Miskolci Egyetem

Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Alkalmazott Informatikai Intézeti Tanszék

Oktatást segítő alkalmazás kibővítése felhasználók közötti csevegéssel és részletes kereséssel

Készítette:
Vécsi Ádám

Tervezésvezető:
Dr. Krizsán Zoltán
egyetemi docens

Miskolc, 2021.

Eredetiségi Nyilatkozat

Alulírott **Vécsi Ádám**; Neptun kód: *IZBTF9* a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős, *gépészmérnök* szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy

Oktatást segítő alkalmazás kibővítése felhasználók közötti csevegéssel és részletes kereséssel

című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.
Miskolc, 2021.

(*Vécsi Ádám*)

Tartalomjegyzék

1. fejezet

Bevezetés

A mai világban az egyik legjobban használatos kommunikációs forma az interneten való csevegés. Használjuk munkahelyen és otthon is, akár számítógépen, mobiltelefonon, tábla gépen vagy okos órán. Talán a legismertebb alkalmazások a Skype, Slack, WhatsApp, de a legtöbb közösségi média platform lehetővé teszi számunkra ezt a kommunikációs eszközt. Az Instagram, Facebook messenger, Snapchat, LinkedIn is ad ilyen lehetőséget így a barátainkkal, ismerőseinkkel tudunk csevegni. De nem csak magán jellegű beszélgetésekre alkalmas ez, sokan használjuk munkahelyen is információ csere céljából.

Legnagyobb előnye, hogy valós időben történik az üzenetváltás bizonyos esetekben akár kép és videó megosztás, fájl csere is. Ami fontos még a chat alkalmazásokban, hogy az üzenetek naplózva lesznek, így korábbi beszélgetéseket is vissza lehet olvasni és nem vész el az információ. Ezen kívül általában keresni is lehet a beszélgetésekben.

Dolgozatomban egy chat alkalmazás működését szeretném bemutatni, hogyan jutnak el az üzenetek a küldőtől a célig, hogyan történik az üzenetek naplózása, valamint az üzenetekben való keresés. Szeretnék készíteni egy chat alkalmazást egy oktatást segítő alkalmazás kiegészítéseként, melyben pontosan az előbb említett funkciókat valósítanám meg. Backend oldalon Java, Spring boot, Apache Kafka és Elastic search segítségével.

2. fejezet

Chat alkalmazások működése

//TODO: ide, hogy socketen mennek az üzenetek stb.

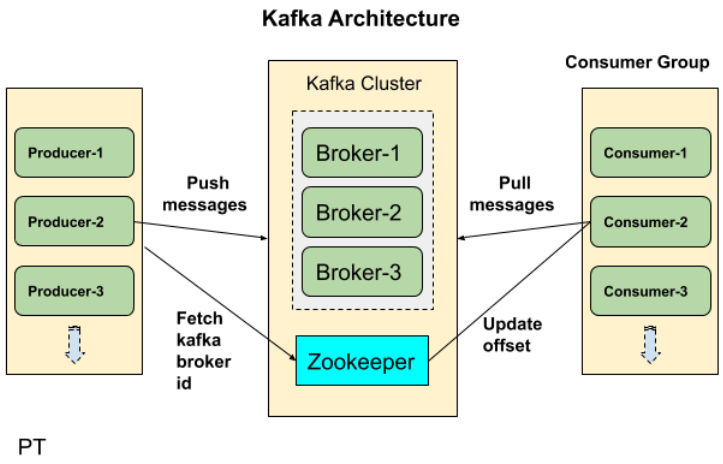
3. fejezet

Apache Kafka

Az Apache Kafka Scala és Java nyelven íródott, és a korábbi LinkedIn adatmérnökök alkotása. Már 2011-ben a technológiát erősen skálázható üzenetküldő rendszerként adták át amely nyílt forráskódú. Ma az Apache Kafka a Confluent Stream Platform része és napi események billióit kezeli. Az Apache Kafka számos megbízható társasággal körbeépítette magát a piacon.

A mai összetett rendszerekben szereplő adatokat és naplókat feldolgozni, újrafeldolgozni, elemezni és kezelni kell - gyakran valós időben. És ezért az Apache Kafka jelentős szerepet játszik az üzenet streaming környezetében. A Kafka kulcsfontosságú tervezési alapelveit az egyre növekvő igény alapján alakítják ki a nagy teljesítményű architektúrák, amelyek könnyen skálázhatóak, és lehetővé teszik az adatok tárolását, feldolgozását és újrafeldolgozását.

3.1. Kafka architektúra



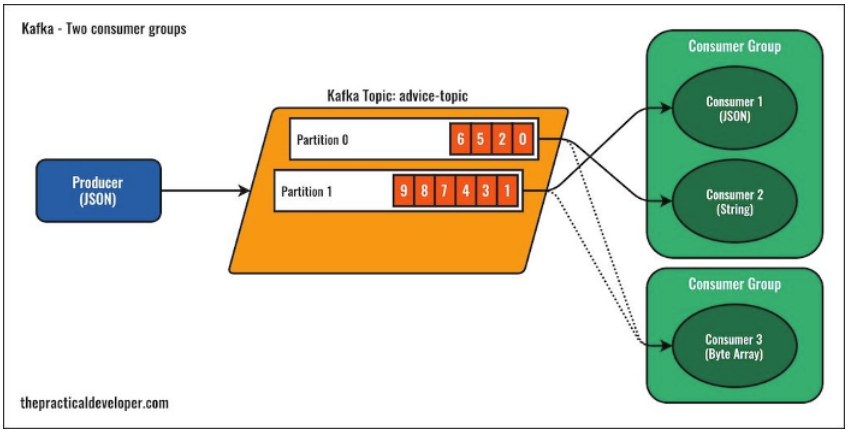
3.1. ábra. Kafka architektúra

Egy Kafka architektúra legalább egy Kafka szerverből (bróker) áll ami a konfigurációját kötelezően a Zookeeper nevű elosztott konfigurációs management rendszerben tárolja. A Kafka borker-hez csatlakoznak a termelők és fogyasztók. A Kafka cluster-ben úgynevezett topic-ok találhatók. A termelők mindig egy dedikált topik-ra írnak, és a

fogyasztók mindig egy dedikált topic-ról olvasnak, tehát a topic az a logikai egység, ami egy termelő-fogyasztó páros számára az üzeneteket tárolja és továbbítja.Mikor elindítunk egy Kafa példányt, akkor valójában egy kafka brokert indítunk el. Ha producer-ek mindig egy brokerhez csatlakoznak. A teljes konfiguráció zookeeper-ben van tárolva. A zookeeper tudja értesíteni a klienseket ha a konfiguráció változik, ezért hamar elterjed a hálózaton a változás. A 3.1 ábrán láthatjuk az architektúrát.

3.2. Kafka működése

Egy topic úgynevezett partíciókra van osztva. Minden üzenet csak egy partícióba kerül be. A 3.2 ábrán látható a partíció. A producer-ek egy megadott topic-kra dobálják be az üzeneteket, amit onnan a consumer-ek kiolvasnak. Egy topic tetszőleges számú partícióból állhat. Egy partíció az a logikai egység, aminek rá kell férnie egy lemezre. A topic-kot úgy kell felskálázni, hogy egyre több partíciót adunk hozzá, amik különböző brokereken fognak létrejönni. Minden partíciónak lehet egy vagy több replikája, amik biztonsági másolatok. Mikor a producer beküld egy üzenetet egy partícióba, akkor fog committed üzenetnek minősülni, ha minden replikára is eljutott. Azt, hogy egy



3.2. ábra. Kafka partíciók

producer melyik partícióba dobja az üzenetet vagy a kulcs határozza meg, vagy round-robin módon mindig egy másikba teszi. Ha van kulcs, akkor az abból készült hash fogja meghatározni, hogy melyik partícióba kerüljön. Ugyan az a kulcs így mindig ugyan abba a partícióba fog kerülni. De a kulcs nem kötelező. A sorrend tartás csak egy partíción belül garantált, de ott nagyon. Ha nagyon kritikus bizonyos üzenetek sorrendje, akkor azokat egy partícióba kell rakni azonos kulcsot használva. Loggolásnál ez nem kritikus, egyrészt mert a logstash sorba rakja az üzeneteket, másrészt mikor elastichsearch-be szúrjuk, ott a dátum lesz az egyik attribútum, ami alapján már sorba lehet majd újra rendezni a logokat. Az meg amúgy sem kritikus, ha a log egy része enyhe csúszással kerül be az adatbázisba, lényeg, hogy végül helyes lesz a sorrend.

A comsumer-eket úgynevezett consumer-group-okba szervezzük az azonosítójuk szerint. Egy csoport mindig ugyan azon topic üzeneteit olvassa, de minden egyes consumer a csoporotban más és más partícióból. Minden partíció csak egy consumer-hez rendelhető hozzá egy csoporton belül. De ha nincs annyi consumer a csoportban mind ahány

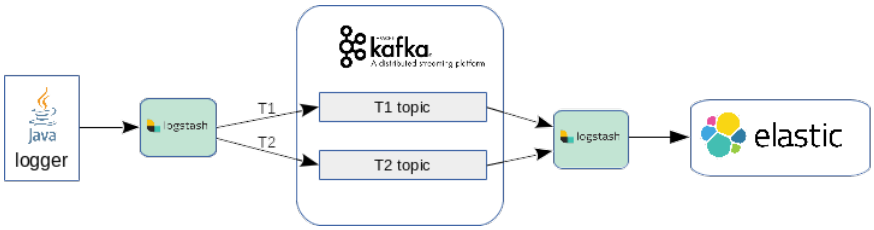
partíció, akkor egy consumer több partíciót is fog olvasni (ahogy ez a fenti ábrán is látszik, az alsó consumer két partíciót olvas. Viszont ha több consumer van mint partíció egy csoportban, akkor bizonyos consumer-ek mindig idle állapotban lesznek. Minden csoporton belül van egy vezető consumer, általában az aki először csatlakozott. Ő teríti a többieknek a cluster információkat.

A Kafka nem tudja értelmezni sem a kulcsot sem az üzenetet. Ez számára egy bájt tömb. Az, hogy egy objektumból hogy lesz bájt tömb kulcs és bájt tömb üzenet a producer-ben lévő serializátor dolga. A consumer-ben pedig a deserializázor dolga, hogy a bájt folyamból újra értelmes objektumot állítson elő.

Minden partíció új üzenete mindig a partíció végére íródik. A partíció elejétől számoljuk az üzenetek sorszámát, ezt hívjuk offset-nek. Mikor egy consumer kiolvas egy üzentet, attól az még ott marad a partícióba egészen addig, amíg len nem jár, alapértelmezetten ez egy nap. Tehát ez eltér a hagyományos sor kezeléstől. A Kafka nyilvántartja, hogy melyik consumer egy adott partícióban melyik offset-nél tartott. Ezt egy speciális topic-ban tartja nyilván: Ha újra is indul a világ, akkor is tudni fogják a consumer-ek hogy hol tartottak, és onnan folytatják.

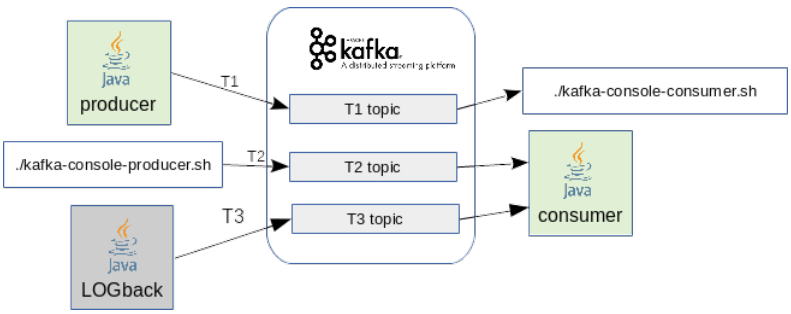
3.3. Kafka és a logolás

A docker alapú cloud világban egy tipikus architektúra a logok centralizált gyűjtésére, mikor egy logstash példány a producer és egy másik logstash példány a consumer. A konténer logokat a producer logstash kapja meg, aki a log sorok különböző paraméterei mentén a megfelelő Topic-ba tudja irányítani az üzeneteket. A consumer logstash pedig leszedi a Topic-rol az üzenetet és beírja Elasticsearch-be. A 3.3



3.3. ábra. Kafka és a Java

A Kafka világban nagyon széles a választéka a producer-eknek és consumer-eknek, akik képesek közvetlenül Kafka-ba írni és onnan olvasni. A Java világban a megfelelő Kafka lib-ek segítségével írhatunk Java producer-eket és consumer-eket amik olyan Java programok, amik közvetlenül írják ill. olvassák a Kafka topic-ot. A másik lehetőség a producer-re, hogy a logger keretrendszerünk Kafka kliens appender-jét használjuk, ami a rendszer logokat képes kapásból Kafka-ba írni. Ha letöltjük a Kafka programot, akkor abban található parancssori producer és consumer is, ami képes tesztelés céljából közvetlen beírni és kiolvasni egy topic-ból, ami nagyon hasznos a tesztelés során.



3.4. ábra. Kafka logs

4. fejezet

Elasticsearch, kibana, logstash, beats

4.1. Elasticsearch

Az Elasticsearch egy nagyon skálázható, nyílt forrású, teljes szövegű kereső és elemző motor. Ez lehetővé teszi a nagy mennyiségű adat gyors tárolását, keresését és elemzését gyors és szinte valós időben. Általában olyan alkalmazások használják motorként/ technológiaként, amely bonyolult keresési funkciókkal és követelményekkel rendelkeznek. Séma nélküli, néhány alapértelmezett értéket használ az adatok indexeléséhez.

A Relációs adatbázis viszonylag lassan működik hatalmas adatkészletek esetében, ami lassabb keresési eredményeket eredményez az adatbázisból származó lekérdezés esetén. Természetesen az RDBMS optimalizálható, de ez magában foglalja a korlátozások halmazát is, például, hogy minden mezőt nem lehet indexelni, és a sorok frissítése erősen indexált táblázatokba hosszadalmas folyamat.

A vállalkozások manapság alternatív módszereket keresnek, ahol az adatokat olyan módon tárolják, hogy a visszakeresés gyors. Ez úgy érhető el, ha az adatok tárolására az RDBMS helyett NoSQL-t alkalmazunk. Az Elasticsearch egy ilyen NoSQL elosztott adatbázis. Az Elasticsearch rugalmas adatmodelleken alapszik és kis késleltetésű, majdhogy nem valós idejű keresést tesz lehetővé.

- Közel a valós idejű: Az Elasticsearch közeli valósidejű keresési platform, amely a keresést olyan gyorsan hajtja végre, mint amikor egy dokumentumot indexel.
- Klaszter: A klaszter egy vagy több csomópont gyűjteménye, amely együttesen a teljes adatot tárolja. Egyesített indexelési és keresési képességeket biztosít minden csomópontban, és egyedi névvel azonosítják (alapértelmezés szerint „elasticsearch").
- Csomópont: A csomópont egyetlen kiszolgáló, amely a klaszter része, adatokat tárol és részt vesz a klaszter indexelés és keresésben.
- Index: Az index olyan dokumentumok gyűjteménye, amelyek hasonló tulajdonságokkal rendelkeznek, és névvel azonosíthatók. Ez a név hivatkozik az indexre, miközben indexeli, keres, frissít és töröl műveleteket a benne lévő dokumentumok alapján.

- Típus: A típus olyan index logikai típusa, amelynek szemantikája komplett. Ez a dokumentum a közös mezőkből álló dokumentumok számára van meghatározva. meghatározhat egynél több típust az indexében.
- Dokumentum: A dokumentum alapvető információegység, amely indexálható. Ezt JSON-ban mutatják be, amely egy globális internetes adatcsere-formátum.
- Shards: Az elasticsearch lehetővé teszi az index felosztását több darabra, úgynevezett shards-ra. Mindegyik shard önmagában egy teljesen funkcionális és független „index", amely a klaszter bármelyik csomópontján elhelyezhető.
- Replikák: Az Elasticsearch lehetővé teszi, hogy egy vagy több példányt készítsen az index szeleteiről, amelyeket replikáknak vagy replikáknak hívnak.

4.2. Kibana

A Kibana egy adatmegjelenítő és -kezelő eszköz az Elasticsearch számára, amely valós idejű hisztogramokat, vonaldiagramokat, kördiagramokat és térképeket biztosít. Ez lehetővé teszi az Elasticsearch adatok megjelenítését és az Elastic Stack navigálását. Az egyik kérdéssel megválaszthatja, hogy hogyan alakítsa ki az adatait, és megtudja, hová vezet az interaktív megjelenítés. Például, mivel a Kibanát gyakran használják naplóelemzéshez, ez lehetővé teszi a kérdések megválaszolását arról, hogy honnan származnak a webes találatai, a terjesztési URL-ek stb. Ha nem saját alkalmazását épít az Elasticsearch tetején, akkor a Kibana remek módja az indexének keresésére és megjelenítésére egy hatékony és rugalmas felhasználói felülettel. Fontos hátránya azonban, hogy minden megjelenítés csak egyetlen index / index mintázat alapján működhet. Tehát ha szigorúan eltérő adatokkal rendelkező indexekkel rendelkezik, akkor mindegyikhez külön megjelenítést kell létrehoznia. A fejlettebb használati esetekben a Knowi jó lehetőség. Ez lehetővé teszi, hogy az Elasticsearch adatait összekapcsolja több index között, és összekeverje más SQL / NoSQL / REST-API adatforrásokkal, majd vizualizációkat készíthet belőlük egy üzleti felhasználóbarát felhasználói felületen.

4.3. Logstash

A Logstash az adatokat összesíti és feldolgozza, és elküldi az Elasticsearch-nek. Ez egy nyílt forráskódú, szerveroldalú adatfeldolgozási folyamat, amely sok forrásból származó adatokat egyidejűleg vesz fel, átalakítja és gyűjtésre továbbítja. Emellett formátumoktól függetlenül átalakítja és előkészíti az adatokat, azonosítva a megnevezett mezőket a struktúra felépítéséhez, és átalakítja azokat egy közös formátumba való konvergáláshoz. Mivel például az adatok gyakran különböző rendszerekben vannak szétszórva, különböző formátumokban, a Logstash lehetővé teszi a különböző rendszerek összekapcsolását, például webszerverek, adatbázisok, Amazon szolgáltatások stb., És az adatok közzétételét bárhol, ahol folyamatos adatfolyamon kell mennie.

4.4. Beats

A Beats egy egyszerű, egycélú adatátviteli ügynökök gyűjteménye, amelyeket száz vagy több ezer gép és rendszer adatainak küldésére használnak a Logstash vagy az Elasticsearch számára. Az Beats kiválóan alkalmasak az adatok gyűjtésére, mivel a szerverekre ülhetnek, a tárolókkal együtt, vagy funkcióként telepíthetők, majd az adatokat az Elasticsearchbe központosíthatják. Például a Filebeat ülhet a szerveren, figyelheti a bejövő naplófájlokat, elemzi azokat, és importálhatja az Elasticsearch rendszerbe valós időben.

5. fejezet

Chat alkalmazás implementálása

6. fejezet

Összefoglalás

Ebben a fejezetben kell összefoglalni a szakdolgozat eredményeit, sajátosságait és a témában való elhelyezkedését. A fejezet címe az „Összefoglalás” NEM módosítható! Lehet benne több alfejezet is, de nem ajánlott. Minimum 1 maximum 4 oldal a terjedelem.

Summary

Irodalomjegyzék

[1] Neha Narkhede, Gwen Shapira, and Todd Palino: *Kafka: The Definitive Guide, Real-Time Data and Stream Processing at Scale*. O'Reilly Media, Inc., 2017.

[2] https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

[3] https://wiki.berki.org/index.php/Apache_Kafka#Kafka_bemutat.C3.A1sa

[4] https://programmertoday.com/apache-kafka-architecture-and-components/

[5] https://www.elastic.co/what-is/elasticsearch

[6] https://programmertoday.com/apache-kafka-architecture-and-components/