 Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Mérnökinformatikus mesterképzési szak

Alkalmazásfejlesztői szakirány

Oktatást segítő alkalmazás kibővítése felhasználók közötti csevegéssel és részletes kereséssel

Diplomamunka

(szenvedés árán összehozott értelmetlen hulladék, mely egy semmitérő képzés nullához konvergáló terméke)

Vécsi Ádám

IZBTF9

2020

|  |  |  |  |
| --- | --- | --- | --- |
| **MISKOLCI EGYETEM**  **GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR** | | | |
| **Alkalmazott Informatikai Intézeti Tanszék**  **H-3515 Miskolc-Egyetemváros** | | |
|  | Mérnökinformatikus mesterképzési szak  Alkalmazásfejlesztői szakirány  **Diplomamunka azonosító: IAL/IZBTF9/MSc/2021**  **Intézmény azonosító: FI 87515** |  |

**DIPLOMAMUNKA FELADAT**

|  |  |
| --- | --- |
| **VÉCSI ÁDÁM** | |
| MSc mérnökinformatikus jelölt részére | |
| A tervezés tárgyköre: | | **Alkalmazás készítés** | |
| A diplomaterv címe: | | **Oktatást segítő alkalmazás kibővítése felhasználók közötti csevegésel és részletes kereséssel** | |

**A feladat részletezése:**

* TODO

|  |  |  |
| --- | --- | --- |
| Tervezésvezető: | **Dr. Krizsán Zoltán, egyetemi docens** | **Alkalmazott Informatikai Intézeti Tanszék** |
| Konzulens: | **Dr. Krizsán Zoltán, egyetemi docens** | **Alkalmazott Informatikai Intézeti Tanszék** |

|  |  |
| --- | --- |
| Diplomaterv kiadásának időpontja: | **2020.09.18.** |
| Diplomaterv beadásának határideje: | **2020.11.20.** |

Miskolc, 2020.09.18.

**Dr. Nehéz Károly**

tanszékvezető, egyetemi docens

1. A diplomamunka beadható:

............................... ................................. ...........................................  
 dátum tervezésvezető konzulens

2. A diplomamunka ...............................szövegoldalt,  
 ...............................db nyomtatott mellékletet,  
 ...............................db CD / DVD lemezt,  
 ...............................db egyéb mellékletet  
 tartalmaz.

................................. ....................................................  
 dátum tervezésvezető

3. A diplomamunka bírálatra bocsátható

nem bocsátható  
  
A bíráló neve, beosztása, munkahelye:.……….........................................................................................

…………………….................................……….......................................................................................

…….................... ..........................................  
 dátum tanszékvezető

4. A diplomamunka osztályzata:  
  
A bíráló javaslata: ...............................................  
  
A tanszék javaslata: ...............................................  
  
**A ZVB döntése:** ...............................................

miskolc, ......................................................

..................................................

**ZVB** elnök

**Eredetiségi nyilatkozat**

Alulírott *Vécsi Ádám.* (neptun kód: IZBTF9)

a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy a

*Oktatást segítő alkalmazás kibővítése felhasználók közötti csevegéssel és részletes kereséssel*

című diplomamunkám saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályi szerint történt.

Tudomásul veszem, hogy 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 a szakdolgozat visszavonásra kerül.

Miskolc, 2020.11.11.

Hallgató aláírása

Tartalomjegyzék

[1. Bevezetés 6](#_Toc55329869)

[2. Chat alkalmazások vizsgálata?? 7](#_Toc55329870)

[3. Chat alkalmazáshoz használt technológiák bemutatása 8](#_Toc55329871)

[3.1 Spring boot (fél oldal) 8](#_Toc55329872)

[3.2 Docker (1 oldal) 8](#_Toc55329873)

[3.3 Apache Kafka (3-5 oldal) 10](#_Toc55329874)

[3.3.1 Architektúra 10](#_Toc55329875)

[3.3.2 Kafka Cluster 11](#_Toc55329876)

[3.3.3 Kafka Broker 11](#_Toc55329877)

[3.3.4 Producer API 13](#_Toc55329878)

[3.3.5 Consumer API 14](#_Toc55329879)

[3.4 Elasticsearch (3-5 oldal) 17](#_Toc55329880)

[3.4.1 Kibana 18](#_Toc55329881)

[3.4.2 Beats 19](#_Toc55329882)

[3.4.3 Logstash 19](#_Toc55329883)

[4. Chat alkalmazás implementáció 20](#_Toc55329884)

[4.1 Docker Compose 20](#_Toc55329885)

[4.2 Felhasználók, szobák tárolása 20](#_Toc55329886)

[4.3 Apache Kafka integrálódás 20](#_Toc55329887)

[4.3.1 Topic létrehozása és csatlakozás hozzá 20](#_Toc55329888)

[4.3.2 Producer 20](#_Toc55329889)

[4.3.3 Consumer 20](#_Toc55329890)

[4.4 Elasticsearch integrálódás 20](#_Toc55329891)

[4.4.1 Index létrehozása 20](#_Toc55329892)

[4.4.2 Lekérdezések 20](#_Toc55329893)

[4.5 Keresés, szűrés implementálása 20](#_Toc55329894)

[4.6 Websocket 20](#_Toc55329895)

[4.7 JQuery 20](#_Toc55329896)

[5. Összegzés 21](#_Toc55329897)

[Summary 22](#_Toc55329898)

[Irodalomjegyzék 23](#_Toc55329899)

[Mellékletek 24](#_Toc55329900)

# 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 Elasticsearch segítségével

# Chat alkalmazások vizsgálata??

TODO:

# Chat alkalmazáshoz használt technológiák bemutatása

## Spring boot (fél oldal)

## Docker (1 oldal)

A Docker egy olyan eszköz, amelyet az alkalmazások létrehozása, telepítése és futtatása konténerek segítségével könnyebbé tesz. A tárolók lehetővé teszik a fejlesztők számára, hogy csomagolják az alkalmazást az összes szükséges alkatrésszel, például könyvtárakkal és más függőségekkel, és egy csomagként telepítsék. Ezzel a tárolónak köszönhetően a fejlesztő megnyugodhat, hogy az alkalmazás bármely más Linux gépen futni fog, függetlenül attól, hogy a gépnek milyen testreszabott beállításai lehetnek, amelyek eltérhetnek a kód írásához és teszteléséhez használt géptől.

Bizonyos értelemben a Docker egy kicsit olyan, mint egy virtuális gép. De a virtuális géppel ellentétben, a teljes virtuális operációs rendszer létrehozása helyett, a Docker lehetővé teszi az alkalmazások számára, hogy ugyanazt a Linux-kernelt használják, mint az általuk futtatott rendszer, és csak olyan alkalmazásokat igényel, amelyek még nem futnak a gazdagépen. Ez jelentős teljesítménynövekedést eredményez és csökkenti az alkalmazás méretét.

Ami fontos, a Docker nyílt forráskódú. Ez azt jelenti, hogy bárki hozzájárulhat a Docker fejlesztéséhez és kibővítheti saját igényeinek kielégítéséhez, ha további funkciókra van szüksége, amelyek már nem kaphatók.

Kinek szól Docker?

A Docker egy olyan eszköz, amelyet mind a fejlesztők, mind a rendszergazdák számára előnyössé alakítottak ki, így számos DevOps (fejlesztő + művelet) eszköztár részévé teszi. A fejlesztők számára ez azt jelenti, hogy a kód írására összpontosíthatnak, anélkül, hogy aggódnának a rendszer mellett, amelyen végső soron futni fog. Ez azt is lehetővé teszi számukra, hogy előnyt szerezzenek, ha alkalmazásuk részeként a Docker konténerben futtatásra tervezett több ezer program egyikét használják. Az operatív személyzet számára a Docker rugalmasságot biztosít és potenciálisan csökkenti a szükséges rendszerek számát a kis lábnyoma és az alacsonyabb rezsi miatt.

A konténerek megértése

A konténerek három szoftverkategóriát igényelhetnek:

* Építő: a konténer építéséhez használt technológia.
* Motor: a konténer futtatásához használt technológia.
* Orchestration: sok konténer kezeléséhez használt technológia.

A konténerek használatának egyik vonzereje az, hogy képesek kecsesen meghalni, és igény szerint újjáéledni. Akár a konténer megszűnését összeomlás okozza, akár azért, mert egyszerűen nincs rá szükség, ha a kiszolgáló forgalma alacsony, a konténerek olcsók elindítani, és úgy vannak kialakítva, hogy zökkenőmentesen jelennek meg és tűnjenek el. Mivel a konténerek elmúlóak, és új példányokat hoznak létre, ahányszor csak szükséges, várható, hogy a megfigyelésüket és kezelésüket nem az ember valós időben végzi el, hanem automatizált módon.

A Linux-konténerek elősegítették a nagy rendelkezésre állású számítástechnika hatalmas változását, és számos eszközkészlet van a szolgáltatások (vagy akár az egész operációs rendszer) konténerekben történő futtatásához. A Docker az egyik lehetőség a sok közül, amelyet az Open Container Initiative (OCI) meghatároz, az ipari szabványügyi szervezet, amely az innovációt ösztönzi, miközben elkerüli a gyártók bezáródásának veszélyét. Az OCI-nak köszönhetően választása van a tárolóeszköz-lánc kiválasztásakor, beleértve a Docker, az OKD, a Podman, az rkt, az OpenShift és másokat.

Ha úgy dönt, hogy a szolgáltatásokat konténerekben futtatja, akkor valószínűleg szüksége van a tárolók tárolására és kezelésére tervezett szoftverre. Ezt széles körben konteiner hangszerelésnek nevezik. A Kubernetes biztosítja a konténerek hangszerelését a konténerek különféle futási idejéhez.

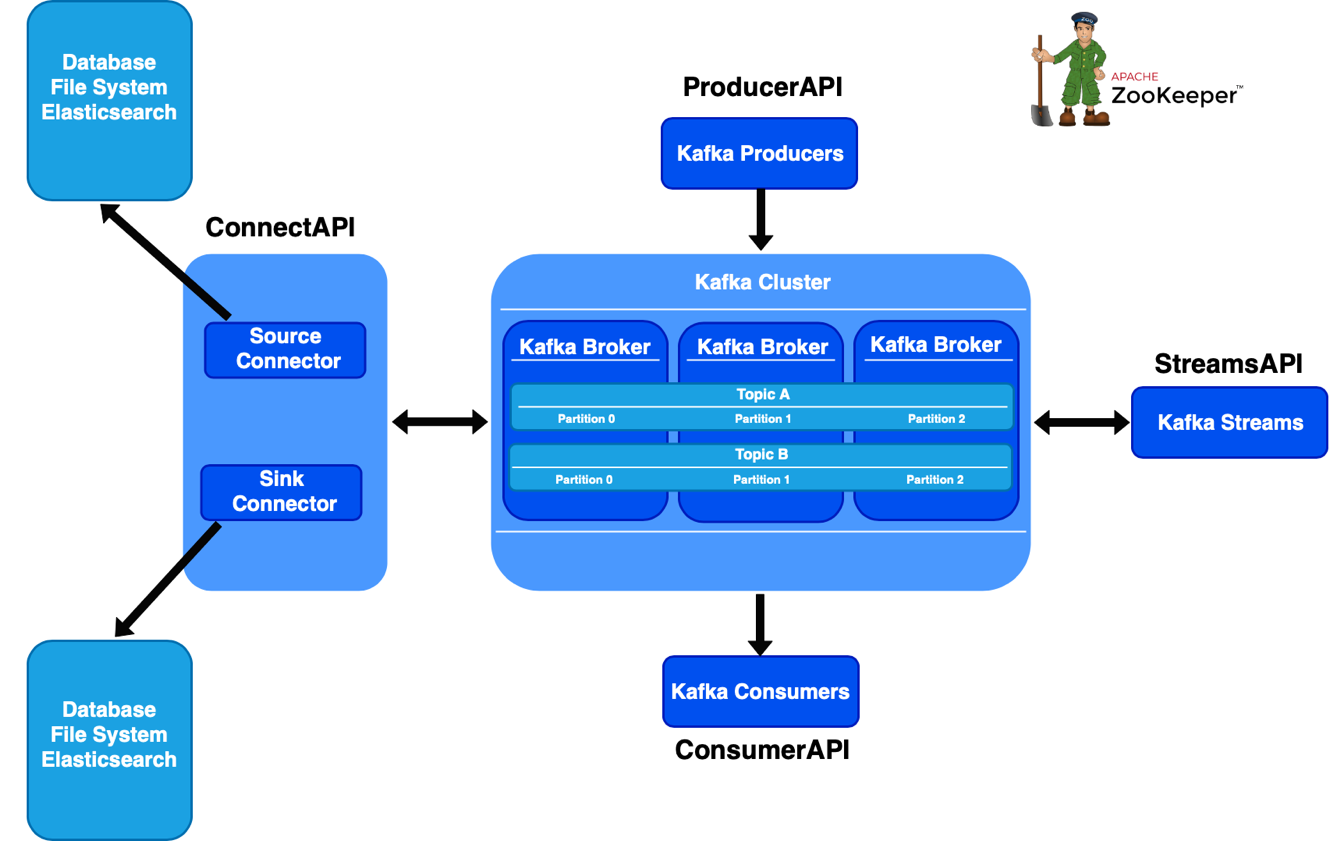
## Apache Kafka (3-5 oldal)

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.

### 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.



### Kafka Cluster

### Kafka Broker

#### Topic

Egy topic úgynevezett partíciókra van osztva. Minden üzenet csak egy partícióba kerül be. \Aref{fig:kafka-partitions} ábrán látható a partíció.

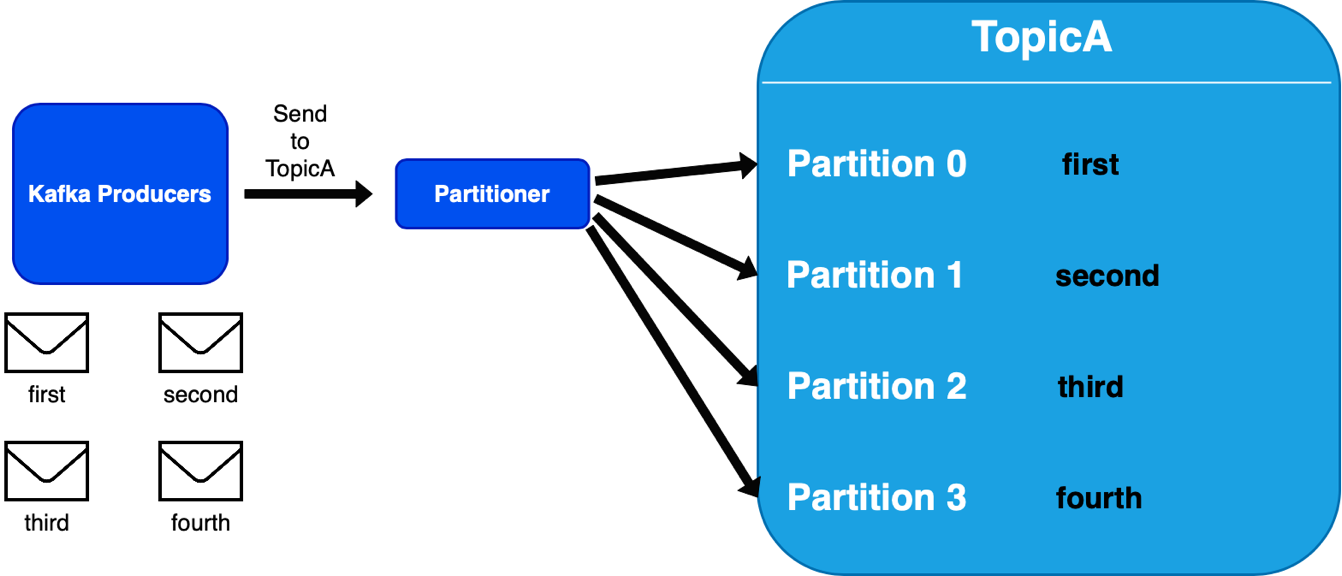
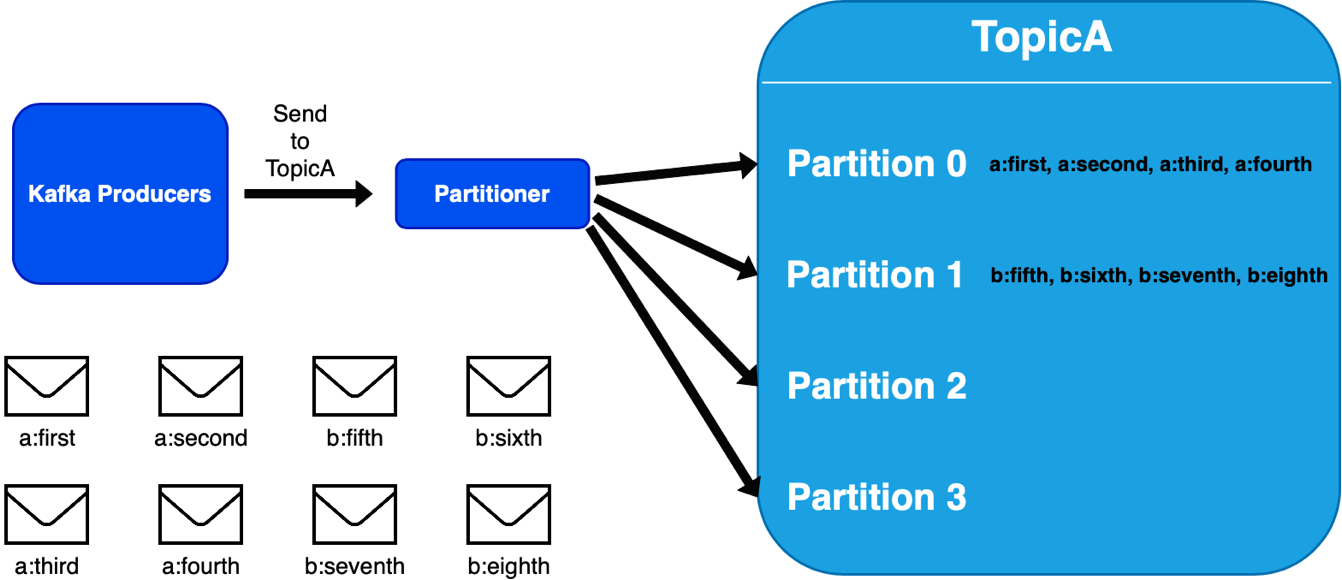


#### Partíció

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.

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. A képen szöveg látható

Automatikusan generált leírás

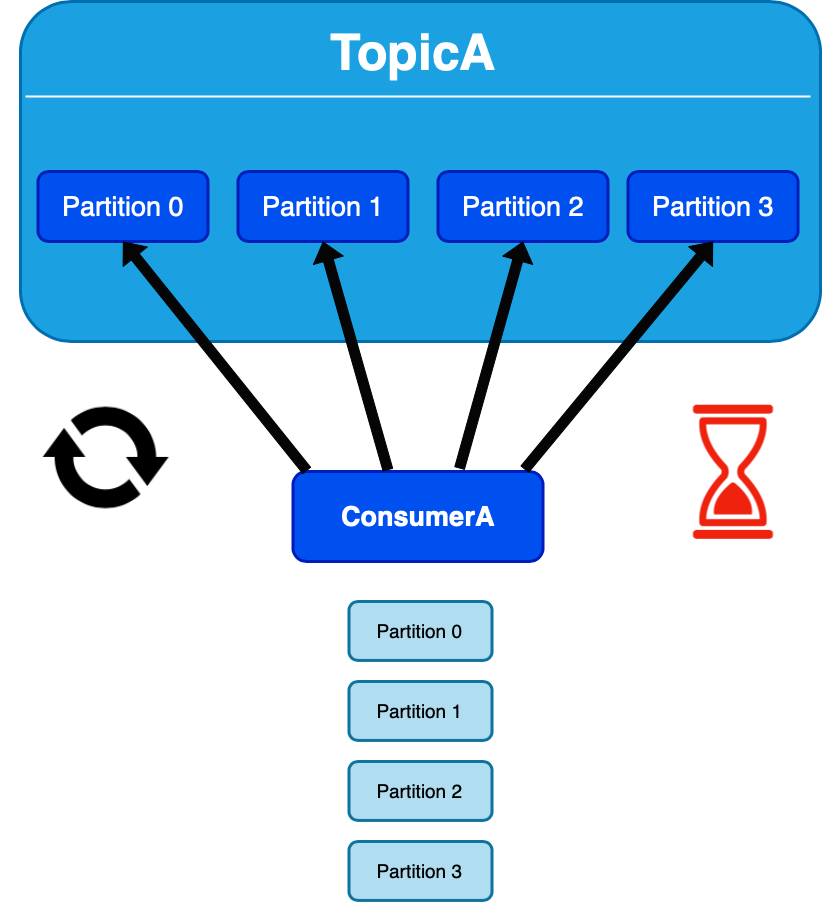
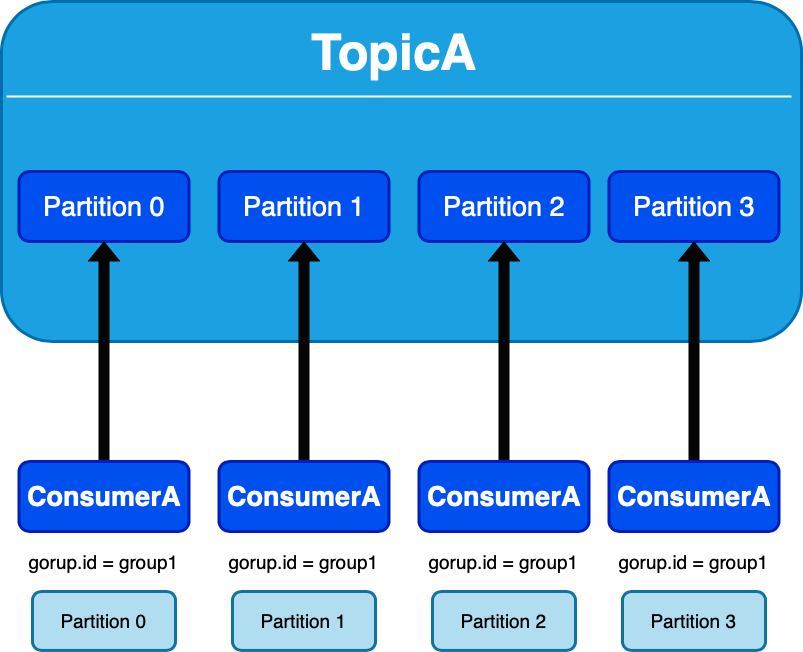


### Producer API

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 re Azt, hogy egy 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.plikára is eljutott.

### Consumer API

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.

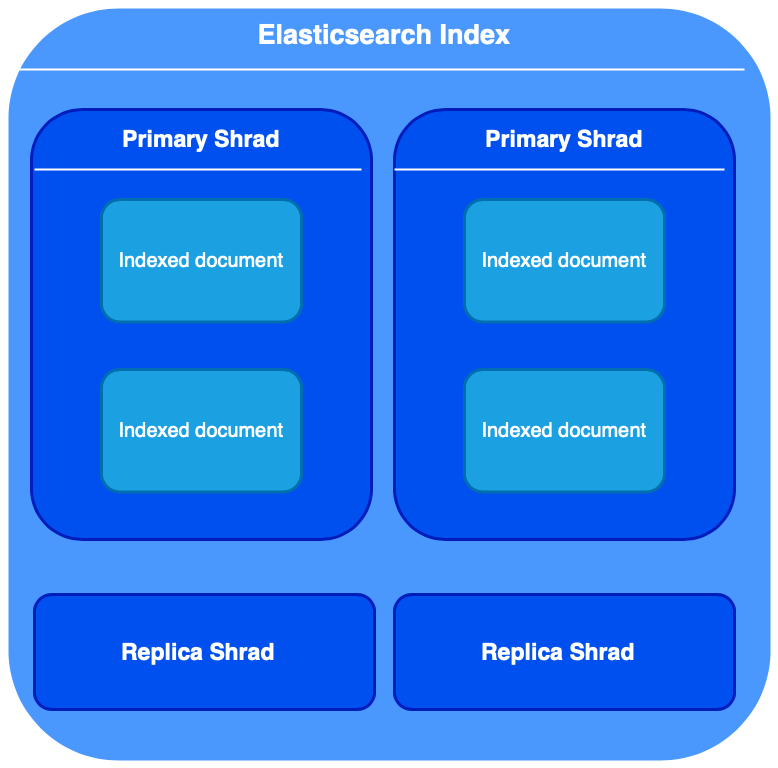


TODO:

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 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.

## Elasticsearch (3-5 oldal)

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.

### 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.

### 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.

### 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.

# Chat alkalmazás implementáció

Ahhoz, hogy az alkalmazás működőképes legyen szükség van arra, hogy megteremtésük az előfeltételeket. Szükség van Elasticsearch-re, Apache Kafka broker-re, Zookeeper-re és végül nem feltétlenül szükséges, de a könnyű tesztelhetőség kedvéért egy Kibana-ra is. Mivel bonyolult lenne ezt mind telepíteni, konfigurálni úgy döntöttem, hogy készítek hozzá egy Docker compose fájlt. Ami ezt mind összerakja a fejlesztéshez.

## Docker Compose

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | version: '3'  services:  zookeeper:  image: wurstmeister/zookeeper  container\_name: zookeeper  ports:  - "2181:2181"  kafka:  image: wurstmeister/kafka  container\_name: kafka  ports:  - "9092:9092"  environment:  KAFKA\_ADVERTISED\_HOST\_NAME: localhost  KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  elasticsearch:  image: docker.elastic.co/elasticsearch/elasticsearch:7.9.2  container\_name: elasticsearch  environment:  - node.name=elasticsearch  - discovery.seed\_hosts=elasticsearch  - cluster.initial\_master\_nodes=elasticsearch  - cluster.name=docker-cluster  - bootstrap.memory\_lock=true  - "ES\_JAVA\_OPTS=-Xms512m -Xmx512m"  ulimits:  memlock:  soft: -1  hard: -1  volumes:  - esdata1:/usr/share/elasticsearch/data  ports:  - 9200:9200  kibana:  image: docker.elastic.co/kibana/kibana:7.9.2  container\_name: kibana  environment:  ELASTICSEARCH\_URL: "http://elasticsearch:9200"  ports:  - 5601:5601  depends\_on:  - elasticsearch  volumes:  esdata1:  driver: local |

## Felhasználók, szobák tárolása

Ahhoz, hogy a szobákat, felhasználókat tudjuk tárolni szükség van több függőségre is.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-core</artifactId>  </dependency>  <groupId>com.h2database</groupId>  <artifactId>h2</artifactId>  <scope>runtime</scope>  </dependency> |

hilite.me

H2 embedded adatbázist használok adatbázisként. Liquibase segítségével hozom létre a szükséges táblákat. Spring data jpa segítségével történik az adatok tárolása.

## Apache Kafka integrálódás

### Topic létrehozása és csatlakozás hozzá

### Producer

### Consumer

## Elasticsearch integrálódás

### Index létrehozása

### Lekérdezések

## Keresés, szűrés implementálása

## Websocket

## JQuery

# Összegzés

# Summary

# Irodalomjegyzék

[1] https://opensource.com/resources/what-docker

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

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

[4] https://wiki.berki.org/index.php/Apache\_Kafka#Kafka\_bemutat.C3.A1sa

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

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

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

[10]

# Mellékletek

CD melléklet tartalma