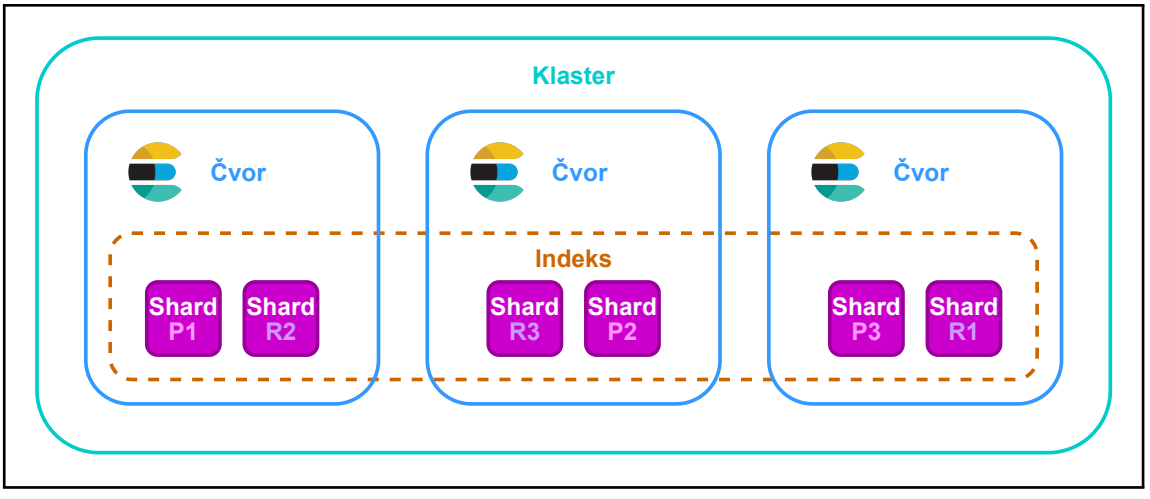
Elasticsearch je besplatan, distribuirani, otvoreni alat za skladištenje, pretragu i analizu *(eng. search and analytics engine)*, raznih tipova podataka, uključujući tekstualne, numeričke, geo-prostorne *(eng. geospatial)*, strukturne i nestrukturne [8]. Manipulacija podataka je ostvarena *REST API*-jem, a alat omogućava izvanrednu horizontalnu skalabilnost. ***Elasticsearch*** je zasnovan na ***Apache Lucene* biblioteci** [12], tj. ideji indeksiranja podataka na osnovu samog sadržaja. Podaci se skladište kao dokumenti, koji pripadaju jednom indeksu, dok je uz pomoć distribuiranog modela, indekse moguće podeliti na manje komponente, odnosno krhotine *(eng. shard)*, koji mogu biti rasprostranjeni kroz veći broj čvorova *(eng. node)*. Trenutno aktuelna verzija na kojoj se zasniva opis alata jeste verzija 8.6.

2.2.1 Osnove *Elasticsearch*-a

Elasticsearch arhitektura je projektovana za prikupljanje dokumenata, koji se skladište kao *JSON* objekti [13]. Alat podržava ugnježdene strukture, što olakšava obradu kompleksnih podataka i upita. Za praćenje informacija o dokumentima dodeljuju se specijalni atributi, odnosno meta-podaci, koji počinju donjom crtom. Važni meta-podaci su:

- **_index** – predstavlja kom indeksu dokument pripada. U analogiji, npr. sa relacionim bazama, ovo bi predstavljalo jednu bazu podataka.
- **_type** – predstavlja klasu, odnosno mapiranje koje bi trebalo da ima svaki dokument koji pripada istom tipu. Ovo je analogno tabeli u relacionim bazama podataka. Trenutna verzija alata ovo polje tretira kao zastarelo *(eng. deprecated)*, ostavljeno je samo radi kompatibilnosti sa starijim verzijama alata.
- **_id** – jedinstveni identifikator dokumenta u okviru tipa.
- **_version** – predstavlja broj izmena dokumenta, odnosno koliko puta je dokument bio kreiran, ažuriran, obrisan.

Glavni elementi arhitekture ***Elasticseach***-a su: **klaster**, **čvor**, **krhotina** i **analizator**, čija je organizacija predstavljena na slici 2.2.



Slika 2.2: *Arhitektura* ***Elasticsearch*** *alata*

Klaster predstavlja grupu čvorova koji skladište podatke. U okviru konfiguracionog fajla ***config/Elasticsearch.yml*** [1] se može precizirati broj čvorova koji bivaju startovani u klasteru, kao i fizička ili virtualna adresa svakog od čvorova. Čvorovi u okviru jednog klastera su logički povezani i mogu razmenjivati podatke jedni sa drugima. Pri pokretanju, sistem automatski kreira jedan klaster sa jednim čvorom u njemu, što je dovoljno za osnovne potrebe prosečnog korisnika.

Čvor ne predstavlja server, već jednu instancu ***Elasticsearch***-a, odnosno proces. Svaka instanca pripada jednom klasteru i zajedno, sa ostalim čvorovima u klasteru radi na rešavanju istog zadatka. Svaki čvor se može konfigurisati tako da obavlja barem jednu, a može obavljati više uloga u klasteru. Uloge koje čvoru mogu biti dodeljene obuhvataju:

- **Master čvor** – vrši kontrolu nad klasterom u vidu koordinacije čvorova koji pripadaju klasteru. Ovi čvorovi su odgovorni za operacije nad klasterom *(eng. clasterwide operations)*, poput kreiranja i brisanja indeksa.
- **Data čvor** – vrši skladištenje i odgovoran je za invertovani indeks podataka. Ovo je podrazumevana uloga čvora.
- **Klijentski čvor** – predstavlja raspoređivač opterećenja *(eng. load balancer)*[15] koji vrši usmeravanje pristiglih zahteva različitim čvorovima u klasteru.

Za ostvarivanje komunikacije se koriste dva glavna porta:

- **Port 9200** – koristi se za filtriranje zahteva koji dolaze izvan klastera. Ovo su zahtevi *REST API*-ja koji se koriste za izvršavanje upita, indeksiranje i slično.
- **Port 9300** – koristi se za među-čvornu *(eng. inter-node)* komunikaciju.

Krhotina predstavlja podskup dokumenata u okviru jednog indeksa. Naime, indeks nema ograničen broj dokumenata, niti ograničen memorijski kapacitet koji može da skladišti. Ukoliko se desi prekoračenje memorije na jednom od čvorova, ***Elasticsearch*** prestaje sa radom i javlja grešku. Kako bi se ovaj problem rešio, indeksi se dele na krhotine, koje označavaju skalabilnu jedinicu za indeksiranje i

[1] *YAML* format je dostupan u okviru [14]

omogućavaju distribuciju jednog indeksa na više čvorova. Ovim se postiže horizontalna skalabilnost sistema. Svaka krhotina funkcioniše kao nezavisan *Lucene* indeks [12], koji može biti skladišten bilo gde u klasteru.

Replika predstavlja kopiju krhotine indeksa, dok se originalna krhotina naziva primarnom *(eng. primary shard)*. Redundantnost podataka je glavni mehanizam na koji se sistemi otporni na greške oslanjaju [15]. S obzirom da se radi o distribuiranom sistemom, ukoliko bi neki od čvorova otkazao, krhotine indeksa koje sadrži bi bile nedostupne, a samim tim i svi dokumenti koji se nalaze u nedostupnim podskupovima. Iz tog razloga se formiraju replike i dodeljuju različitim čvorovima u sistemu. Pored otpornosti na otkaze, ovo omogućava i brže pretraživanje podataka, jer više čvorova koji sadrže istu kopiju krhotine indeksa mogu istovremeno da vrše njenu pretragu i time ubrzaju proces pretrage indeksa u celini. Broj replika nije ograničen i mogu se precizirati nakon kreiranja indeksa, međutim, treba voditi računa jer iako je ovim čitanje ubrzano, svaka modifikacija je time znatno složenija.

Analizatori imaju zadatak da vrše parsiranje fraza i izraza u konzistentne termine. Ovo se odvija u procesu indeksiranja. Svaki analizator je sačinjen od jednog tokenizatora *(eng. tokenizer)* [2] i većeg broja filtera tokena. Kada se naiđe na određeni izraz, tokenizator može da podeli izraz u prethodno definisane termine. U ovome se krije još jedna tajna brze pretrage podataka.

2.3 *Logstash*

Logstash je alat otvorenog koda za prikupljanje podataka sa mogućnostima nadovezivanja *(eng. pipelining)* u realnom vremenu. Alat može dinamičnim putem objediniti podatke iz različitih izvora, obogatiti ih, normalizovati i na kraju dostaviti podatke na unapred definisana odredišta. Ovim se pruža mogućnost prečišćavanja podataka za različite slučajeve upotrebe, naprednu analizu ili vizuelizaciju [9].

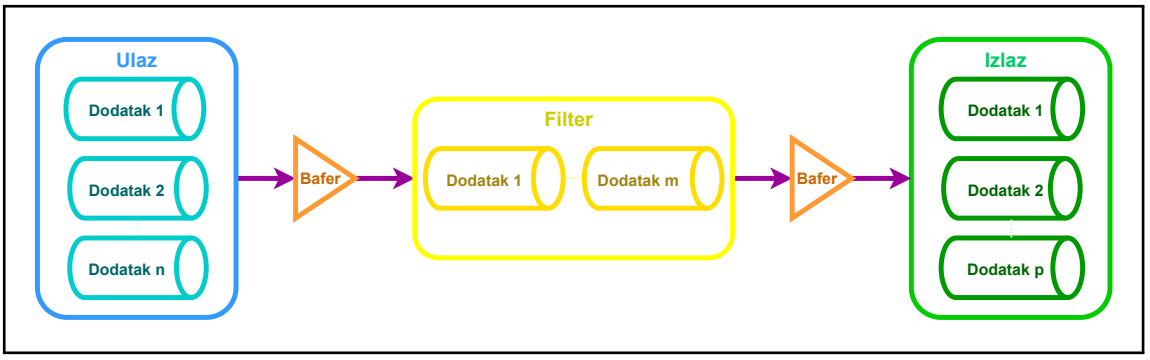
Na početku je ***Logstash*** bio korišćen za prikupljanje podataka evidencije *(eng. log)*, međutim, mogućnosti koje pruža su prevazišle taj slučaj korišćenja. Naime, bilo koja vrsta događaja može biti prosleđena alatu, uz pomoć širokog spektra ulaznih dodataka *(eng. input plugins)*, primljeni događaji mogu biti integrisani i transformisani uz pomoć dodataka za filtriranje *(eng. filter plugins)* i na kraju tako formatirani

[2] Tokenizacija je proces razgraničenja i moguće klasifikacije delova niza ulaznih znakova. Dobijeni tokeni se zatim prosleđuju nekom drugom obliku obrade. Proces se može smatrati podzadatkom raščlanjivanja ulaza.[16]

podaci mogu biti prosleđeni na razne vrste izlaza, uz pomoć izlazih dodataka *(eng. output plugins)*. Sam proces obrade je podržan velikim brojem često korišćenih kodera, a pritom je alat dizajniran za obradu velike količine podataka u realnom vremenu. Alat je razvijen u programskom jeziku JRuby i izvršava se na javinoj virtuelnoj mašini *(eng. Java Virtual Machine)*, skraćeno *JVM* [17].

2.3.1 Kako *Logstash* funkcioniše?

Logstash za sebe ima vezane cevovode, ili tokove podataka *(eng. pipeline)* definisane ulaznim, transformacionim i izlaznim etapama. Ulazi očitavaju događaje, filteri ih modifikuju, a izlazi tako modifikovane događaje dostavljaju na definisana odredišta. Ulazi i izlazi imaju unapred podržane kodeke [3], koji omogućavaju kodiranje i dekodiranje podataka pri ulazu i izlazu iz toka obrade, bez korišćenja transformacionih filtera. Primer jednog toka podataka prikazan je na slici 2.3.



Slika 2.3: *Primer* ***Logstash*** *toka podataka*

Svaki dodatak koji se koristi u okviru ulazne etape se izvršava u sopstvenoj niti i svaki od njih upisuje događaje u centralizovani red čekanja koji je ili u radnoj memoriji pokrenutog procesa, ili može biti na disku. Svaki radnik toka podataka *(eng. pipeline worker)* uzima blok događaja iz reda čekanja, sprovodi blok kroz definisane dodatke u okviru filter etape i na kraju šalje obrađene događaje na svaki od definisanih dodataka izlazne etape. Veličina bloka, kao i broj radnika toka podataka, odnosno niti, je podesiv kroz konfiguracione fajlove ***Logstash***-a.

Logstash podrazumevano koristi redove čekanja koji se nalaze u radnoj memo-

[3] Kodek *(eng. codec)* kompresuje ili dekompresuje bilo koji sadržaj u i iz bajtova.[18]

riji za baferovanje događaja između etapa, ulaz i filter etapa, odnosno filter i izlaz etapa, što je brže rešenje. Međutim pri nastanku greške, odnosno obustavi procesa, baferovani podaci bivaju izgubljeni.

Ulaznim dodacima se definiše izvor podataka koji ulaze u sistem. Najčešće korišćeni ulazni dodaci podrazumevaju:

- **file** – vrši čitanje fajla sa fajl sistema nalik *Unix* komandi `tail -0F`,
- **syslog** – osluškuje na podrazumevanom portu 514, koji predstavlja port za *syslog* poruke u *Unix*-u i parsira ih na osnovu RFC3164 formata [19],
- **redis** – vrši čitanje sa *Redis* servera, konkretno sa *Redis* kanala i listi. *Redis* [20] je često korišćen kao berzijanac poruka *(eng. message broker)* u centralizovanoj instalaciji ***Logstash***-a i služi za skladištenje događaja u okviru reda čekanja,
- **beats** – vrši obradu podataka koji se dobijaju uz pomoć *Beats* alata [11].

textbfFilter dodaci predstavljaju procesne uređaje koji posreduju tokom podataka. Filteri mogu biti kombinovani uslovima koji diriguju da li se određena faza obrade izvršava ili ne, u zavisnosti od kriterijuma koji se ispituje. Najčešće korišćeni dodaci za filtriranje su:

- **grok** – parsira i gradi proizvoljan tekst. Ovaj dodatak je trenutno najpogodnije rešenje za parsiranje nestrukturnih tokova podataka u podatke koji imaju strukturu i moguće je nad njima izvršavati upite,
- **mutate** – izvršava generalne transformacije nad poljima događaja, poput: preimenovanja, uklanjanja, zamene, modifikacije i slično,
- **drop** – izvršava obustavu događaja u potpunosti. Korisno pri događajima koji imaju debug nivo ozbiljnosti,
- **clone** – vrši replikaciju događaja, sa mogućnošću za dalje dodavanje i uklanjanje pojedinih polja,
- **geoip** – nadograđuje *IP* adresu geo-prostornim koordinatama, kao i dostupnim informacijama za iste.

Izlazni dodaci predstavljaju destinacije konačne etape u izvršavanju toka podataka ***Logstash***-a. Događaj može proći kroz više izlaza, međutim, njegova obrada

je završena kada prođe kroz sve navedene izlaze. Najčešće korišćeni dodaci za izlaz podrazumevaju:

- **elasticsearch** – šalje događaj na *Elasticsearch*, samim tim se vrši indeksiranje podataka koji dalje mogu lako da se pretražuju i vizualizuju u *Kibani*.
- **stdout** – ispisuje događaj na standardni izlaz.
- **file** – skladišti formatiran događaj na fajl u fajl sistemu.
- **graphite** – šalje događaj na *graphite*, koji predstavlja popularan alat otvorenog koda za skladištenje podataka metrike. [21]
- **statsd** – šalje podatke na *statsd*. Ovo je pozadinski servis *(eng. deamon)*, koji se izvršava na *Node.js* platformi, a čija je funkcija osluškivanje podataka statistike poslate *UDP* transportnim protokolom i prosleđivanje primljenih podataka na jedan ili više priključivih *(eng. pluggable)* pozadinskih *(eng. backend)* servisa, poput gore navedenog *graphite* servisa. [22]

2.3.2 *Logstash* konfiguracioni fajlovi

Logstash sadrži dve vrste konfiguracionih fajlova, a to su:

- **Pipeline konfiguracioni fajlovi**, koji definišu sam tok podataka. Izgled ovih fajlova će biti detaljnije objašnjen u nastavku.
- **Settings konfiguracioni fajlovi**, koji definišu inicijalizaciju i tok rada samog ***Logstash*** procesa. Ovi fajlovi su generisani pri instaliranju samog ***Logstash*** servisa i podrazumevaju fajlove:
  - **Logstash.yml** – sadrži konfiguracione indikatore *(eng. flags)* za instancu programa. Ovde se mogu definisati: tip i lokacija bafera koji se koristi za red čekanja između etapa, veličina bloka događaja koju obrađuje jedan radnik, nivo evidentiranja *(eng. logging level)* i mnoga druga podešavanja.
  - **Pipelines.yml** – sadrži konfiguraciju većeg broja tokova podataka koji će se izvršavati u okviru jedne instance ***Logstash***-a. Ovde se između ostalog navodi putanja do same konfiguracije toka podataka, kao i broj radnika u okviru toka, identifikator toka i slično.

- **Jvm.options** – sadrži podešavanja za *JVM* indikatore. Ovde se između ostalog može definisati minimalna i maksimalna veličina radne memorije dodeljene procesu.
- **Log4j2.properties** – sadrži podrazumevana podešavanja za *log4j* biblioteku [23].

Struktura konfiguracionih fajlova

Logstash konfiguracioni fajlovi za tokove podataka su pisani u specijalnom jeziku [24] koji je razvijen od strane samih osnivača programa. Jezik podseća na *JSON* format sa par izuzetaka, ali je vrlo jednostavan za razumevanje. Tok podataka, *pipeline* ima odvojene segmente za ulaznu, filter i izlaznu etapu, čija je oblast važenja uokvirena vitičastim zagradama. U okviru svake od etapa se mogu navesti dodaci koji se koriste. Bitno je naglasiti da se dodaci u okviru ulazne etape izvršavaju konkurentno, dok se u filter i izlaznoj etapi izvršavaju sekvencijalno.

Primer jednog konfiguracionog fajla je dat u okviru slike 1.

```
input {
    http {
        port => 3333
        tags => gateway
    }
}
filter {
    . . .
}
output {
    . . .
}
```

Izvorni kod 1: *Primer konfiguracije toka podataka*

Dodaci mogu zahtevati vrednosti za određena podešavanja. Tipovi vrednosti koje postoje u okviru konfiguracije su:

- **Lista** – definiše se uglastim zagradama, dok su elementi unutar zagrada odvojeni zarezima,
- **Logička vrednost** – definiše se vrednostima `true` i `false`,
- **Bajt** – definiše se kao string koji sadrži broj bajtova sa mernom jedinicom, bilo da je u pitanju SI (osnova 1000), ili binarna (osnova 1024) merna jedinica. Polje nije osetljivo na velika slova *(eng. case-insensitive)* i prepoznaje razmak između vrednosti i jedinice. Ukoliko se ne napiše jedinica, već samo celobrojna vrednost, podrazumeva se da vrednost predstavlja egzaktan broj bajtova,
- **Kodek** – definiše ime jednog od ***Logstash*** kodeka, koji se može naznačiti i u ulaznim i u izlaznim etapama. Kada se nađe u ulaznoj etapi, predstavlja način dekodiranja podataka koji ulaze u tok, dok na izlazu predstavljaju način kodiranja u određeni format,
- **Heš mapa** – definiše kolekciju ključ-vrednost elemenata u formatu *"polje" => "vrednost"*, gde se elementi razdvajaju blanko znacima, ne zapetama kao kod listi i nizova,
- **Broj** – obuhvata realne brojeve, odnosno *integer* i *float* vrednosti,
- **Lozinka** – string vrednost koja ne biva evidentirana,
- **URI** – string vrednost koja predstavlja identifikator resursa, može biti potpun ili parcijalni *URL*. Ukoliko sadrži korisničko ime i lozinku, takođe se neće evidentirati,
- **Putanja** – definiše validnu putanju do resursa na sistemu,
- **String** – sekvenca karaktera, ukoliko sadrži blanko znake, neophodno je ograditi ga jednostrukim ili dvostrukim znakom navoda,
- **Referenca na polje** – predstavlja putanju do polja u događaju, ukoliko se radi o polju koje je u korenu događaja, onda se može koristiti notacija `[polje]` ili se izostaviti uglaste zagrade, polje. Ukoliko se referencira ugnježđeno polje, onda je neophodno koristiti notaciju sa uglastim zagradama i to po ugledu na oblik `[polje prvog nivoa][polje drugog nivoa]...[polje n-tog nivoa]`.
- **Komentari** se označavaju kao u programskim jezicima *Perl*, *Ruby* i *Python*, odnosno počinju znakom taraba (#).

Uslovne strukture *(eng. conditionals)* su podržane, imaju poznatu strukturu kao i u ostalim programskim jezicima `if`, `else if`, `else`, a koriste se za dodatno

kontrolisanje filter i izlazne etape toka podataka. Primer strukture je dat u nastavku 2.

```
if EXPRESSION {
    ...
} else if EXPRESSION {
    ...
} else {
    ...
}
```

Izvorni kod 2: *Uslovne strukture u konfiguracionom fajlu toka podataka*

Izraz *(eng. expression)* predstavlja logički izraz koji se svodi na jednu od logičkih vrednosti `true`, ili `false`. Ukoliko izraz sadrži referencu na polje, vrednost izraza će biti `false` ukoliko: polje ne postoji, polje ima nedefinisanu vrednost `null` ili postoji i ima vrednost `false`. Podržani operatori u okviru jezika podrazumevaju:

- Operatore poređenja:
  - Jednakost: ==, **!**=, <, >, <=, >=
  - Regularni izraz: = , **!**
  - Provera sadržaja: **in**, **not in**
- Logičke operatore: **and**, **or**, **nand**, **xor**
- Unarni operator: **!**

Konkretan primer uslovne strukture sa izrazima je dat na slici 3.

```
filter {
    if [fooVal] in ["test", "debug"] and ([logLevel] != "debug") {
        mutate { add_tag => "testing" }
    }
    if [foo] =~ "[0-9]+" {
        mutate { add_tag => "contains numbers" }
    }
    if "%{+HH}" < "16" {
      mutate { add_tag => "Before 16h" }
    }
}
output {
    if "testing" not in [tags] {
        elasticsearch{
            ...
        }
    }
}
```

Izvorni kod 3: *Primer izraza u okviru uslovnih struktura*

Svakom događaju koji se generiše u ulaznoj etapi obrade se pridružuju polja koja bliže opisuju događaj. Ova polja su rezervisana, njihov tip zavisi od implementacije samog ulaznog dodatka koji ih generiše, a namenjeni su boljoj kontroli toka obrade:

- **@metadata** – heš mapa namenjena za skladištenje pomoćnih podataka u toku obrade. Ovo polje je dostupno u filter i izlaznoj etapi, međutim, podrazumevano se ne šalje na izlazne dodatke.
- **@timestamp** – predstavlja vremenski trenutak u kom je generisan događaj. Iz ovog polja se čitaju podaci koje koristi *sprintf* format, koji omogućava referenciranje vrednosti iz ostalih polja događaja.
- **@version** – predstavlja verziju dodatka koji je generisao događaj.
- **tags** – niz vrednosti koji se koriste radi bližeg opisivanja događaja.

2.3.3 Dodaci toka obrade

Dodaci toka obrade *(eng. pipeline plugins)* su programi napisani u programskom jeziku *Java* ili *Ruby*, koji predstavljaju alat za olakšavanje kontrole samog toka

obrade, od ulaza, preko obrade, do izlaza samih podataka. Ove programe može napisati bilo ko u programskom jeziku *Java* ili *Ruby* i publikovati ih na javni *Github* repozitorijum sa kog ostali korisnici mogu da ih povlače i koriste u svojim sistemima. Samim tim, ***Logstash*** ima jako veliki skup dodataka koji mogu biti korišćeni, dok će ovaj rad nadalje obraditi samo one koji su neophodni za razumevanje sistema koji se implementira.

TCP ulazni dodatak [25] vrši čitanje poruka poslatih preko mreže, tako što osluškuje poruke TCP protokola, generišući odgovarajuće događaje za tok podataka u kom je konfigurisan.

Grok filter dodatak [26] vrši parsiranje proizvoljnog teksta i njegovo prevođenje u strukturni podatak, nad kojim se mogu pisati upiti. Funkcioniše tako što koristi regularne izraze za pronalaženje šablona u okviru tekstualnih polja. Sintaksa koju prati ovaj dodatak je **%SINTAKSA:SEMANTIKA**. Sintaksa predstavlja ime unapred definisanog šablona regularnog izraza koji se koristi za identifikovanje dela teksta, dok semantika predstavlja identifikator pronađenog teksta koji se dalje može koristiti u toku podataka. Pored unapred definisanih šablona, mogu se direktno pisati regularni izrazi za pronalaženje šablona. Sintaksa regularnih izraza koja se koristi je *Onigurama* [27], koja ima izgled *(?<identifikator polja>šablon)*. Još jedan način jeste kreiranje sopstvenih šablona u odvojenim fajlovima (putanje do fajlova se definišu u konfiguraciji dodatka), dok novo-definisani šablon ima izgled *IME_ŠABLONA ŠABLON*.

Mutate filter dodatak [28] omogućava generalizaciju mutacija nad poljima događaja. Moguće su operacije: preimenovanja, zamene, modifikacije polja u okviru događaja i slično. Redosled operacija u okviru dodatka je unapred ustanovljen, odnosno ne prati redosled navođenja operacija u okviru konfiguracije. Ukoliko je neophodno izvršiti više operacija koje ne prate ovakav redosled, neophodno je definisati više *mutate* filter dodataka u odgovarajućem redosledu.

Geoip filter dodatak [29] nadograđuje događaje informacijama o geo-prostornoj lokaciji prosleđene *IP* adrese. Ovaj dodatak je zasnovan na podacima iz *MaxMind GeoLite2* baze podataka [30], koja dolazi kao podrazumevana baza dodatka. Moguće je kroz opcije konfigurisati bazu podataka, odnosno promeniti podrazumevanu. Primer odgovora koji se dobija za prosleđenu *IP* adresu dat je u okviru slike 4.

```
{
    "ip": "12.34.56.78",
    "geo": {
        "city_name": "Seattle",
        "country_name": "United States",
        "cotinent_code": "NA",
        "continent_name": "North America",
        "country_iso_code": "US",
        "postal_code": 98106,
        "region_name": "Washington",
        "region_code": "WA",
        "region_iso_code": "US-WA",
        "timezone": "America/Los_Angeles",
        "location": {
            "lat": 47.6062,
            "lon": -122.3321,
        }
    },
    "domain": "example.com",
    "asn": {
        "number": 98765,
        "organization": {
            "name": "Elastic, NV"
        }
    },
    "mmdb": {
        "isp": "InterLi Supra LLC",
        "dma_code": 819,
        "organization": "Elastic, NV"
    }
}
```

Izvorni kod 4: *Primer podataka koje pruža geoip filter dodatak za prosleđenu IP adresu*

Http filter dodatak [31] omogućava integraciju sa eksternim *Web* servisima ili *REST API*-jima, tako što osposobljava ***Logstash*** da šalje zahteve podesive strukture na odgovarajuće krajnje tačke *(eng. endpoint)*, kako bi obogatio događaj koji se

obrađuje dodatnim informacijama. Parametar koji definiše zaglavlja zahteva *(eng. headers)* koji se šalju bivaju definisana u okviru **@metadata** polja, kako ne bi okupirali destinacije na izlazu. Ovo je vrlo podesiv dodatak i ima veliki broj opcija za konfiguraciju.

Dns filter dodatak [32] pruža pretragu stavki, kanoničkih imena *(eng. CNAME)*, ili stavki adresa *(eng. A record)*, odnosno ukoliko je u pitanju inverzni *DNS*, pretražuju se pointer stavke *(eng. pointer record, PTR)*. Bitno je napomenuti da ovaj dodatak može da obradi samo jednu stavku, dakle za veći broj stavki je potrebno više puta iskoristiti ovaj dodatak, s tim što ovakvo podešavanje može drastično da uspori sistem.

Elasticseach izlazni dodatak [33] omogućava skladištenje vremenskih serija podataka, poput podataka evidencije, događaja i metrike, kao i podataka čije skladištenje nije bazirano na vremenskim intervalima, direktno u *Elasticsearch*. S obzirom da se radi o dodatku koji radi sa osnovnim proizvodom kompanije, ovaj dodatak je jako fleksibilan i može kontrolisati razne segmente vezane za operacije u *Elasticsearch*-u. Najbitnije opcije u okviru dodatka, koje se vezuju za temu ovog rada podrazumevaju:

- **action** – enumeracija(*index, delete, create, update*) koja definiše koja vrsta operacije biva izvršavana. Podrazumevana vrednost za vremenske serije podataka, odnosno tokove podataka je *create*, dok je *index* podrazumevana vrednost za podatke koji se ne vezuju za vremenske intervale.
- **index** – string koji predstavlja indeks u koji se upisuju podaci obrađeni u okviru toka podataka
- **hosts** – uri tip vrednosti, koji predstavlja jednu ili više adresa čvorova na kojima je pokrenuta instanca *Elasticsearch*-a. Podrazumevana vrednost je `[//127.0.0.1]`.

2.4 *Kibana*

Kibana je aplikacija otvorenog koda koja pruža pregledan korisnički interfejs ka *Elastic stack*-u, uz mogućnosti pretraživanja, vizuelizacije i analize podataka indeksiranih *Elasticsearch*-om. Takođe poznata i kao alat za kreiranje raznovrsnih grafikona *(eng. charts)*, ***Kibana*** omogućava nadgledanje, upravljanje i održavanje

bezbednosti klastera u *Elastic stack*-u. Prvi put je postala dostupna javnosti 2013. godine, dok je trenutna aktuelna verzija 8.6, na osnovu koje će biti opisani pojedini detalji same aplikacije [10].

Glavna primena ***Kibana***-e podrazumeva: pretraživanje, vizualizaciju indeksiranih podataka u okviru *Elasticsearch*-a, kao i analizu podataka kroz kreiranje grafikona, poput: poluga, pita, tabela, histograma i mapa. Komandna tabla *(eng. dashboard)* kombinuje ove elemente na jedan pano i time omogućava analitički pregled podataka u realnom vremenu za razne slučajeve korišćenja, poput:

1. analize podataka evidencije,
2. nadgledanja metričkih podataka infrastrukture i kontejnera,
3. vizuelizaciju geo-prostornih podataka,
4. analizu sigurnosnih podataka,
5. analizu podataka biznis logike.

2.4.1 Upitni jezik *Kibana*-e

Kibana koristi specifični upitni jezik nazvan ***Kibana Query Language*** [34], ili skraćeno ***KQL***, kojim je omogućeno jednostavno filtriranje podataka u svrhu vizualizacije. Ovaj jezik se razlikuje od standardnog *Lucene* upitnog jezika, jer ne omogućava pretragu uz pomoć regularnih izraza ili takozvanu pomućenu *(eng. fuzzy)* pretragu podataka, međutim omogućena je pretraga ugnježdenih polja kao i takozvanih skriptovanih polja *(eng. scripted fields)*.

U okviru jezika se mogu identifikovati podvrste upitnog jezika: **upiti termina**, **upiti Bulove algebre**, **opsežni upiti**, **upiti koji koriste džokere** i **upiti nad ugnježdenim poljima**.

Upiti termina *(eng. terms query)*, koji koriste egzaktan stil pretrage. Sintaksa ovog upita ima oblik *<putanja do polja> : <list termina>*, gde putanja do polja predstavlja ugnježdena polja počevši od korena dokumenta, sve do polja koje se pretražuje, nivoi su razdvojeni tačkom. Lista termina predstavlja prihvatljive vrednosti u okviru navedenog polja, dok se različiti termini odvajaju razmakom, a ukoliko se

pretražuju egzaktne fraze, koriste se znaci navođenja. Putanja do polja i lista termina se odvajaju specijalnim karakterom dvotačka, na koji može da se gleda kao na operator *in*.

Upiti Bulove algebre *(eng. boolean queries)* predstavljaju kombinovanje prethodno navedene podvrste upita logičkim operatorima: **or**, **and** i **not**. Po pravilu, **and** ima viši prioritet od operatora **or**, ukoliko je neophodna drugačija logika, koriste se zagrade.

Opsežni upiti *(eng. range queries)* predstavljaju upite koji numeričke i vrednosti datuma navedenih polja porede operatorima jednakosti: >, >=, < i <=. Moguće je koristiti i matematičke izraze pri poređenju vrednosti, što je jako pogodno za datume.

Upiti koji koriste džokere *(eng. wildcard)*, koji je označen znakom * i može biti ili u delu putanje polja, čime se pokriva veći broj polja, ili u okviru vrednosti koje se traže u poljima i time pokriva veći spektar vrednosti, jer menja odgovarajući deo bilo kojom sekvencom bilo koje dužine.

Upiti nad ugnježdenim poljima *(eng. nested field queries)* omogućavaju dva pristupa u filtriranju ugnježdenih dokumenata:

- Filtriranje jednog dokumenta na osnovu određenih delova upita
  <polje>:{ <ugnježdeno polje 1>:<izraz 1> and <ugnježdeno polje 2>:<izraz 2> }
- Filtriranje više dokumenata na osnovu određenih delova upita
  <polje>:{ <ugnježdeno polje 1>:<izraz 1> } and <polje>:{ <ugnježdeno polje 2>:<izraz 2> }

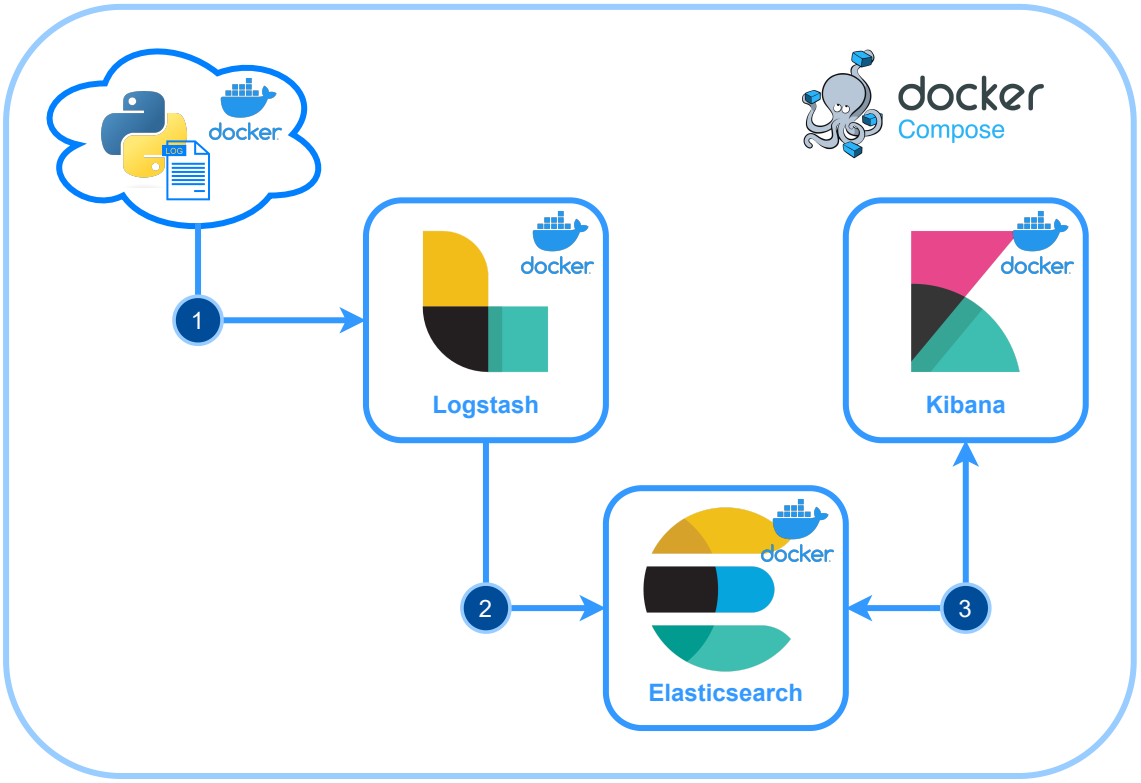
Glava 3

Primer korišćenja *Elastic stack*-a

U ovom poglavlju će biti opisana implementacija sistema koji je iskorišćen kao jedan od mnogih primera korišćenja ***Elastic stack*** alata. Sistem koji je implementiran ima ulogu obrađivanja podataka mrežnog saobraćaja u realnom vremenu, kroz njihovo parsiranje, transfomaciju, nadogradnju, skladištenje i vizuelizaciju. Kompletna implementacija je ostvarena korišćenjem odgovarajućih komponenti ***Elastic stack***-a i kontejnarizovana u okviru ***Docker*** [35] kontejnera uz pomoć ***Docker compose*** alata [36].

3.1 Arhitektura implementiranog sistema

Kompletna arhitektura sistema je prikazana na dijagramu 3.1. U okviru sistema se mogu prepoznati već opisane komponente iz prethodnog poglavlja.



Slika 3.1: *Dijagram arhitekture sistema*

Podaci o mrežnom saobraćaju su simulirani kratkom ***Python*** skriptom koja čita podatke iz dataset-a [37], povezuje se na ***Logstash*** ulazni dodatak uz pomoć *TCP* konekcije i šalje sadržaj u određenim vremenskim intervalima. Ovim se ostvaruje unošenje sirovih podataka za obrađivanje u realnom vremenu, što je označeno kao **korak 1** na dijagramu. ***Logstash*** komponenta je zadužena za prihvatanje sirovih podataka u sistem. U okviru pokrenutog toka podataka, ***Logstash*** parsira sirove podatke, unosi u njih semantiku, vrši njihovo obogaćivanje i tako obrađene podatke šalje na ***Elasticsearch***, što je označeno kao **korak 2** na dijagramu.

Komponenta ***Elasticsearch*** je zadužena za indeksiranje, skladištenje i pretraživanje podataka na osnovu polja koja je ***Logstash*** kreirao. Ovi podaci su dostupni ***Kibana*** komponenti, koja može da na osnovu njih kreira vizualnu reprezentaciju iz koje se mogu izvući nova znanja, odnosno zaključci. Komunikacija između alata ***Kibana*** i ***Elasticsearch*** je dualna, označena kao **korak 3** na dijagramu, s obzirom da se dijagrami koji se kreiraju u okviru alata ***Kibana*** skladište u okviru

Elasticsearch instance.

Uzeći u obzir performanse računara na kome je pokrenut ovaj sistem, nijedna komponenta nije replicirana. Svaka komponenta ima jednu instancu koja je pokrenuta u svom kontejneru i komunikacija se odvija isključivo direktnom mrežnom komunikacijom, odnosno ne postoje balanseri opterećenja. U još realnim primeru, instance bi bile replicirane i pokrenute sa mnogo više alocirane memorije u okviru *JVM*-e, kao i činjenica da bi postojali balanseri opterećenja za različite instance ***Logstash*** komponente.

3.2 Konfiguracioni fajlovi sistema

Svaka od komponenti sistema je pokrenuta u okviru sopstvenog ***Docker*** kontejnera, a za pokretanje sistema se koristi ***Docker compose*** alat, te je celokupan sistem moguće konfigurisati u okviru par fajlova i pokrenuti samo jednom komandom `docker-compose up`. Sadržaj ***docker-compose.yml*** fajla se nalazi na isečku 5.

Sa isečka 5, u okviru linija 4-27 možemo primetiti servis pod nazivom ***setup***, koji je odgovoran za konfigurisanje određenih parametara instanci ***ELK stack***-a. Ovaj servis se pokreće nakon što je servis ***Elasticsearch*** podignut i izvršava ***setup*** skriptu. ***Elasticsearch*** servis je konfigurisan u okviru linija 29-47. Za ovaj servis su otvoreni portovi 9200 i 9300 čija funkcija je upisana u sekciji 2.2.1. Parametri *JVM*-e su prosleđeni na liniji 42 i to sa minimalno neophodnom memorijom od 512MB.

Logstash servis je konfigurisan u okviru linija 49-71 i to sa otvorenim portovima 5044, 50000 i 9600. Port 5044 se koristi za **Beats** ulazni dodatak, dok je port 50000 otvoren za *TCP* i *UDP* ulazne dodatke. Port 9600 se koristi za komunikaciju sa ***ELK stack***-om. Ova komponenta takođe ima minimalne memorijske parametre za izvršavanje u *JVM*-i. U okviru linija 73-88 je konfigurisan servis ***Kibana***, koji je pokrenut na portu 5601, što je standardni port za ovu komponentu.

Konfiguracija svih komponenti, osim ***Logstash***-a je standardna konfiguracija koju preporučuje kompanija Elastic u svojoj dokumentaciji, odnosno dostupna na github repozirotijumu [38].

Logstash konfiguracija je izmenjena kako bi se prilagodila ovom primeru. Odnosno tok podataka je konfigurisan na sledeći način (isečak 6).

```yaml
version: '3.7'

services:
  setup:
    build:
      context: setup/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    init: true
    volumes:
      - ./setup/entrypoint.sh:/entrypoint.sh:ro,Z
      - ./setup/helpers.sh:/helpers.sh:ro,Z
      - ./setup/roles:/roles:ro,Z
      - setup:/state:Z
    environment:
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
      LOGSTASH_INTERNAL_PASSWORD: ${LOGSTASH_INTERNAL_PASSWORD:-}
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
      METRICBEAT_INTERNAL_PASSWORD: ${METRICBEAT_INTERNAL_PASSWORD:-}
      FILEBEAT_INTERNAL_PASSWORD: ${FILEBEAT_INTERNAL_PASSWORD:-}
      HEARTBEAT_INTERNAL_PASSWORD: ${HEARTBEAT_INTERNAL_PASSWORD:-}
      MONITORING_INTERNAL_PASSWORD: ${MONITORING_INTERNAL_PASSWORD:-}
      BEATS_SYSTEM_PASSWORD: ${BEATS_SYSTEM_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch

  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml:ro,Z
      - elasticsearch:/usr/share/elasticsearch/data:Z
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      node.name: elasticsearch
      ES_JAVA_OPTS: -Xms512m -Xmx512m
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
      discovery.type: single-node
    networks:
      - elk
    restart: unless-stopped

  logstash:
    build:
      context: logstash/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro,Z
      - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
    ports:
      - 5044:5044
      - 50000:50000/tcp
      - 50000:50000/udp
      - 9600:9600
    environment:
      LS_JAVA_OPTS: -Xms256m -Xmx256m
      LOGSTASH_INTERNAL_PASSWORD: ${LOGSTASH_INTERNAL_PASSWORD:-}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-changeme}
      ELASTIC_USERNAME: ${ELASTIC_USERNAME:-elastic}
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      - 5601:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
    networks:
      - elk
    depends_on:
      - elasticsearch
    restart: unless-stopped

networks:
  elk:
    driver: bridge

volumes:
  setup:
  elasticsearch:
```

Izvorni kod 5: *Konfiguracija* ***docker-compose.yml*** *fajla*

```
input {
        tcp {
                port => 50000
                codec => plain
        }
}

filter {
  grok {
    match => { "message" => "(?<[flow][id]>[0-9\.\-]*)(?:[\s\,]*%{IP:[source][ip]})(?:[\s\,]*%{INT:[source][port]})
    ↪  (?:[\s\,]*%{IP:[destination][ip]})(?:[\s\,]*%{INT:[destination][port]})(?:[\s\,]*%{INT:[protocol]})
    ↪  (?:[\s\,]*%{INT:[timestamp][day]})(?:[\s\/]*%{INT:[timestamp][month]})(?:[\s\/]*(?<[timestamp][year]>[0-9]{4}))
    ↪  (?:[\s]*%{INT:[timestamp][hour]})(?:[\s\:]*%{INT:[timestamp][minute]})(?:[\s\:]*%{INT:[timestamp][second]})" }
        tag_on_failure => "parsing_data_failed"
        tag_on_timeout => "parsing_data_timeout"
        add_field => {
                gen => "network-traffic"
        }
        target => "parsed_data"
  }

  if [gen] == "network-traffic" {
                http {
                        url => "https://api.blocklist.de/api.php?ip=%{[parsed_data][source][ip]}&start=1&format=json"
                        target_body => "[parsed_data][source][reputation]"
                }
                json{
                        source => "[parsed_data][source][reputation]"
                        target => "[parsed_data][source][reputation]"
                }

                http {
                        url => "https://api.blocklist.de/api.php?ip=%{[parsed_data][destination][ip]}&start=1&format=json"
                        target_body => "[parsed_data][destination][reputation]"
                }
                json{
                        source=> "[parsed_data][destination][reputation]"
                        target => "[parsed_data][destination][reputation]"
                }

                geoip {
                        source => "[parsed_data][source][ip]"
                        target => "[parsed_data][source][geoip]"
                }

                geoip {
                        source => "[parsed_data][destination][ip]"
                        target => "[parsed_data][destination][geoip]"
                }

                if [parsed_data][source][geoip][ip] and [parsed_data][source][geoip][ip] != "" {
                        mutate {
                                add_field => {
                                        "[parsed_data][source][reverse_dns]" => "%{[parsed_data][source][geoip][ip]}"
                                }
                                add_tag => "geoip_src_success"
                        }
                }

                if [parsed_data][destination][geoip][ip] and [parsed_data][destination][geoip][ip] != "" {
                        mutate {
                                add_field => {
                                        "[parsed_data][destination][reverse_dns]" => "%{[parsed_data][destination][geoip][ip]}"
                                }
                                add_tag => "geoip_dst_success"
                        }
                }

                if "geoip_src_success" in [tags]{
                        dns {
                                reverse => ["[parsed_data][source][reverse_dns]"]
                                action => "replace"
                        }
                }

                if "geoip_dst_success" in [tags]{
                        dns {
                                reverse => ["[parsed_data][destination][reverse_dns]"]
                                action => "replace"
                        }
                }
    }
}

output {
        if [gen] == "network-traffic" {
                elasticsearch {
                       hosts => ["elasticsearch:9200"]
                        user => "elastic"
                        password => "changeme"
                        index => "network-traffic-%{+YYYY.MM}"
                }
    }
        stdout {
                codec => rubydebug
        }
}
```

Izvorni kod 6: *Tok podataka* ***Logstash.conf***

Ulazna etapa toka podataka je konfigurisana u okviru linija 1-6. U okviru nje je definisan jedna ulazni dodatak, odnosno *TCP*, koji osluškuje na portu 5000 i koristi ***plain*** kodek za dekodiranje ulaznih podataka. Ovaj kodek podržava tekstualne podatke, odnosno vrši konverziju iz bajtova u string i podrazumevano koristi *utf-8* format.

Obrađivanje podataka se odvija u okviru filter etape definisane na linijama 8--80. S obzirom da se dodaci filter etape izvršavaju sekvencijalno, redosledom kojim su napisani, prvi koji se izvršava jeste ***grok filter dodatak***. U okviru njega se iz polja *"message"*, koje predstavlja sadržaj ulazne etape, vrši parsiranje delova poruke i njihov upis se vrši u odgovarajuća polja pridružena šablonu. Takođe, standardna konfiguracija koju ima svaki dodatak jeste mogućnost ubacivanja tagova, odnosno polja, kao i *target* polje koje precizira mesto u okviru događaja, koji se trenutno obrađuje, gde će se smestiti rezultat dodatka. U okviru linije 19 je postavljen uslov da ukoliko se na polju *gen* događaja, koji se obrađuje, ne nalazi vrednost *network--traffic*, filter etapa se završava.

Za događaje koji su označeni kao *network-traffic*, sledi obrada u okviru linija 20--78. S obzirom da su navedeni dodaci već objašnjeni, trebalo bi da je jasno šta se u pozadini dešava sa događajem. U okviru linija 20-23 se izvršava ***http filter dodatak***, koji prosleđuje *IP* adresu izvorišta kako bi proverilo njenu reputaciju, koristeći javni *API* dostupan na *endpoint*-u u okviru linije 21. Rezultat se smešta u odgovarajuće ugnježdeno polje događaja. S obzirom da se upisuje string vrednost u polje, u okviru linija 24-27 se ovo polje parsira i na njegovo mesto dolazi *JSON*, odnosno vrednost tipa *hash*, koje sadrži odgovarajuća polja. Identična logika se izvršava i za odredišnu *IP* adresu, u okviru linija 29-36.

Na linijama 38-41 i 43-45 se pomoću ***geoip filter dodatka***, ispituju podaci o geo-protornim informacijama izvorišne i odredišne *IP* adrese, respektivno. Nakon toga se za svaku od njih, u zavisnosti od toga da li je geo-prostorna pretraga bila uspešna, pripremaju polja za povratnu *DNS* pretragu, koristeći filter dodatak ***mutate*** u okviru linija 48-55 za izvorišnu, odnosno 57-64 za odredišnu *IP* adresu.

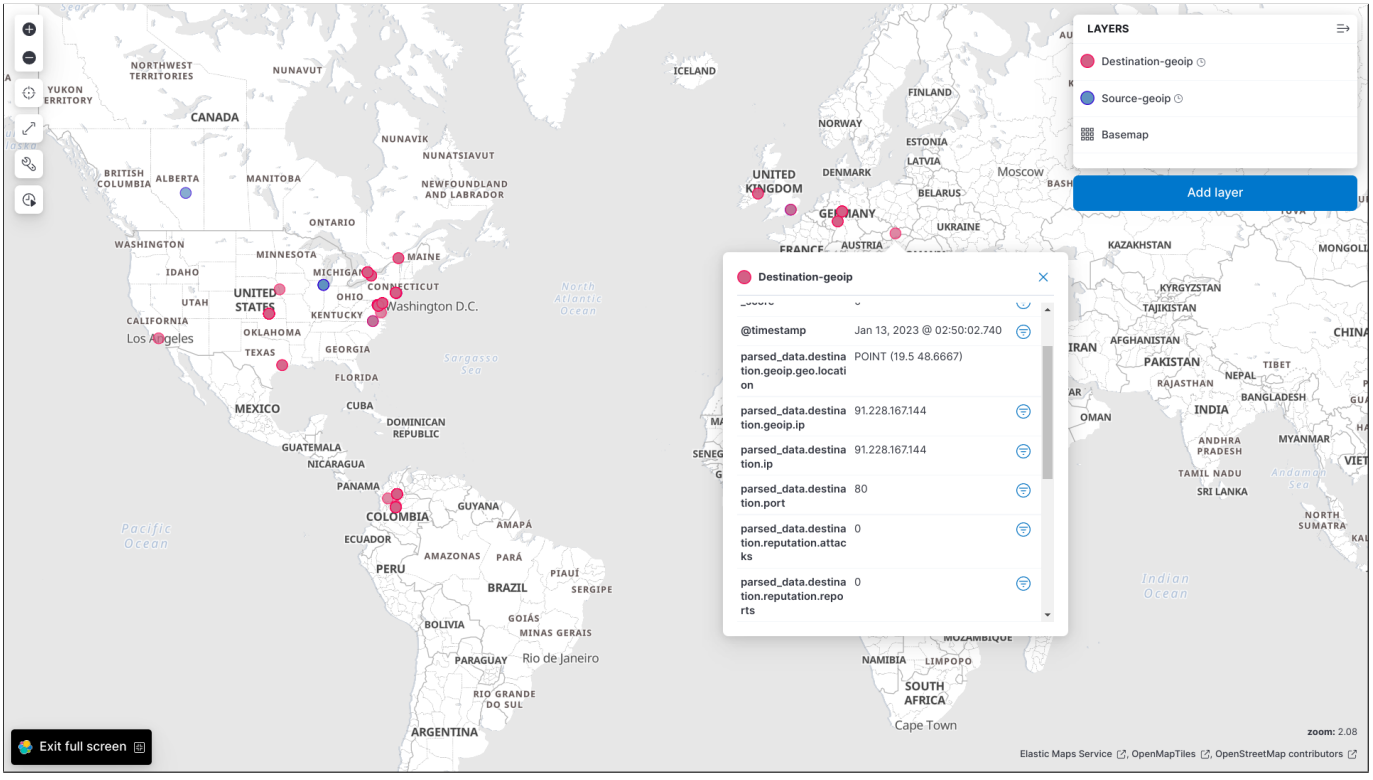
Konačno, ukoliko se radi o uspešnim geo-prostrnim vrednostima, vrši se povratna *DNS* pretraga, uz pomoć ***dns filter dodatka*** i vrednost pripremljenih polja se zamenjuje sa rezultatima pretrage, u okviru linija 66-70 za izvorišnu, odnosno 73-78 za odredišnu *IP* adresu. Ovime je filter etapa okončana.

U okviru izlazne etape, podaci koji sadrže *network-traffic* vrednost u okviru *gen* polja bivaju upisani na ***Elasticsearch*** instancu, koristeći istoimeni izlazni dodatak.

Specifična stvar jeste dobra praksa indeksiranja, na liniji 88, gde se indeksi grupišu na osnovu datuma, po mesecu kada su generisani. Time se olakšava dalja analiza skladištenih dokumenata. Paralelno sa ovim se izvršava i neobavezan ***izlazni dodatak stdout*** za ispis na standardni izlaz, koristeći ***enkoder rubydebug***.

3.3 Vizuelizacija podataka

Nakon što je komponenta ***Logstash*** uspešno obradila podatke i upisala ih u ***Elasticsearch*** komponentu, podaci su dostupni alatu ***Kibana*** za njihovu vizuelizaciju. Za vizuelizaciju je odabrano prikazivanje geo-prostornih koordinata, s obzirom na filter etapu ***Logstash*** toka podataka, te su podaci predstavljeni na svetskoj mapi, kao lokacije gde su se nalazili najbliži mrežni centri iz kojih su slati paketi. Odnosno, kreiran je pogled tipa mapa, na kome su ilustrovani podaci. Vizuelizacija podataka je prikazana na slici 3.2.



Slika 3.2: *Vizuelizacija podataka uz pomoć alata* ***Kibana***

Ovde je moguće koristiti ***KQL*** jezik za dodatno filtriranje podataka sa mape, odnosno prelazeći preko lokacija na mapi, prikazuje se *tooltip* kao na slici 3.2, koji pruža kratak uvid u bitnija polja skladištenih dokumenata.

Glava 4

Zaključak

Elasticsearch-ov model podataka je zaista fleksibilan i intuitivan. Moguće je ugnježdavanje polja, neograničenost u broju polja, bilo kakve tranformacije nad poljima, a nije unapred potrebno definisati šemu podataka. *Logstash* pruža neverovatnu fleksibilnost pri obrađivanju podataka u realnom vremenu. Postoji veliki broj gotovih dodataka koji dolaze unapred instalirani uz *Logstash*, a činjenica da je moguće ručno pisati dodatke u *Java*-i zaista pruža još veću fleksibilnost. *Kibana*, kao alat za vizuelizaciju ima veliki broj mogućnosti, mnoge od kojih ovaj rad nije pokrio. Sve u svemu, *ELK stack* je veoma stabilan alat. Tokom realizacije sistema opisanog u radu, konfiguracija je bila vrlo jednostavna, a rušenje i ponovno podizanje kontejnera nije pravilo nikakve probleme u međusobnoj komunikaciji između komponenata.

Kada se poredi sa tehnologijama poput *Apache Spark*-a ili *Hadoop*-a, mogu se naći zajedničke karakteristike. To je, u suštini, rezultat toga da svaki radni okvir želi da pruži način za obradu skupa velikih podataka, čime se granice raznih tehnologija zamagljuju. Ipak svaki alat služi određenoj svrsi i moramo da izaberemo ono što najbolje odgovara datim zahtevima. Ako jednostavno želimo da lociramo dokumente prema ključnoj reči i izvršimo jednostavnu analizu, onda *ELK stack* može da posluži kao sjajan alat za to. Ako imamo ogromnu količinu podataka za koje je potreban širok spektar različitih tipova složene obrade i analize, onda *Hadoop* pruža najširi spektar alata i najveću fleksibilnost.

Nismo ograničeni na korišćenje samo jedne tehnologije, već je zaista lepo i korisno biti upoznat sa što više različitih, jer se tada isti problem može sagledati iz više različitih uglova i pružiti mnogo više potencijalnih ideja i rešenja.

U okviru prvog poglavlja je dat uvod u rad. U drugom poglavlju su opisane osnove *Elastic stack*-a, zajedno sa njegovim komponentama i njihovim specifikacijama. U okviru trećeg poglavlja, je prikazan i objašnjen primer korišćenja jednog sistema koji se zasniva na alatu *Elastic stack*. U okviru ovog, poslednjeg, poglavlja su izneti zaključci rada.

Literatura

[1] *Energy efficiency in data centers and clouds*. San Diego, CA: Academic Press, Jan. 2016.

[2] M. P. I. Forum, *MPI : A Message-Passing Interface Standard Version 4.0*, june 2021.

[3] D. Miner, *MapReduce design patterns*. Sebastopol, CA: O'Reilly Media, dec 2012.

[4] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, 3 ed., jun 2012.

[5] B. Chambers and M. Zaharia, *Spark - the definitive guide*. Sebastopol, CA: O'Reilly Media, Mar. 2018.

[6] Elastic, "Elastic stack - documentation." https://www.elastic.co/guide/en/elastic-stack/current/index.html, 2023.

[7] Elastic, "Elastic - site." https://www.elastic.co/, 2023.

[8] Elastic, "Elastic - elasticsearch." https://www.elastic.co/elasticsearch/, 2023.

[9] Elastic, "Elastic - logstash." https://www.elastic.co/logstash/, 2023.

[10] Elastic, "Elastic - kibana." https://www.elastic.co/kibana/, 2023.

[11] Elastic, "Elastic - beats." https://www.elastic.co/beats/, 2023.

[12] Apache, "Apache - lucene." https://lucene.apache.org/, 2022.

[13] E. International, "Json." https://www.ecma-international.org/publications-and-standards/standards/ecma-404/, 2022.

[14] T. Y. Project, "Yaml." https://github.com/yaml/, 2021.

[15] P. Sanders, K. Mehlhorn, M. Dietzfelbinger, and R. Dementiev, *Sequential and parallel algorithms and data structures: The basic toolbox*. Cham, Switzerland: Springer Nature, 1 ed., 2019.

[16] "Anatomy of a compiler." http://www.cs.man.ac.uk/~pjj/farrell/comp3.html.

[17] "Java platform, standard edition java virtual machine guide." https://docs.oracle.com/javase/9/vm/java-virtual-machine-technology-overview.htm#JSJVM-GUID-982B244A-9B01-479A-8651-CB6475019281, Oct 2017.

[18] "Codecs." https://support.microsoft.com/en-us/windows/codecs-faq-392483a0-b9ac-27c7-0f61-5a7f18d408af.

[19] R. Gerhards, "The syslog protocol," RFC 5424, RFC Editor, March 2009. http://www.rfc-editor.org/rfc/rfc5424.txt.

[20] J. L. Carlson, *Redis in Action*. New York, NY: Manning Publications, May 2013.

[21] Graphite-Project, "Graphite." https://github.com/graphite-project/graphite-web, 2022.

[22] statsd, "statsd." https://github.com/statsd/statsd, 2022.

[23] Apache, "Logging services." https://logging.apache.org/log4j/2.x/, 2022.

[24] Elastic, "Logstash pipeline language specification." https://github.com/elastic/logstash/blob/main/logstash-core/spec/logstash/java_pipeline_spec.rb, 2023.

[25] Elastic, "Logstash tcp input plugin." https://www.elastic.co/guide/en/logstash/current/plugins-inputs-tcp.html, 2023.

[26] Elastic, "Logstash grok filter plugin." https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html, 2023.

[27] kkos, "Onigurama regex specification." https://github.com/kkos/oniguruma/blob/master/doc/RE, 2022.

[28] Elastic, "Logstash mutate filter plugin." https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html, 2023.

[29] Elastic, "Logstash geoip filter plugin." https://www.elastic.co/guide/en/logstash/current/plugins-filters-geoip.html, 2023.

[30] MaxMind, "Maxmind geolite api specification." https://dev.maxmind.com/minfraud/api-documentation?lang=en, 2023.

[31] Elastic, "Logstash http filter plugin." https://www.elastic.co/guide/en/logstash/current/plugins-filters-http.html, 2023.

[32] Elastic, "Logstash dns filter plugin." https://www.elastic.co/guide/en/logstash/current/plugins-filters-dns.html, 2023.

[33] Elastic, "Logstash elasticsearch output plugin." https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html, 2023.

[34] Elastic, "Kibana query language." https://www.elastic.co/guide/en/kibana/current/kuery-query.html, 2023.

[35] Docker, "Docker documentation." https://docs.docker.com/, 2023.

[36] Docker, "Docker compose documentation." https://docs.docker.com/compose/, 2023.

[37] J. S. ROJAS, "Ip network traffic flows labeled with 75 apps." https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps, 2018.

[38] Devianthony, "Docker-elk." https://github.com/deviantony/docker-elk, 2022.