Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát

Diplomová práca

2020 **Bc. Kamil Strachan**

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát

Diplomová práca

Študijný program:	Hospodárska informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra kybernetiky a umelej inteligencie (KKUI)
Školiteľ:	Ing. Peter Bednár, PhD.
Konzultant:	Ing. Peter Bednár, PhD.

Košice 2020 **Bc. Kamil Strachan**

Abstrakt v SJ

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát je práca, ktorá sa zaoberá zefektívnením výmeny elektronických obchodných dokumentov. Práca podáva teoretický prehľad o elektronickej výmene dát, streamovaní dát a takisto aj o štandarde Universal Business Language. Dôraz sa kladie najmä na technológie veľkých dát akými sú Apache Kafka, ElasticSearch, Kibana a Logstash. Pomocou týchto technológií je navrhnutá architektúra na spracovanie a analýzu dát. Táto architektúra sa spúšťa prostredníctvom docker kontajnerov. Ďalším bodom tejto práce je implementácia softvéru, ktorý je založený na princípe komunikácie medzi firmami. Komunikácia v tomto softvéri je inicializovaná na základe troch vstupných parametrov - počet firiem, počet obchodných partnerov a počet správ posielaných jednou firmou druhej firme. Naimplementový klient je následne otestovaný v troch testovacích scenároch, kde sú rôzne nastavenia vstupných parametrov. Zaindexované dokumenty sa následne vizualizujú v Kibane. Záverečná časť práce obsahuje vyhodnotenie testovania, na základe ktorého je uvedené, ktorý parameter najviac vplýva na komunikáciu.

Kľúčové slová

edi, streamovanie dát, ubl, faktúra, apache kafka, elasticsearch, kibana, logstash

Abstrakt v AJ

Design of architecture for the processing and analysis of the transaction business data is a thesis which engage in efficiency improvement of exchange of electronic business data. The thesis administer theoretical overview about electrical data exchange, data streaming, and as well as Universal Business Language. It lays emphasis on big data technologies such as Apache Kafka, ElasticSearch, Kibana and Logstash. By means of these named technologies the architecture is designed for data processing and analysis. The architecture turns on via docker containers. Another item in thesis is software implementation, which is established on principle of communication between companies. Communication in this particular software is initiated on three basic input parameters - quantity of companies, quantity of business partners and quantity of messages send by one firm to another. Implemented client is then tested in three scenarios, where each has different setting of input parameters. Indexed documents are then visualised in Kibana. Final part of the thesis consist of test evaluation, which presents whichever parameter influence the communication most.

Kľúčové slová v AJ

edi, data streaming, ubl, invoice, apache kafka, elasticsearch, kibana, logstash

57103

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra kybernetiky a umelej inteligencie

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Hospodárska informatika**

Názov práce:

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát.
Design of architecture for the processing and analysis of the transaction business data.

Student: **Bc. Kamil Strachan**
Školiteľ: **Ing. Peter Bednár, PhD.**
Školiace pracovisko: **Katedra kybernetiky a umelej inteligencie**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:
1. Podať teoretický prehľad problematiky elektronického obchodovania a výmeny obchodných dokumentov
2. Navrhnúť a implementovať vhodnú architektúru pre spracovanie obchodných transakcií s využitím technológii spracovania Veľkých Dát
3. Otestovať a vyhodnotiť implementovanú architektúru pomocou vhodných testovacích scenárov
4. Vypracovať dokumentáciu podľa pokynov katedry a vedúceho práce

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 04.05.2020
Dátum zadania diplomovej práce: 31.10.2019

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice 4. 5. 2020 .....

Vlastnoručný podpis

Poďakovanie

Touto formou by som sa rád poďakoval vedúcemu tejto diplomovej práce, Ing. Petrovi Bednárovi, PhD., za pripomienky a odbornú pomoc pri vypracovaní diplomovej práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Ľudia vo všeobecnosti obľubujú zjednodušovanie a zefektívňovanie práce. Preto sa kladie veľký dôraz na vývoj technológií, ktoré vyššie spomenuté veci zabezpečujú. Obchodný sektor je jedno z odvetví, ktorých sa to týka. Elektronická výmena obchodných dokumentov je ekonomickejšia a navyše aj ekologickejšia oproti papierovej výmene týchto dokumentov. Dokonca, elektronická výmena zabezpečuje aj to, že takáto komunikácia prebieha veľmi rýchlo. V dnešnej dobe môžu tento proces uľahčiť aj technológie Veľkých dát. Tieto technológie zabezpečujú to, že vedia pracovať s obrovským množstvom dát, rôzneho druhu a s veľkou rýchlosťou.

Cieľom tejto záverečnej práce je návrhnúť architektúru na spracovanie a analýzu transkačných obchodných dát. Daná architektúra bude využívať technológie veľkých dát. V kapitole 2 sa zaoberáme podobnými existujúcimu riešeniami danej problematiky. V nasledujúcej kapitole si urobíme úvod do problematiky elektronického obchodovania a výmeny elektronických obchodných dokumentov. Venovať sa budeme aj streamovaniu dát, ktoré daný proces urýchľuje. Uvedieme si rozdiely medzi streamovaním a dávkovým spracovaním dát, prečo využívať streamovanie a uvedieme si aj príklady, kde sa stretávame so streamovaním. Elektronický prenos obchodných dokumentov bude v našej práci mať za úlohu technológia Apache Kafka, ktorá je považovaná za populárnu streamovaciu platformu. Program, ktorý naimplementujeme bude fungovať na princípe výmený dokumentov medzi firmami v prostredí. Dáta, ktoré si medzi sebou firmy budú posielať sú faktúry v štandarde Universal Business Language. UBL je XML štandard, v ktorom sú zostavované obchodné dokumenty ako faktúra, objednávka a podobne. Štandard UBL a Apache Kafka si bližšie špecifikujeme v kapitole 6 a 7. Časťou našej práce bude aj implementovanie konverzie zo štandardu UBL do JSON formátu na jednoduchšiu prácu a pristupovanie k atribútom v rámci komunikácie medzi firmami. Konvertovanie a takisto softvérové prostredie firiem si implementujeme v jazyku Python, kedže tento jazyk je v dnešnej dobe nesmierne populárny a má jednoduchú syntax. Ak už budeme mať vytvorené softvérové prostredie, kde medzi sebou budú komunikovať firmy, tak prejdeme k testovaniu, ktoré si vytvoríme na základe vhodných testovacích scenárov. Po otestovaní si chceme dané obchodné dokumenty aj vizualizovať, k čomu nám dopomôže funkčný ekosystém medzi databázou ElasticSearch a webovým rozhraním Kibana, kde sa dáta vedia ľahko vizualizovať. Implementáciu navrhnutej architektúry si popíšeme v kapitole 5. Navrhnuté testovacie scenáre a implementovaného testovacieho klienta si popíšeme v kapitole 9.

V poslednej kapitole tejto záverečnej práce si vyhodnotíme dané testovanie a bližšie si popíšeme výsledky, ktoré sme prostredníctvom tohto testovania získali.

1 Formulácia úlohy

Naša diplomová práca obsahuje jednotlivé úlohy, ktoré je nevyhnutné vypracovať, aby sme mohli túto diplomovú prácu považovať za úspešnú.

Úlohy na vypracovanie:

- Jednou z hlavných úloh tejto práce je oboznámiť sa s elektronickou výmenou dát a touto problematikou. Táto časť následne bude zahrňať aj priblíženie práce so štandardizovaným formátom obchodných dokumentov Universal Business Language. Keďže v tejto práci budeme pracovať výhradne s faktúrami, je nevyhnutné aby sme si naštudovali aké atribúty obsahuje tento obchodný dokument a následne odkonzultovali dôležitosť týchto atrobútov a zvolili si, s ktorými budeme ďalej pracovať.

- Našou ďalšou úlohou bude naimplementovať konverziu z formátu XML na formát JSON. Po konzultácií s vedúcim práce sme sa rozhodli, že budeme pouzívať jazyk Python, vzhľadom na jeho obrovskú popolaritu v dnešnej dobe. K jednotlivým atribútom budeme pristupovať pomocou dictonaries.

- Kedže chceme spracovávať obchodné dokumenty v reálnom čase je nevyhnutné aby sme použili streamovaciu platformu na báze publisher/subscriber. Po konzultácií sme sa rozhodli, že použijeme Apache Kafku. Pomocou tejto technológie navrhneme architektúru výmeny obchodných dokumentov.

- Obchodné dokumenty chceme takisto uložiť do databázy. Databázu, ktorú sme sa rozhodli použiť je databáza ElasticSearch. V tomto prípade budeme musieť prepojiť Kafku s ElasticSearchom a zaindexovať tieto obchodné dokumenty aby sme následne vedeli s nimi pracovať. Prepojenie týchto dvoch technológii je možné prostredníctvom dátového kanálu, akým je napríklad dátový kanál Logstash.

- Implementácia testovacieho klienta a otestovanie výkonu je poslednou dôležitou časťou našej záverečnej práce. Testovať budeme aké výsledky má komunikácia, ktorá prebieha medzi firmami a to tak, že budeme meniť jednotlivé parametre pre všetky testovacie scenáre. Výsledky jednotlivých testovacích scenárov je nevyhnutné na záver popísať a vyhodnotiť.

2 Architektúry spracovania dát pomocou technológií Veľkých dát

Aby si používateľ dokázal vytvoriť akýkoľvek dashboard potrebuje dáta ako zdroj informácií, ktoré si chce zobraziť. V dnešnom svete množstvo dát pribúda a zväčšuje sa deň za dňom. To môžeme považovať za jeden z hlavných dôvodov zväčšujúceho sa záujmu a používania technológií Veľkých dát, ktoré predstavujú obrovské množstvo dát, ktoré prichádzajú z rôznych zdrojov, zložitých na spracovanie a analyzovanie softvérami a technológiami.

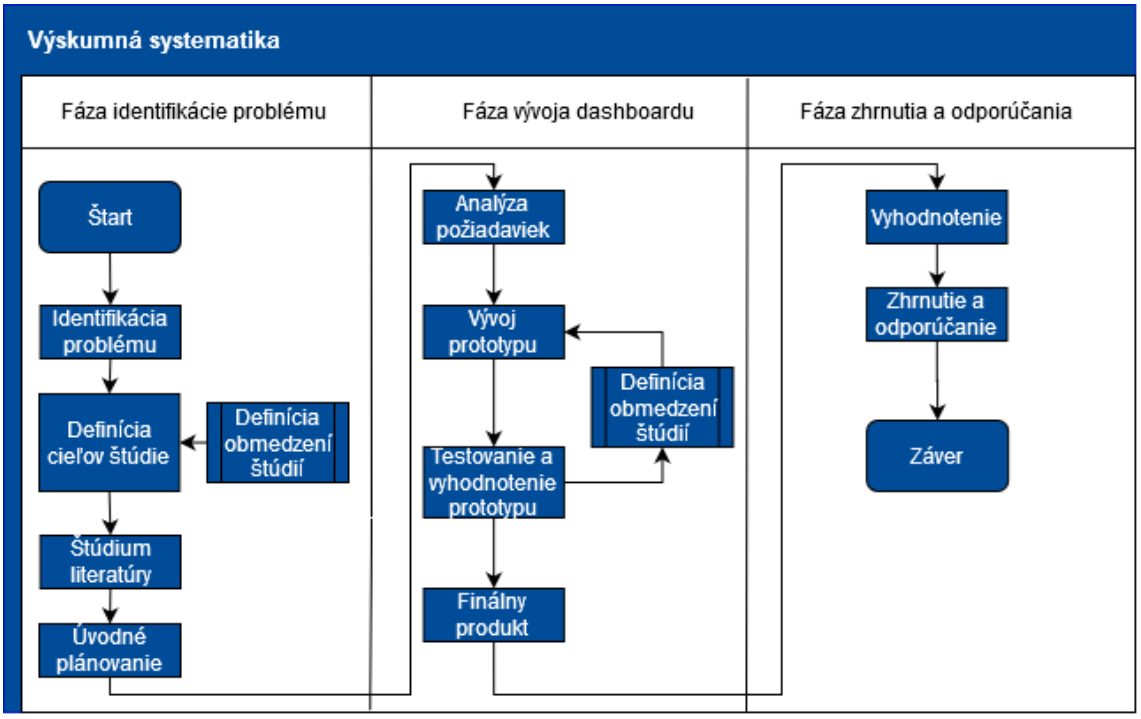
2.1 Architektúra spracovania dát na zrýchlenie transakcií

Prvé riešenie[17], ktorému budeme venovať pozornosť sa zaoberá architektúrou spracovania dát s využitím veľkých dát na zvýšenie rýchlosti transakcií. Autori si prácu na tomto projekte rozdelili na viacero fáz. Prvou fázou v tomto prípade bola identifikácia problému. Ten si takisto identifikovali na základe vypracovania podobných prác. Práca je založená na princípoch veľkých dát ich a vizualizácii.

Vo fáze vývoja dashboardu, autori využili metódu prototypovania, kedy je vývoj rozdelený na 4 fázy a to na - analýza požiadaviek, vývoj prototypov, testovanie a vyhodnotenie a na záver finálny produkt. Vo fáze analýzy požiadaviek autori identifikovali to, aké sú požiadavky na vývoj dashboardu. Na základe analýz sa architektúra a dizajn dashbordu vyvíjajú vo forme prototypu a následne je tento prototyp implemetovaný. Po implementácii bol produkt testovaný používateľom. Vo fáze testovania používateľ poskytoval svoj názor a odporúčania na používanie dashboardu. V prípade ak používateľ dal akékoľvek odporučanie tak sa prototyp predizajnoval, nainmplementoval a znova začal testovať používateľom. Tento proces sa opakoval pokiaľ používatelia nemali žiadne pripomienky alebo odporúčania. Tento prototyp sa nazýva finálny produkt. V poslednej fáze sa vývoji zameriali na vyhodnotenie ich výskumu, zhrnuli si ho a dali odporúčania ako pre budúcich výskumníkov. Popísané fázy sú zobrazená na obrázku nižšie, ktorý sme sa rozhodli pre lepšie porozumenie preložiť do slovenského jazyka.

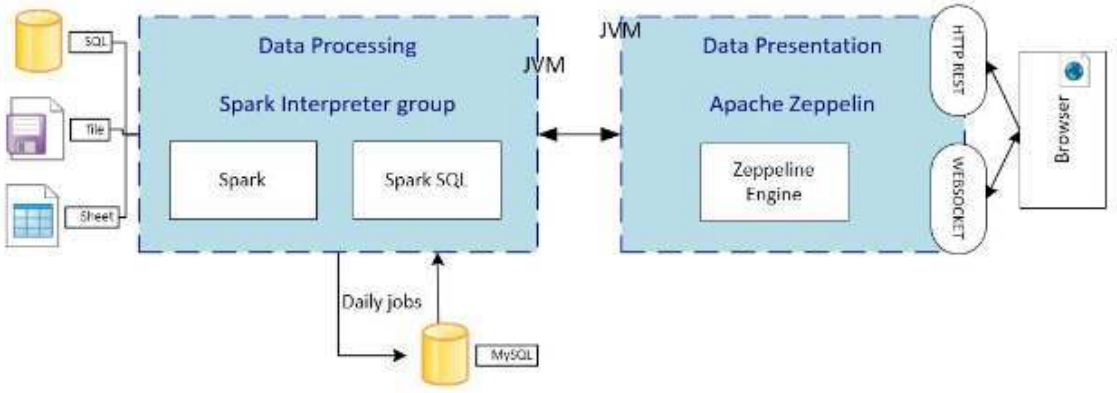
Táto štúdia sa zameriava na spracovanie dát, ktoré sú zobrené v dashboarde vo webovom rozhraní, použitím technológií veľkých dát, ktoré sú open-source. Tento výskum sa venuje oblastiam, ktoré zahŕňajú organizáciu dát, spracovanie dát, uchovávanie dat a ich následné zobrazenie prostredníctvom dashboardu.

Výskum od navrhnutej architektúry očakával to, aby boli dáta zobrazené pomocou dashboardu, ktorý používa Apache Zeppelin s interpreterom Apache Spark. Študia obsahuje 2 experimenty. Prvý experiment sa zameriava na spracovanie zdrojových dát prostredníctvom nástroja Apache Spark, ktoré sú zobrazené priamo v Apache Zeppelin. Druhý experiment využíva databázu SQL ako úložisko dát a denne vytvára joby spracovania dát. Pridávanie týchto jobov umožňuje automa-



Obrázok 2 – 1 Metodológia výskumu

tizovať spracovanie dát a tieto výsledky sa ukladajú do SQL databázy, aby sa tieto dáta zobrazovali ako dashboard. Návrh architektúry tejto štúdie je zobrazený na obrázku nižšie.



Obrázok 2 – 2 Architektúra spracovania dát na zrýchlenie transakcií

Spark je open-source nástroj, ktorý sa používa na spracovanie dát, zatiaľ čo výsledky sú prezentované v Apache Zeppelin. Každý nástroj beží na inom Java Virtual

Machine, rovnako ako výsledky spracovaných dát, ktoré sú uložené v databáze MySQL. Apache Zeppelin predstavuje webovú aplikáciu na tvorbu dashboardov, ktorý pracuje s rôznymi interpretermi dát a jedným z nich je aj Spark.

Zdrojom dát v tomto výskume boli súbor, SQL databáza a csv tabuľka, ktoré číta Spark a následne ich sparsuje a spracuje podľa obchodných potrieb. Dáta sa čítajú v podľa dátumu súboru. Následne sa vytvorí dočasný dátový rámec založený na transakčných dátových záznamov. Tento dátový rámec je dataset, ktorý sa ukladá do pomenovaných stĺpcov. To predstavuje akýsi konceptuálny ekvivalent k tabuľke v klasickej relačnej databáze. V prípade, že počet súborov vracia hodnotu null, tak by sa mal vytvoriť prázdny dátovy rámec. V žiadnom inom prípade by však nemal byť prázdny. Prázdny dátovy rámec sa vytvorí pretože null hodnoty by nemali byt pridané do agregovanej databázy. Dáta, ktoré sa konvertujú do dátových rámcov nie sú len transakčné dátové záznamy ale aj vyhľadávacie tabuľky z databázy MySQL, ktoré sa používajú na procesy dopytov.

Zosumarizovaním tejto štúdie prichádzame k záveru, že zoskupené dáta zo Spark SQL sú uložené v relačných tabuľkách, ktoré sú zdrojom informácii na ich zobrazenie v dashboarde. Na základe tohto výskumu a zistilo, že nástrojom, ktorý môže byť použitý na vývoj architektúry dashboardu fakturačného systému je Apache Zeppelin. Architektúra spracovania dát je rozdelená medzi denné spracovania dát a zobrazovanie týchto dát v dashboarde využitím toho istého nástroja, ktorým je Zeppelin[17].

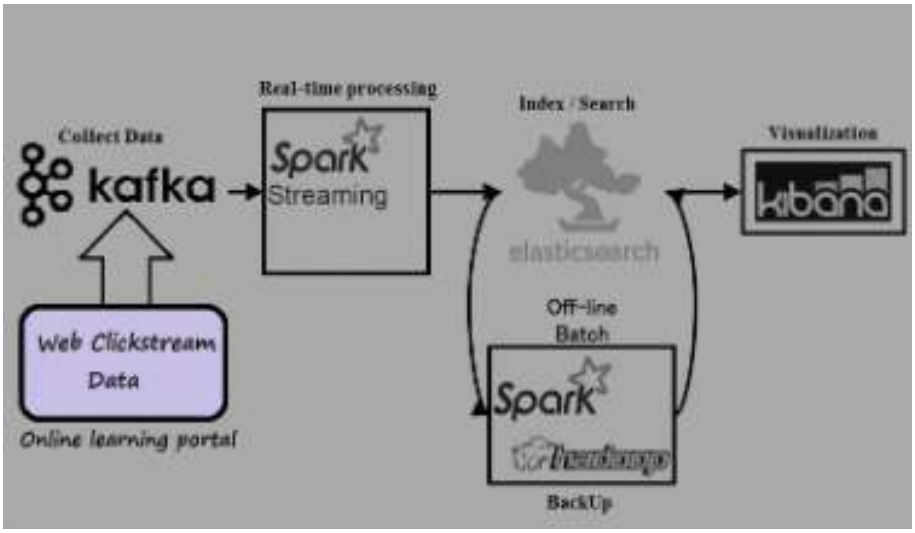
2.2 Architektúra na spracovanie Clickstream dát

Druhú študiu[8], ktorú si popíšeme je Analýza a vizualizácia Clickstream dát v reálnom čase. Clickstream dáta sú dáta o zaznamenávaní kliknutí klientov na stránky pri prezeraní daných stránok alebo využívaní iných aplikácií keď klient klikne kdekoľvek na webovej stránke alebo v aplikáciách. Táto aktivita sa zaznamená na webový server alebo na zákazníka, prípadne aj do smerovača, proxy servera alebo webového prehľadávača.

V tejto štúdii sa výskumníci zameriali na kompletné riešenie spracovania streamov a analyzovania týchto dát. Toto riešenie v sebe zahrňa použitie viacerých technológií veľkých dát. Technológia Hadoop sa používa na ukladanie a spracovanie veľkých datasetov akým sú napríklad Map Reduce joby. Ekosystém Hadoop-u sa skladá z viacerých nástrojov akými sú Spark, Kafka, Storm, Pig, Hive, Hbase, Oozie, Ambari, Mahour, Pheonix a podobne. Tieto nástroje môžu byť použité na riešenie rôznych problematík. Hadoop podporuje rôzne typy dát ako sú štrukturované, semištrukturované a neštrukturované dáta.

Informácie sa zbierajú zo vzdelávacieho online portálu. Výskumnici využili Apache Kafku na dátovú výmenu a Spark streaming na spracovanie prichádzajúcich dát. Následne sa zozbierané a spracované dáta vo formáte JSON pomocou spark streamingu odošlú do ElasticSearchu, ktorý sa používa na indexovanie, vyhľadávanie a

analýzu. Nasleduje využitie Kibany, ktorá sa používa na dátovú vizualizáciu.



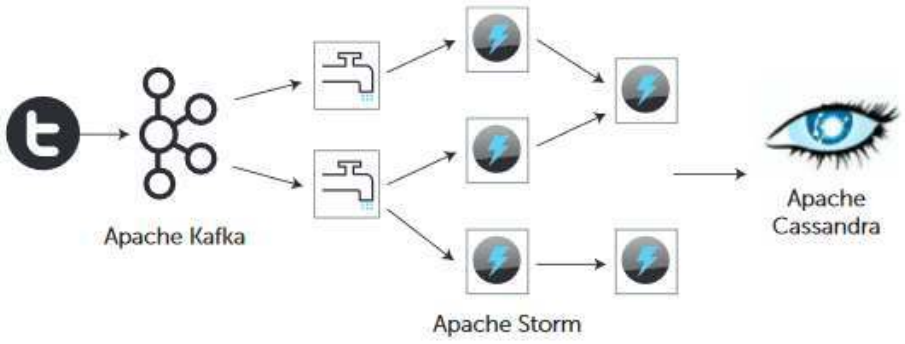
Obrázok 2 – 3 Návrh architektúry na spracovanie Clickstream dát

Všetky tieto technológie môžu byť použité na analýzu clickstream dát v reálnom čase. Obrázok vyššie opisuje všeobecnú štruktúru a dizajn procesu analýzy clickstream dát. Riešenie je implementované s využívaním týchto dát. To je založené na ich nepretržitom extrahovaní. Jeden klik na webovej stránke vytvorí kompletné informácie o používateľovi, čo zahŕňa dáta, ktoré boli nakonfigurované pri tomto výskume. Clickstream dáta z reálneho času sa ukladajú ako záloha v Hadoop-e pre pokročilejšie analýzy. Vedci v tomto výstupe vytvorili jedno uzlový Hadoop klaster, ktorý pracuje na štandardnom bežne dostupnom hardvéri[8].

2.3 Návrh architektúry na spracovanie dát v Cloude

Tretia štúdia[18], ktorej venujeme pozornosť je Streamovanie a spracovanie veľkých dát v cloudových dátových centrách. Štúdia popisuje to ako efektívne pomáhajú technológie veľkých dát ku znižovaniu hardvérových nárokov. Výskumníci sa zamerali takisto na využitie cloudových dátových centier, ktoré poskytujú priamy prístup k rozsiahlym zdrojom výpočtovej techniky.

Obrázok nižšie zobrazuje navrhnutú architektúru. Apache Kafka slúži ako vysoko výkonný distribuovaný systém zasielania správ. Technológia Apache Storm slúži na distribuované výpočty prebiehajúce v reálnom čase, ktoré sú odolné voči chybám a Apache Casandra v tejto architektúre predstavuje úložisko NoSQL databázy. Výskumníci chceli zjednodušiť proces spracovania dát v reálnom čase a to tak, že si vytvorili zoznam kritérií, ktoré im mali pomôcť pri tomto vývoji - architektúru,



Obrázok 2 – 4 Návrh architektúry na spracovanie dát v Cloude

podporu programovacieho jazyka, integráciu s inými technológiami, kvalitnú dokumentáciu a to aby mali podporu komunity.

Dimenzie architektúr sa delia na centralizované, distribuované a paralelne distribuované systémy. Centralizované systémy streamovania v pamäti sú vhodné na manipuláciu s dopytmi, ktoré majú požiadavku na nízku latenciu a nevytvárajú veľa dát o prechodných stavoch. Ako príklad si môžeme uviesť Esper, ktorý predstavuje streamovací systém s centralizovanou architektúrou, ktorý beží na jednom uzle a uchováva všetky dáta v pamäti. Ak sú však nepretržité dopyty prílíš veľké a mohli by si vyžadovať milióny n-tíc za sekundu je vhodnejšie použiť systém s paralelne distribuovanou architektúrou, akú predstavuje napríklad Apache Samza, ktorá umožňuje rozdeliť toky na viaceré partície a takisto aj paralelizovať výkon operátorov naprieč celým klastrom strojov.

Podpora jazyka ukazuje na flexibilitu framework-u tým, že umožní pracovníkom tímu vyvinúť aplikáciu podľa ich vlastného výberu jazyka. V tomto prípade je dobrým príkladom Apache Storm, ktorý podporuje JVM ale aj iné programovacie jazyky. Ďalšou dôležitou dimenziou je technologická integrácia. Tá predstavuje dostupnosť knižníc pripravených na pripojenie systému k rôznym technológiám. Napríklad v tomto prípade, Apache Kafka je vysoko priepustný systém distribúcie správ v pamäti, ktorý dopĺňa každý systém spracovania toku a má pripravenú súčasť pre túto integráciu[18].

3 Elektronická výmena dát - EDI

EDI prebieha vo forme computer-to-computer, čiže z počítača na počítač. Elektronická výmená dát je výmena obchodných documentov v štandardnom elektronickom formáte medzi obchodnými partnermi[28].

Výhody elektronickej výmeny dát:

- Procesy sú jednoduchšie, čo znamená, že náklady sú nižšie.
- Rýchlejšie a transparentnejšie obchodné procesy.
- Zlepšovanie vzťahov medzi obchodnými partnermi vďaka koordinovaným procesom.
- Vysoká kvalita dát, ktorá nevyžaduje žiaden manuálny zásah.
- Veľmi veľká prispôsobivosť[27].

Hlavným cieľom elektronickej výmený dát je nahradiť papierové dokumenty elektronickými dokumentami. S týmto cieľom sa spája aj zníženie nákladov, ktoré sú použité na ich výmenu a takisto aj zefektívnenie kvality týchto procesov. Právne majú rovnakú váhu elektrické dokumenty, tak ako papierové.

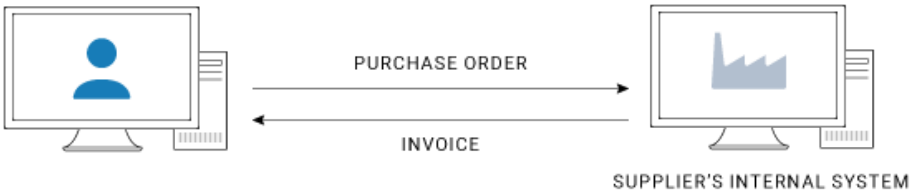
Rozdiel medzi EDI a papierovou výmenou dokumentov je následovný. Objednanie tovaru bez využitia EDI:

- Zákazník vytvorí objednávku vo svojom informačnom systéme.
- Zákazník vytlačí objednávku zo systému a pošle ju emailom alebo faxom.
- Dodávateľ objednávku príjme emailom alebo faxom a prenesie si ju do papierovej formy tým, že si ju vytlačí.
- Dodávateľ si objednávku prepíše z papierovej podoby do svojho informačného systému.
- V poslednom kroku sa objednávka spracováva, vyskladní, tovar sa doručí a vyfaktúruje.

Obrázok 3 – 1 Proces papierovej výmeny obchodných dokumentov

Objednanie tovaru s využitím elektronickej výmeny dát:

- Zákazník vo svojom informačnom systéme vytvorí objednávku. Objednávka, ktorú zákazník vytvoril odchádza cez EDI dodávateľovi v elektronickej podobe.
- Dodávateľ príjme cez EDI objednávku do svojho systému. V momente keď dorazí objednávka elektronicky, prijatie vyzerá tak, ako keby objednávka automaticky vznikla v informačnom systéme dodávateľa.
- Nasleduje priebeh spracovania objednávky, vyskladnenie, dodanie tovaru a fakturácia. Výmena všetkých dalších dokladov môže takisto prebiehať cez EDI[19].



Obrázok 3 – 2 Proces eletronickej výmeny obchodných dokumentov

3.1 Správy elektronickej výmený dát

Vďaka elektronickej výmene dát je umožnené obchodným partnerom vymieňať si medzi sebou typy správ, ktoré sú rôzne štandardizované. Ide o akékoľvek dokumenty, ktoré si vymieňajú medzi sebou obchodní partneri. Najčastejšie vymieňanými dokumentami sú dokumenty o faktúrach, objednávkach alebo aj elektronické dodacie

listy. Týchto správ je obrovské množstvo, keďže EDI dokáže pokryť v podstate všetky obchodné a logistické obchodné procesy.

Typy:

- ORDERS - jedná sa o objednávky a je to typ správy, ktorú posiela zákazník dodávateľovi, aby si objednal tovar alebo napríklad aj služby v určitom množstve. V tomto type správy môže byť špecifikovaný termín a miesto dodania.
- ORDRSP – je to potvrdenie objednávky. Táto správa je odpoveďou na správu ORDERS, ktorú posiela dodávateľ zákazníkovi. Potvrdenie objednávky sa posiela zákazníkovi aby vedel, či mu objednávka bude dodaná alebo nie.
- INVOIC - je to dokument, ktorý popisuje faktúru a vypovedá o prehľadu zásob. Tento typ správy umožňuje dodavateľovi a aj zákazníkovi výmenu informacií o súčasných resp. plánovaných skladových zásobách. Táto správa je väčšinou doplnená o informaciu o počiatočnom a aj súčasnom stave skladu.
- DESADV - je to správa, ktorá podľa podmienok špecifikuje podrobnosti o dodanom tovare. Tieto podmienky si medzi sebou dohodli predávajúci a kupujúci. Správne by mala byť tato správa zasielaná vždy ešte před tým, ako tovar fyzicky dorazí. Takto má zákazník presné informácie o tovare a môže sa pripraviť na prevzatie tovaru. Dokument DESADV je obdobná verzia dodacieho listu.
- COMDIS - obchodná námietka je typ správy, ktorú posiela odberateľ predávajúcemu v prípade, že sa v prijatej faktúre vyskytne nejaká nejasnosť. Touto nejasnosťou môže byť zlá cena, zlý popis tovaru alebo aj situácia, že tovar nebol vôbec dodaný. Ak dodávateľ obdrží túto správu, znamená to, že odberateľ odmieta faktúru, ktorá obsahuje tieto chyby a požaduje korekciu týchto chýb.
- PRICAT - ide o katalóg, ktorý obsahuje ponúkaný tovar a jeho ceny. Táto správa sa využíva dvoma spôsobami. Aktuálny katalóg ponúkaného tovaru dodávateľom je prvý prípad. Druhý prípad je keď dodávateľ chce oznámiť zmenu v ponúkanom sortimente[23].

3.2 Elektronický podpis

Dokumenty je možné a niekedy aj nutné ochrániť podpisom. V niektorých prípadoch stačí vlastnoručný podpis ale inokedy je nutné notársky overiť vlastnoručný podpis. Rozdiel medzi vlastnoručným podpisom a elektronickým podpisom je taký, že ak by sme chceli preskúmať vlastnoručný podpis tak by sme potrebovali k takémuto skúmaniu grafológa, ktorý by pozorne skúmal akým ťahom bolo pero vedené. Ďalej by sa zaoberal sklonom a tvarom jednotlivých písmen. Takisto by sa mohol venovať

aj tomu, aký atrament bol použítý, prípadne na aký papier bol podpis nanesený. Ak by sme sa už ale bavili o elektronickom podpise, ktorý sa prikladá k elektronickým dokumentom, tak skúmať veci ako pri normálnom podpise by nemalo význam. Keď ide o elektronický podpis, tak relevantná by bola debata o hodnote elektornického podpisu a overovaní toho, či hodnota tohto podpisu je je skutočne taká, aká má byť[16].

Elektronický podpis sú vlastne údaje, ktoré majú dátovú podobu a v počítači nahrádzajú podpis, ktorý sme vytvorili vlastnoručne. E-podpis je s dátovou správou logický spojený alebo sa dá priamo pripojiť k nej. Využívá sa v dátových správach na jednoznačné overenie identity osoby, ktorá je podpísaná. Tým, že do elektronického dokumentu pridáme e-podpis, tak zabezpečíme dôveryhodnosť daného dokumentu.

To, či je elektronický podpis dôveryhodný sa zaisťuje tým, že prenesieme dôveru na tretiu stranu, ktorú považujeme takisto za dôveryhodnú. Tým, že spojíme dáta verejnej časti certifikátu s privátnou časťou pomocou certifikačných autorít, tým zabezpečíme overenie podpisu. Verejná časť certifikátu je verejný kľúč, ktorý posielame a privátnú časť máme uloženú v systéme svojho počítača.

Dátová správa, ktorú vydáva poskytovateľ certifikačných služieb je certifikát. Ten pája dáta na overovanie elektronických podpisov s osobou, ktorá je podpísaná. Týmto krokom sa overuje identita. Môžeme to prirovnať ku overeniu vlastnoručného podpisu pri listine s tým rozdielom, že sa jedná o elektronický podpis[25].

Celý princíp prenosu dôvery je založený na asymetrickej kryptografii. E-podpis si však nepredstavujte ako obrázok vášho vlastnoručného podpisu, ktorý ste vložili do faktúry. Jedná sa o matematický výpočet, čiže nie je to odfotený alebo naskenovaný obrázok. Pri tvorbe elektornického podpisu sa ako prvé vypočíta odtlačok dokumentu, ktorý sa nazýva hash. Hash je malé číslo, ktoré má veľkosť niekoľkých stoviek bitov. Jednou z vlastností hashu je napríklad to, že ak vstupný dokument minimálne upravíme, výsledný hash sa veľmi výrazne zmení. To, že by dve správy mali rovnaký hash je matematicky nepravdepodobné. To znamená, že podľa hashu vieme jednoducho identifikovať práve jeden dokument. Elektronický podpis vzniká tak, že v momente keď sa vytvorí hash, systém ho zašifruje autorovým privátnym kľúčom.

Elektronický podpis je autentickejší a miera jeho bezpečnosti je väčšia ako pri klasickom papierovom podpise vytvoreným vlastnoručne. E-podpis je navyše možné jednoducho aplikovať na veľké množstvo dokladov[26].

Elektronický podpis môžeme získať výhradne od kvalifikovaných certifikačných autorít. Na Slovensku tieto služby poskytujú podľa Národného bezpečnostného úradu napríklad "Prvá slovenská certifikačná autorita (PSCA)", "Certifikačná autorita CA Disig", alebo aj "Certifikačná autorita Ministerstva obrany SR (CAMOSR)". Platnosť elektronického podpisu je 1 rok[25].

4 Streamovanie dát

Streamovanie patrí medzi jednu z kľúčových technológií, ktoré sa používajú na získanie informácii z veľkých dát. Je to proces neustáleho prenosu, príjmania a spracovania dát. Streamovanie dát je kľúčovou schopnosťou pre organizácie, ktoré chcú generovať analytické výsledky v reálnom čase. Hodnota dát zo streamu spočíva v schopnosti ich spracovať a analyzovať ihneď po ich príchode[4].

Aby sme lepšie pochopili streamovanie dát, tak si ho porovnáme s tradičným dávkovým spracovaním. Dávkové spracovanie sa môže použiť na výpočet ľubovoľných dopytov na rôzne súbory údajov. Spravidla počíta výsledky, ktoré sú odvodené zo všetkých údajov, ktoré obsahuje, a umožňuje hĺbkovú analýzu veľkých súborov dát. Systémy založené na MapReduce, ako napríklad Amazon EMR, sú príklady platforiem, ktoré podporujú dávkové úlohy. Na rozdiel od toho spracovanie stremov vyžaduje príjem sekvencií údajov a postupnú aktualizáciu metrík, správ a súhrnných štatistík v reakcii na každý prichádzajúci dátový záznam. Stremovanie dát je vhodnejšie na monitorovanie v reálnom čase. Pre lepší prehľad sme si uviedli porovnanie streamovania a dávkového spracovania v tabuľke, ktorá je zobrazená nižšie[29].

	Streamovanie	Dávkové spracovanie
Rozsah dát	Dopytovanie alebo spracovanie údajov v rámci priebežného časového okna alebo iba v najnovších dátových záznamoch.	Dopytovanie alebo spracovanie nad väčšinou alebo nad všetkými dátami v datasete
Veľkosť dát	Individuálne záznamy, mikro dávky, ktoré sa skladajú z niekoľkých záznamov	Veľké dávky dát
Výkon	Trvanie v sekundách alebo milisekundách	Trvanie v minútach až hodinách
Analýzy	Jednoduché funkcie odozvy, agregácie a metriky	Komplexné analýzy

Tabuľka 4 – 1 Streamovanie vs. dávkové spracovanie

Dobrými príkladmi streamovania údajov sú napríklad obchody s akciami v reálnom čase, spravovanie skladových zásob a takisto aj aplikácie na zdieľanie jazdenia. Napríklad, keď si cestujúci zavolá taxi službu, aplikácia vie nielen to, ktorého vodiča prideliť, ale vie aj to ako dlho bude trvať príchod vodiča a to na základe údajov o polohe v reálnom čase ale takisto aj z historických údajov o premávke. Aplikácie vie takisto aj to koľko by mala jazda stáť na základe dát v reálnom čase ale aj údajov z minulosti[20].

Streamovanie dát je výkonný nástroj avšak je tu pár výziev s ktorými sa stretá-

vame. Nasledujúci zoznam ukazuje zopár vecí, ktoré by sme si mali naplánovať ak sa rozhodneme využívať streamovanie dát[1]:

- Škálovateľnosť - ak náhodou dôjde k tomu, že systém zlyhá, množstvo záznamov prichádzajúcich z každého zariadenia sa môže zvyšiť. Pridávaním väčšej kapacity, zdrojov a serverov nastáva škálovanie aplikácie okamžite a exponenciálne sa zvyšuje množstvo generovaných nespracovaných dát. Pri práci so streamovaním je rozhodujúce navrhnúť aplikáciu tak aby sme ju mohli škálovať.

- Usporiadávanie - nie je jednoduché určiť postupnosť dát v streame a v mnohých aplikáciach je to veľmi dôležité. Chat alebo iná konverzácia by nedávala zmysel ak by nebola usporiadaná. Často sa vyskytujú nezrovnalosti medzi poradím vygenerovaného dátového balíka a poradím, v ktorom sa dostane do cieľa. Vyskytujú sa aj nezrovnalosti v časových pečiatkach a hodinách zariadení, ktoré generujú dáta. Pri analýze streamov musia byť aplikácie navhrnuté tak aby transakcie boli nedeliteľné, konzistentné, nezávislé a odolné, čiže ACID.

- Konzistentnosť a odolnosť - konzistentnosť dát a prístup k nim je vždy veľký problém pri spracovaní dát zo streamu. Dáta, ktoré sú už prečítané v akomkoľvek danom čase už mohli byť upravené a zastaralé v inom dátovom centre kdekoľvek na svete. Odolnosť dát považujeme za výzvu takisto pri práci so streamami v Cloude.

- Tolerancia chýb a garancia dát - patria medzi dôležité aspekty pri práci s dátami, spracovaním streamu alebo akýmikoľvek distribuovanými systémami. Keď dáta prichádzajú z rôznych zdrojov, miest a v rôznych formátoch a množstve je dôležité aby sa systém vedel vyhnúť rôznym narušeniam a takisto aj ukladáť dátové streamy, tak aby boli dostupné a odolné[20].

S narastom využívania streamovania prichádza takisto aj množstvo riešení zameraných na prácu so streamovaním. Nasledujúci zoznam zobrazuje niekoľko populárnych nástrojov na prácu s dátovými streamami:

- Amazon Kinesis Firehose - táto technológia je spravovaná, škálovateľná Cloud služba, ktorá umožňuje spracovanie veľkých dátových streamov v reálnom čase.

- Apache Kafka - je distribuovaný systém zasielania správ na princípe publish-subscribe, ktorý je integrovaný do rôznych aplikácii a dátových streamov.

- Apache Flink - je to nástroj na spracovanie stremov, ktorý poskytuje prostriedky na distribuovaný výpočet cez dátové streamy.

- Apache Storm - je to distribuovaný výpočtový systém v reálnom čase. Apache Storm sa používa na distribuované strojové učenie, analýzu v reálnom čase a na mnoho dalších prípadov, kde sa pracuje s veľkým objemom dát[1].

V dnešnej dobe svet biznisu nechce čakať na dávkové spracovanie dát, namiesto toho chce aby všetko prebiehalo v reálnom čase od detekcie podvodov, akciového trhu, aplikácií so zdieľanou polohou a takisto aj komerčných webových stránok. To sa dosiahne tým, že aplikácie začnú používať streamovanie dát, ktoré nielen integrujú dáta, ale takisto ich aj spracovávajú, filtrujú, analyzujú a reagujú na tieto dáta v reálnom čase, v momente keď sú prijaté. Každé odvetvie, ktoré sa zaoberá veľkými dátami, môže mať úžitok z dát, ktoré prichádzajú nepretržite v reálnom čase.

Systémy spracovania streamov ako Apache Kafka a Confluent ponúkajú analýzu dát z reálneho času. V každom odvetví existujú prípady použitia na streamovanie dát. Schopnosť ich integrovať, analyzovať, odstraňovať problémy alebo robiť predikciu dát v reálnom čase nám ponúka nové prípady použitia akými sú:

- Dáta o polohe
- Detekcia podvodov
- Akciový trh v reálnom čase
- Marketing, predaje a biznis analýza
- Aktivita zákazníka
- Monitorovania a spravovanie interných informačných systémov
- Monitorovanie záznamov
- Strojové učenie a umelá inteligencia - prediktívna analýza prináša nové možnosti kombináciou súčasných dát a dát z minulosti pre jeden centrálny nervový systém[20].

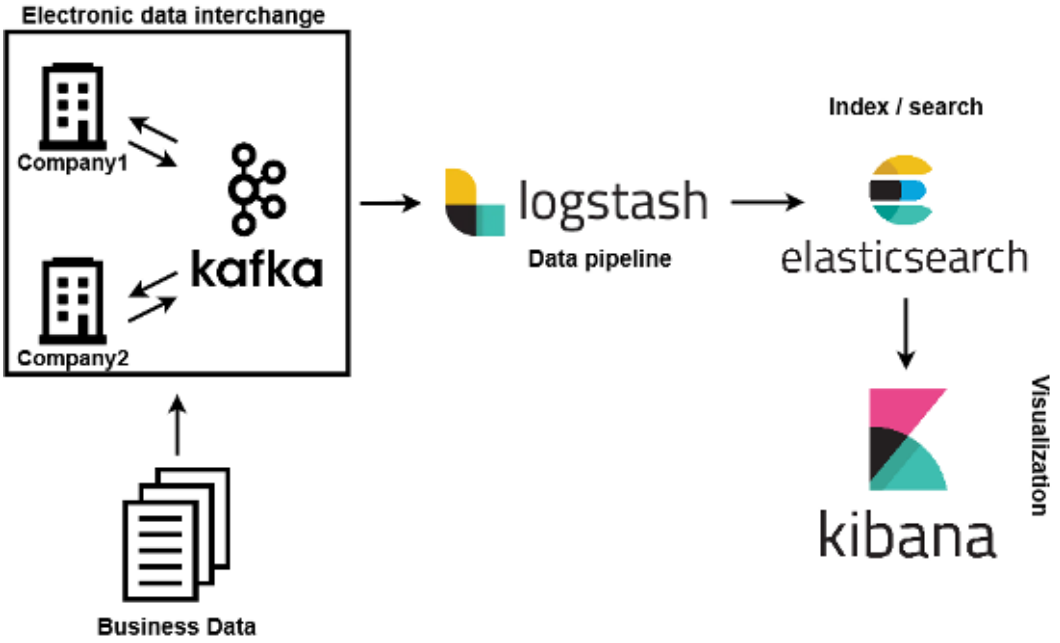
Ak chceme začať používať stremovanie dát, tak musíme dodržať a splniť požiadavky architektúry streamovania, ktorá sa skladá z týchto komponentov:

- Dátový zdroj, Producer - Najdôležitejšou požiadavkou spracovania stremov je jeden alebo viac zdrojov dát, ktorými sú Produceri. Produceri sú aplikácie, ktoré komunikujú s entitami, ktoré generujú dáta, a prenášajú ich na sprostredkovateľa správ.
- Sprostredkovateľ správ, broker - prijíma údaje od Producera a prevádza ich do štandardného formátu správ a potom posiela správy do nepretržitého streamu, ktorým je topic.

- Procesor streamov - prijíma dátové streamy od jedného alebo viacerých sprostredkovateľov a na dáta aplikuje užívateľom definované dopyty, aby ich pripravil na spotrebu a analýzu.
- Aplikácia consumera - Po tom, čo procesor streamov dát pripravil dáta, je možné tieto dáta streamovať do jednej alebo viacerých spotrebiteľských aplikácií[4].

5 Návrh architektúry na spracovanie a analýzu

Hlavnou časťou našej práce je navrhnúť architektúru, ktorá dokáže spracovať a analyzovať obchodné dokumenty, v našom prípade faktúry. Navrhnutá architektúra vyzerá následovne:



Obrázok 5 – 1 Navrhnutá architektúra

Prvým krokom je načítanie obchodných dokumentov, faktúr, zo zdrojovej databazy súborového systému. Po skonvertovaní faktúr na JSON formát môžeme začať s implementáciou Kafka klienta. Implementácia klienta je navrhnutá na princípe firiem, ktorej súčasťou sú Produceri a Consumeri. Tí nám umožňujú výmenú faktúr prostredníctvom Kafky. V tomto kroku pracujeme s faktúrami vo formáte JSON a cez Kafku tieto dokumenty posielame a príjmame. Princípom tejto architektúry je správne nastavenie klienta Apache Kafky, ktorý bude zabezpečovať prenos obchodných dokumentov medzi producermi a consumermi v určitých časových intervaloch. Dôležité je vytvoriť si topic, prostredníctvom ktorého bude Kafka vedieť, ktorému consumerovi má obchodné dokumenty posielať.

Následne je nevyhnutné zabezpečiť to, aby sa tieto dokumenty aj ukladali do databázy. Databázu, ktorú sme v tomto prípade použili sa volá ElasticSearch. ElasticSearch sme prepojili s Kafkou cez dátový kanál Logstash prostredníctvom konfiguračného súboru zmieneho dátového kanálu. V tomto kroku tento dátový kanál aj zaindexuje obchodné dokumenty, aby sme následne vedeli pracovať ďalej s týmito dátami. Konfiguračný súbor sme upravili tak, že na vstupe bude mať správy z Kafky, ktoré sa budú ukladať do ElasticSearchu.

Je nevyhnutné aby sme si vedeli dáta, ktoré sú uložené v elasticu zobraziť v nejakom rozhraní. Na tento úkon sme sa rozhodli, že použijeme webové rozhranie Kibana, kde budeme sledovať jednotlivé operácie vykonané nad dátami. V tomto rozhraní vieme takisto sledovať určité metriky databázy ElasticSearch.

5.1 Prepojenie Kafky s Elasticsearch

Na to aby sme prepojili Kafku s databázou ElasticSearch, budeme potrebovať dátový kanál. Na tento úkon sme sa rozhodli, že využijeme dátový kanál Logstash. Dôležité je správne vytvoriť konfiguračný súbor, cez ktorý sa budú správy z Kafky posielať priamo do elasticu a zároveň sa budú správy indexovať. Tento dátový kanál obsahuje 3 parametre, vstupný parameter, parameter filtrovania a výstupný parameter.

Vyššie spomenutý konfiguračný súbor logstash-u vyzerá následovne:

```
input {
   kafka {
       bootstrap_servers => "147.232.202.105:29094"
       topics_pattern => ["company.*"]
    }
}

filter {
   json {
      source => "message"
      remove_field => ["message"]
      target => "doc"
   }
}

output {
   elasticsearch {
      hosts => ["147.232.202.105:9200"]
      index => "businessdata"
      workers => 1
   }
}
```

Tento konfiguračný súbor obsahuje 3 parametre. Prvý parameter určuje, čo má mať tento dátový kánal na vstupe. Konfiguračný súbor má na vstupe Apache Kafku, na ktorú sa pripája na vzdialený server na port 29094 a na vstupe má všetky správy z topicov ktoré majú v názve company. Druhý parameter je v našom prípade filtračný parameter. Parameter sa zameriava na to, že jeho zdrojovými údajmi budú

údaje z atribútu message. Následne sa pole message vymaže a dáta sa uložia ako typ doc. Výstup konfiguračného súboru je nastavený na elasticsearch, ktorý je spustený takisto na vzdialenom serveri na porte 9092. Logstash zabezpečuje takisto aj indexovanie. Index je v našom prípade nazvaný "businessdata".

6 Universal Business Language

UBL je dostupný, prispôsobiteľný a rozšíriteľný XML slovník pre obchodné dokumenty. Obsahuje dokumenty ako sú objednávky, faktúry, sprievodky, ktoré sú posielané medzi obchodnými partnermi. Títo obchodní partneri sú kupujúci, predávajúci, posielajúci a sklady. UBL je určený na to, aby pomohol vyriešiť problémy definovaním všeobecného XML formátu pre obchodné dokumenty, ktoré môžu byť obmedzené alebo rozšírené tak, aby spĺňali požiadavky jednotlivých odvetví.

UBL ponúka následovné:

- Súbor štruktúrovaných obchodných objektov a ich pridružených sémantík, ktoré sú vyjadrené ako viac krát použiteľné dátové komponenty a bežné obchodné dokumenty.
- Knižnicu XML schém pre znova použiteľné dátové komponenty ako sú "adresa", "položka", "platba"- sú to bežné dátové štruktúry, ktoré sa vo svete obchodu používajú denno-denne.
- Sada XML schém pre bežné obchodné dokumenty ako sú "objednávka", "poradenstvo pri posielaní" a "faktúra" - sú vytvorené z dátových komponentov knižnice UBL a môžu byť použité v kontexte generického obstarávania a prepravy.

Štandardná báza pre XML obchodné schémy poskytuje tieto výhody:

- Nižšie náklady na integráciu v rámci podnikov a medzi podnikmi, prostredníctvom opätovného používania bežných dátových štruktúr.
- Lacnejší komerčný softvér, pretože softvér, ktorý je napísaný aby spracovával XML množinu značiek je jednoduchší na vývoj ako softvér, ktorý pracuje s neobmedzeným počtom množín značiek.
- Jednoduchšiu krivku učenia, pretože používatelia potrebujú ovládať jedinú knižnicu.
- Nižšie náklady na vstup a tým aj rýchlejšie prijatie zo strany malých a stredných podnikov.
- Štandardizované školenie, ktoré vedie k faktu, že je množstvo kvalifikovaných pracovníkov.
- Univerzálna a dostupná skupina systémových integrátorov.
- Štandardizované a lacné tooly pre vstup a výstup údajov.
- Štandardný cieľ pre lacný off-the-shelf obchodný softvér[9].

6.1 Obchodné procesy UBL

UBL má 8 základných obchodných procesov. Týmito procesmi sú:

- objednávka,
- odpoveď objednávky (jednoduchá),
- odpoveď objednávky (detailná),
- zmena objednávky,
- zrušenie objednávky,
- informácie o odoslaní,
- potvrdenie prijatia,
- faktúra[14].

6.1.1 Proces objednávania

Proces objednávky je vlastne vytvorenie zmluvného záväzku medzi predávajúcim a kupujúcim. Dokumenty, ktoré sa nachádzajú v tomto procese sú, objednávka, odpoveď objednávky, zmena objednávky, zrušenie objednávky.

Objednávka môže špecifikovať pokyny, ktoré sa týkajú zľiav a poplatkov, ktoré určujú typ poplatku a to, kto tie poplatky platí. Objednávka poskytuje viac riadkov objednávky. Objednávka môže špecifikovať dodacie podmienky, na druhej strane, riadok objednávky môže obsahovať pokyny na dodanie. Kupujúci môže tiež uviesť potenciálne prijateľné alternatívy.

6.1.2 Odpoveď objednávky (jednoduchá)

Odpoveď objednávky je prostriedok, v ktorom predajca potvrdzuje prijatie objednávky od kupujúceho, pričom uvádza, že záväzok splní bez zmeny alebo to, že objednávka bola odmietnutá.

6.1.3 Odpoveď objednávky (detailná)

Zmeny, ktoré sú navrhované v objednávke zo strany predávajúceho, sa uskutočňujú prostredníctvom úplného dokumentu odpoveď objednávky. Odpoveď objednávky navrhuje nahradiť originálnu objednávku. Odzrkadľuje nový stav transakcie objednávky.

Dokument objednávky je tiež prostriedok, ktorým predajca dodáva alebo potvrdzuje kupujúcemu detaily objednávky, ktoré v čase objednávky neboli k dispozícií alebo neboli špecifikované kupujúcemu v čase objednania.

To zahŕňa napríklad:

- dátum doručenia, ktorý ponúkne predajca ak dátum nebol špeciálne vyžiadaný kupujúcim,
- cenu,
- zľavy,
- poplatky,
- klasifikačné kódy položiek.

Predajca takisto môže poskytovať rady o výmene, náhradách alebo iných nevyhnutných zmenách prostredníctvom tohto dokumentu.

6.1.4 Zmena objednávky

Kupujúci môže zmeniť objednávku dvomi spôsobmi – podľa právnej zmluvy alebo dohody o obchodnom partnerovi. V prvom prípade, zaslaním zmeny objednávky alebo v druhom prípade zaslaním zrušenia objednávky, následným vytvorením novej úplnej objednávky. Zmena objednávky odráža aktuálny stav transakcie objednávky.

Kupujúci môže vyžadovať zmenu prijatej objednávky z rôznych dôvodov, ako sú - zmena objednaných položiek, množstva, zmena dátumu doručenia, zmena dodacej adresy a mnoho ďalších dôvodov. Dodávateľ môže prijať ale aj odmietnuť zmenu objednávky prostredníctvom odpovede na objednávku.

6.1.5 Zrušenie objednávky

Kupujúci môže zrušiť ustanovenú transakciu objednávky použitím dokumentu zrušenia objednávky v ktoromkoľvek bode procesu. To, či bude zrušenie objednávky prijaté alebo zamietnuté určujú právne zmluvy, dohody o obchodných partneroch a obchodné pravidlá.

6.1.6 Informácie o odoslaní

Informácie o odoslaní sú posielané odosielacou stranou pre doručovaciu stranu preto, aby potvrdili odoslanie položiek. Informácie o odoslaní nám poskytujú tieto 2 situácie:

- Prepravná jednotka sa stará o zásielku. Tým pádom môže príjemca kontrolovať zásielku a položky, ktoré prepravná jednotka obsahuje. Množstvo rovnakých položiek na objednávke sa môžu rozdeliť do rôzných prepravných jednotiek. To spôsobí, že sa objavia na samostatných expedičných linkách v rámci prepravej jednotky.

- Zásielky anotované prepravnou jednotkou, v ktorej sú umiestnené uľahčujú kontrolu oproti objednávke. Kvôli pohodliu sa každý riadok objednávky rozdelí medzi viacero prepravných jednotiek. Výsledkom toho bude expedičná linka pre každú prepravnú jednotku.

Informácie o odoslaní môžu v obidvoch prípadoch odporúčiť:

- Úplné dodanie – to znamená, že všetky položky objednávky budú alebo sú doručované príjemcovi alebo kupujúcemu v jednej kompletnej zásielke v daný dátum.
- Čiastočné dodanie - to znamená, že položky objednávky budú alebo sú doručované príjemcovi alebo kupujúcemu v samostatných zásielkach v daný dátum.

6.1.7 Potvrdenie prijatia

Tento dokument sa posiela dodacou stranou pre odosielaciu stranu na to, aby sa potvrdilo prijatie objednávky. Tento dokument takisto môže informovať aj o tom, že zásielka je poškodená alebo nedorazili všetky položky zásielky.

Tento dokument poskytuje 2 situácie:

- Údaj o tom, že zásielka bola prijatá od prepravnej jednotky a obsahoval potvrdenie s informáciami o odoslaní ako uvádza strana predajcu.
- Údaj o tom, že potvrdenia boli prijaté prepravnou jednotkou s informáciami o odoslaní ako uviedla strana predajcu.

6.1.8 Faktúry

Faktúra definuje finančné dôsledky obchodnej transakcie. Zvyčajne sa vydáva na základe jednej expedičnej udalosti, ktorá vyvolá jednu faktúru. Faktúru je možné vystaviť aj na predplatenie na celom alebo čiastočnom základe. Možnosti sú:

- Vopred zaplatená faktúra.
- Pro forma faktúra – neočakáva sa platba.
- Normálna faktúra – pri odoslaní expedovaných položiek.
- Faktúra po vrátení potvrdenia o prijatí.

Faktúra obsahuje len také informácie, ktoré sú nevyhnutné na fakturačné účely. Nejedná sa o informácie, ktoré už boli obsiahnuté v iných dokumentoch, ktoré nie sú potrebné pri fakturácií. V prípade nevyhnutnosti sa faktúra odkazuje na iné dokumenty.

Poplatky sa môžu určiť buď ako paušálna suma, alebo percentuálnym podielom, ktorý sa vzťahuje na celú hodnotu faktúry pred výpočtom daní. Tieto poplatky zahŕňajú:

- balenie,
- dodanie,
- poštovné,
- dokumentáciu[9].

6.2 Rozširenie UBL verzie - 2.2

Verzia UBL 2.2 obohacuje funkcionalitu UBL 2.1 zvýšením počtu definovaných dokumentov zo 65 na 81 dokumentov. Tieto dokumenty sú k dispozícií pre eTendering, obchodné adresáre a dohody a nový dopravný dokument o hmotnosti[10].

6.3 UBL Faktúra verzia 2.2

Po preštudovaní knižnice UBL sme si určili, ktoré komponenty budeme používať a ktoré používať nebudeme, čiže povinné a nepovinné. Faktúra má v UBL verzii 2.2 vo svojej štruktúre 54 základných komponentov. Po následnom výbere nám ostali tieto základné komponenty:

- **<cbc:ID>** - atribút, ktorý prideľuje faktúre ID, má typ string .
- **<cbc:IssueDate>** - atribút, ktorý hovorí o dátume, kedy bol tento dokument vydaný, prideľuje ho odosielateľ, typ je datum.
- **<cbc:IssueTime>** - atribút, ktorý hovorí o čase, kedy bol tento dokument vydaný, prideľuje ho odosielateľ, typ je čas .
- **<cbc:DueDate>** - atribút, ktorý hovorí o splatnosti faktúry, typ je dátum.
- **<cbc:PaymentCurrencyCode>** - atribút, kód označujúci menu použitú na platbu faktúry, typ je string.
- **<cac:InvoicePeriod>** - entita, ktorá popisuje obdobie, na ktoré sa faktúra vzťahuje, obsahuje subatribúty.
- **<cac:OrderReference>** - entita, ktorá odkazuje na objednávku, obsahuje subatribúty.
- **<cac:AccountingSupplierParty>** - entita účtovnej dodávateľskej strany.

- **<cac:AccountingCustomerParty>** - entita účtovnej zákaznickej strany.
- **<cac:Delivery>** - entita, ktorá popisuje dodanie.
- **<cac:PaymentMeans>** - entita, ktorá popisuje aké sú očakávané platobné prostriedky.
- **<cac:InvoiceLine>** - entita, ktorá popisuje položku faktúry.

Komponenty, ktoré v tagu začínajú písmenami CAC sú entity, ktoré obsahujú ďalšie vnorené atribúty. Pre príklad si uvedieme entitu, ktorá popisuje dodanie, čiže cac:Delivery:

- **<cbc:ID>** - identifikátor pre dodávku, má typ string.
- **<cbc:Quantity>** - atribút, ktorý hovorí o tom, aké je množstvo položiek obsahuje zásielka, má typ decimal.
- **<cbc:ActualDeliveryDate>** - atribút, ktorý hovorí aký je skutočný datum dodania, má typ dátum.
- **<cbc:ActualDeliveryTime>** - atribút, ktorý hovorí aký je skutočný čas dodania, má typ čas.
- **<cbc:LatestDeliveryDate>** - atribút, ktorý hovorí aký je najnovší dátum dodania, má typ dátum.
- **<cbc:LatestDeliveryTime>** - atribút, ktorý hovorí aký je najnovší čas dodania, má typ čas.
- **<cbc:TrackingID>** - identifikátor sledovania dodávky, má typ string.
- **<cac:DeliveryAddress>** - entita, ktorá popisuje adresu dodania, obsahuje tieto atribúty.
- **<cac:RequestedDeliveryPeriod>** - entita, ktorá popisuje požadovanú lehotu na dodanie.
- **<cac:PromisedDeliveryPeriod>** - entita, ktorá popisuje sľúbenú lehotu dodania.
- **<cac:EstimatedDeliveryPeriod>** - entita, ktorá popisuje odhadovanú lehotu dodania.
- **<cac:DeliveryTerms>** - entita, ktorá popisuje dodacie podmienky.

Ostatné komponenty, ktoré sme sa rozhodli používať majú podobné vetvenie a sú k dispozícií v jednej z príloh tejto záverečnej práce.

6.4 Konvertovanie UBL dokumentu do formátu JSON

Konverziu UBL dokumentov, faktúr sme implementovali prostredníctvom jazyka Python. Na túto konverziu sme využili Python knižnice xmltodict, json. Ukážeme si následný príklad.

Majme jednoduchú UBL faktúru vo formáte XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Invoice xmlns="urn:oasis:names:specification:ubl:schema:
   xsd:Invoice-2"
         xmlns:cac="urn:oasis:names:specification:ubl:
            schema:xsd:CommonAggregateComponents-2"
         xmlns:cbc="urn:oasis:names:specification:ubl:
            schema:xsd:CommonBasicComponents-2">
         <cbc:ID>TOSL108</cbc:ID>
         <cbc:IssueDate>2009-12-15</cbc:IssueDate>
         <cac:OrderReference>
                  <cbc:ID>123</cbc:ID>
         </cac:OrderReference>
         <cac:InvoiceLine>
                  <cac:Price>
                           <cbc:PriceAmount currencyID="EUR
                              ">0.75</cbc:PriceAmount>
                           <cbc:BaseQuantity unitCode="C62">
                              1</cbc:BaseQuantity>
                  </cac:Price>
         </cac:InvoiceLine>
</Invoice>
```

Z vyššie uvedených komponentov si urobíme slovník aby následná práca bola pre nás jednoduchšia. Kedže nám názvy jednotlivých atribútov resp. entít nevyhovujú, tak zmenu názvu týchto atribútov urobíme prostredníctvom funkcie pop(). Implementácia konverzie na JSON pre túto jednoduchú UBL faktúru bude vyzerať následovne:

```
import json
import xmltodict

with open('C:\\Users\\Kam o\\Desktop\\xmltojson\\
   xmlskuska.xml') as fd:
     doc = xmltodict.parse(fd.read())

doc['Invoice']['ID'] = doc['Invoice'].pop('cbc:ID')
doc['Invoice']['IssueDate'] = doc['Invoice'].pop('cbc:
```

```
    IssueDate')
doc['Invoice']['OrderReference'] = doc['Invoice'].pop('
    cac:OrderReference')
doc['Invoice']['OrderReference']['ID'] = doc['Invoice']['
    OrderReference'].pop('cbc:ID')
doc['Invoice']['InvoiceLine'] = doc['Invoice'].pop('cac:
    InvoiceLine')
doc['Invoice']['InvoiceLine']['Price'] = doc['Invoice']['
    InvoiceLine'].pop('cac:Price')
doc['Invoice']['InvoiceLine']['Price']['PriceAmount'] =
    doc['Invoice']['InvoiceLine']['Price'].pop('cbc:
    PriceAmount')
doc['Invoice']['InvoiceLine']['Price']['BaseQuantity'] =
    doc['Invoice']['InvoiceLine']['Price'].pop('cbc:
    BaseQuantity')

with open('ubl.json', 'w') as json_file:
    json.dump(doc, json_file, indent = 2)
```

Ak sme faktúru upravili podľa naších preferencií, tak ju uložíme do súboru. Ak daný súbor ešte neexistuje, tak ho program sám vytvorí.

7 Apache Kafka

Apache Kafka je open-source distribuovaný systém na zasielanie a príjmanie správ. Systém spĺňa tieto charakteristiky:

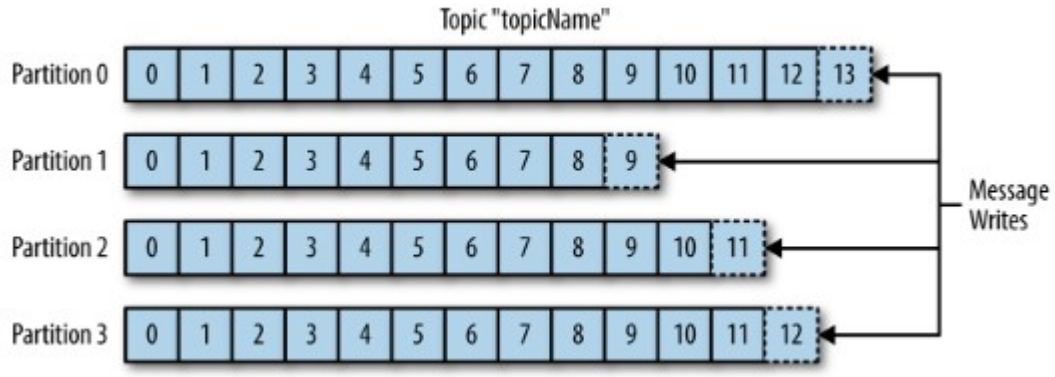
- **Neustále posielanie správ** - čerpá reálne hodnoty z veľkých dát, to znamená, že stratu akýchkoľvek dát si nemôže dovoliť.
- **Vysoká priepustnosť** - vzhľadom na Veľké dáta, je systém navrhnutý aby pracoval na komoditnom hardveri a podporoval milióny správ za sekundu.
- **Distribuovanosť** - explicitne podporuje rozdeľovanie správ cez Kafka servery.
- **Podpora viacerých klientov** - systém podporuje klientov z rôznych platforiem ako Java, .Net, PHP, Ruby a Python.
- **Reálny čas** - správy, ktoré vytvoril Producer by mali byť ihneď viditeľné pre Consumera.

Kafka ponúka riešenie publish/subscribe v reálnom čase[6]. Kafku vytvorila spoločnosť LinkedIn aby vyriešila problém s dátovým kanálom. Apache Kafka bol navrhnutý tak, aby poskytoval vysoko výkonný systém posielania správ, ktorý zvládne spracovávať veľké množstvo údajov. Kafka takisto poskytuje čisté a štrukturované dáta o aktivitách používateľa a systémových metrikách v reálnom čase.

7.1 Topicy a partície

Kafka kategorizuje správy do topicov. Databázova tabuľka alebo priečinok v súborovom systéme sú najbližšie analógie pre topic. Topicy sa ďalej delia na partície. Partícia popisuje jeden jediný log. Správy sa čítajú po poradí, od začiatku do konca.

Keď sa rozprávame o dátach v rámci systémov ako Kafka, tak často spomíname termín stream, čo je vlastne tok údajov. Stream sa často považuje za jeden topic dát, bezohľadu na to koľko má partícii. Predstavuje to jediný tok údajov, ktoré sa presúvajú z producerov do consumerov. Tento spôsob odkazovania na správy je najbežnejší keď ide o spracovanie dátového toku. To sa deje vtedy, keď frameworky - ako napríklad Kafka Streams, Apache Samza a Storm, operujú so správami v reálnom čase.



Obrázok 7 – 1 Topicy a partície

7.2 Producer and Consumer

Kafka má 2 typy používateľov, Producera a Consumera.

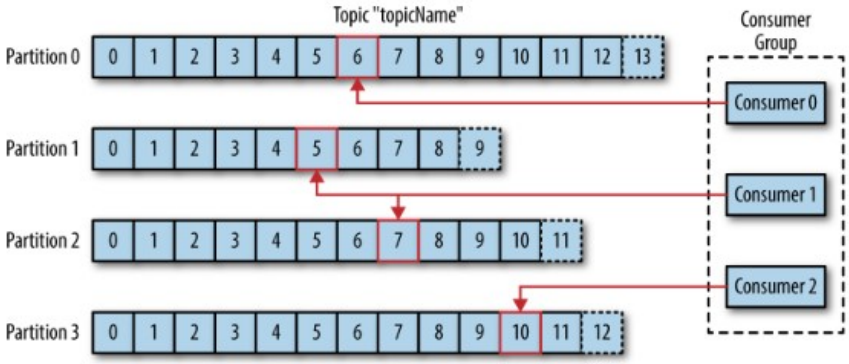
7.2.1 Producer

Úlohou Producera je vytvárať nové správy. V iných publish-subscribe ho poznáme ako publisher alebo writer. Správy sa vytvárajú ku konkrétnemu topicu. V pôvodnom stave sa Producer nestará o to, ktorá správa ma byť zapísaná do akej partície a rovnomerne vyvažuje správy medzi partície. V niektorých prípadoch Producer nasmeruje správy do špecifických partícií. Vykonáva sa to pomocou kľúča správ a oddielu, ktorý generuje hash kľúča a mapuje ho do špecifického oddielu. To znamená, že správy s rovnakým kľúčom budú zapísané do rovnakej partície. Producer môže využívať takisto aj vlastné oddiely, ktoré majú iné biznis pravidlá pre mapovanie správ ku partíciam.

7.2.2 Consumer

Consumer číta správy. V iných publish-subscribe ho poznáme ako subscriber alebo reader. Odoberá jeden alebo viac topicov a prečíta správy v takom poradí, v akom boli odoslané resp. vyprodukované. Consumer sleduje offset správ, ktorý ukazuje, ktoré správy už boli prečítané. Každá správa v danej partícií ma jedinečný offset. Uložením offsetu poslednej spotrebovanej správy pre každú partíciu, či už v ZooKeeperi alebo v Kafke, sa môže Consumer zastaviť a reštartovať bez toho, aby stratil svoje miesto. Viacero Consumerov spolu tvorí skupinu, ktorá spolu príjma správy na jednom topicu. Táto skupina zaisťuje to, že každá partícia má jedného člena, consumera.

Na obrázku nižšie sú traja consumeri v jednej skupine, ktoré prijmajú správy z

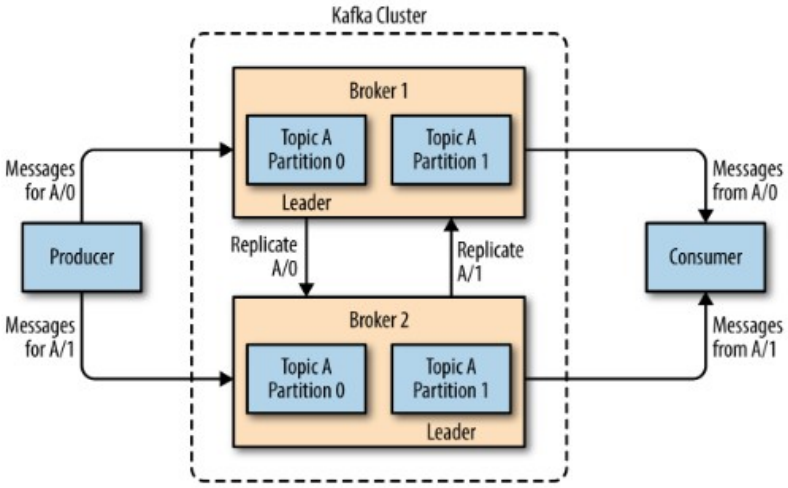


Obrázok 7 – 2 Skupina Consumerov, ktorá číta správy z jednotlivých partícií topicu

topicu. Dvaja Consumeri pracujú s jednou partíciou zatiaľ čo, tretí pracuje s dvoma partíciami. Vlastníctvo partície consumerom je názov, ktorý sa používa pre mapovanie Consumera k partícií.

7.3 Brokre a klastre

Broker je samostatný server Kafky. Brokre sú navrhnuté tak aby operovali ako súčasť klastra. V klastri má jeden broker úlohu kontrolóra klastra, je zvolený automaticky z aktívných brokerov. Tento broker je zodpovedný za administratívne operácie. Medzi tieto operácie patrí prideľovanie partícií ku brokerom a monitorovanie chybných brokerov. Leader je broker v klastri, ktorý vlastní partíciu. Partícia môže byť pridelená k viacerým brokerom, čo má za následok replikáciu partície[15].



Obrázok 7 – 3 Kafka klaster a brokre

7.4 Prípady použitia Kafky

Viaceré spoločnsoti používajú technológiu Apache Kafka následovne[22]:

- **LinkedIn (www.linkedin.com-)** - využíva Kafku na streamovanie operačných metrík a dát o činnosti. Vďaka tomuto fungujú rôzne produkty ako LinkedIn news feed a LinkedIn Today ako aj offline analytické systémy, ako je Hadoop.
- **DataSift (www.datasift.com/)** - používa Kafku ako zberateľa na monitorovanie udalostí a ako sledovateľa spotreby dátových tokov používateľmi v reálnom čase.
- **Twitter (www.twitter.com/)** - používa Kafku ako súčasť svojho Stormu - infraštruktúra spracovania streamovania.
- **Foursquare (www.foursquare.com/)** - Kafka pre Foursquare zabezpečuje online-to-online a online-to-offline posielanie správ. Používajú to na integrovanie monitorovacích a produkčných systémov, ktoré sú založené na offline infraštruktúrach Hadoop-u.
- **Square (www.squareup.com/)** - využíva Kafku ako zbernicu na všetkých systémových udalostí v rámci svojich dátových centier[22].

7.5 Kafka Streams

Je to klientská knižnica na vytváranie aplikácií a mikroservís, kde vstupné a výstupné dáta sa ukladajú na klaster Kafky.

Výhody Kafka Streams:

- Elasticita, veľká škálovateľnosť, tolerancia chýb.
- Nasadenie do kontajnerov, virtuálných mašín, cloud.
- Rovnako využiteľné pre malé, stredné a veľké prípady použitia.
- Úplná integrácia s Kafka Security.
- Štandard, ktorý dovoľuje písať aplikácie v Jave a Scale.
- Jednorázová možnosť spracovania.
- Nevyžaduje samostatný procesujúci klaster.
- Možnosť vývoja na Linuxe, Windowse a Macu.

7.5.1 Prípady pužitia Kafka Streams

- **New York Times** - využívajú Kafka Streams na ukladanie a distribuovanie publikovaného obsahu do rôzných aplikácií a systémov, ktoré robia tento obsah viditeľným pre čitateľov v reálnom čase.
- **Pinterest** - využíva Apache Kafku a Kafka streams vo veľkom rozsahu na poháňanie prediktívneho rozpočtového systému svojej reklamnej infraštruktúry v reálnom čase. Kafka Streams pre nich vykazuje najpresnejšie predpovede výdavkov, aké kedy predtým robili.
- **Zalando** - využívajú Kafku ako ESB, čo im pomohlo prejsť na architektúru mikroservisov. Použitím Kafka Streams na spracovanie streamov udalostí, Zalando umožňuje svojmu teamu špecialistov robiť podnikovú inteligenciu v reálnom čase.
- **Rabobank** - Kafka poháňa ich digitálny nervový systém Business Event Bus. Rabobank ho využíva na to, aby zvýšil počet finančných procesov a služieb. Službu Rabo Alerts poháňa Kafka Streams. Táto služba v reálnom čase upozorňuje svojich zákazníkov o finančných udalostiach.
- **LINE** - využíva Kafku ako hlavné dátové stredisko pre svoje služby, aby komunikovali s inými službami. Denne produkuje veľké množstvo správ, ktoré využíva na vykonávanie napríklad biznis logiky, detekcie hrozby, indexovanie vyhľadávania a dátovú analýzu. Line využíva Kafka Streams na spoľahlivú transformáciu a filtrovanie topicov, vďaka ktorým môžu spotrebitelia efektívne konzumovať subtopicy.
- **trivago** - trivago využíva Kafka Streams aby umožnil svojim vývojarov voľný prístup k dátam. Kafka Streams poháňa niektorý z ich analytických kanálov a ponúka nekonečné možnosti na skúmanie a fungovanie dátových zdrojov, ktoré majú k dispozícii[12].

7.6 Metriky Apache Kafky

Keďže Kafka je závislá na Zookeeperovi, tak je nevyhnutné aby sme takisto sledovali ZooKeeper.

Z predchádzajúcich kapitol už vieme, že Kafka je zložená z troch hlavných častí a to, Producerov, Consumerov a Brokerov. Každá jedná z častí, ktoré tvoria spoločne systém Kafka ma svoje vlastné metriky, ktoré je možné monitorovať. Členia sa tieto hlavné metriky:

- **Metriky Kafka Broker-u**

- **Metriky JVM**
- **Host/Server metriky**

Ak sa používateľ rozhodne, že chce monitorovať Kafku, musí zohľadniť tieto kategórie svojich komponentov:

- **Dáta Producera a Consumera.**
- **Veľkosť správy a údajové typy.**
- **Brokre a servre na prenos dát.**
- **Programovací jazyk a platforma.**
- **Sieťová architektúra.**
- **Open-source balíky v produkcii.**

Vzhľadom na uvedené kategórie, existujú kľúčové metriky, ktoré dokáže používateľ sledovať. Tieto metriky ho takisto vedia upozorniť ak sa vyskytne nejaký problém.

7.6.1 Metrika sieťových požiadaviek

Vzhľadom na to, že hlavnou úlohou Kafka brokerov je zhromažďovať a presúvať dáta na spracovanie, môže dochádzať k veľkej sieťovej premávke. Sledovaním počtu sieťových požiadaviek za sekundu môže používateľ monitorovať a porovnávať sieťovú priepustnosť serveru v každom dátovom centre a poskytovateľov cloudu.

Ak sieťová šírka pásma konkrétneho brokera prekročí alebo klesne pod prahovú hodnotu, môže to znamenať to, že je potrebné zväčšiť počet brokerov alebo to, že niektorá podmienka spôsobuje oneskorenie. Takisto to môže naznačovať potrebu implementovať back-off protokol consumera na efektívnejšie spracovanie dátových tokov.

7.6.2 Metrika sieťovej chyby

Krížové odkazy na sieťovú priepustnosť so súvisiacimi sieťovými chybami môžu pomôcť diagnostikovať, čo spôsobuje latenciu. Chybové podmienky zahŕňajú vynechané sieťové pakety, mieru chybovosti v odpovediach podľa typu žiadosti a typy vyskytujúcich sa chýb.

Ak sa sieťová priepustnosť znižuje bez toho aby sa zvyšovala chybovosť, môže to znamenať to, že je potrebné zväčšiť počet consumerov. V prípade vynechávanie paketov, to môže znamenať, že používateľ má problém s hardvérom.

7.6.3 Metrika nízkej replikácií partícii

Tým, že používateľ zvýši počet replík topicu, môže zaistiť odolnosť dát a takisto aj to, že broker bude vždy schopný doručovať. Výsledkom tohto je to, že dáta sa replikujú do viac ako jedného brokera, ktoré je možné spracovať aj v prípade zlyhania jedného brokera.

Táto Kafka metrika používateľa upozorňuje na prípady, keď je v danom topicu menej ako minimálny počet aktívnych brokerov. Spravidla by nemali existovať žiadne nedostatočne replikované partície v Kafka produkcii, to znamená, že táto hodnota by mala byť vždy rovná hodnote 0.

7.6.4 Metrika počtu offline partícií

Offline partície predstavujú dátové úložiská, ktoré nie sú k dispozícii pre aplikácie používateľov z dôvodu zlyhania alebo reštartu servera. V klastri Kafke slúži jeden z brokerov ako kontrolór zodpovedný za správu stavov partícií a replík a podľa potreby priradenie partícií.

Moment kedy je nevyhnutné zvýšiť replikovanie partícií nastáva vtedy keď partície na konkrétnych serveroch sa zvyšujú a znižujú v dôsledku pripojenia na cloud alebo iných prechodných sieťových udalostí.

7.6.5 Metrika všetkých partícií broker-a

Predchádzajúce dve metriky označujú potenciálne chyby. Musíme však rozpoznať, koľko partícií spravuje broker, aby sa používateľ mohol vyhnúť chybám a vedieť, kedy je čas na zvýšenie. Cieľom by malo byť udržanie rovnováhy medzi brokermi.

7.6.6 Metrika latencie presunu logov

Pripojením k existujúcim log súborom Kafka ukladá dáta. Zápisy vyrovnávacej pamäte sa presunú do fyzického úložiska, ktoré je založené na viacerých interných faktoroch Kafky. Vykonávajú asynchrónne procesy na to, aby sa optimalizoval výkon a odolnosť. Pomocou systémového volania fsync je možné takýto zápis vykonať. Kedy sa má vykonať tento presun dokažeme Kafke nariadiť použitím nastavení v log.flush.interval.messages a log.flush.interval.ms. Odporúča sa, aby sa tieto súbory nenastavovali manuálne. Tým, že používateľ povolí systemové volanie fsync bežať na pozadí, tým zefektívni spôsob presunu.

Stratégia monitorovania odolnosti dát by mala zahŕňať kombináciu replikácie dát a latencie v čase premiestnenia logu asynchrónneho disku. To, že používateľ meria čas presunu logov na základe svojho plánu, ukazuje latenciu a tá môže ukázať, že je potrebné zvýšiť počet replík a škálovania, zabezpečiť rýchlejšie úložisko alebo to takisto môže ukážať problémy s hardvérom.

7.6.7 Metrika správ Consumera

Tým, že používateľ nastaví základné línie očakávanej priepustnosti správ consumerov a zmeria fluktáciu, tým dokáže určiť latenciu a to, či je potrebné zmenšiť alebo zväčšiť počet consumerov. Aby používateľ zistil napríklad príčinu toho, že správy sú veľmi veľké, využíva krížovú referenciu dát s meraním bajtov za sekundu a veľkosť fronty. Pri zmene maximálnej veľkosti správy musí používateľ zmeniť súbor max.message.bytes. Tento súbor určuje najväčšiu veľkosť záznamu, ktorú Kafka povoľuje. Ak používateľ túto hodnotu zvýši, je takisto potrebné zvýšiť aj pamäť consumerov, aby bolo možné načítať záznam tak veľkého obsahu.

7.6.8 Metrika maximálneho frontu Consumera

Aby používateľ dokázal vytvoriť presný systém na posielanie správ v reálnom čase je doležité aby pochopil to, ako veľký je front prichádzajúcich správ do topicu. Meraním a sledovaním tejto metriky consumerom umožňuje to, aby presne vedeli, kedy je potrebné zvýšiť danú hodnotu. Podstatné pre používateľa je zapamätať si, že túto metriku meriame v rámci consumerov a partícii. To znamená, že každá partícia v každom topicu má pre daného consumera určité oneskorenie.

7.6.9 Metrika oneskorenia správ

Táto metrika popisuje oneskorenie počtu správ na repliku sledovateľa. Naznačuje, že replikácia mohla byť zastavená alebo prerušená. Keďže spracovanie prichádzajúcich dát v reálnom čase je jedným z hlavných problémov tak je túto metriku je potrebné monitorovať. Monitorovaním konfiguračného parametra replica.lag.time.max.ms môžeme zmerať čas, počas ktorého replika nenačítava nové dáta od leadra.

7.6.10 Metrika sledovania pamäte

V prípade, že výmena prebieha minimálne, Kafka má najlepší výkon. Ak chceme spustiť tento program, je potrebné nastaviť maximálnu veľkosť hromady JVM. Týmto sa vyhneme zberu nepotrebných dát. Ak chceme túto veľkosť správne upraviť je potrebné sledovať, či voľná pamäť servera klesne pod prahovú hodnotu, a tiež čítanie disku. Ak brokre musia čítať nedávno zapísané údaje z disku namiesto medzipamäte, je potrebné vyladiť nastavenia pamäte JVM alebo nastáva potreba zvýšiť pamäť RAM. Okrem toho musíme sledovať využitie výmeny. Ak je výmena povolená, sledujeme zvýšenie aktivity výmeny serverov. Môže to viesť k vypršaniu časového limitu operácie Kafky[3].

7.7 Apache ZooKeeper

Apache ZooKeeper je distribuované, open-source koordinačné riešenie. ZooKeeper plní úlohu správcu clustera a je neoddeliteľnou spčasťou pri práci s Apache Kafka. Podľa oficiálnych Apache Kafka dokumentácií je spustenie Zookeepera uvádzané ako nutná podmienka pred spustením servru Kafky [11].

Kafka využíva služby Apache ZooKeeper na to aby dokázala zkoordinovať brokre a celý cluster. Apache ZooKeeper pomáha Kafke v 2 oblastiach - pri správe brokerov a consumerov.

U brokerov sa ZooKeeper stará o tieto oblasti:

- Stav - podáva informácie uzlom o tom, či nezlyhal niektorý z uzlov. Vďaka posielaniu tzv. heartbeats žiadostí, ktoré je neustále, prebieha kontrola. ZooKeeper má na starosť výber nového leadra skupiny odberateľov ak nastane situácia, že sa jeden z uzlov odpojí.

- Kvóty - niektorým uzlom dovoľuje mať odlišné kvóty na produkovanie a spracovanie záznamov.

- Replikácie - pre každú z kategórií uchováva tzv. ISR (in-sync replicas) replikácie. Jedná sa o synchronizovanú replikáciu naprieč celým klastrom. To znamená, že ak jeden z uzlov zlyhá, ISR záznam je aktualizovaný. ZooKeeper je zodpovedný za výber nového vodcu, ak uzol, ktorý zlyhal bol leadrom jednej z kategórií.

- Registre uzlov a topicov - vďaka tomu, že uchováva registre uzlov a klastrov, sa vieme dostať k údajom ako zoznam všetkých dostupných uzlov v rámci klastra spolu so zoznamom kategórií, ktoré drží konkrétny uzol. Tieto informácie sa udržiavajú aktívne po dobu spojenia so systémom ZooKeeper, to znamená, že záznamy nie sú trvalé. V momente keď sa ukončí spojenie ukončí, sú zahodené.

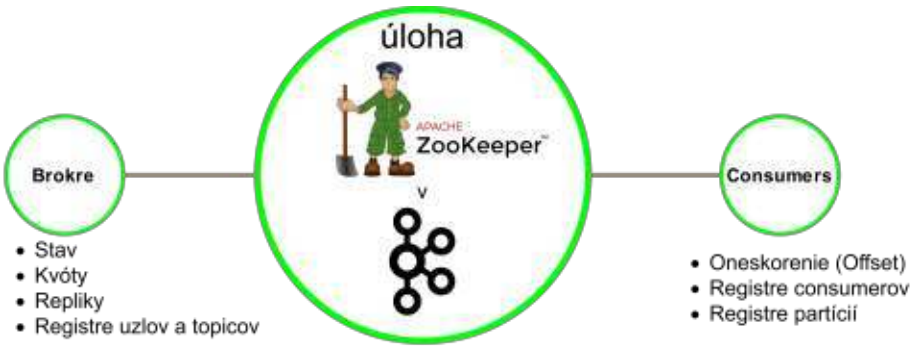
Pri spravovaní consumerov sa Zookeeper stará o následovné:

- Oneskorenie(offset) - ZooKeeper je predvolené úložisko pre nastavenia oneskorenia pre konkrétneho consumera. Plány do budúcnosti su také, že na tento účel bude vytvorený dedikovaný Kafka uzol nazvaný Offset Manager.

- Registre consumerov - podobne ako uzly majú aj consumerovia, vlastný register a platia pre ne rovnaké pravidlá. V momente keď sa odberateľ odpojí, záznam o ňom sa vymaže.

- Registre partícií - vzhľadom na to, že partícia je vždy viazaná práve na jedného consumera, ktorý spadá do konkrétnej skupiny, systém potrebuje vedieť väzbu medzi partíciou a consumerom. Takisto ako u predchádzajúcich registroch, v momente ked partícia prestane existovať, záznam o nej sa vymaže[13].

Na obrázku nižšie je znázornené, aké úlohy ZooKeeper plní pri spolupráci s Kafkou.



Obrázok 7 – 4 ZooKeeper v Apache Kafka

7.8 Apache Avro

Apache Avro predstavuje kompaktný formát serializácie binárnych dát, ktoré poskytujú rôzne dátové štruktúry. Avro využíva JSON notáciu na serializáciu a deserializáciu dát. Avro ponúka:

- Bohaté dátové štruktúry.
- Kompaktné, rýchle, binárne dátove formáty.
- Kontajner na ukladanie pretrvávajúcich údajov.
- Vzdialené volanie procedúr(RPC).
- Jednoduchú integráciu s dynamickými programovacími jazykmi[2].

7.8.1 Vytvorenie Avro schémy

Avro schéma je v JSON formáte a popisuje polia, ktoré berú do úvahy hodnoty vrátane ich údajových typov. Avro popisuje záznam a má nasledujúci formát.

```
{
 "type": "record,
 "namespace": "schema.namespace"
 "name": "schema.name"
 "fields": [
 { "name": "field1", "type": "string" },
 { "name": "field2", "type": "int" }
```

```
8   ]
9 }
```

Záznam má 4 polia: type, namespace, name a fields. Pole type popisuje JSON, ktoré nesie názov "record"pre Avro schémy. Pole namespace odlišuje jednu schému skupiny od druhej. Polia name a namespace majú ľubovoľné hodnoty. Polia fields sú aktuálné definácie schém a popisujú hodnoty záznamov vrátane ich typov. Pole type má primitívne hodnoty ako null, int, long, string, boolean, float, double, bytes. Definícia schémy sa môže skladať z jedného alebo viacerých polí[24].

7.9 Práca s Apache Kafka

V tejto časti si ukážeme ako sa pracuje s technológiou Apache Kafka. Začneme postupne a ukážeme si ako spustiť Kafku, ako vytvoriť topicy a producera a consumera prostredníctvom príkazového riadku.

Po stiahnutí súborov, ktoré sú nevyhnutné na prácu a spustenie Apache Kafky je nevyhnutné aby sme spustili ZooKeepera, Kafka Server a vytvorili topic. Spustenie Zookeepera je zabezpečené prostredníctvom nasledujúceho príkazu príkazu:

Obrázok 7 – 5 Príkaz na Zookeepera cez príkazový riadok

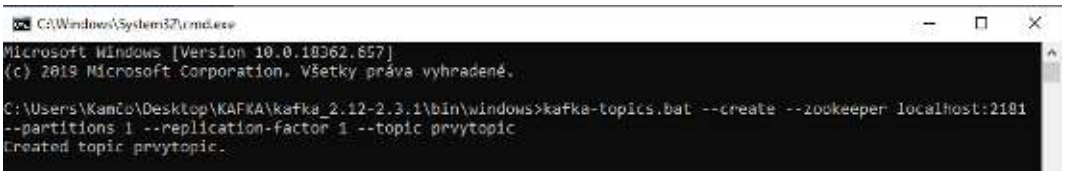
ZooKeeper štandardne od momentu spustenia tohto príkazu pracuje na porte 2181 ak to nenastavíme inak v konfiguračnom súbore zookeeper.properties.

Po spustení ZooKeepera nastal čas na to aby sme spustili Kafka server. Naštartovanie servera Kafky je vykonané prostredníctvom nasledujúceho príkazu:

Obrázok 7 – 6 Príkaz na spustenie Kafka serveru cez príkazový riadok

Kafka server beží od momentu spustenia tohto príkazu na porte 9092. Tento port môžeme takisto zmeniť v konfiguračnom súbore, tentokrát ide o súbor server.properties.

Kedže Kafka smeruje správy do jednotlivých topicov, je nevyhnutné aby sme vytvorili aj topic. Topic vytvoríme pomocou nasledujúceho príkazu:



Obrázok 7 – 7 Príkaz na vytvorenie topicu cez príkazový riadok

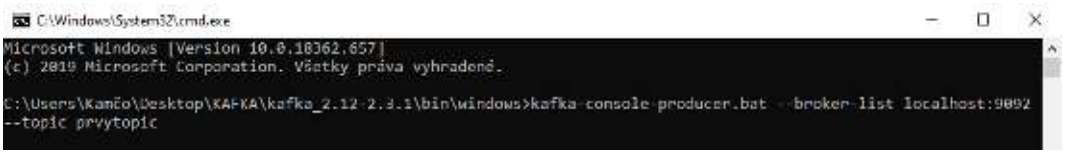
Vytvorili sme topic s názvom "prvytopic". Tento topic obsahuje jednu partíciu a takisto aj jednu repliku.

Ak sme zvládli naštartovanie ZooKeepera a Kafka servru a takisto vytvorenie topicu, tak môžeme prístupiť k otestovaniu triviálnej komunikácie medzi producerom a consumerom. Ako prvého si spustíme consumera prostredníctvom príkazu zobrazeného na obrázku nižšie.

Obrázok 7 – 8 Príkaz na vytvorenie Consumera v príkazovom riadku

Jednotlivé prepínače v príkaze ukazujú to, že consumer sa pripája na Kafka server a aj to, že bude odoberať správy posielané do topicu "pvytopic". Posledný prepínač hovorí o tom, že consumer bude čítať všetky správy poslané do consumera od začiatku.

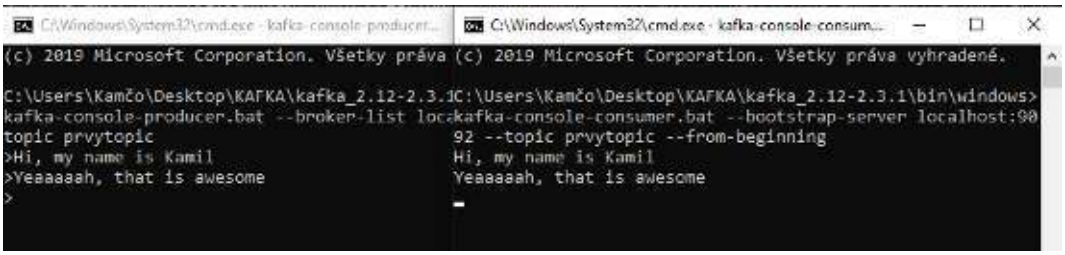
Po spustení consumera môžeme spustiť producera. Toho zapneme prostredníctvom príkazu zobrazeného na obrázku nižšie.



Obrázok 7 – 9 Príkaz na vytvorenie Producera v príkazovom riadku

Na základe prepínačov v príkaze vidíme, že producer sa takisto pripája na Kafka server a že správy, bude posielať, tak Kafka zapíše do topicu, ktorý sme vytvorili predtým s názvom "prvytopic". Keď už máme vytvoreného producera a aj consumera

môžeme otestovať či komunikácia medzi nimi funguje. Na nasledujúcom obrázku je zobrazená jednoduchá komunikácia medzi producerom a consumerom na princípe "Hello World".



Obrázok 7 – 10 Komunikácia medzi producerom a consumerom cez príkazový riadok

Pristupovať ku Kafke, tak ako k jej Producerom a Consumerom vieme aj inak ako len cez príkazový riadok. Vďaka API Kafka Producera a Consumera vieme s nimi pracovať v jazykoch, ktoré majú pre Apache Kafka dostupného klienta. To znamená, že vieme napríklad vytvoriť Producera v príkazovom riadku a Consumera vytvoríme v danom programovacom jazyku. Takisto to vieme urobiť aj opačne, že Producer bude vytvorený v programovacom jazyku a Consumer bude spustený prostredníctvom príkazového riadku. Tretia možnosť vytvorenia je vytvoriť oboch, aj producera a consumera v danom programovacom jazyku. Kedže naša doterajšia práca bola vytvorená v jazyku Python, rozhodli sme sa, že tento jazyk použijeme aj na náš ďalší vývoj. Takto vyzerá jednoduchá implementácia Producera, ktorý sme vytvorili v Pythone.

```
1  from kafka import KafkaProducer
2  from json import dumps
3  import time
4
5  producer = KafkaProducer(
6      bootstrap_servers ='localhost:9092',
7      value_serializer=lambda x: dumps(x).encode('utf-8'))
8
9  message = None
10
11 for i in range(3):
12     i = i + 1
13     message = {}
14     print("Kamil's message to others: " + str(i))
15     message["message_id"] = str(i)
16     message["message_value"] = "Kafka is fun!"
17     print("Message to be sent: ", message)
```

```
18      producer.send("skuska", message)
19      time.sleep(1)
```

Ak chceme vytvoriť producera, musíme si nainštalovať knižnicu kafka a následne si do zdrojového kódu naimportujeme modul KafkaProducer. Následne si vytvoríme premennú z ktorej urobíme Producera. Producerovi musíme povedať aj to na aký broker sa má pripojiť. V našom prípade je to lokálny broker, ktorý pracuje na porte 9092. Vzhľadom na to, že v tejto vzorovej implementácii budeme posielať správu vo tvare JSON-u, tak je nevyhnutné aby sme producerovi pridali aj JSON serializer. Vzhľadom na to, je nutné naimportovať modul dumps z json knižnice. Posledný riadok kódu prikazuje producerovi aby tieto správy poslal každú sekundu. Výstup zo vzorovej implementácie Producera je následovný:

```
Kamil's message to others: 1
Message to be sent:  {'message_id': '1', 'message_value': 'Kafka is fun!'}
Kamil's message to others: 2
Message to be sent:  {'message_id': '2', 'message_value': 'Kafka is fun!'}
Kamil's message to others: 3
Message to be sent:  {'message_id': '3', 'message_value': 'Kafka is fun!'}

Process finished with exit code 0
```

Obrázok 7 – 11 Výstup producera implementovaného v pythone

Nasledujúci zdrojový kód zobrazuje jednoduchú implementáciu Kafka Consumera v jazyku Python:

```
1  from kafka import KafkaConsumer
2  from json import loads
3
4  consumer = KafkaConsumer(
5      'skuska',
6      group_id='group1',
7      bootstrap_servers=['localhost:9092'],
8      value_deserializer=lambda x: loads(x.decode('utf-8'))
           )
9
10 print("Messages from Kafka, which Kamil sent")
11 msg_count = 0
12 for message in consumer:
13     msg = message.value
14     print("Message received: ", msg)
15     msg_count += 1
```

```
16      if msg_count >= 3:
17          break
18  consumer.close()
```

Keď vytvárame Consumera je nutné naimportovať modul KafkaConsumer z knižnice kafka, ktorú sme si nainštalovali pred vytvorením Producera. Z premennej consumer sme vytvorili Kafka consumera, ktorého parametre hovoria o tom, že ma odoberať správy z topicu skuska. Ďalej sme určili, že tento consumer patrí do skupiny consumerov "group1ä pripojili sme ho na lokálny broker. Vzhľadom na to, že sme do Kafky poslali JSON správy, tak musíme consumerovi priradiť aj JSON deserializér. Ak chceme aby implementácia prebehla v poriadku, je nevyhnutné vypnúť consumera pomocou príkazu close(). Výstup zo vzorovej implementácie je zobrazený na obrázku nižšie:

```
I am reading messages from Kafka, which Kamil sent
Message received:  {'message_id': '1', 'message_value': 'Kafka is fun!'}
Message received:  {'message_id': '2', 'message_value': 'Kafka is fun!'}
Message received:  {'message_id': '3', 'message_value': 'Kafka is fun!'}

Process finished with exit code 0
```

Obrázok 7 – 12 Výstup consumera implementovaného v pythone

8 ElasticSearch

Je to distribuovaný vyhľadávací a analytický nástroj, ktorý má využitie v reálnom čase. Používa sa na vyhľadávanie textu, štruktúrované hľadanie, analýzu a kombinuje tieto tri vlastnosti dokopy.

ElasticSearch môže byť popísaný ako:

- Distribuované úložisko dokumentov v reálnom čase, kde každé pole je zaindexované a vyhľadávateľné.
- Distribuovaný analytický nástroj v reálnom čase.
- Schopný škálovať stovky serverov a petabajtov štruktúrovaných a neštruktúrovaných dát.

Na to aby používateľ komunikoval s ElasticSearch je potrebné aby používal RESTful API. Aké API sa má používať zaleží na tom, aký programovací jazyk chce používateľ použiť.

Ak sa používateľ rozhodne používať programovací jazyk Java, tak bude využívať Java API. V tomto prípade ElasticSearch využíva dvoch vstavaných klientov, ktorý sa používajú v samotnej implementácii. Jedná sa o týchto klientov:

- Uzlový klient - pripája sa na lokálny klaster ako nedátový uzol. To znamená, že tento uzol vie na ktorom uzly v klastri sa dáta nachádzajú. Takisto môže posielať žiadosti konkrétnemu uzlu.
- Transportný klient - môže sa používať na posielanie žiadostí na vzdialený klaster. Nepripája sa na neho, iba posiela ďalej žiadosti na uzol v klastri.

Obidvaja klienti komunikujú s klastrom prostredníctvom portu 9300, kde využívajú natívny transferový protokol Elasticsearchu. Uzly v klastri medzi sebou komunikujú takisto prostredníctvom tohto portu.

Pre používateľa je takisto možné použiť API s JSON-om cez HTTP. Takúto komunikáciu využívajú všetky ostatné programovacie jazyky. S ElasticSearchom komunikujú prostredníctvom portu 9200. Pristúpiť k tomuto portu môžeme prostredníctom webového prehliadača tak ako aj cez príkazový riadok pomocou príkazu curl. ElasticSearch má okrem Javy podporu týchto programovacích jazykov - Python, .NET, PHP, JavaScript, GROOVY, Perl a Ruby.

ElasticSearch je dokumentovo orientovaná databáza, čo znamená, že ukladá celé dokumenty alebo objekty. Takisto indexuje obsah každého dokumentu v takom poradí aby ich bolo možné vyhľadávať. Používateľ dokáže indexovať, vyhľadávať, triediť, filtrovať dokumenty. Táto možnosť umožňuje ElasticSearchu celotextové vyhľadávanie.

Serializačným formátom pre ElasticSearch je JSON. JSON serializácia je podporovaná vo väčšine programovacích jazykov a tento jazyk sa stal štandardom pre NoSQL databázy.

8.1 Prípady použitia ElasticSearch-u

- Wikipedia - využívajú ElasticSearch na poskytutnutie celotextového vyhľadávania so zvýraznenými úryvkami vyhľadávania, vyhľadávanie search-as-you-type a návrhy did-you-mean[7].
- The Guardian - využívajú ElasticSearch aby skombinovali logy návštevníkov s dátami socialných sieti, ktoré slúžia ako hodnotenie v reálnom čase pre editorov o názore čitateľov na publikovaný článok.
- Stack Overflow - kombinuje vyhľadávanie celého textu s geologickými pripomienkami a využíva more-like-this na vyhľadávanie súvisiacích otázok a odpovedí.
- GitHub - využíva ElasticSearch na vyhľadávanie 130 miliard riadkov kódu[7].

8.2 Kibana

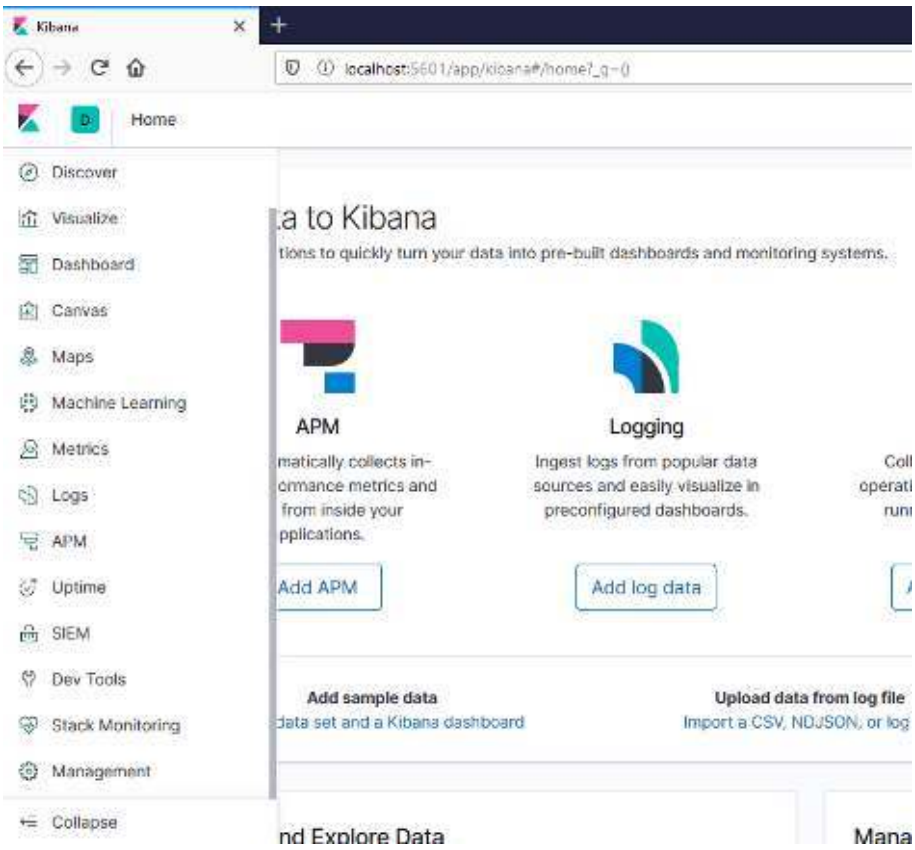
Kibana je vizualizačný a analytický nástroj, ktorý nám umožňuje pracovať s dátami, ktoré sú v ElasticSearchi. Kibana je súčašťou tzv. ELK architektúry, ktorá predstavuje databázu ElasticSearch, dátový kanál Logstash a webové rozhranie Kibana.

Základnými vlastnosťami webového rozhrania Kibana sú analýza dát a dopytovanie. Tento nástroj nám umožňuje vizualizáciu dát prostredníctvom rôznych spôsobov ako sú tepelné mapy, histrogramy, čiarové grafy, koláčové grafy. Pomocou rôznych metód možeme v Kibane vyhľadávať dáta, ktoré sú uložené v elasticu aby sme ich použili na analýzu koreňových príčin. Používateľ pomocou Kibany ľahko pochopí veľké dáta a je schopný si rýchlo vytvoriť a zdieľať dashboard, ktorý zobrazuje zmeny, ktoré sa dejú v elasticu v reálnom čase[21].

Kibana štandardne pracuje na porte 5601. Obrázok nižšie zobrazuje ako vyzerá webové rozhranie Kibana.

Vlastnosti Kibany:

- Výkonný dashboard, ktorý nám umožňuje vizualizovať zaindexované dáta, ktoré sú v klasti elasticu.
- Umožňuje vyhľadávať zaindexované dáta v reálnom čase.
- Používateľom umožňuje nie len vyhľadávať ale aj prezerať a ďalej pracovať s dátami, ktoré sú uložené v databáze ElasticSearch.



Obrázok 8 – 1 Webové rozhranie Kibany

- Umožnuje nám vytvárať dopyty na dátach a výsledky následne zobraziť v grafoch, tabuľkách a mapách.
- Konfigurovateľný dashboard, ktorý rozdeľuje logy z Logstashu na menšie časti v elasticu.
- Kibana nám vie takisto poskytnúť historické dáta prostredníctom grafov.
- Predstavuje real-time dashboard, ktorý je jednoduchý na konfiguráciu[5].

9 Testovanie výmeny obchodných dokumentov prostredníctvom Apache Kafka

V tejto kapitole sa zameriame na testovanie výkonu systému Apache Kafka. To, že prebieha komunikácia medzi jednotlivými firmami zabezpečujú systémy zasielania správ. Tým, že budeme aplikáciu postupne rozširovať, tak sa požiadavky, ako maximálna priepustnosť systému, zvyšujú. Testovacie prostredie si chceme namodelovať na princípe elektronickej výmeny dát medzi firmami. Na základe tohto rozdodnutia sme si pri príprave testovania určili tieto základné pravidlá:

- Každá firma bude mať svojho Producera a zároveň aj Consumera.
- Každá firma bude mať priradený jeden topic - to znamená, že počet topicov sa rovná počtu vytvorených firiem.
- Do topicu konkrétnej firmy sa budú zapisovať dokumenty, ktoré má táto firma prijať.
- Firma môže odoslať dokument inej firme - to znamená, že danú správu bude zapisovať do jej topicu.
- Súčasťou posielanej správy musí byť aj topic kam má príjmajúca firma odoslať potvrdenie, že danú správu dostala. V prípade, že firma príjma správu, ktorá odpoveďou už je, ďalej na takúto správu odpovedať nebude.

Z vecí, ktoré sme si určili vyššie teda musíme pripraviť a naimplementovať klineta, kde si budeme vedieť určiť následovné vstupné inicializačné parametre:

- Počet firiem - tým určíme aj počet topicov a samozrejme počet Consumerov a Producerov.
- Počet obchodných partnerov firmy - týmto firmám bude firma posielať obchodné dokumenty.
- Počet dokumentov, ktorý má firmá odoslať druhej firme.

Parametre, ktoré sme si zvolili budú takisto určovať ako veľmi sa bude v systéme komunikovať. Testovanie je rozdelené na 3 testovacie scenárov. Každý testovací scenár má iné nastavenie vstupného parametra, ktorý určuje počet firiem, ktoré mezdi sebou komunikujú. V každom testovacom scenári budeme merať za ako dlho prebehla celá komunikácia medzi firmami. Merať budeme aj jednotlivo čas trvania odosielania, príjmania a odpovedania na dané obchodné dokumenty. K jednotlivým nameraným časom si uvedieme aj rýchlosť posielania resp. príjmania správ.

9.1 Prostredie na testovanie systému

Testovanie bolo vykonané na štyroch serveroch Technickej univerzity v Košiciach, ktoré nám boli na tento účel priradené. Na hlavnom serveri, na ktorom je operačný systém Ubuntu 16.04 budeme spúšťať architektúru prostredníctvom docker images. Architektúru spustíme naraz vykonaním príkazu *docker-compose up*. Pri testovaní bol použitý docker image confluentinc/cp-zookeeper a confluentinc/cp-kafka, prostredníctvom ktorých spustíme ZooKeeper a Apache Kafku, docker image ElasticSearch-u a Kibany od spoločnosti Bitnami s verziou 7 a verzia Logstash 7.6.2.

Na ďalších troch serveroch si budeme postupne spúšťať jednotlivých klientov a testovať ich. Tieto servery majú operačný systém CentOS 6.10. Každý testovací scenár pracuje s množinou topicov a počet topicov sa bude odvýjať od počtu firiem, ktoré vytvoríme. Topicy sme vytvorili so základnými nastaveniami. Vzhľadom na to, že každá firma má jeden topic tak replikačný faktor a hodnota nastavenia počtu partícií topicov sa rovná hodnote 1. Keďže máme tri testovacie scenáre, tak na jednom serveri budeme spúšťať klientov k danému testovaciemu scenáru.

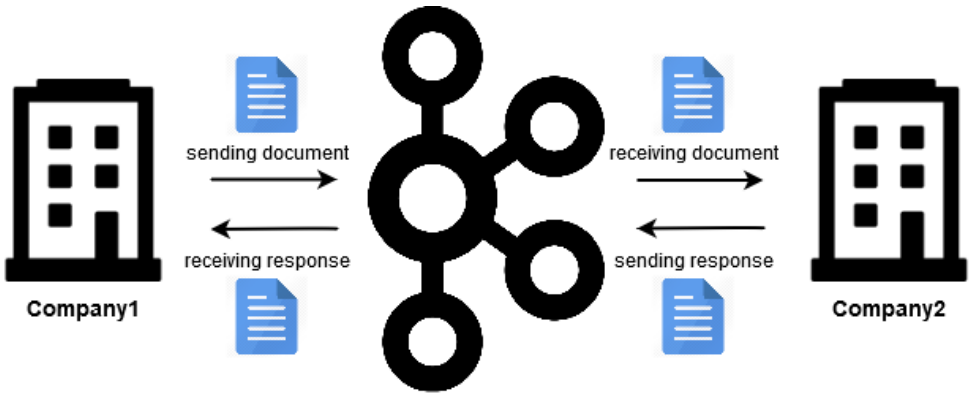
Podrobné a presné popísanie jednotlivých testovacích scenárov sa nachádza v kapitolách, ktoré budú ďalej následovať. Hodnoty, ktoré sme namerali sme si zobrazili prostredníctvom grafov. Na záver si vyhodnotíme výsledky jednotlivých testovacích scenárov systému Apache Kafka, v nami navrhnutej architektúre systému na odosielania, spracovanie a analýzu transakčných obchodných dát.

9.1.1 Prvý testovací scenár

Rozhodli sme sa, že prvý testovací scenár si rozdelíme na 5 podscenárov. Keďže testovanie je navrhnuté na princípe firiem, na začiatku je dôležité si určiť 3 parametre. Parameter, ktorý určuje, koľko firiem bude v systéme. Parameter, ktorý určí koľko partnerov majú mať dané firmy a posledný parameter určí koľko správ ma firma odoslať druhej firme. V prvom testovacom scenári sme prvému parametru určili hodnotu 50, čiže vytvoríme 50 firiem.

Na nasledujúcom obrázku je zobrazené ako prebieha komunikácia medzi firmami prostredníctvom systému Apache Kafka:

V prvom testovacom scenári, v ktorom sme si vytvorili 50 firiem budeme meniť druhý parameter, ktorý predstavuje počet partnerov, ktoré jedna firma má. Začneme tým, že počet partnerov pre jednu firmu bude 15. Každá z vytvorených firiem má tým pádom po 15 partnerov. Náhodný výber obchodných partnerov zapríčiní to, že počet firiem od ktorých daná firma odoberá správy sa môže odlišovať od inej firmy. Parameter, ktorý určuje koľko správ sa má poslať sme si nastavili na hodnotu 25. Následne po spustení daného skriptu sa najprv vytvorí 50 firiem, ktoré si postupne vyberú zo zoznamu vytvorených firiem 15 partnerov. Po tom ako už má daná firma svojich partnerov, začína postupne každej jednej posielať obchodné dokumenty. Najprv každá z daných firiem odosiela dokumenty svojim obchodným



Obrázok 9 – 1 Proces výmeny obchodných dokumentov medzi firmami prostredníctvom Kafky

partnerom a až potom ich príjma. Po prijatí všetkých správ, firmy začnú odpovedať svojim partnerom. Firma odpovedá naspäť firme, od ktorej daný obchodný dokument dostala. Odpoveď zapíše naspäť do jej kanála. Po tom ako firmy odpovedia svojim obchodným partnerom o tom, že prijali ich dokumenty, nasleduje to, že konkrétnym firmám sa vracajú správy o prijatí dokumentov od ich partnerov. Firmy na odpoveď o prijatí už ďalej neodpovedajú. Komunikácia, kde majú firmy po 15 partnerov a posielajú svojim partnerom 25 obchodných dokumentov trvala 489,12 sekúnd s celkovou rýchlosťou 223,52 správ za sekundu. Odosielanie malo rýchlosť 132,47 správ za sekundu. Príjmanie faktúr prebiehalo s rýchlosťou 334,84 správ za sekundu.

V druhom prípade prvého testovacieho scenára zmeníme len parameter počet obchodných partnerov. Hodnotu tohto parametra sme nastavili na hodnotu 25. Parameter, ktorý určuje počet odoslaných správ jednou firmou každému zo svojich obchodných partnerov sa v druhom prípade prvého testovacieho scenára nemení. Proces výmeny obchodných dokumentov medzi 50 firmami, ktoré mali po 25 partnerov a posielali si po 25 správ, trval 588,51 sekúnd s rýchlosťou 212,40 správ za sekundu. Za jednu sekundu v druhom prípade prvého testovacieho scenára sa odošle 109,84 správ a príjme 325,55 správ.

Tretí prípad prvého testovacieho scenára spočíva v tom, že druhý parameter má rovnakú hodnotu ako v druhom prípade a to konkrétne 25. Parameter, ktorý určuje počet odoslaných správ jednou firmou každému zo svojich obchodných partnerov sme sa rozhodli v treťom prípade prvého testovacieho scenára nastaviť na hodnotu 35. Komunikácia medzi firmami v treťom prípade prvého testovacieho scenára trvala celkom 807,84 sekúnd s celkovou rýchlosťou 216,63 správ za sekundu. Za jednu sekundu v treťom prípade prvého testovacieho scenára sa odošle 113,13 správ a príjme 409,31 správ.

Ďalší, v poradí už štvrtý prípad prvého testovacieho scenáru spočíva v tom, že zmeníme parameter počtu obchodných partnerov. Počet partnerov zvýšime na

hodnotu 35. Parameter, počtu odoslaných správ jednej firme bude 25. Celý proces výmeny obchodných dokumentov v tomto prípade trval 954,79 sekúnd s rýchlosťou 183,29 správ za sekundu. Odosielanie má v štvrtom prípade prvého testovacieho scenára rýchlosť 90,92 správ za sekundu a príjmanie 403,59 správ za sekundu.

Piaty a zároveň posledný prípad prvého testovacieho scenára má takisto rovnaký počet firiem a hodnotu, ktorá určuje počet partnerov, ponecháme na rovnakej hodnote ako v predchádzajúcom prípade a to na hodnote 35. V tomto prípade zmeníme len tretí parameter. Jeho hodnotu nastavíme na hodnotu 35, čiže jedna firma pošle druhej 35 správ resp. obchodných dokumentov. Proces komunikácie v poslednom prípade prvého testovacieho scenára trval dokopy 1474,45 sekúnd s rýchlosťou 166,16 správ za sekundu. Za jednu sekundu v piatom prípade prvého testovacieho scenára sa odošle 95,89 správ a príjme 499,10 správ.

Testovací scenár si podrobnejšie rozoberieme v kapitole o vyhodnocovaní testovania, ktoré sme vykonávali. Meriame celkový čas trvania komunikácie, počet poslaných správ za sekundu, trvanie odosielania, príjmania a takisto odpovedania. Pre jednoduchšií prehľad nastavenia parametrov pri jednotlivých prípadoch komunikácie v prvom testovacom scenári sme si urobili nasledujúcu tabuľku.

Prvý testovací scenár			
	Počet firiem	Počet partnerov	Počet odoslaných správ
1.prípad	50	15	25
2.prípad	50	25	25
3.prípad	50	25	35
4.prípad	50	35	25
5.prípad	50	35	35

Tabuľka 9 – 1 Nastavenie parametrov v prvom testovacom scenári

9.1.2 Druhý testovací scenár

V druhom testovacom scenári sme sa rozhodli, že počet vytvorených firiem si navýšime o 25, čiže tentokrát budeme vytvárať 75 firiem. Tento testovací scenár si takisto rozdelíme na 5 jednotlivých prípadov, kde budeme meniť ďalšie parametre. Princíp výmeny obchodných dokumentov prebieha na rovnakom princípe ako v prvom testovacom scenári.

V prvom prípade druhého testovacieho scenára sme nastavili počet obchodných partnerov pre jednu firmu na hodnotu 15. Počet obchodných dokumentov, ktoré firma posiela svojim partnerom bude 25. Komunikácia medzi firmami v tomto prípade druhého testovacieho scenára trvala 675,87 sekúnd s rýchlosťou 166,45 správ za

sekundu. Za jednu sekundu v prvom prípade druhého testovacieho scenára sa odošle 107,12 správ a príjme 210,46 správ.

Druhý prípad tohto testovacieho scenára bude mať iný počet obchodných partnerov, teda 25. V tomto prípade zmeníme aj hodnotu počtu posielaných správ jednou firmou druhej firme. Tento parameter sme nastavili na hodnotu 25. Úplná komunikácia medzi firmami v tomto prípade trvala 1040,74 sekúnd s rýchlosťou 180,16 správ za sekundu. Odosielanie faktúr malo rýchlosť 105,32 správ za sekundu a príjmanie 317,28 správ za sekundu.

V treťom prípade druhého testovacieho scenára sme ponechali parameter počtu obchodných partnerov na hodnote 25. Tretí parameter, ktorý predstavuje počet obchodných dokumentov, ktoré firma posiela svojim partnerom, sme nastavili na hodnotu 35. Komunikácia medzi firmami v treťom prípade druhého testovacieho scenára trvala 1876,89 sekúnd s rýchlosťou posielania 139,86 správ za sekundu. Prijímanie správ v treťom prípade druhého testovacieho scenára prebiehalo rýchlosťou 329,09 správ za sekundu. Odosielanie malo rýchlosť 127,26 odoslaných správ za sekundu.

Štvrtý prípad tohto testovacieho scenára bude mať rozdielný počet obchodných partnerov ako mal predchadzajúci prípad testovania, a to 35. V tomto prípade zmeníme takisto hodnotu počtu posielaných správ jednou firmou druhej firme. Tento parameter sme nastavili na hodnotu 25. Úplná komunikácia medzi firmami v tomto prípade trvala 2071,91 sekúnd s rýchlosťou 126,69 správ za sekundu. Za jednu sekundu v štvrtom prípade druhého testovacieho scenára sa odošle 97,64 správ a príjme 382,92 správ.

V piatom prípade druhého testovacieho scenára sme zmenili parameter počtu obchodných partnerov, a to na hodnotu 35, čo znamená, že každý z firiem bude odosielať správy svojim 40-tim obchodným partnerom. Tretí parameter, ktorý predstavuje počet obchodných dokumentov, ktoré firma posiela svojim partnerom, sme nastavili na hodnotu 35. Komunikácia medzi firmami v piatom prípade druhého testovacieho scenára trvala 2690,25 sekúnd s rýchlosťou 136,60 správ za sekundu. Odosielanie faktúr malo rýchlosť 145,55 správ za sekundu a príjmanie 88,94 správ za sekundu.

Testovací scenár si podrobnejšie rozoberieme v kapitole o vyhodnocovaní testovania, ktoré sme vykonávali. Meriame celkový čas trvania komunikácie, počet poslaných správ za sekundu, trvanie odosielania, príjmania a takisto odpovedania. Pre jednoduchšií prehľad nastavenia parametrov pri jednotlivých prípadoch komunikácie, ktoré prebiehali v rámci druhého testovacieho scenára, sme si urobili nasledujúcu tabuľku.

Druhý testovací scenár			
	Počet firiem	Počet partnerov	Počet odoslaných správ
1.prípad	75	15	25
2.prípad	75	25	25
3.prípad	75	25	35
4.prípad	75	35	25
5.prípad	75	35	35

Tabuľka 9 – 2 Nastavenie parametrov v druhom testovacom scenári

9.1.3 Tretí testovací scenár

Tento testovací scenár si takisto rozdelíme na 5 jednotlivých prípadov rozličných nastavení parametrov. Prostredie, kde budú firmy tentokrát komunikovať si znova navýšime a znova o 25 firiem. V prípade tretieho testovacieho scenáru sme si teda určili, že v systéme komunikácie si budeme vytvárať 100 firiem, ktoré medzi sebou budú komunikovať.

Pre prvý prípad tretieho testovacieho scenára sme nastavili počet obchodných partnerov pre jednu firmu na hodnotu 15. Parameter, ktorý určuje koľko obchodných dokumentov pošle jedna firme druhej firme sme nastavili na hodnotu 25. Komunikácia medzi firmami v prvom prípade druhého testovacieho scenára trvala 983,96 sekúnd s rýchlosťou 152,45 správ za sekundu. V tomto prípade tretieho testovacieho scenára sa za jednu sekundu sa odošle 79,64 správ a príjme 196,88 správ.

Nasledujúcemu prípadu, v poradí druhému, tretieho testovacieho scenára sme nastavili hodnoty parametra následovne. Počet obchodných partnerov pre jednu firmu sme nastavili na hodnotu 25. Počet obchodných dokumentov, ktoré firma posiela svojim partnerom bude 25. Výmena obchodných dokumentov medzi jednotlivými firmami v tomto prípade tretieho testovacieho scenára trvala 1694,98 sekúnd s rýchlosťou 147,49 správ za sekundu. Odosielanie malo rýchlosť posielania 71,65 správ za sekundu. Prijatie malo rýchlosť 226,84 správ za sekundu.

V poradí tretí prípad tohto testovacieho scenára sme vykonali následovne. Počet obchodných partnerov pre jednu firmu sme ponechali na hodnote 25 ako bolo nastavené v prechádzajúcom prípade testovania. Zmenili sme ale tretí parameter, ktorý určuje koľko dokumentov má firma poslať druhej firme. Tento parameter sme nastavili na hodnotu 35. V tomto prípade komunikácia medzi firmami trvala 3331,21 sekúnd s rýchlosťou 105,07 správ za sekundu. Odosielanie malo v treťom prípade tretieho testovacieho scenáru rýchlosť 76,65 správ za sekundu a príjmanie malo rýchlosť 78,65 správ za sekundu.

Tretí testovací scenár ďalej pokračuje štvrtým prípadom. Teraz sme nastavili počet obchodných partnerov pre jednu firmu na hodnotu 35.Tretí parameter, počtu

odosielaných dokumentov druhej firme, sme nastavili na hodnotu 25. Komunikácia medzi firmami v štvrtom prípade druhého testovacieho scenára trvala 3558,20 sekúnd s rýchlosťou 98,36 správ za sekundu. V tomto prípade tretieho testovacieho scenára sa za jednu sekundu odošle 68,45 správ a príjme 86,73 správ.

Posledný prípad tretieho testovacieho scenára má nasledujúce nastavenie parametrov. Parameter počtu obchodných partnerov sme nastavili na hodnotu 35. Tretí parameter, ktorý určuje koľko obchodných dokumentov sa má poslať partnerovi sme nastavili na hodnotu 35. Výmena obchodných dokumentov v poslednom prípade tretieho testovacieho scenára trvala 5668,20 sekúnd s rýchlosťou 86,45 správ za sekundu. V piatom prípade tretieho testovacieho scenára sa za jednu sekundu odošle 61,15 správ a príjme 81,62 správ.

Testovací scenár si podrobnejšie rozoberieme v kapitole o vyhodnocovaní testovania, ktoré sme vykonávali. Meriame celkový čas trvania komunikácie, počet poslaných správ za sekundu, trvanie odosielania, príjmania a takisto odpovedania. Pre jednoduchšií prehľad nastavenia parametrov pri jednotlivých prípadoch komunikácie v treťom testovacom scenári sme si urobili nasledujúcu tabuľku.

Tretí testovací scenár			
	Počet firiem	Počet partnerov	Počet odoslaných správ
1.prípad	100	15	25
2.prípad	100	25	25
3.prípad	100	25	35
4.prípad	100	35	25
5.prípad	100	35	35

Tabuľka 9 – 3 Nastavenie parametrov v treťom testovacom scenári

9.2 Vizualizácia dát

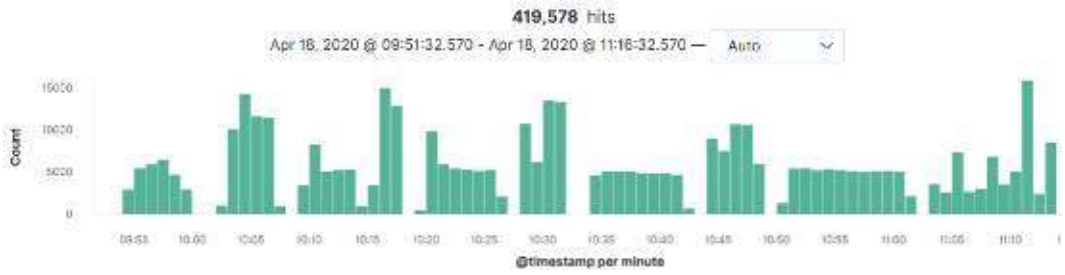
Dokumenty, ktoré si v komunikácii vymieňali firmy sa počas tohto procesu zaindexovali do ElasticSearchu a v tejto časti si tieto dáta budeme vizualizovať. V našej práci pracujeme s verziou Elasticsearch 7 a verziou Kibana 7. Dáta, ktoré si ukladáme do ElasticsSarch-u si vieme vizualizovať pomocou webového rozhrania Kibana. Ak si tieto dáta chceme zobrazovať a pracovať s nimi, musíme postupovať následovnými krokmi. Po spustení Kibany si na bočnom menu vyberieme možnosť "Management". Dostaneme sa do sekcie manažmentu indexov. V tejto sekcii ďalej klikneme na Ïndex Patterns"pod malým logom Kibany ako je zobrazené na obrázku nižšie.

Na to aby vedela Kibana pristupovať k dátam z elasticu je nevyhnutné si vytvoriť index aj v nej. Ak chceme pristúpiť k indexu "businessdata"z ElasticSearchu musíme

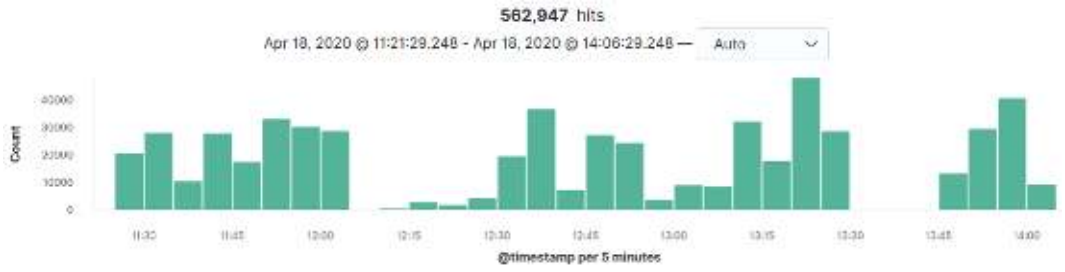
Obrázok 9 – 2 Kibana - Index Patterns

si vytvoriť index s rovnakým názvom aj v Kibane.

Keď sme si vytvorili index už aj v Kibane, vieme si tieto dáta už aj vizualizovať. Pre vizualizáciu je nutné v menu kliknúť na možnosť "Discover". Obrázky, ktoré sú zobrazené nižšie, ukazujú histogramy, kde môžeme vidieť koľko dokumentov bolo zaindexovaných do ElasticSearchu v nami vykonaných testovacích scenároch.

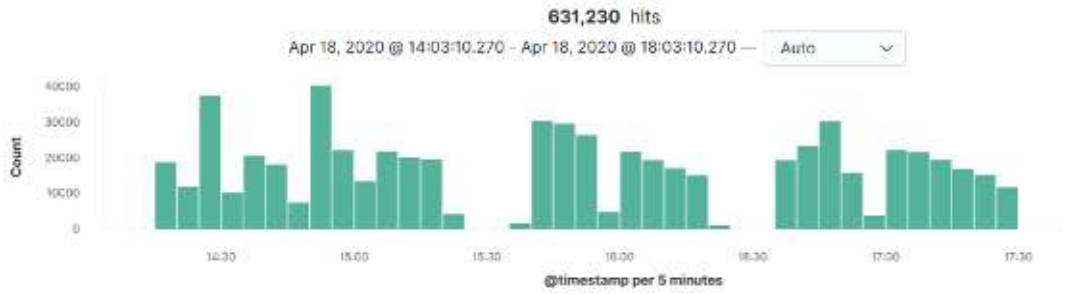


Obrázok 9 – 3 Zaindexované dokumenty - prvý testovací scenár



Obrázok 9 – 4 Zaindexované dokumenty - druhý testovací scenár

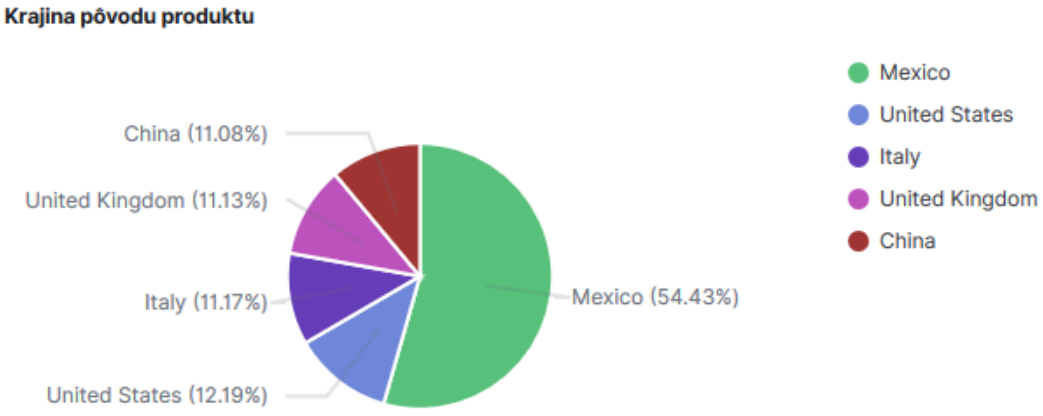
Ako môžeme vidieť na týchto obrázkoch, najviac dokumentov bolo zaindexovaných počas vykonávania tretieho testovacieho scenára. Dáta z daného indexu si



Obrázok 9 – 5 Zaindexované dokumenty - druhý testovací scenár

vieme filtrovať na základe toho, ktoré polia v dokumentoch si chceme zobraziť. Ak klikneme na možnosť z menu Kibany "Vizualize", Kibana nám ponúka možnosť si vytvárať rôzne typy grafov. Následne je dôležité si vytvoriť novú vizualizáciu a z pop-up okna, ktoré sa nám zobrazí si zvolíme typ grafu, ktorý si chceme vytvoriť. Ako príklad sme si vytvorili 4 jednoduché vizualizácie.

Prvou vizualizáciou je koláčový graf, ktorý nám zobrazuje krajinu pôvodu kupovaného produktu. Tento graf nám zobrazuje aj percentuálne hodnoty. Určili sme si aj to, že graf má mať tvar koláča a nie tvar šišky. Legendu sme si chceli dali zobraziť na pravej strane. Graf nám hovorí to, že najviac produktov z obchodných dát, ktoré sme získali bolo vyrobených v Mexicu, ktoré ako krajina pôvodu predstavuje hodnotu 54,43 % všetkých produktov. Naopak najmenej produktov bolo vyrobených v Číne(11,08 %). Vytvorený graf v Kibane je zobrazený na obrázku nižšie.



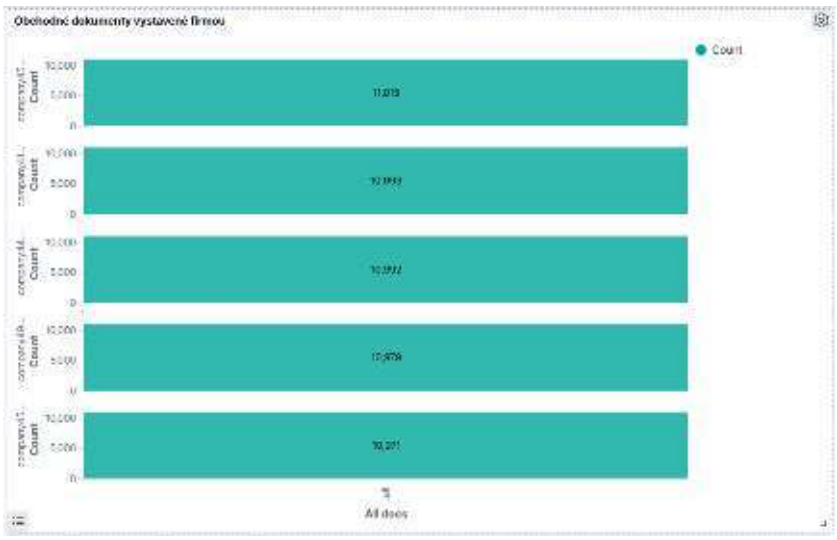
Obrázok 9 – 6 Krajina pôvodu produktu

Ďalšou vizualizáciou, ktorú sme si vytvorili bol graf typu "Goal". Tento typ grafu slúži na to aby sme si určili hlavný cieľ napríklad počtu dokumentov. Cieľ si napríklad

nastavíme na 5 miliónov a danou agregáciou si vypočítame celkový počet dokumentov, ktorý máme v databáze aktuálne. Celkový počet dokumentov je aktuálne 1 613 755 z cieľa 5 miliónov. Ďalšou vizualizáciou je tabuľka najviac kupovaných značiek. V danej tabuľke sme si zvolili, že sa má zobraziť 5 najviac kupovaných značiek s ich celkovým počtom a aby boli značky zoradené od najväčšieho zastúpenia po najmenšie v tomto podarí - Bauer, Adidas, Lenovo, Wilson, Fanatics. Tieto dve vizualizácie sú zobrazené na obrázku nižšie.

Obrázok 9 – 7 Počet dokumentov a značky

Štvrtá vizualizácia je vertikálny graf, ktorý nám zobrazuje 5 firiem, ktoré vystavili najviac faktúr zo všetkých všetkých firiem. Na grafe vidíme, že najviac faktúr vystavila firma "company45"s celkovým počtom 11 015 faktúr, nasleduje firma "company41", ktorá vystavila 10 993 faktúr. Treťou firmou s najviac vystavenými faktúrami je firma "company44"s celkovým počtom 10 992 fakúr. Nasleduje firma "company49"s 10 979 vystavenými faktúrami. Top 5 uzatvára firma "company46", ktorá vystavila 10 971 faktúr.



Obrázok 9 – 8 Firmy, ktoré vystavili najviac faktúr

Ak sme si už navizualizovali jednotlivé grafy, ktoré sme chceli, tak môžeme v menu kliknúť na možnosť "Dashboard". Dashboard nám umožní aby sme si vedeli zobraziť jednotlivé grafy a výstupy dát na jednej stránke. Dashboard si vieme upravovať na základe našich preferencií, čiže upravovať usporiadanie jednotlivých grafov a podobne. Dôležité je to, že dáta v jednotlivých vizualizáciach sa nám zobrazujú na základe toho aké časové rozpätie dát zaindexovaných do ElasticSearchu si vyberieme. To znamená, že výstup jednotlivých grafov môže byť iný za obdobie posledných 7 dní ako výstup posledných 3 dní.

Obrázok 9 – 9 Dashboard vizualizácií

9.3 Vyhodnotenie testovania výkonu Apache Kafka

V tejto kapitole si bližšie rozoberieme výsledky, ktoré sme získali prostredníctvom testovania komunikácie medzi firmami. Na výsledky vplýva rýchlosť internetového pripojenie ale aj výpočtová kapacita a pamäť RAM daných serverov. Za zmienku je nutné podotknúť to, že testovanie bolo vykonávané počas ideálnych podmienok rýchlosti internetového pripojenia. Technické parametre využitých serverov sú špecifikované v systémovej príručke, ktorá je mezdi prílohami tejto práce. Naimplementovaný skript klienta je bližšie špecifikovaný v systémovej príručke tejto závarečnej práce.

9.3.1 Vyhodnotenie - prvý testovací scenár

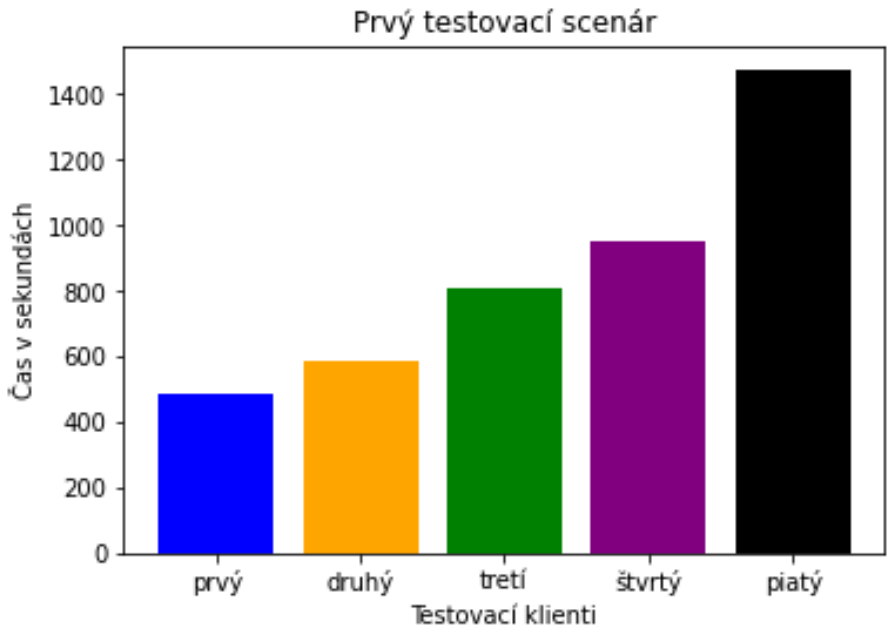
Nasledujúca tabuľka nám zobrazuje výsledky z prvého testovacieho scenára, kde počet firiem v systéme sme nastavili na hodnotu 50.

	Dĺžka komunikácie (sec)	Rýchlosť (msg/sec)	Dĺžka odosielania (sec)	Dĺžka príjmania (sec)	Dĺžka odpovedania (sec)
1.prípad	489,12	223,52	234,39	102,42	100,96
2.prípad	588,51	212,40	284,50	95,99	126,24
3.prípad	807,84	216,63	386,73	106,89	225,05
4.prípad	954,79	183,29	481,21	108,40	275,20
5.prípad	1474,45	166,16	638,76	122,72	616,76

Tabuľka 9 – 4 Tabuľka výsledkov prvého testovacieho scenáru

Graf a hodnoty v tabuľkách, ktoré sme získali na základe prvého testovacieho scenára nám ukázali, že nárast času trvania komunikácie medzi firmami je exponenciálny. Rozdiely medzi celkovým trvaním komunikácie v jednotlivých prípadoch nie je však enormný. Vyšší nárasť môžeme bádať pri poslednom piatom prípade testovacieho scenára, kde bol oproti štvrtému prípadu zmenený tretí parameter. Počet obchodných partnerov sa však nezmenil. Teda najväčší vplyv na dĺžku trvania danej komunikácie v tomto prípade mal tretí parameter. Rozdiel medzi trvaním prvým prípadom a posledným prípadom komunikácie je trojnásobný.

Dĺžka odosielania faktúr sa zvyšuje v jednotlivých prípadoch miernym stúpaním, najväčší nárasť je vidieť takisto pri treťom a poslednom testovacom prípade. Príjmanie obchodných dokumentov si drží stabilné trvanie a trvá podstatne kratšie ako odosielanie. Vysoký nárasť trvania má však odpovedanie na prijaté faktúry, ktoré ma v rámci jednotlivých prípadoch až šesťnásobný nárast. To môže byť zapríčninené väčším náporom na pamäť RAM.



Obrázok 9 – 10 Graf výsledkov prvého testovacieho scenára

Na základe výsledkov z prvého testovacieho scenára môžeme povedať, že najväčšiu zátaž na systém spôsobuje odpovedanie na prijatie obchodných dokumentov a tá zátaž je spôsobená najmä zväčšením tretieho parametra, ktorý určuje počet správ, ktoré má firma poslať inej firme.

9.3.2 Vyhodnotenie - druhý testovací scenár

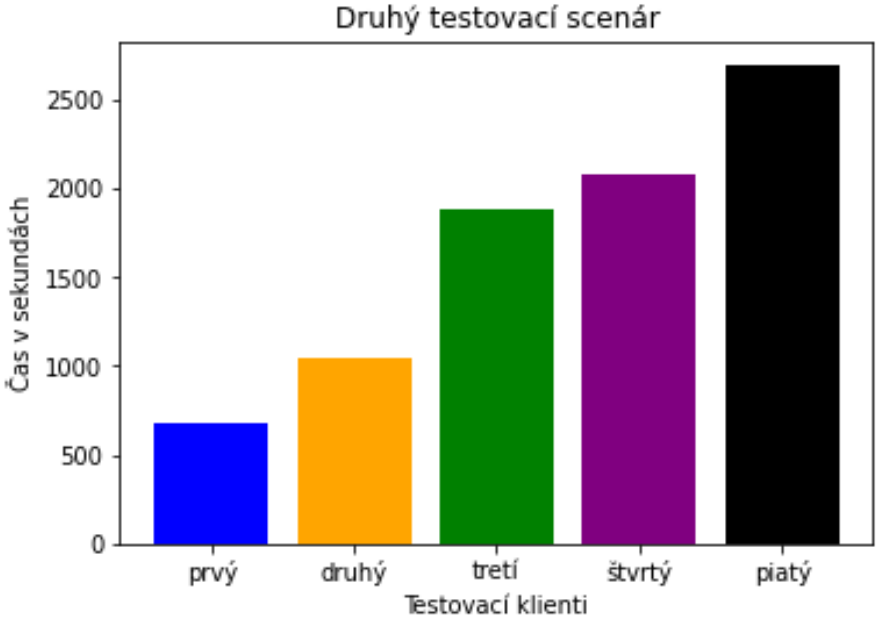
Nasledujúca tabuľka nám zobrazuje výsledky z druhého testovacieho scenára, kde počet firiem v systéme sme nastavili na hodnotu 75.

	Dĺžka komunikácie (sec)	Rýchlosť (msg/sec)	Dĺžka odosielania (sec)	Dĺžka príjmania (sec)	Dĺžka odpovedania (sec)
1.prípad	675,87	166,45	262,55	133,64	159,67
2.prípad	1040,74	180,16	445,07	147,69	318,60
3.prípad	1876,89	139,86	515,68	167,37	1060,67
4.prípad	2071,91	126,69	672,10	171,38	1081,47
5.prípad	2690,25	130,60	631,24	1033,04	866,95

Tabuľka 9 – 5 Tabuľka výsledkov druhého testovacieho scenáru

Graf a hodnoty v tabuľkách, ktoré sme získali vykonavávaním druhého testova-

cieho scenára nám ukázali, že nárast času trvania komunikácie medzi firmami je, rovnako ako v prvom testovacom scenári, exponenciálny. Vyšší nárasť trvania komunikácie si môžeme všimnúť pri treťom a poslednom piatom prípade testovacieho scenára, kde bol oproti druhému resp. štvrtému prípadu zmenený len tretí parameter, parameter odoslaných správ. To znamená, že najväčší vplyv na rýchlosť danej komunikácie v tomto prípade mal tretí parameter. Rozdiel medzi trvaním prvým prípadom a posledným prípadom komunikácie je približne štvornásobný.



Obrázok 9 – 11 Graf výsledov druhého testovacieho scenára

Dĺžka odosielania obchodných dokumentov sa zvyšuje pri každom prípade s výnimkou posledného prípadu. Najvyšší nárasť však v tomto meraní vidíme pri druhom prípade, kde sme zvýšili parameter počtu obchodných partnerov. Príjmanie obchodných dokumentov trvá podstatne kratšie ako odosielanie a má mierny nárast v každom ďalšom prípade. Výnimkou v druhom testovacom scenári je posledný prípad, kde si môžeme všimnúť enormný nárasť času prijímania obchodných dát. Odpovedanie na prijatie obchodných dokumentov pri prvých dvoch prípadoch nárast relatívne mierny avšak vyšší nárasť vidíme pri treťom prípade, kde sme zvýšili hodnotu parametra počtu odoslaných správ, čo mohlo byť znova zapríčinené zvýšeným náporom na pamäť RAM. Výnimkou tohto merania je takisto posledný, piaty prípad testovacieho scenára, kde sa dĺžka odpovedania skrátila.

Na základe nameraných výsledkov druhého testovacieho scenára, môžeme povedať, že najvyšší vplyv na výkon testovaného systému má tretí parameter. V tomto meraní sme našli aj anomáliu merania odosielania, príjmania a odpovedania, ktorá

sa vyskytla v piatom prípade testovania. Tento prípad bol pravdepodobne ovplyvnený prudkým náporom na pamäť RAM alebo prípadne náhlym nárastom rýchlosti internetového pripojenia, následnym poklesom a opätovným nárastom tejto rýchlosti.

9.3.3 Vyhodnotenie - tretí testovací scenár

Nasledujúca tabuľka nám zobrazuje výsledky z tretieho testovacieho scenára, kde počet firiem v systéme sme nastavili na hodnotu 100.

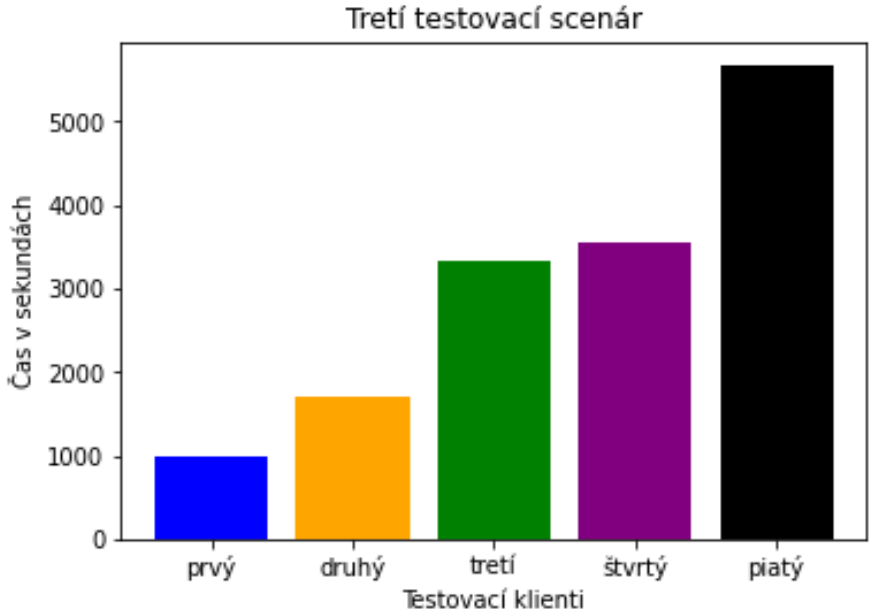
	Dĺžka komunikácie (sec)	Rýchlosť (msg/sec)	Dĺžka odosielania (sec)	Dĺžka príjmania (sec)	Dĺžka odpovedania (sec)
1.prípad	983,96	152,45	470,87	190,48	147,15
2.prípad	1694,98	147,49	872,27	275,53	330,43
3.prípad	3331,21	105,07	1143,24	1112,53	802,62
4.prípad	3558,20	98,36	1278,35	1008,82	920,74
5.prípad	5668,20	86,45	2003,27	1519,47	1493,36

Tabuľka 9 – 6 Tabuľka výsledkov tretieho testovacieho scenáru

Graf a hodnoty v tabuľkách, ktoré sme získali na základe vykonávania tretieho testovacieho scenára nám ukázali, že nárast času trvania komunikácie medzi firmami je rovnako ako v predchádzajúcich dvoch testovacích scenároch exponenciálny. Rozdiely medzi posielaním v jednotlivých prípadoch sú badateľné najmä v treťom a piatom prípade testovania. V týchto prípadoch sme oproti prechádzajúcemu prípadu zmenili iba parameter počtu odoslaných obchodných dokumentov. Počet obchodných partnerov sa však nezmenil. Výsledky v tomto testovacom scenári ukázali, že najväčší vplyv na rýchlosť danej komunikácie v treťom testovacom scenári mal tretí parameter.

Dĺžka odosielania narastala v každom prípade tohto testovacieho scenára, najvyšší nárasť vidíme v piatom prípade, kde sme zmenili parameter, počtu odosielaných obchodných dokumentov inej firme. Príjmanie dokumentov trvalo v každom prípade tretieho testovacieho scenára kratšie ako odosielanie. Dĺžka trvania príjmania obchodných dokumentov sa zvyšuje pri každom ďalšom prípade. Prudký nárast však zaznamenávame v treťom a piatom prípade, kde sme v porovnaní s druhý resp. štvrtým prípadom zmenili iba tretí parameter. Merali sme aj to ako dlho trvalo odpovedanie na obchodné dokumenty. Dĺžka trvania odpovedania sa v každom prípade zvyšuje, avšak najvyšší nárast je znova v treťom a piatom prípade tohto testovacieho scenáru, kde sme zvýšili hodnotu počtu odosielaných faktúr inej firme.

Na základe výsledkov, ktoré sme získali testovaním klientov, ktorých sme vytvorili v rámci tretieho testovacieho scenára, môžeme povedať, že najvýšší vplyv na



Obrázok 9 – 12 Graf výsledkov tretieho testovacieho scenára

výkon testovaného systému má tretí parameter. Zvyšujúce sa trvanie celkovej komunikácie môže mať za príčinu to, že je vyšší nápor na systém a takisto aj vyššia záťaž na pamäť RAM.

Po vykonaní všetkých troch testovacích scenárov môžeme konštatovať, že ak zvyšujeme hodnoty jednotlivých parametrov, komunikácia trvá dlhšie. Na základe výsledkov môžeme takisto skonštatovať, že odosielanie obchodných dokumentov trvá dlhšie ako príjmanie týchto dokumentov. Pri výpočtovej sile, ktorú sme mali k dispozícii má najväčší vplyv na výkon systému tretí parameter, ktorý určuje koľko správ má odoslať jedna firma každej a jednej firme zo svojho zoznamu obchodných partnerov. Tento parameter výrazne ovplývňuje odpovedanie na prijaté obchodné dokumenty. Výsledky však môžu byť iné ak by sme jednotlivých klientov spúšťali na zariadeniach, ktoré by mali výpočtovú silu väčšiu alebo na druhej strane, menšiu ako sme mali k dispozícii.

10 Záver

Cieľom tejto diplomovej práce bolo to, aby sme navrhli architektúru na spracovanie a analýzu transakčných obchodných dát, ako to vyplýva z názvu tejto práce. Túto architektúru sme mali navrhnúť tak, aby sme využili technológie veľkých dát, keďže veľké dáta sú aktuálne na stúpajucom trende vo svete IT. Postup vypracovania tejto práce pozostával z nasledujúcich krokov:

1. Podať teoretický prehľad problematiky elektronického obchodovania a výmeny obchodných dokumentov.

V teoretickej časti tejto záverečnej práce sme sa sústredili na zanalyzovanie súčasného stavu danej problematiky, kde sme si rozobrali jednotlivé riešenia ohľadom návrhu architektúry na spracovnaia a analýzu dát s využitím technológie veľkých dát. Spomenuli sme si technológie ako Apache Spark, Apache Zeppelin, Spark Streaming, Apache Storm, Apache Cassandra, Apache Kafka, ElasticSearch a ďalšie. V tejto časti sme si takisto opísali elektronickú výmenu dát(EDI), jej princípy, výhody oproti papierovej výmene dokumentov, druhy správ EDI, ale takisto aj elektronický podpis. Následne sme si zvolili, že ako EDI štandard využijeme Universal Business Language, ktorý sme popísali v kapitole 6, kde sme spomenuli aké obchodné procesy UBL ponúka a jednotlivé procesy sme si popísali. Z obchodných procesov sme si vybrali, že budeme pracovať s obchodným dokumentom faktúra, ktorý sme si podľa našich preferencií upravili. V teoretickej časti diplomovej práci sme opísali takisto aj streamovania dát, kde sme si porovnali streamovanie a dávkové spracovanie, povedali sme si aké výhody streamovanie prináša a uviedli sme si aj zopár prípadov, kde sa streamovanie v dnešnej dobe využíva.

2. Navrhnúť a implementovať vhodnú architektúru pre spracovanie obchodných transakcií s využitím technológií spracovania Veľkých Dát.

Našim ďalším krokom bolo zvolenie technológie, ktorá mala zabezpečiť data streaming obchodných dát. Vybrali sme si technoogiu Apache Kafka, ktorá je popisaná v kapitole 7. V tejto kapitole sme popisali princíp fungovania Apache Kafka a takisto aj jej jednotlivé súčasti. Spomenuli sme aj nevyhnutnú časť pre fungovanie Kafky, Apache Zookeeper a takisto aj formát avro. Nasledovala fáza výberu databázy, kde sme si zvolili, že použijeme databázu ElasticSearch, ktorá sa orientuje na dokumenty a ich indexovanie. Teoretický opis tejto databázy ElasticSearch sa nachádza v kapitole 8. V ďalšom kroku sme riešili problematiku ako dostať dáta z Apache Kafky do databázy ElasticSearch. Na tento úkon sme využili dátovy kanál Logstash, ktorý nám poskytuje túto možnosť. Po dostaní dát z Kafky do ElasticSearchu sme si chceli aj dané obchodné dokumenty aj vizualizovať. Rozhodli sme sa, že dáta z ElasticSearchu si zvizualizujeme pomocou webového rozhrania Kibana. Následne sme sa rozhodli, že na spojazdnenie tejto architektúry využijeme Docker. Pomocou jedného .yml súboru si spustíme služby Apache Kafka, Logstash, ElasticSearch a aj Kibanu. Postup implementácie navrhnutej architektúry je popísaný v kapitole 5.

3. Otestovať a vyhodnotiť implementovanú architektúru pomocou vhodných testovacích scenárov.
Ďalším krokom v tejto diplomovej práci bol vývoj softvérového prostredia, ktoré sme si namodelovali na princípe komunikácie medzi firmami. Komunikáciu medzi jednotlivými firmami sme si inicializovali na základe troch vstupných parametrov - parameter počtu firiem, parameter počtu obchodných partnerov a parameter počtu odoslaných dokumentov jednou firmou druhej. Následným implementovaním klienta sme si vytvorili 3 testovacie scenáre, kde sme testovali ako dlho trvá komunikácia medzi firmami v jednotlivých testovacích scenároch. Každý testovací scenár sme si rozdelili na 5 rôznych prípadov, kde sme menili vstupné parametre. Navrhnutá architektúra bola spustená na hlavnom serveri s operačným systémom Ubuntu 16.04 a na ostatných troch serveroch sme spúštali klientov pre jednotlivé testovacie scenáre. Po otestovaní sme si vytvorili v Kibane zopár vizualizácií obchodných dát. Výsledky z testovania nám ukázali, že s využitím serverov, ktoré nám boli poskytnuté na túto fázu, kde sme merali trvanie komunikácie medzi firmami, najviac vplýva tretí parameter, ktorý určoval počet obchodných dokumentov, ktoré má firma odoslať druhej firme. Ná vývoj sme využili programovací jazyk Python, kde sme využili rôzne knižnice, ako je popísané v systémovej príručke tejto záverečnej práce.

Odporúčaním do budúcna v oblasti tejto práce je návrh zmeny dátového kanála, kedže dátový kanál Logstash spôsobuje veľkú záťaž na pamäť RAM. Na to by mohol byť využitý konektor Kafka Connect, ktorý nie len prepája rôzne služby ale môže v sebe obsahovať aj serializér a deserializér na využívané formáty. Navrhnutá architektúra by takisto mohla byť implementovaná napríklad v systéme komunikácie dodávateľov a odberateľov, ktorý od nich dané produkty odoberajú, na to aby sa zefektívnila ich komunikácia a aby dodávatelia mali jednoduchší proces analyzovania nákupov svojich zákazníkov.

Literatúra

[1] ALLEY, Garrett. *What Is Data Streaming?* [online]. DZone. 2018. Dostupné z: https://dzone.com/articles/what-is-data-streaming.

[2] *Apache Avro™ 1.9.2 Documentation.* [online]. 2020. Dostupné z: http://avro.apache.org/docs/current/index.html

[3] BRUNO, Eric. *Kafka Metrics to Monitor.* [online]. 2019. Dostupné z: https://sematext.com/blog/kafka-metrics-to-monitor

[4] DEBUZNA, Joe. *What Is Data Streaming? A Data Architect's Guide.* [online]. Dataversity. 2019. Dostupné z: https://www.dataversity.net/what-is-data-streaming-a-data-architects-guide/.

[5] *ELK Stack Tutorial: Learn Elasticsearch, Logstash, and Kibana.* [online]. Dostupné z: https://www.guru99.com/elk-stack-tutorial.html

[6] GARG, Nishant. *Apache Kafka.* [online]. Packt Publishing Ltd. 2013. ISBN 978-1-78216-793-8. Dostupné z: https://books.google.sk/books?hl=en&lr=&id=qS9eAQAAQBAJ

[7] GORMLEY, Clinton.;TONG, Zachary. *Elasticsearch: The Definitive Guide* [online]. O'Reilly Media, Inc. 2015. ISBN 978-1-449-35854-9. Dostupné z: https://books.google.sk/books?id=d19aBgAAQBAJ

[8] HANAMANTHRAO, Ramanna; THEJASWINI, S. *Real-time clickstream data analytics and visualization.* [online]. Institute of Electrical and Electronics Engineers. 2017. ISBN 978-1-5090-3704-9. Dostupné z: https://ieeexplore.ieee.org/document/8256978

[9] HOLMAN, G.Ken. *Universal business language version 2.2.* [online]. OASIS Open. 2018. Dostupné z: http://docs.oasis-open.org/ubl/os-UBL-2.2/UBL-2.2.pdf

[10] HOLMAN, G.Ken.; BENGTSSON, Kenneth. *OASIS Universal Business Language (UBL) TC* [online]. OASIS Open. 2019. Dostupné z: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl#enh-2.2

[11] *Kafka 2.5 Documentation.* [online]. Apache Foundation. Dostupné z: https://kafka.apache.org/documentation/#zk

[12] *Kafka Streams.* [online]. Apache Foundation. Dostupné z: https://kafka.apache.org/documentation/streams

[13] KONIECZNY, Bartosz. *The role of Apache ZooKeeper in Apache Kafka.* [online]. Waiting for code. 2016. Dostupné z: https://www.waitingforcode.com/apache-kafka/the-role-of-apache-zookeeper-in-apache-kafka/read

[14] MEADOWS, Bill; SEABURG, Lisa. *Universal Business Language 1.0.* [online]. Oasis Open. 2004. Dostupné z: http://docs.oasis-open.org/ubl/cd-UBL-1.0/

[15] NARKHEDE, Neha; SHAPIRA, Gwen; PALINO, Todd. *Kafka: The DefinitiveGuide: REAL-TIME DATA AND STREAM PROCESSING AT SCALE.* [online]. O'Reilly Media, Inc. 2017. ISBN 978-1-491-93616-0. Dostupné z: https://books.google.sk/books?hl=enlr=id=a3wzDwAAQBAJ. s. 5-8

[16] PETERKA, Jiří. *Báječný svět elektronického podpisu* [online]. CZ.NIC, z. s. p. o. 2001. ISBN 978-80-904248-3-8. Dostupné z: https://knihy.nic.cz/files/edice/bajecny_svet_elektronickeho_podpisu_cznic.pdf. s. 28-30

[17] PRATAMA, M. A. L. et al. *Data Processing Architecture using Opensource Bigdata Technology to Increase Transaction Speed.* [online]. 2018 Third International Conference on Informatics and Computing (ICIC). Institute of Electrical and Electronics Engineers. 2018. ISBN 978-1-5386-6921-1. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8780434. s. 1–6

[18] RANJAN, Rajiv. *Data Processing Architecture using Opensource Bigdata Technology to Increase Transaction Speed.* [online]. Institute of Electrical and Electronics Engineers. ISSN 2325-6095, 2014, vol.1. Dostupné z: https://ieeexplore.ieee.org/abstract/document/6848734. s. 1–6

[19] REICHEL, David. *Jak na elektronickou výměnu dat?* [online]. CCV Informační systémy. 2009. Dostupné z: https://data.businessworld.cz/file/elektronicka-vymena-dat.pdf. s. 19

[20] *Streaming Data - A Complete Guide* [online]. Confluent. Dostupné z: https://www.confluent.io/learn/data-streaming/.

[21] SUREKA, Akash. *What is Kibana Used for? –10 Important Features to Know.* [online]. Clarion Technologies. Dostupné z: https://www.clariontech.com/platform-blog/what-is-kibana-used-for-10-important-features-to-know.

[22] THEIN, K. M. M. *Data Processing Architecture using Opensource Bigdata Technology to Increase Transaction Speed.* [online]. International Journal of Scientific Engineering and Technology Research. ISSN 2319-8885, 2014, vol.3, no.47. Dostupné z: http://ijsetr.com/uploads/436215IJSETR3636-621.pdf. s. 1–6

[23] *Typy EDI správ a ich použitie.* [online]. EDI Zone. Dostupné z: http://www.edizone.sk/elektronicka-vymena-dat/typy-edi-sprav-a-ich-pouzitie/.

[24] VOHRA, Deepak. *Apache Avro.* [online]. Apress, Berkeley, CA. 2016. ISBN 978-1-4842-2198-3. Dostupné z: https://link.springer.com/chapter/10.1007/978-1-4842-2199-0_7.

[25] *Vyznajte sa v elektronickej archivácii II: Elektronický podpis nie je naskenovaný obrázok.* [online]. EDI Zone. Dostupné z: http://www.edizone.sk/elektronicka-fakturacia/elektronicka-archivacia-dokumentov/vyznajte-sa-v-elektronickej-archivacii-ii-elektronicky-podpis-nie-je-naskenovany-obrazok/.

[26] *Vyznajte sa v elektronickej archivácii III: Ako funguje elektronický podpis* [online]. EDI Zone. Dostupné z: http://www.edizone.sk/elektronicka-fakturacia/elektronicka-archivacia-dokumentov/vyznajte-sa-v-elektronickej-archivacii-iii-ako-funguje-elektronicky-podpis/.

[27] *What is EDI?* [online]. SEAL Systems, The Digital Paper Factory. Dostupné z: https://www.sealsystems.com/service-support/glossary/electronic-data-interchange/.

[28] *What is EDI (Electronic Data Interchange)?* [online]. EDI Basics. Dostupné z: https://www.edibasics.com/what-is-edi/.

[29] *What is Streaming Data?* [online]. Amazon Web Services. Dostupné z: https://aws.amazon.com/streaming-data/.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka

Príloha C UBL Faktúra 2.2

Príloha D CD médium - záverečná práca v elektronickej podobe

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a umelej inteligencie

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát

Diplomová práca

Príloha A - Používateľská príručka

Vedúci diplomovej práce:

Ing. Peter Bednár, PhD.

Diplomant:

Bc. Kamil Strachan

Konzultant diplomovej práce:

Ing. Peter Bednár, PhD.

Košice 2020

Obsah

1 Funkcia programu

Používateľská príručka slúži ako návod ako spojazdniť prostredie na serveri,kde bude bežať navrhnutá architektúra. Príručka slúži aj ako návod ako spojazdniť serveri, kde bežia jednotlivé testovacie scenáre, vykonávaním python skriptov.

Tento program bol vytvorený v jazyku python a slúži na simulovanie prostredia, kde medzi sebou komunikujú firmy prostredníctvom architektúry, ktorú sme navrhli. Komunikácia v systéme sa inicializuje prostredníctvom troch parametrov:

- Počet firiem
- Počet obchodných partnerov
- Počet správ, ktoré má firma odoslať druhej firme

Naimplementovaný klient bol naprogramovaný v jazyku Python sa skladá z niekoľkých hlavných častí:

- Trieda Company
- funkcia odosielania obchodných dokumentov
- funkcia príjmania obchodných dokumentov
- funkcia odpovedania na prijaté obchodné dokumenty

2 Súpis obsahu dodávky

- Diplomová práca v elektronickej forme,
- Používateľska príručka v elektronickej forme,
- Systémova príručka v elektronickej forme,
- Príloha UBL faktúra v elektronickej forme,
- súbor, ktorý obsahuje dáta, testovacie skripty a konfiguračný súbor pre prvý testovací scenár,
- súbor, ktorý obsahuje dáta, testovacie skripty a konfiguračný súbor pre druhý testovací scenár,
- súbor, ktorý obsahuje dáta, testovacie skripty a konfiguračný súbor pre tretí testovací scenár,
- Avro schéma,
- docker-compose.yml súbor na spustenie navrhnutej architektúry prostredníctvom dockru,
- Výsledky jednotlivých testov,
- Skript na konvertovanie XML obchodných dokumentov.

3 Inštalácia navrhnutej architektúry

Na tento proces budeme využívať server Technickej Univerzity v Košiciach, ku ktorému vieme pristúpiť na IP adrese 147.232.202.105. Ak sa však používateľ chce pripojiť na tento server, je nevyhnutné aby bol pripojený na VPN univerzity. Návod ako si spojazdniť VPN na vašom PC nájdete na tejto stránke:

- **https://uvt.tuke.sk/wps/ portal/uv/sluzby/vzdialeny-pristup-vpn**

Prvotným krokom je inštalácia Dockru na server. Návod ako si nainštalovať Docker na operačný systém ubuntu nájdete na tomto linku:

- **https://docs.docker.com/ engine/install/ubuntu/**

Po nainštalovaní Dockru si vytvoríme súbor pomocou príkazu "sudo vim docker-compose.yml", kde vložíme obsah nášho .yml súboru. Po vytvorení .yml súboru si vytvoríme súbor logstash.conf, ktorý predstavuje konfiguračný súbor pre dátový kanál Logstash. Po vykonaní týchto krokov pristúpime ku kroku spustenia .yml súboru. Práca s docker-compose obsahuje tieto príkazy:

- *docker-compose up* - príkaz slúži na pustenie .yml súboru, ktorý zapne všetky kontajneri, ktoré sa v ňom nachádzajú
- *docker-compose down* - príkaz vypne a vymaže vytvorené kontajneri

Tieto príkazy môžu vyžadovať to, aby boli vykonané s príkazmi administrátora preto pred príkaz pre istotu pridáme "sudo". K ElasticSearchu vieme pristúpiť na tejto adrese - 147.232.202.105:9200. Kibanu si spustíme na následovnej adrese - 147.232.202.105:5601.

3.1 Proces prípravy prostredia na PC

Keďže testovacieho klienta sme vyvíjali prostredníctvom laptopu, je nevyhnutné nainštalovať Python do systému.

Návod ako nainštalovať Python na Windows nájdete na tomto odkaze -

- **https://phoenixnap.com/kb/how-to-install-python-3-windows**

Po procese inštálacie si môžeme jednoducho overiť, či sme python nainštalovali správne. Otvoríme si príkazový riadok a napíšeme do neho príkaz "python –version". Po vykonaní tohto príkazu by sa nám mala zobraziť verzia Pythona, ktorú používame. V našom prípade sme naištalovali verziu 3.7.4.

Ak sme si už nainštalovali správne Python do svojho systému, prejdeme ku kroku inštalácie vývojové prostredia, kde budeme vytvárať daný program. Ako vývojové prostredie sme si zvolili JetBrains PyCharm Community Edition 2019.2.5. Návod ako si nainšalovať toto vývojové prostredie na Windows 10 nájdete na nasledujúcom odkaze:

- **https://www.youtube.com/watch?v=SZUNUB6nz3g**

Predtým ako začneme s akýmkoľvek vývojom, je nutné aby si sme si nastavili v PyCharme interpretera, prípadne aj nainštalovali potrebné knižnice na vývoj. Návod ako správne nastaviť interpretera nájdete na tomto linku:

- **https://www.jetbrains.com/help/pycharm/configuring-python-interpreter.html**

Návod ako nainštalovať knižnice do PyCharmu nájdete na tomto linku:

- **https://www.jetbrains.com/help/pycharm/installing-uninstalling-and-upgrading-packages.html**

3.2 Proces prípravy serverov na testovanie

Na tento proces budeme využívať 3 školské servre, ktoré sú dostupné na týchto IP adresách.

- 147.232.202.122
- 147.232.202.123

- 147.232.202.124

Ak sa však používateľ chce pripojiť na tieto servre, je nevyhnutné aby bol pripojený na VPN univerzity. Návod ako si spojazdniť VPN na vašom PC nájdete na tejto stránke:

- **https://uvt.tuke.sk/wps/ portal/uv/sluzby/vzdialeny-pristup-vpn**

Nasledujúcim krokom je inštalácia interpretera Python3. Návod ako nainštalovať Python na CentOS nájdete na nasledujúcom linku:

- **https://linuxize.com/post/how-to-install-python-3-on-centos-7/**

Po inštalácii Pythona je ešte nevyhnutné vykonať príkaz "scl enable rh-python36 bash", ktorý nastaví interpretera na verziu 3.6.

Ďalším krokom v procese prípravy je vytvorenie virtuálneho prostredia pre python a nainštalovanie nástroja pip, ktorý slúži na inštaláciu knižníc. Tento krok vykonáme vykonaním týchto príkazov:

- *python3.6 -m venv env –without-pip*
- *source env/bin/activate*
- *curl https://bootstrap.pypa.io/get-pip.py | python3*

3.3 Programové použitie

Po vykonaní prechádzajúcich krokov, môžeme pristúpiť k spúšťaniu jednotlivých python skriptov. Python skript vieme spustiť vykonaním napríklad takéhoto príkazu:

- *python test1_case1.py*

3.4 Programové obmedzenia

Hlavným programovým obmedzením tohto programu je pamäť RAM. S narastajúcimi vstupnými parametrami sa zvyšuje aj nápor na pamäť RAM. Napríklad akprogram vyčerpá celú dostupnú pamäť RAM, systém ukončí vykonávanie skriptu chybovou hláškou "KILLED". Tento problém môžeme vyriešiť navýšením pamäte RAM.

4 Zhodnotenie riešenia

Riešenie, ktoré sme pripravili slúži výlučne na vedecké účely. Naimplementovaný program má na starosti komunikáciu medzi firmami. Firmy v ekosystéme, ktorý sme navrhli. Firmy si medzi sebou navzájom posielajú obchodné dokumenty, ktoré následne príjmajú a potom na prijaté obchodné dokumenty odpovedajú. Dokumenty sa počas odosielania indexujú a ukladajú do databázy ElasticSearch, prostredníctvom dátového kanálu Logstash, ktorý má na vstupe dáta z Apache Kafky, Následne si dáta, ktoré sú v ElasticSearch-i, vieme vizualizovať prostredníctvom webového rozhrania Kibana. Riešenie bolo otestované tromi testovacími scenármi, ktorého výsledky sú popísané v jadre tejto diplomovej práce. Študentom, ktorí budú pokračovať v tomto výskume ďalej, odporúčame si naštudovať problematiku, ktorú sme v tejto práci rozoberali, architektúru a program, ktorý sme implementovali a takisto aj priloženú dokumentáciu s prílohami.

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a umelej inteligencie

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát

Diplomová práca

Príloha B - Systémová príručka

Vedúci diplomovej práce:

Ing. Peter Bednár, PhD.

Diplomant:

Bc. Kamil Strachan

Konzultant diplomovej práce:

Ing. Peter Bednár, PhD

Košice 2020

Obsah

1 Funkcia programu

Tento program bol vytvorený v jazyku python a slúži na simulovanie prostredia, kde medzi sebou komunikujú firmy prostredníctvom architektúry, ktorú sme navrhli. Komunikácia v systéme sa inicializuje prostredníctvom troch parametrov:

- Počet firiem,
- počet obchodných partnerov,
- počet správ, ktoré má firma odoslať druhej firme.

Naimplementovaný klient bol naprogramovaný v jazyku Python sa skladá z niekoľkých hlavných častí:

- Trieda Company,
- funkcia odosielania obchodných dokumentov,
- funkcia príjmania obchodných dokumentov,
- funkcia odpovedania na prijaté obchodné dokumenty,
- hlavná funkcia programu.

2 Analýza riešenia

Teoretický základ navrhnutého a vypracovaného riešenia je popísaný v hlavnej časti diplomovej práce.

2.1 Programovací jazyk Python

Praktická časť tejto diplomovej práce bola vyvíjaná v jazyku Python. Python je moderný programovací jazyk, ktorého popularita stále rastie. Na rozdiel od mnohých iných jazykov, ktoré sú kompilačné (C/C++, C) je Python interpreter. To znamená, že:

- interpreter nevytvára spustiteľný kód,
- na spustenie programu musí byť v počítači nainštalovaný Python,
- interpreter umožňuje aj interaktívnu prácu s prostredím.

Hlavné vlastnosti jazyka Python:

- veľmi jednoduchá a dobre čitateľná syntax a keďže Python je aj vysoko interaktívny, je veľmi vhodný aj pre vyučovanie programovania,
- na rozdiel od staticky typovaných jazykov, pri ktorých je treba dopredu deklarovať typy všetkých dát, je Python dynamicky typovaný, čo znamená, že neexistujú žiadne deklarácie,
- Python obsahuje pokročilé črty moderných programovacích jazykov, napríklad podpora práce s dátovými štruktúrami, objektovo-orientovaná tvorba softvéru,
- je to univerzálny programovací jazyk, ktorý poskytuje prostriedky na tvorbu moderných aplikácií, takých ako analýza dát, spracovanie médií, sieťové aplikácie a pod.,
- Python má obrovskú komunitu programátorov a expertov, ktorí sú ochotní svojimi radami pomôcť aj začiatočníkom.

3 Popis programu

Python skript, ktorý bol vytvorený obsahuje jednu hlavnú triedu s názvom Company. Python skript je vytvorený na princípe elektronickej výmeny dát medzi firmami. Komunikácia prebieha na základe týchto pravidiel:

- Každá firma má svojho Producera a zároveň aj Consumera.
- Každá firma má priradený jeden topic - to znamená, že počet topicov sa rovná počtu vytvorených firiem.
- Do topicu konkrétnej firmy sa zapisujú dokumenty, ktoré firma príjme.
- Firma môže odoslať dokument inej firme - to znamená, že danú správa sa zapíše do jej topicu.
- Súčasťou posielanej správy je aj topic kam príjmajúca firma odosiela potvrdenie, že danú správu dostala. V prípade, že firma prijala správu, ktorá odpoveďou už je, ďalej na takúto správu odpovedať nebude.

Z pravidiel, ktoré sme si určili sme navrhli a naimplementovali klienta , kde si učujeme následovné parametre:

- Počet firiem - tým určíme aj počet topicov a samozrejme počet Consumerov a Producerov
- Počet obchodných partnerov firmy - týmto firmám bude môcť odoslať dokumenty
- Počet dokumentov, ktorý má firmá odoslať druhej firme.

Parametre, ktoré sme si zvolili určujú ako veľmi sa v systéme komunikuje. Spustením tohto skriptu, firmy medzi sebou komunikujú a meriame celkové trvanie komunikácie a takisto aj to, koľko obchodných dokumentov sa prostredníctvom Apache Kafka odošle a takisto aj príjme za jednu sekundu.

3.1 Funkcia ___init___(self, name)

Táto funkcia inicializuje konštruktor danej triedy Company. Konštuktor triedy Company vyzerá následovne:

```
1     def __init__(self, name):
2         self.name = name
3         self.messages = []
4         self.set_topic()
5         self.set_producer()
```

Konštruktor triedy Company, obsahuje jeden vstupný parameter, "name". Týmto parameter určíme názov firme. V koštruktore takito inicializujeme zoznam správ, ktoré daná firma prijala. V konštruktore sa takisto nastaví topic firmy a Kafka Producer.

3.2 Funkcia set_topic(self)

Táto funkcia nastavuje názov topicu a pridavá daný topic do zoznamu topicov. Podľa prvého riadku funkcie môžeme vidieť, že názov vytvorenej firmy vlastne predstavuje aj názov topicu, kedže premennú "nameï berie z konštruktora. Funkcia je zobrazená v zdrojovom kóde nižšie.

```
1     def set_topic(self):
2         self.topic = self.name
3         topics.append(self.topic)
```

3.3 Funkcia set_partner(self)

Funkcia set_partner(self) slúži na náhodný výber obchodných partnerov zo zoznamu "partners". Využítá je funkcia sample aby si náhodou firma nezvolila za svojho partnera seba samého. Firma si vyberie toľko partnerov, koľko určuje parameter m.

Funkcia takisto vypíše zoznam partnerov, ktoré daná firma po výbere má. Táto funkcia je zobrazená na ukážke nižšie.

```
1    def set_partner(self):
2        self.partners = random.sample([i for i in topics
             if i != self.topic], k = m)
3        print(self.topic, "business partners: ", self.
             partners)
```

3.4 Funkcia set_consumer(self)

Funkcia set_consumer(self) slúži na inicializáciu Kafka Consumera pre firmu. Kafka Consumera inicializujeme niekoľkými parametrami. Priradíme mu topic, skupinu. Ďalej určíme to od kedy má Consumer čítať dané správy. Nastavenie ëarliest"znamená, že čítať bude všetky správy, ktoré prišli do topicu od začiatku, aj tie, ktoré prišli predtým, ako bol Consumer vytvorený. Parameter "consumer_timeout_msürčuje kedy sa má Consumer vypnúť od posledného prijatia správy. Parameter "bootstrap_serversürčuje to, na aký broker sa má Consumer pripojiť. Táto funkcia je zobrazená na nasledujúcej ukážke.

```
1    def set_consumer(self):
2        self.consumer = KafkaConsumer(self.topic,
3                        group_id="group1",
4                        auto_offset_reset='earliest',
5                        consumer_timeout_ms=1000,
6                        value_deserializer=lambda m:json.
                             loads(m.decode('utf-8')),
7                        bootstrap_servers=server)
8        print("I am consumer", self.topic)
```

3.5 Funkcia set_producer(self)

Funkcia set_producer(self) slúži na inicializáciu Kafka Producera pre firmu. Pre Producera nastavujeme jeden parameter, ktorý určuje na aký broker sa má Producer pripojiť. Táto funkcia je zobrazená na nasledujúcej ukážke.

```
1    def set_producer(self):
2        self.producer = KafkaProducer(bootstrap_servers=
            server)
```

3.6 Funkcia send_messages(self)

Táto funkcia slúži na odosielanie obchodných dokumentov. Daná firma postupne pristupuje ku svojim partnerov, zo svojho zoznamu a následne konkrétnej firme posiela počet dokumentov, ktorý určuje paramerter "d". Funkcia pridáva do správy jej súčasť aj atribút topic, ktorý má v sebe hodnotu firmy, ktorá dokumenty odosiela, aby príjmajúca firma vedela, kde má poslať odpoveď, že faktúru prijala. Táto funkcia je zobrazená na zdrojovom kóde, ktorý je zobrazený nižšie.

```
1    def send_messages(self):
2        for partner in self.partners:
3            for e in range(d):
4                with open('/home/student/test1_scripts/
                    UBLjsons/'+ random.choice(os.listdir(
                    FOLDER_PATH)), 'r') as json_file:
5                    data = json.load(json_file)
6                    message = data
7                    message["topic"] = self.topic
8                self.producer.send(partner, json.dumps(
                    message).encode('utf-8'))
```

3.7 Funkcia receive(self)

Funkcia receive(self) zabezpečuje prijímanie obchodných dokumentov firmou. Vo funkcii sa najprv nainicializuje Kafka Consumer a následné Consumer firmy postúpne číta správy zo svojho topicu a firma ukladá správy do svojho zoznamu prijatých správ aby firma vedela spätne k týmto správam pristupovať. Na záver funkcie sa consumer vypína aby sa progrma mohol ďalej vykonávať. Táto funkcia je zobrazená na zdrojovom kóde, ktorý je zobrazený nižšie.

```
1     def receive(self):
2         self.set_consumer()
3         for message in self.consumer:
4             message = message.value
5             self.messages.append(message)
6         self.consumer.close()
```

3.8 Funkcia respond(self)

Táto funkcia slúži na to aby firma odpovedala na správy ktoré prijala. Funkcia postupne prehľadáva zoznam správ danej firmy. následne zisťuje či daný obchodný dokument obsahuje atribút, ktorý určuje kam má firma poslať potvrdenie o prijatí obchodného dokumentu. Ak ten obsahuje atribút "topic"tak firma pošle konkrétnej firme potvrdenie o prijatí. Ak ten atribút neobsahuje, vypíše sa iba hláška do konzoly o tom, že firma nebude odpovedať na tento dokument. Funkcia je zobrazená na zdrojovom kóde nizšie.

```
1     def respond(self):
2         for i in self.messages:
3             data = json.dumps(i)
4             data2 = json.loads(data)
5             if 'topic' not in data2:
```

```
6                 print(self.topic,": I won't respond to
                      response from you")
7             else:
8                 respond = data2['topic']
9                 response["from"] = self.topic
10                response["response"] = "I just received
                      document from your company"
11                self.producer.send(data2['topic'], json.
                      dumps(response).encode('utf-8'))
12        self.messages.clear()
```

3.9 Funkcia ___main___

Táto funkcia je hlavnou funkciou programu, ktorá sa vykonáva. Funkcia má na vstupe 3 parametre, ktoré inicializujú komunikáciu medzi firmami. Vstupné parametre sa načítavajú z konfiguračného súboru config.txt. Premenná "n"predstavuje parameter, ktorý určuje koľko firiem sa má vytvoriť. Premenná "m"je parameter, ktorý určuje koľko obchodných partnerov má firma mať. Tretia premenná "d", predstavuje parameter, ktorý určuje počet správ, ktoré má firma poslať druhej firme. Premenná "server"predstavuje Kafka Broker na ktorý sa má klient pripojiť. Následne sme definovali aj cestu ku obchodným dokumentom odkiaľ si firma berie faktúry. Potom sme si definovali zoznamy, kde sa uklada nameraný čas v rámci firmy, pre posielanie, príjmanie, odpovedanie a pod. Premenná "n_messages"predstavuje počet odoslaných dokumentov celkovo. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
if __name__ == "__main__":
    configParser = configparser.RawConfigParser()
    configFilePath = r'C:\\Users\\Kam o\\Desktop\\config
       .txt'
    configParser.read(configFilePath)
    n = int(configParser.get('First-test', '
       number_of_companies'))
    print("Number of companies:", n)
    m = int(configParser.get('First-test', '
       number_of_partners'))
    print("Number of partners:", m)
    d = int(configParser.get('First-test', '
       number_of_messages'))
    print("Number of messages which should be send:", d)
    server = '147.232.202.105:29094'
    FOLDER_PATH = r'C:\\Users\\Kam o\\Desktop\\UBLjsons'
    fileNames = os.listdir(FOLDER_PATH)
    response = {}
    sending = []
    receiving = []
    responding = []
    receiving_response = []
    recieving2 = []
    n_messages = n * m * d
```

Program ďalej pokračuje overením toho, či su vstupné parametre správne. Ak sa počet firiem rovná počtu obchodných partnetov, program vypíše chybovú hlášku o tom, aby sme skontrolovali inicializačné parametre. Rovnaký prípad nastane ak počet firiem je menší alebo sa rovná hodnote 0. V prípade, že vstupné parametre sú

nastavené správne, začne sa komunikácia medzi firmami. Ako prvé sa vytvorí počet firiem, ktorý určuje parameter "n". Pri vytvorení nastavujeme aj aké meno majú firmy mať. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
21      if m>=n or m<=0:
22          print("ERROR!!! Check initial parameters")
23      else:
24          print("EDI in Kafka starts...")
25          companies = [Company('company'+str(i)) for i in
                range(n)]
```

Prvou časťou komunikácie medzi firmami je posielanie obchodných dokumentov. Premenná "tempor1"predstavuje premennú, do ktorej sa uloží čas trvania odosielania obchodných dokumentov. Následne sa vykoná prvá slučka pre firmy, kde každá firma postupne posiela obchodné dokumenty svojim partnerom - firma volá funkciu "set_partner()", následne funkciu "send_messages()". Vo funkcii taktiež meriame čas trvania tejto slučky. Počiatočný čas inicializujeme v premennej "start". Do "tempor1"následne ukladáme čas, ktorý sa vykonal v rámci jednej firmy, následne sa ďalej pracuje s touto hodnotou v rámci posielania ďalšou firmou, kde sa znová táto hodnota zmení. Proces takto prebieha do kým sa nevykoná celá slučka a na záver hodnota v tejto premennej obsahuje výsledný čas slučky v sekundách. Premenná "timers_öbsahuje čas odosielania práve pre jednu firmu a danú hodnotu ukladá do zoznamu "sending", kde budú uložené hodnoty trvania času odosielania všetkých firiem v komunikácii. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
26          print("Sending...")
27          tempor1 = 0
28
29          for comp in companies:
30              timer = 0
31              start = time.time()
32              comp.set_partner()
33              comp.send_messages()
34              tempor1 = (time.time() - start) + tempor1
35              timers = (time.time() - start) + timer
36              timers_ = round(timers, 2)
37              sending.append(timers_)
```

Ďalšou časťou komunikácie medzi firmami je príjmanie obchodných dokumentov. Premenná "tempor2"predstavuje premennú, do ktorej sa uloží čas trvania príjmania obchodných dokumentov. Následne sa vykoná druhá slučka pre firmy, kde každá firma postupne prečíta obchodné dokumenty, ktoré prijala od firiem, ktoré si ju vybrali za partnera - firma volá funkciu "receive()". Vo funkcii taktiež meriame čas trvania tejto slučky. Počiatočný čas inicializujeme v premennej "start". Do "tempor2"následne ukladáme čas, ktorý sa vykonal v rámci jednej firmy, následne sa ďalej pracuje s touto hodnotou v rámci príjmania správ ďalšou firmou, kde sa znová táto hodnota zmení. Proces takto prebieha do kým sa nevykoná celá slučka a na záver hodnota v tejto premennej obsahuje výsledný čas slučky v sekundách. Premenná "timers_öbsahuje čas príjmania obchodných správ práve pre jednu firmu a danú hodnotu ukladá do zoznamu "receiving", kde budú uložené hodnoty trvania času príjmania obchodných dokumnetov všetkých firiem v komunikácii. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
38          print("Receiving...")
39          tempor2 = 0
40
41          for comp in companies:
42              timer = 0
43              start = time.time()
44              comp.receive()
45              tempor2 =(time.time() - start) + tempor2
46              timers = (time.time() - start) + timer
47              timers_ = round(timers, 2)
48              receiving.append(timers_)
```

Nasledujúcou časťou komunikácie medzi firmami je odpovedanie na prijaté obchodné dokumenty. Premenná "tempor3"predstavuje premennú, do ktorej sa uloží čas trvania odpovedania na prijaté obchodné dokumenty. Následne sa vykoná tretia slučka pre firmy, kde každá firma postupne posiela potvrdenie o tom, že prijala obchodné dokumenty - firma volá funkciu "respond()". Vo funkcii taktiež meriame čas trvania tejto slučky. Počiatočný čas inicializujeme v premennej "start". Do "tempor3ükladáme čas, ktorý sa vykonal v rámci jednej firmy, následne sa dalej pracuje s touto hodnotou v rámci posielania odpovedí ďalšou firmou, kde sa znová táto hodnota zmení. Proces takto prebieha do kým sa nevykoná celá slučka a na záver hodnota v tejto premennej obsahuje výsledný čas slučky v sekundách. Premenná "timers_öbsahuje čas posielania odpovedí práve pre jednu firmu a danú hodnotu ukladá do zoznamu "responding", kde budú uložené hodnoty trvania času odpovedania všetkých firiem v komunikácii. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
49          print("Responding...")
50          tempor3 = 0
51
52          for comp in companies:
53              timer = 0
54              start = time.time()
55              comp.respond()
56              tempor3 = (time.time() - start) + tempor3
57              timers = (time.time() - start) + timer
58              timers_ = round(timers, 2)
59              responding.append(timers_)
```

Ďalšou časťou komunikácie medzi firmami je príjmanie správ, o tom, že firmy prijali obchodné dokumenty. Premenná "tempor4"predstavuje premennú, do ktorej sa uloží čas trvania tohto procesu. Následne sa vykoná slučka pre firmy, kde každá firma postupne vykonáva daný proces - firma volá funkciu "receive()". Vo funkcii taktiež meriame čas trvania tejto slučky. Počiatočný čas inicializujeme v premennej "start". Do "tempor4ükladáme čas, ktorý sa vykonal v rámci jednej firmy, následne sa dalej pracuje s touto hodnotou v rámci procesu vykonaného ďalšou firmou, kde sa znová táto hodnota zmení. Proces takto prebieha do kým sa nevykoná celá slučka a na záver hodnota v tejto premennej obsahuje výsledný čas slučky v sekundách. Premenná "timers_öbsahuje čas vykonanie tohto procesu práve pre jednu firmu a danú hodnotu ukladá do zoznamu "receiving_response", kde budú uložené hodnoty trvania času tohto procesu všetkých firiem v komunikácii. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
60          print("Receiving responses...")
61          tempor4 = 0
62
63          for comp in companies:
64              timer = 0
65              start = time.time()
66              comp.receive()
67              tempor4 = (time.time() - start) + tempor4
68              timers = (time.time() - start) + timer
69              timers_ = round(timers, 2)
70              receiving_response.append(timers_)
```

Poslednou časťou komunikácie medzi firmami si overujeme, či firmy budú odpoveď už na prijaté potvrdenie od firmy. Premenná "tempor5"predstavuje premennú, do ktorej sa uloží čas trvania tohto procesu. Následne sa vykoná slučka pre firmy, kde každá firma prechádza zoznam svojich správ a kontroluje či má odpoveď na prijatú správu alebo nie - firma volá funkciu "respond()". Vo funkcii taktiež meriame čas trvania tejto slučky. Počiatočný čas inicializujeme v premennej "start". Do "tempor5ükladáme čas, za ktorý sa proces vykonal v rámci jednej firmy, následne sa dalej pracuje s touto hodnotou v rámci vykonávania procesu ďalšou firmou, kde sa znová táto hodnota zmení. Proces takto prebieha do kým sa nevykoná celá slučka a na záver hodnota v tejto premennej obsahuje výsledný čas slučky v sekundách. Premenná "timers_öbsahuje čas trvania procesu práve pre jednu firmu a danú hodnotu ukladá do zoznamu "receiving2", kde budú uložené hodnoty trvania času všetkých firiem v komunikácii. Časť tohto zdrojového kódu je zobrazená na nasledujúcom úryvku.

```
71          print("I want to be sure if companies wouldn't
               respond")
72          tempor5 = 0
73
74          for comp in companies:
75              timer = 0
76              start = time.time()
77              comp.respond()
78              tempor5 = (time.time() - start) + tempor5
79              timers = (time.time() - start) + timer
80              timers_ = round(timers, 2)
81              recieving2.append(timers_)
```

Po vykonaní všetkých slučiek prichádzame k vypisovaniu výsledkov z danej komunikácie. Postupne sa vypisujú hodnoty namerané v jednotlivých slučkách. Vypisujeme čas trvania daného procesu, a rýchlosť posielania správ v danej slučke. Postupne sa to vypisuje pre odosielanie, príjmanie, odpovedanie, prijmanie odpovedí a opätovné odpovedanie. Trvanie sme v každom prípade zaokrúhlili na 2 desatinné miesta. Časť tohto zdrojového kódu je zobrazená na úryvku nižšie.

```
82          print("OK, communication works well!!!")
83          print("Sending lasts {0:.2f} seconds".format(
               tempor1), "with sending {0:.2f} messages per
               second".format(n_messages / tempor1))
84          print("Receiving lasts {0:.2f} seconds".format(
               tempor2), "with consuming {0:.2f} messages per
                second".format(n_messages / tempor2))
85          print("Responding lasts {0:.2f} seconds".format(
               tempor3), "with sending {0:.2f} messages per
```

```
            second".format(n_messages / tempor3))
86          print("Receiving responds lasts {0:.2f} seconds".
               format(tempor4), "with consuming {0:.2f}
               messages per second".format(n_messages /
               tempor4)) Msgs/s".format(n_messages / tempor4)
               )
```

Posledná časť funkcie main a výpisu výsledkov z komunikácie sa skladá z vypísania celkové času trvania komunikácie a to tak, že sme si vytvorili premennú "final", kde sme spočítali namerané hodnoty zo slučiek, ktoré boli vykonané pred tým. Výsledné trvanie je v sekundách a zaokrúhlené na 2 desatinné miesta. Ďalej sme si vytvorili zoznam "final_list", kde si uložíme čas trvania komunikácie pre každu jednu firmu osobitne. To vykonáme zavolaním funkcie "zip", ktorá nám spojí zoznamy "sending", "receiving", "responding", "receiving_responseä "receiving2"do jedného zoznamu. Na záver si vypíšeme najdlhšie a najkratšie hodnotu trvania komunikácie firiem. Časť tohto zdrojového kódu je zobrazená na úryvku nižšie.

```
82          final = tempor1 + tempor2 + tempor3 + tempor4 +
               tempor5
83
84          print("The whole communication lasts {0:.2f}
               seconds".format(final), "with speed of {0:.2f}
                messages per second".format((n_messages*4) /
               final))
85
86          final_list= []
87
88          for (item1, item2, item3, item4, item5) in zip(
               sending,receiving,responding,
               receiving_response,recieving2):
```

```
89              final_list.append(item1 + item2 + item3 +
                    item4 + item5)
90
91          print(final_list)
92          print("The longest communication by a company :",
                max(final_list), "seconds.", "The shortest
                communication by company:", min(final_list), "
                seconds.")
```

4 Konfiguračný súbor

Navrhnutý program si načítava vstupné parametre z konfiguračného súboru. Nasledujúca ukážka predstavuje konfiguračný súbor, ktorý sme použili v rámci prvého testovacieho scenára.

```
[First-test]
number_of_companies = 50
number_of_partners = 15
number_of_messages = 25

[Second-test]
number_of_companies = 50
number_of_partners = 25
number_of_messages = 25

[Third-test]
number_of_companies = 50
number_of_partners = 25
number_of_messages = 35

[Fourth-test]
number_of_companies = 50
number_of_partners = 35
number_of_messages = 25

[Fifth-test]
number_of_companies = 50
number_of_partners = 35
number_of_messages = 35
```

Tento konfiguračný súbor sa skladá z piatich častí, kedže v každý testovací scenár bol rozdelený na 5 prípadov. Pre každý prípad sú parametre nastavené na iné hodnoty. Jeden testovací prípad má 3 vstupné parametre. Parametre, ktoré obsahuje konfiguračný súbor sú následovné:

- number_of_companies - parameter určujúci počet firiem v komunikácii,
- number_of_partners - parameter, ktorý určuje, koľko obchodných partnerov bude mať jedna firma,
- number_of_messages - parameter, ktorý určuje, koľko obchodných dokumentov má jedna firma poslať druhej.

5 Súbor docker compose

Navrhnutá architektúra funguje prostredníctvom docker kontajnerov, ktoré spúšťame prostredníctvom nasledujúceho docker-compose.yml súboru:

```
version: '2'
services:
 elasticsearch:
    image: 'bitnami/elasticsearch:7'
    ports:
      - '9200:9200'
    volumes:
      - 'elasticsearch_data:/bitnami'
  kibana:
    image: 'bitnami/kibana:7'
    ports:
      - '5601:5601'
    volumes:
      - 'kibana_data:/bitnami'
    depends_on:
      - elasticsearch
  zookeeper:
    image: confluentinc/cp-zookeeper
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
```

```
26   kafka:
27     image: confluentinc/cp-kafka
28     hostname: kafka
29     container_name: kafka
30     depends_on:
31       - zookeeper
32     ports:
33       - "9092:9092"
34       - "29094:29094"
35     environment:
36       KAFKA_BROKER_ID: 1
37       KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
38       KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:
            PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT,
            PLAINTEXT_HOST2:PLAINTEXT
39       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092
            ,PLAINTEXT_HOST://localhost:9092, PLAINTEXT_HOST
            2://147.232.202.105:29094
40       KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
41       KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
42   logstash:
43     image: "logstash:7.6.2"
44     volumes:
45       - ./:/home/ubuntu/main
46     command: logstash -f /home/ubuntu/main/logstash.conf
47     depends_on:
48       - elasticsearch
49
50 volumes:
```

```
51  elasticsearch_data:
52    driver: local
53  kibana_data:
54    driver: local
```

Tento súbor obsahuje nasledujúce služby:

- Elasticsearch - využívame docker image bitnami/elasticsearch:7, ktorý spúšťa databázu na porte 9200
- Kibana - využívame docker image bitnami/kibana:7, ktorý spúšťa službu Kibana na porte 5601
- Apache ZooKeeper - využívame docker image confluentinc/cp-zookeeper, ktorý spúšťa službu Zookeeper na porte 2181
- Apache Kafka - využívame docker image confluentinc/cp-kafka, ktorý spúšťa službu Apache Kafka na porte 9092 a pre vzdialený prístup na porte 29094.
- Logstash - využívame docker image logstash:7.6.2, ktorý súšťa službu službu Logstash

Podstatnou časťou v tomto .yml súbore je časť, kde sa konfiguruje Apache Kafka, konkrétne nastavenie "KAFKA_ADVERTISED_LISTENERS". Nastavenie listenerov je dôležité kvôli tomu, aby me vedeli ku Kafka dockru pristúpovať lokálne aj zo vzdialených serverov. "PLAINTEXT://kafka:29092"je hlavný listener, ktorý je napojený na broker. "PLAINTEXT_HOST://localhost:9092"je listener, ktorý umožňuje aby sme mohli ku Kafke pristúpovať lokálne na porte 9092. Posledný listener "PLAINTEXT_HOST2://147.232.202.105:29094"je pre našu prácu najpodstatnejší, kedže vďaka nemu sa vieme na Apache Kafku pripojiť prostredníctvom vzdialeného serveru.

6 Požiadavky na programové vybavenie

Na to aby sa dal naimplementovaný program spustiť je nutné naimportovanie následovných modulov:

- modul kafka - KafkaProducer, KafkaConsumer
- modul random
- modul json
- modul os
- modul time
- modul configparser

6.1 Programové obmedzenia

Hlavným programovým obmedzením tohto programu je pamäť RAM. S narastajúcimi vstupnými parametrami sa zvyšuje aj nápor na pamäť RAM. Napríklad ak program vyčerpá celú dostupnú pamäť RAM, systém ukončí vykonávanie skriptu s hláškou "KILLED". Tento problém môžeme vyriešiť navýšením pamäte RAM.

6.2 Technické parametre

Na vyhotovenie tejto diplomovej práce sme použili osobný laptop a servre Technickej univerzity v Košiciach.

Osobný laptop, ktorý sme použili bol laptop Toshiba Satellite C55-A-1NU. Tento počítač má nasledujúce techické parametre:

- Procesor - dvojjadrový procesor Intel Core i5–4200M pracujúci na frekvencii 2,50 GHz(Turbo 3,1)
- Operačná pamäť - 8 GB DDR3 RAM

- Pevný disk - 256 GB SSD
- Operačný systém - Windows 10 Pro
- Java - Java 1.8.0 221
- Python - Python 3.7.4
- Vývojové prostredie - JetBrains PyCharm Community Edition 2019.2.5

Vedúci diplomovej práce poskytol na vyhotovenie tejto práce 4 školské serveri. Na jednom serveri sme spustili navrhovanú architektúru a na ďalších troch serveroch sme spúšťali klientov, ktorých sme naprogramovali v jazyku Python.

Server na ktorom sme spustili architektúru, bol dostupný na IP adrese 147.232.202.105 a mal nasledujúce techické parametre:

- Procesor - Intel Xeon CPU E5-2667 v4 @ 3.20GHz
- Operačná pamäť - 15 GB
- Pevný disk - 100 GB
- Operačný systém - Ubuntu 16.04.6 LTS, xenial
- Java - OpenJDK Runtime Environment 1.8.0 242
- Textový editor - vim
- Docker - Docker 19.03.8, afacb8b7f0
- Kafka docker image - Docker 19.03.8, afacb8b7f0
- Zookeeper docker image - confluentinc/cp-zookeeper
- ElasticSearch docker image - bitnami/elasticsearch:7
- Kibana docker image - bitnami/kibana:7

- Logstash docker image - logstash:7.6.2

Servre, na ktorých sme spúšťali testovacie skripty boli dostupné na týchto IP adresách:

- 147.232.202.122
- 147.232.202.123
- 147.232.202.124

Tieto servre mali nasledujúce technické parametre:

- Procesor - Intel Xeon CPU E5-2640 @ 3.20GHz, dve jádra
- Operačná pamäť - 1.8 GB
- Pevný disk - 10 GB
- Operačný systém - CentOS 6.10
- Python - Python 3.6.7
- Textový editor - vim

7 Zhodnotenie riešenia

Riešenie, ktoré sme pripravili slúži výlučne na vedecké účely. Naimplementovaný program má na starosti komunikáciu medzi firmami. Firmy si medzi sebou navzájom posielajú obchodné dokumenty, ktoré následne príjmajú a potom na prijaté obchodné dokumenty odpovedajú. Dokumenty sa počas odosielania indexujú a ukladajú do databázy ElasticSearch, prostredníctvom dátového kanálu Logstash, ktorý má na vstupe dáta z Apache Kafky, Následne si dáta, ktoré sú v ElasticSearch-i, vieme vizualizovať prostredníctvom webového rozhrania Kibana. Riešenie bolo otestované tromi testovacími scenármi, ktorého výsledky sú popísané v jadre tejto diplomovej práce. Študentom, ktorí budú pokračovať v tomto výskume ďalej, odporúčame si naštudovať problematiku, ktorú sme v tejto práci rozoberali, architektúru a program, ktorý sme implementovali a takisto aj priloženú dokumentáciu s prílohami.

Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a umelej inteligencie

Návrh architektúry pre spracovanie a analýzu transakčných obchodných dát

Diplomová práca

Príloha C - UBL Faktúra 2.2

Vedúci diplomovej práce:

Ing. Peter Bednár, PhD.

Diplomant:

Bc. Kamil Strachan

Konzultant diplomovej práce:

Ing. Peter Bednár, PhD.

Košice 2020

Obsah

Zoznam tabuliek

1 Atribúty UBL faktúry

UBL dokumenty sú štandarne vo formte XML. Kompletná verzia UBL faktúry verzie 2.2 je dostupná na tomto linku.

- http://www.datypic.com/sc/ubl22/e-ns47_Invoice.html

Z kompletnej UBL faktúry sme si vybrali, že budeme pracovať s týmito atribútmi. Urobili sme aj premapovanie z XML formátu do formátu JSON. Hlavné atribúty a entity faktúry:

- **<cbc:ID>** - atribút, ktorý prideľuje faktúre ID, má typ string
- **<cbc:IssueDate>** - atribút, ktorý hovorí o dátume, kedy bol tento dokument vydaný, prideľuje ho odosielateľ, typ je datum
- **<cbc:IssueTime>** - atribút, ktorý hovorí o čase, kedy bol tento dokument vydaný, prideľuje ho odosielateľ, typ je čas
- **<cbc:DueDate>** - atribút, ktorý hovorí o splatnosti faktúry, typ je dátum -
- **<cbc:PaymentCurrencyCode>** - atribút, kód označujúci menu použitú na platbu faktúry, typ je string
- **<cac:InvoicePeriod>** - entita, ktorá popisuje obdobie, na ktoré sa faktúra vzťahuje, obsahuje subatribúty
- **<cac:OrderReference>** - entita, ktorá odkazuje na objednávku, obsahuje subatribúty
- **<cac:Signature>** - entita, ktorá popisuje podpis, ktorý sa použil na tento dokument, obsahuje subatribúty
- **<cac:AccountingSupplierParty>** - entita účtovnej dodávateľskej strany
- **<cac:AccountingCustomerParty>** - entita účtovnej zákaznickej strany

- **<cac:Delivery>** - entita, ktorá popisuje dodanie
- **<cac:PaymentMeans>** - entita, ktorá popisuje aké sú očakávané platobné prostriedky
- **<cac:InvoiceLine>** - entita, ktorá popisuje položku faktúry

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:ID	ID
cbc:IssueDate	IssueDate
cbc:IssueTime	IssueTime
cbc:DueDate	DueDate
cbc:PaymentCurrencyCode	PaymentCurrencyCode
cac:InvoicePeriod	InvoicePeriod
cac:OrderReference	OrderReference
cac:AccountingSupplierParty	AccountingSupplier
cac:AccountingCustomerParty	AccountingCustomer
cac:Delivery	Delivery
cac:PaymentMeans	PaymentMeans
cac:InvoiceLine	InvoiceLine

Tabuľka 1 – 1 UBL - hlavné atribúty a entity

1.1 InvoicePeriod

Entita, ktorá popisuje obdobie, na ktoré sa faktúra vzťahuje, obsahuje nasledujúce atribúty:

- **<cbc:StartDate>** - atribút, ktorého hodnota je dátum začiatku fakturačného obdobia, typ je dátum

- **<cbc:EndDate>** - atribút, ktorého hodnota je dátum konca fakturačného obdobia, typ je dátum
- **<cbc:StartTime>** - atribút, ktorého hodnota je začínajucí čas fakturačného obdobia, typ je čas
- **<cbc:EndTime>** - atribút, ktorého hodnota je končiaci čas fakturačného obdobia, typ je čas
- **<cbc:Description>** atribút, ktorý popisuje obdobie faktúry, typ je string

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:StartDate	StartDate
cbc:EndDate	EndDate
cbc:StartTime	StartTime
cbc:EndTime	EndTime
cbc:Description	Description

Tabuľka 1 – 2 UBL - InvoicePeriod

1.2 OrderReference

Entita, ktorá odkazuje na objednávku, obsahuje nasledujúce atribúty:

- **<cbc:ID>** - identifikátor pre referenciu na objednávku, pridelená kupujúcim, má typ string
- **<cbc:CopyIndicator>** - atribút označuje, či je odkazovaná objednávka kópia alebo original, má typ boolean
- **<cbc:UUID>** - atribút, ktorý popisuje jedinečný univerzálny identifikátor tejto objednávky, má typ string

- **<cbc:IssueDate>** - atribút popisuje dátum vytvorenia objednávky, má typ datum
- **<cbc:IssueTime>** - atribút popisuje čas vytvorenia objednávky, má typ čas

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:ID	ID
cbc:CopyIndicator	CopyIndicator
cbc:UUID	UUID
cbc:IssueDate	IssueDate
cbc:IssueTime	IssueTime

Tabuľka 1 – 3 UBL - OrderReference

1.3 AccountingSupplierParty

Entita účtovnej dodávateľskej strany, obsahuje tieto atribúty:

- **<cbc:CustomerAssignedAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený samotným zákazníkom, má typ string
- **<cbc:SupplierAssignedAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený dodávateľom, má typ string
- **<cbc:AdditionalAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený treťou stranou, má typ string
- **<cac:Party>** - entita, ktorá popisuje stranu dodávateľa
- **<cac:DespatchContact>** - entita, ktorá popisuje kontakt na dodávateľskú stranu pre odoslanie

- **<cac:AccountingContact>** - entita, ktorá popisuje kontakt na dodávateľa pre účtovanie
- **<cac:SellerContact>** - entita, ktorá popisuje primárny kontakt na dodávateľa

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:CustomerAssignedAccountID	CustomerID
cbc:SupplierAssignedAccountID	SupplierID
cbc:AdditionalAccountID	AdditionalID
cac:Party	Party
cac:DespatchContact	DespatchContact
cac:AccountingContact	AccountingContact
cac:SellerContact	SellerContact

Tabuľka 1 – 4 UBL - AccountingSupplier

1.4 AccountingCustomerParty

Entita účtovnej zákazníckej strany, obsahuje tieto atribúty:

- **<cbc:CustomerAssignedAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený samotným zákazníkom, má typ string
- **<cbc:SupplierAssignedAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený dodávateľom, má typ string
- **<cbc:AdditionalAccountID>** - atribút, ktorý popisuje identifikátor pre účet zákazníka, pridelený treťou stranou, má typ string
- **<cac:Party>** - entita, ktorá popisuje stranu zákazníka

- **<cac:DeliveryContact>** - entita, ktorá popisuje dodavací kontakt na zákazníka
- **<cac:AccountingContact>** - entita, ktorá popisuje zákaznícky kontakt pre účtovanie
- **<cac:BuyerContact>** - entita, ktorá popisuje stranu zákazníka

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:CustomerAssignedAccountID	CustomerID
cbc:SupplierAssignedAccountID	SupplierID
cbc:AdditionalAccountID	AdditionalID
cac:Party	Party
cac:DeliveryContact	DeliveryContact
cac:AccountingContact	AccountingContact
cac:BuyerContact	BuyerContact

Tabuľka 1 – 5 UBL - AccountingCustomer

1.5 Delivery

Entita, ktorá popisuje dodanie, obsahuje tieto atribúty:

- **<cbc:ID>** - identifikátor pre dodávku, má typ string
- **<cbc:ActualDeliveryDate>** - atribút, ktorý hovorí aký je skutočný datum dodania, má typ dátum
- **<cbc:ActualDeliveryTime>** - atribút, ktorý hovorí aký je skutočný čas dodania, má typ čas

- **<cbc:LatestDeliveryDate>** - atribút, ktorý hovorí aký je najnovší dátum dodania, má typ dátum
- **<cbc:LatestDeliveryTime>** - atribút, ktorý hovorí aký je najnovší čas dodania, má typ čas
- **<cbc:TrackingID>** - identifikátor sledovania dodávky, má typ string
- **<cac:DeliveryAddress>** - entita, ktorá popisuje adresu dodania
- **<cac:RequestedDeliveryPeriod>** - entita, ktorá popisuje požadovanú lehotu na dodanie,
- **<cac:PromisedDeliveryPeriod>** - entita, ktorá popisuje sľúbenú lehotu dodania
- **<cac:EstimatedDeliveryPeriod>** - entita, ktorá popisuje odhadovanú lehotu dodania
- **<cac:DeliveryTerms>** -entita, ktorá popisuje dodacie podmienky

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:ID	ID
cbc:ActualDeliveryDate	ActualDeliveryDate
cbc:ActualDeliveryTime	ActualDeliveryTime
cbc:LatestDeliveryDate	LatestDeliveryDate
cbc:LatestDeliveryTime	LatestDeliveryTime
cbc:TrackingID	TrackingID
cac:DeliveryAddress	DeliveryAddress
cac:RequestedDeliveryPeriod	RequestedDelivery
cac:PromisedDeliveryPeriod	PromisedDelivery
cac:EstimatedDeliveryPeriod	EstimatedDelivery
cac:DeliveryAddress	DeliveryTerms

Tabuľka 1 – 6 UBL - Delivery

1.6 PaymentMeans

Entita, ktorá popisuje aké sú očakávané platobné prostriedky, obsahuje tieto atribúty:

- **<cbc:ID>** - identifikátor platobného prostriedku, má typ string
- **<cbc:PaymentDueDate>** - datum splatnosti, má typ datum
- **<cbc:PaymentChannelCode>** - atribút je kód označujúci platobný kanál platobného prostriedku, má typ string
- **<cbc:PaymentID>** - atribút je identifikátor platby uskutočnenej pomocou platobného prostriedku, má typ string
- **<cac:CardAccount>** - entita, ktorá bližšie špecifikuje kreditnú, debetnú alebo inú platobnú kartu, ktorá predstavuje platobný prostriedok

- **<cac:PayerFinancialAccount>** - entita, ktorá popisuje finančný účet platiteľa
- **<cac:PayeeFinancialAccount>** - entita, ktorá popisuje finančný účet veriteľa

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:ID	ID
cbc:PaymentDueDate	PaymentDueDate
cbc:PaymentChannelCode	PaymentChannelCode
cbc:PaymentID	PaymentID
cac:CardAccount	CardAccount
cbc:PayerFinancialAccount	PayerFinancialAccount
cac:PayeeFinancialAccount	PayeeFinancialAccount

Tabuľka 1 – 7 UBL - PaymentMeans

1.7 InvoiceLine

- **<cbc:ID>** - atribút, ktorý je indetifikátorom pre riadok faktúry, má typ string
- **<cbc:InvoicedQuantity>** - atribút, ktorý hovorí o tom koľko riadkov je na faktúre, má typ decimal
- **<cbc:LineExtensionAmount>** - atribút, ktorého hodnota je celková suma pre položky faktúry vrátane poplatkov, ale bez daní, má typ decimal
- **<cbc:AccountingCost>** -atribút, ktorý vyjadruje aké je účtovné nákladové stredisko kupujúceho pre položky faktúry, ktoré sú vyjadrené ako text, má typ string
- **<cac:AllowanceCharge>** - entita, ktorá popisuje príspevok alebo poplatok spojený s riadkami faktúry

- **<cac:Item>** - entita, ktorá popisuje položku priradenú k tomuto riadku faktúry
- **<cac:Price>** - entita, ktorá špecifikuje cenu tejto faktúry

Universal Business Language(XML)	Universal Business Language(JSON)
cbc:ID	ID
cbc:InvoicedQuantity	InvoicedQuantity
cbc:LineExtensionAmount	ExtensionAmountID
cbc:AccountingCost	AccountingCost
cac:AllowanceCharge	AllowanceCharge
cac:Item	Item
cac:Price	Price

Tabuľka 1 – 8 UBL - InvoiceLine

Dokument faktúry, ktorý je v stave ako sme opísali v tejto príručke, slúži ako obchodný dokument, ktorý si posielajú firmy v rámci komunikácie.