5. melléklet

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

DIPLOMAMUNKA

OE-NIK Hallgató neve: **Vincze Dénes**

2021.12.15 Hallgató törzskönyvi száma: **T/003790/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Intézet

SZAKDOLGOZAT vagy DIPLOMAMUNKA

FELADATLAP

Hallgató neve: **Vincze Dénes**
Törzskönyvi száma: **T/003790/FI12904/N**

A dolgozat címe:

Hogyan töltsünk adatot Big Data rendszerekbe Kafka-val
How can we load data into Big Data by Kafka

Intézményi konzulens: Dr Holyinka Péter

Külső konzulens: Szakács Balázs

Beadási határidő: 2021.12.15

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat:

Egy szoftverfejlesztő cég az általa fejlesztett szoftver felhasználási módjait kívánja begyűjteni. Határozza meg a cég által elvárt igényeket, és a feladat megoldásához javasolható technikai eszközöket, szoftvereket. Ismerje meg a szoftvereket, tervezze meg és fejlessze ki az alkalmazást. Mutassa be a rendszer működését egy példán, melyben a konkrét cég igényein alapuló adatgyűjtés történik. A bemutatásnál törekedjen arra, hogy a példában bemutatott lépések irányt mutassanak más igénylők esetén történő alkalmazásra.

A dolgozatnak tartalmaznia kell:

1. a probléma ismertetését,
2. a kielégítendő igényeket, és az alkalmazással szemben támasztott követelményeket,
3. a felhasználásra javasolt szoftverek tömör ismertetését,
4. a rendszer terveit,
5. a konkrét példa dokumentációját, kiemelve az egyes lépéseket,
6. az elért eredményeket és a tovább fejlesztési lehetőségeket.

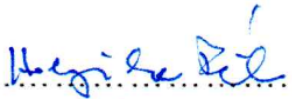
Ph.

intézet igazgató

A dolgozat (OE TVSz. 32.§ 8. pont szerinti) elévülésének határideje: 2022. december 15

A dolgozatot beadásra alkalmasnak tartom:

külső konzulens



intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2021.12.15

hallgató aláírása

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Vincze Dénes..... Neptun Kód: [redacted]..... Tagozat: .....

Telefon: [redacted] Levelezési cím (pl.: lakcím): .....

Szakdolgozat / Diplomamunka[1] címe magyarul:
Hogyan töltsünk adatot Big Data rendszerekbe Kafka-val .....

Szakdolgozat / Diplomamunka[2] címe angolul:
How can we load data to Big data system by Kafka .....

Intézményi konzulens: Dr. Holyinka Péter.....

Külső konzulens: Szakács Balázs.....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2020.03.11	Feladat kiírás	
2.	2021.04.22	Üzleti terv készítés	
3.	2020.04.29	Irodalom kutatás	
4.	2020.05.22	Véglegesítés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

.....
Intézményi konzulens

Budapest, 2020.05.22

[1] Megfelelő aláhúzandó!

[2] Megfelelő aláhúzandó!

[3] Megfelelő aláhúzandó!

[4] Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Vincze Dénes..... Neptun Kód: [redacted] Tagozat: .....

Telefon: [redacted] Levelezési cím (pl.: lakcím): .....

Szakdolgozat / Diplomamunka[5] címe magyarul:
Hogyan töltsünk adatot Big Data rendszerekbe Kafka-val .....

Szakdolgozat / Diplomamunka[6] címe angolul:
How can we load data to Big data system by Kafka .....

Intézményi konzulens: Dr. Holyinka Péter.....

Külső konzulens: Szakács Balázs.....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.11.12	Adatok letöltése	
2.	2021.11.30	Python	
3.	2021.12.02	Kafka-python csomag	
4.	2021 12.06	Adatok kiírása	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 [7] tantárgy követelményét teljesítette, beszámolóra / védésre [8]bocsátható.

.....

Intézményi konzulens

Budapest, 2021.12.15

[5] Megfelelő aláhúzandó!

[6] Megfelelő aláhúzandó!

[7] Megfelelő aláhúzandó!

[8] Megfelelő aláhúzandó!

1. Bevezető és a szakdolgozat célja

Magyarországi szoftverfejlesztő cég életében fontos az, hogy az ügyfél elégedett legyen a termékkel. Az elégedettség mértékétől a szoftverbe épített visszajelző rendszeren tud a cég választ kapni, fontos továbbá megjegyezni, hogy a felhasználóknak, milyen egyéb észrevételei vannak. Ezek a visszajelzések napló állományokban érkeznek be a cég életébe, amiket különböző szervereken tárolnak. Az eltérő helyen tároltnapló állományokat szeretnénk egy központi helyre összegyűjteni, és ezeket egy helyen letárolni, és abban az esetben amikor szükség van rájuk elemezni legyünk képesek őket. A cégnek fontos az információ a szoftverrel kapcsolatban, mivel az információ mindenki számára értéket képvisel. Az adatokat biztonságosan szeretnénk eljuttatni a célrendszerbe, amihez adatvezetékre van szükségünk.

Megértéshez szükséges beszélni arról, hogy valójában mi is azaz adatvezeték és miért van rá szükség, mire lehet használni, és hogyan épül fel ez a valós életben. Többféle adatvezeték típus létezik az iparágban, vannak kifejezetten adattárházhoz és kifejezetten Big Data technológiát használó rendszerekhez.

A mi esetünkben Big Data technológiát használó adatvezetéket fogjuk megvizsgálni, és ezek közül is az Apache Kafka-t, mivel a cég életében Apache Kafka lesz idővel bevezetve. Vizsgálni fogjuk, hogy bevezetés milyen előnyöket eredményez.

Először általánosságban beszélnünk kell az adatvezetékekről, hogy ennek a működését megértsük, és ezt követően lehet áttérni, mint az Apache Kafka vezeték típusra.

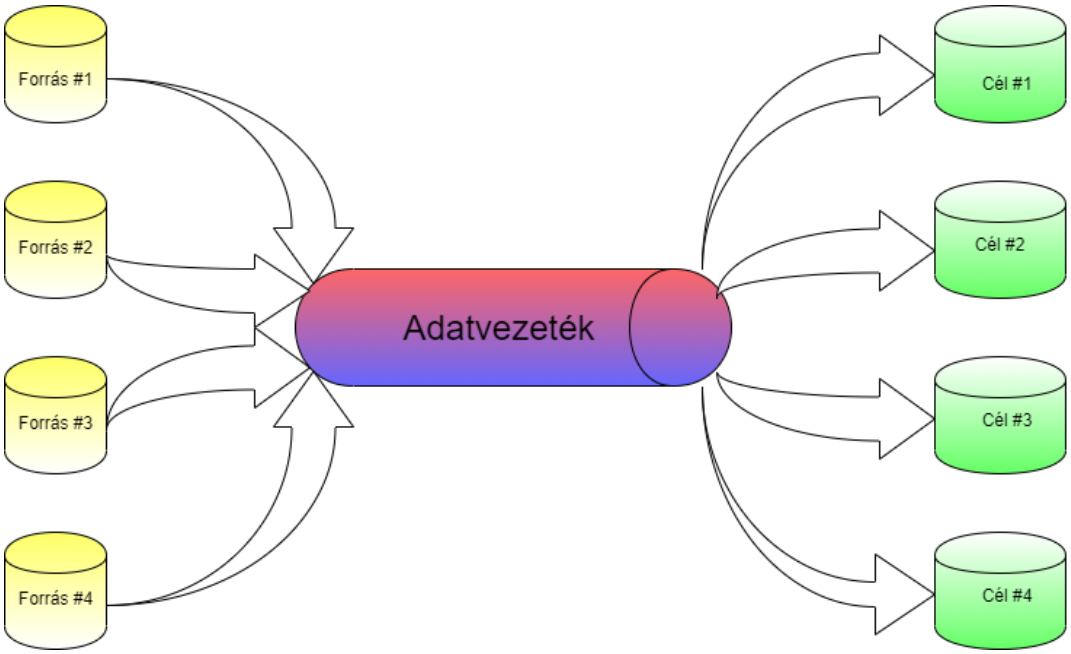
2. Üzleti igények

Egy informatikai cég életében több helyen léteznek naplóállományok, és a feladatom ezeknek a napló állományoknak az eljuttatását megtervezni célrendszerbe Apache Kafka segítségével oly módon, hogy adatokat ne veszítsünk menet közben. Fontos azokra a kritériumokra odafigyelni, amire valós életben igazán lehet számítani, bármilyen típusú adatátvitel esetén. Kafka esetében ilyenek például, ha esetleges hálózat kimaradás, számítógép kiesés számítógép csoportból adatok lassabban jönnek át az áteresztő képességnél, vagy plusz számítógép lép be csoportba. Számítógép csoport létrehozása virtuális környezetben, virtuális környezetre a Kafka telepítése, és a naplóállományok összegyűjtése több forrás rendszerből és Kafka segítségével ezt a virtuális gép csoporthoz eljuttatni.

3. Data pipeline

3.1. Mi az a data pipeline? [2] [4]

Data pipeline magyar nevén “adatcső”-nek vagy “adatvezeték”-nek nevezhetjük, az adatok betöltése és adatok feldolgozásának hatékonyság-növelése céljából lett kifejlesztve. Adatvezeték az adatfeldolgozási lépések sorozata. Első körben meg kell határozni, hogy milyen adatot hova és hogyan gyűjtsük őket össze. A betöltésekben, adatmanipulációs tevékenységekben, adatok érvényesítési folyamatában, és további elemzések céljából lehetőség van rá,ezeket a tevékenységeket automatizálni. Data pipeline egy olyan szoftver, amely képes arra, hogy a meglévő erőforrás teljes kapacitását ki tudja használni a szűk keresztmetszet, hibák és / vagy késleltetés leküzdésével együttesen. Streaming data pipeline-k egy olyan bizonyos data pipeline-k, amelyek képesek valós idő kereteken belül több millió eseményt kezelni, ezek az események lehetnek információk gyűjtése, információk elemzése és ezeknek az információknak akár az eltárolása is egy időben. Streaming data pipeline képessége lehetővé tudja tenni, hogy alkalmazások és az elemzések valós idő kereteken belüli jelentések készítését.



ábra 3.1Data pipeline vizuális megjelenítése

Egyszerre több forrás rendszerrel lehet kapcsolatban a data pipeline. Data pipeline az összes adatot data streaming-ként kezeli, függetlenül attól, hogy statikusan, azaz forrás fájlokból, avagy teljes mértékben valós idejű forrás rendszerekből, azaz tranzakciós forrás rendszerekből származnak az adatok. A data pipeline a forrás rendszerből érkező adatokat kisebb részekre osztja fel, és ebben az esetben valós körülmények között egy erre dedikált cluster szerverünk, avagy szervereink vannak, és ekkor az adatokat párhuzamosan dolgozza fel az adatcsövünk, úgyhogy extra számítási teljesítményt biztosítost a rendszer számára. A data pipeline-nak nincsen szüksége arra, hogy

közvetlen háttértárral rendelkezzen. Az adatokat akár egy külső alkalmazásba is tölthetjük (pl. Salesforce, SAP). Ha az adatok még nincsenek betöltve az előre meghatározott platformra, akkor az adat a data pipeline elején helyezkedik el. Lépések sorozataként értelmezzük, amikor az egyik hely kimeneti adata a másik a hely bemeneti adataként jelenik meg a folyamat lezárásáig.
A data pipeline-ok három nagyon lényeges részből tevődnek össze:

- egy vagy több forrás rendszerből
- egy feldolgozási rendszerből
- egy vagy több cél rendszerből.

Data pipeline-ok lehetővé tudják tenni, az adatok áramlását úgy, hogy az adatok el tudjanak jutni az adattárházakba, data lake-kbe, analitikus adatbázisokba, és tranzakciós rendszerekbe. Hasonló esetekben ezek a különböző lépések történhetnek párhuzamosan is, egy data pipeline-nak egyszerre lehet több bemenete is

3.2. Miben különbözik az adatvezeték az ETL-től? [1] [2] [4]

ETL egy speciális data pipeline-t határoz meg. ETL rövidítése “Extract Transform Load”. Adatok mozgása egy forrás rendszerből való kinyerése például alkalmazásból, egy cél rendszerbe való áttöltéséért felelős, leggyakrabban adattárházakba. “Extract” adatok kinyerését jelenti a forrás rendszerből, “Transform” adatok manipulálását jelenti, hogy az adatokat a cél rendszerbe betölthető formába kerüljenek, “Load” pedig az adatok adattárházba való betöltéséért felelős.

3.3. Big data bevezetés és Big adatvezeték [2]

Sven Löffler-től idézzem, aki Business Development Executive IoT Analytics és Data Economy rendszerekkel foglalkozik. Ő mondta miszerint, Big Data-nak van egy 2 betűs rövidítése, ami a “3V”, ez a mennyiségből (Volume) sebességből (Velocity) és a változatosságból (Variability) ered. Mennyiség másodpercenként előállított adatokból ered, ami lehet videóanyag, szöveges forrás, és még sok forrásanyag, avagy rendszer. Ezeket az adatokat nem szeletenként kapjuk, hanem folyamatos áramlásként, úgy képzeljük mintha nyitva hagyjuk a csapot. Ilyen mennyiségű adat nagyon sok lehetőséget nyújt olyan esetekre, mint prediktív elemzés, valós idejű jelentéskészítés és riasztások. Az adatok struktúrájához az adatvezetékeknek is fejlődnie kellett, hogy tudja támogatni a big data-ba adatok betöltését. Big Data pipeline-k olyan típusú data pipeline-k, amelyeket nagy mennyiségű adatok befogadására terveztek. Big Data adat mennyisége érdekessé teszi pipeline építését. Ezen adatoknak letárolhatóaknak és valós időben feldolgozhatóaknak kell lenniük, ennek köszönhetően bizonyos műveleteket végre lehet velük hajtani. Big Data adatmennyisége megköveteli, hogy a adatvezeték méretezhető legyen, mivel az adatok összessége idővel változhat. Valós körülmények között sok Big Data esemény történik, mint például eszközök közötti kommunikáció, avagy adatgyűjtés, közösségi médián valamilyen életesemény megosztása, amelyeknek az időpontjai közel állhatnak egymáshoz, így tehát Big Data vezetékeknek minden esetben képesnek kell lenniük arra, hogy méretezhetőek

legyenek a jelentős mennyiségű adatok egyidejű feldolgozásaira. Big Data megköveteli, hogy data pipeline-k képesek legyenek adat struktúráját megkülönböztetni, értem itt alatta strukturált, félig strukturált és a strukturálatlan adatokat.

3.4. Data pipeline hatékony építése [2]

3.4.1. Data pipeline építési szempontok [3]

Data pipeline architektúráknál sok szempontot kell figyelembe venni. Például data pipeline-nak kell streaming adatokkal dolgoznia? Milyen adat rátával kell dolgoznia? Mennyi és milyen típusú adatfeldolgozás fog történi data pipeline-on? Adatok cloud-ban vagy on-prem generálódnak és hova kell továbbítani? Tervbe van véve, hogy a data pipeline microservice-kel lesz felépítve? Csapatban van olyan kolléga, aki rendelkezik az adott területnek megfelelő programozási és üzemeltetési szaktudással?

3.4.2. Fejlesztési komplexitás csökkentése [3]

Ne írjunk felesleges kódot, és így nem fogunk elveszni a kód rengetegben.
Ha nincsen szükség arra, hogy valamilyen új betöltő eszközt lefejlesszünk, akkor azt próbáljuk meg elkerülni, de ha mégis szükség van egy ilyen eszközre, ami adatokat egyik helyről másikra tudja áttölteni, és nincsen ehhez hasonló termék a piacon akkor érdemes átgondolni, hogy azt lefejlesszük-e.
Ebben az esetben szükséges egy adatvezetéket felépíteni, amit későbbiekben használunk, és próbáljuk a jó architektúra tervezésre helyezni a hangsúlyt. Abban az esetben, ha van egy jól szakképzett adatmérnök kollégánk, kérjük meg őt, hogy vizsgálja meg, hogy milyen adatok típusok honnan érkeznek milyen rendszerekből, csak naplóállományok jönnek, vagy valamilyen tranzakciós rendszerből érkeznek be az adatok, vagy ennek a kettőnek valamilyen hibrid kivitelből.
Ökölszabályként elmondható, hogy viszonylag könnyebb egy bonyolultabb megoldást összetenni, de gondolkodásra és megfontolásra van szükség, egy egyszerűbb rendszer kiépítéséhez. Ez a mondás valójában nem olyan, mint az adatvezeték építése során, amelyek hagyományosan részt vesznek a kód blokkok írásában, az adatok kinyerése a forrás rendszerekből és ezek az átalakítás és betöltése (ELT) céljából, és az adatok data pipeline-on keresztül data-lake-kbe és adattárházakba tölteni. Ha a csapat szokott ETL folyamatokat írni, valamilyen alkalmazás segítségével, ez akár legyen Microsoft Visual Studio, akkor gyorsan abba a helyzetbe találhatja magát, hogy a forrás állomány és annak az adatbázis kódnak, hasznos funkcióit, mint összekapcsolását vagy összehangolását, vagy infrastruktúra építését ETL folyamatokhoz, spagetti kódhoz kapcsolják különböző mérsékelt forrás- és célrendszerekhez. Fő feladata ennek a rendszernek a kezelése lehetne, ahelyett, hogy megérteni az alapadatok struktúráját, és elgondolkodni a következő adat mennyiség mozgásának a megoldásán.
Kétféle lehetőség van, hogy erre a problémára megtaláljuk a megoldást:

- Stratégia vásárlás

Big data területén robbanás történt kereskedelmi integrációs eszközök körében. Ezen eszközök célja, hogy az adatok és az esemény naplókat, azaz a log-kat összegyűjtse a forrás rendszerekből és ezeket az adatokat célrendszer belépési pontjára szállítsa. Ezek az alkalmazások a folyamat során újratölthetik, tisztíthatják, transzformálhatják, lemásolhatják, rendezhetik és végrehajthatnak rajta más adatmanipulációkat, leggyakrabban drag and drop funkciókkal. Alkalmazások többsége kommunikál más alkalmazásokkal, adatfolyamokkal, IoT rendszerekkel, mobiltelefonokkal, fájlokkal és adatbázisokkal, alapvetően bármilyen olyan eszközzel vagy rendszerrel, ami adatot avagy rekordot képes generálni felhőben vagy földi környezetben. Ezen túl vannak illesztőprogramok és beépülő modulok a leggyakrabban használt eszközökhöz.
Néhány általában használt ETL-integrátor eszköz:

- ❖ Snowplow Analytics
- ❖ Stitch Data
- ❖ Five tran.

Ezek közül mind SaaS-ajánlatok, ezeket már alacsony költségekkel el lehet érni a piacon.

- Stratégia építés

Ha cégen belül már van tapasztalt kolléga, vagy esetleg külsős vállalkozó szerepében van már egy tapasztalt adatmérnöki csapat, akkor támogatásukkal a rendszer felépítése és üzemeltetése egyszerűbbé válhat. Erre a célra léteznek különféle eszközök, melyek akár szép grafikus felhasználói felülettel rendelkező alkalmazást biztosítanak az átíráshoz. Emellett beépített támogatást nyújtanak minden olyan folyamathoz, mint például fájlok megnyitása, olvasása, írása, adatok másolása, ciklusok írása bizonyos feladatokra, ütemezés.
Ilyen alkalmazások például:

- ❖ Pentaho data integration (Kettle néven terjedt el)
- ❖ DBT (Data Build Tool)
- ❖ Python code editor.

Mindig kövessük a clean code elvét. A clean code elv a szoftverfejlesztésben azt jelenti, hogy a forráskódban igyekezzünk kerülni a kód ismétlést, a kódismétlés mellőzése a kód karbantarthatóságát nagymértékben leegyszerűsíti, és ezzel növeli a hatékonyságot mind a fejlesztési és mind üzemeltetési területen. Big Data iparágában paradigma elmozdult a terjedelmes MapReduce irányából az alkalmazás írásának minimalizálásáig. Ha a komplexitást nem kezeljük jól, abban az esetben adatok előállításához szükséges megannyi erőforrás, valamint az adatok tárolására képes adatbázisok és számos eszköz robbanásszerű alkalmazása a működés átláthatatlanságához és kezelhetetlenségéhez vezethet.
Munka kiszervezése egy lényeges lépés, án tartsuk belső hatáskörökben a feladatok üzemezését, rendszer monitorozását, függőségek kezelését tartalmazó kódot. Lehet, hogy nincsenek nyílt forráskódú eszközök, és ebben az esetben érdemes lehet pénzt

költeni ilyen típusú eszközökre, minthogy kód ismétléses forráskódot elemezve csapdákba fussunk.

Építsünk be data pipeline-ba online ellenőrzést. Az adatok célrendszerbe történő betöltésénél problémát tud jelenteni a késedelemesen érkező forrásadat, valamint a helytelen, hibás adatok. Az adatoknak ellenőrizhetőnek kell lenniük, mielőtt betöltjük az éles cél környezetbe. Az utolsó lépést megelőzően töltsük be az adatokat egy átmeneti táblába, majd ezt követően futtassuk le sorban az szükséges érvényesítési lépéseket, amelyek metaadatok által vannak vezérelve. Ezek a modellek lehetővé teszik az SQL-ben írt validációk meghatározását, és a csatolását az ETL folyamatokhoz. A keretrendszer olvassa a metaadatokat az egyes feladatok végén, és azokat végre hajtja, és egyezteti az eredményeket a határértékekkel. Ha esetleg az érvényesítés meghiúsulna, akkor abban az esetben a végrehajtás sikertelen, különben betölti a ideiglenes táblából a végleges helyére az éles környezetbe. Szükséges karbantartani soron következő érvényesítési lépéseket a metaadat-vezérelt architektúrán keresztül. Automatizáljuk azokat a dolgokat, amikre van lehetőség. Érvényesítési lépéseknek tartalmazniuk kell az összes folyamat ellenőrzését (számlálás másolás stb...), mind funkcionalitást (számoknak van-e értelmük, összehasonlítás egy ismert halmazzal).

3.4.3. Megbízhatóság és a méretezés [2] [3]

Egy jól megtervezett adatvezetéknek a következőket tartalmaznia kell:

- Futtatás

A forrásadatok (bármilyen okból történő) újrabeállítása végzetes hiba folytán néha szükséges lehet az ETL folyamat újbóli futtatására. Az ETL folyamatnak kell, hogy rendelkezzen egy visszaállítási ponttal, és képesnek kell lennie arra, hogy a visszaállítási pontból vissza tudjon úgy állni, hogy bármilyen emberi, avagy kódbeli beavatkozásra ne legyen szükség (munka újraindításához szükséges kivétel). Ez a rendszer működésére jellemző adatokon (metaadatokon) alapuló ETL data pipeline-on keresztül érhető el, ahol minden folyamat ellenőrzi az állapotát, és minden átalakítás (transzformáció) újrafuttatható.

- Felügyelet

Bármikor egy lekérdezés (vagy felhasználói felület) meg kell, hogy tudja mondani, hogy mi a rendszer állapota. Milyen folyamatok futnak a rendszeren és hol történik a végrehajtás folyamata, ha esetleg megakadna a futtatás. Nagymérétkben meg tudja könnyíteni a felhasználó életét, abban az esetben, ha szerkezetileg felügyeleti szoftver adatvezeték tetején helyezkedik el, mivel ebben az esetben könnyebben lehet megtekinteni az adatvezeték helyes működését. Nagyon sok felügyeleti szoftvert kínáló cég található meg az iparágban (ilyen például az AirFlow), azonban ezeknek a lehetőségeknek további kód és fejlesztési leírására lehet igény.

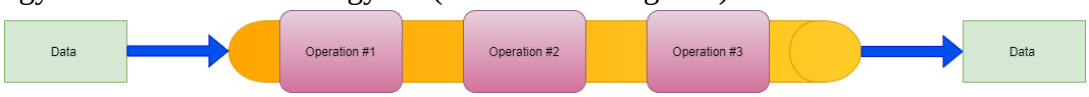
- Automatikus méretezés

Az data pipeline-t a várható terhelés szerint kell méretezni. Fontos elkerülni az alulméretezett infrastruktúrát, mivel ez egy data pipeline esetén azt jelenti, hogy

“bedugul” és nem ér időben célba az adat, vagy rosszabb esetben az adatok el is veszthet. A másik véglet is kerülendő hiszen, ha a nap 24 órájában és a hét hét napján számítógép csoportokat üzemeltetünk terhelés nélkül az rendkívül pazarló tud lenni. Szerencsére a jelenlegi technológiákkal könnyen megoldható ez a probléma. A nagy felhő szolgáltatók, mint az Amazon vagy Google olyan keretrendszert adnak, amiben könnyen tudunk az igényeknek megfelelően akár perceken belül számítógép csoportokat elindítani, leállítani vagy fel és leméretezni már meglévő csoportokat.

3.4.4. Architektúra példák

Data pipeline-t több lehetséges úton lehet felépíteni. Az egyik legelterjedtebb módja a data pipeline felépítésnek a batch típusú data pipeline felépítés. A lenti ábrában lehet egy olyan alkalmazás, amiben például egy tranzakciós forrás rendszernek az eredményeit be kell tölteni egy adattárházba és ezt a folyamatot követően az adatok egy elemzési más néven egy BI (Business Intelligence) adatbázisba kell betölteni.



ábra 3.2Data pipeline architektúra

- Forrásrendszer

Forrás az adat áttöltési folyamat adatbetöltési lépéseinek az első pontja. Forrásrendszer lehet vállalatok jelentés készítésének és elemzések adatai, környezeti források adatai, tranzakciós rendszerek, IoT-eszközök és ezeknek az érzékelői, szociális média, alkalmazások API-jai vagy ehhez kapcsolódó bármilyen egyéb forrás rendszer.

- Célrendszerek

A célrendszer az a végpont, ahol szeretnénk az adatokat végül tárolni. Ez lehet egy adatbázis/adattárház (pl. MySQL adatbázis, Google BigQuery), fájl alapú adattároló (FTP szerver, Amazon S3) vagy valamilyen alkalmazás (Pl Salesforce, SAP).

3.5. Data pipeline típusok [11]

Nagyon sok eltérő data pipeline megoldás létezik, amelyeket eltérő célokra és megoldásokra lehet felhasználni. Megtörténhet az az eset is, egy felhő típusú szolgáltatásokat használva, hogy a felhő natív eszközt is használhat abban az esetben, ha át szeretnénk helyezni az adatokat a felhős környezetbe.

Az alábbi lista bemutatja melyek a leginkább elterjedtebb a data pipeline típusok. Figyelembe kell venni, hogy ezek a rendszerek nem zárják vagy zárhatják ki egymást. Előfordulhat az az eset is, hogy van egy data pipeline, amely felhős szolgáltatásokra míg egy másik például valós idejű streaming-re van optimalizálva.

- Batch

Batch típusú feldolgozás akkor lehet a legalkalmasabb, ha nagy mennyiségű adatot szükséges feldolgozni rendszeres időközönként, nincsen arra szükség, hogy az adatok valós időben jutassuk el “A” pontból “B” pontba. Ez az eljárás hasznos lehet abban az esetben, ha például marketing adatokat szeretnénk eljuttatni naponta egyszer egy

nagyobb rendszerbe például egy elemzés céljából, ilyen típusú adatok lehetnek például: számlázó, pénzügyi piacokról.

- Real-time

Real time eszközök valós időben történő adatfeldolgozásra lettek optimalizálva. A valósidejű szolgáltatás abban az esetben tud hasznos lenni, amikor az adatok streaming típusú forrás rendszerből érkeznek, azaz valamilyen tranzakciós rendszerből kapjuk őket, ilyen rendszerek (például: telemetriai eszköztől az adatok lehetnek)

- Cloud-native

Cloud-native eszközök felhőalapú adatfeldolgozásra lettek kifejlesztve, hogy felhőalapú adatokkal tudjanak együtt dolgozni, ilyen például AWS (Amazon Web Service) verem szolgáltatás. Ebben az esetben az adatok felhő rendszerben vannak tárolva. Ennek köszönhetően nem kell a cégnek beruháznia teljes, avagy rész infrastruktúrára és ehhez szakmai humán jellegű erőforrásra. Az adatok fent vannak felhő környezetben és a felhőt biztosító szolgáltatóra lehet hagyatkozni mind humán erőforrásokban és mind informatikai erőforrások üzemeltetésében.

- Open-source

Open-source magyar nevén nyílt forráskódú rendszerek abban az esetben a leghasznosabbak, ha az adott rendszer kiépítésére alacsony a költségvetése a cégnek, de rendelkezik szakmai háttérrel, hogy ezt a rendszert tudja bővíteni és módosítani a saját céljai alapján. Nyílt forráskódú rendszerek gyakrabban olcsóbbak, mint a kereskedelemben megtalálható vetélytársaik, mert ezen az alapon szolgáló felhasználók számára módosítani és / vagy bővíteni szándékoznak a fejlesztők.

3.6. Első lépések

Vállalaton belül meggyőződtünk arról, hogy nekünk szükségünk van egy adatvezeték kiépítésre, hogy az adatokat el tudjuk juttatni a feldolgozó egységhez, vagy ezeket az adatokat el tudjuk tárolni, hogy ezeket későbbiekben tudjuk majd elemezni. Mi esetünkben ez a Kafka lesz, mint adatvezeték. Kiválasztottuk milyen típusú adatvezetékünk lesz. Meg kell határozni, hogy milyen felületen fog futni az adatvezetékünk, azaz a Kafka, mivel van Windows-s és van Linux-s környezetre egyaránt. Mert a két rendszernek más-más a követelménye. Miután eldöntöttük, hogy milyen rendszert fogunk használni, ki kell építeni a számítógép csoportokat.

4. Kafka [5]

A Kafka egy publikusan elérhető, egy tartós üzenetküldő rendszer, amely képes adatcseréket végre hajtani folyamatok, alkalmazások és szerverek között. Ez a fejezet röviden szeretné az olvasó közönséggel megértetni az üzenetküldést az elosztott napló állományokat, meghatározni a Kafka fontos fogalmait, és végezetül ez a fejezet elmagyaráza a kapcsolat felépítésének a lépéseit.

4.1. Bevezető a Kafka-ba

Kafka-t úgy kell elképzelni, mint egy postást ahogy a hétköznapi életben dolgozik. 3 részre lehet a Kafka lépéseit felosztani ugyanúgy, mint ahogy az adatvezeték esetében lettek felosztva. Van a forrásrendszer, ami Kafka esetében a feladó. Van az data pipeline-unk ami ebben az esetben a Kafka broker más néven a postás és posta rendszere, és van a célállomás ami fogyasztó vagy a végfelhasználó az esetünkben. Apache Kafka Scala-ban és Java-ban íródott, LinkedIn-nél dolgozó adatmérnökök kezei által. Korábban már 2011 körül, egy olyan rendszert adtak át a nyílt forráskódú rendszert kedvelő csoportok és emberek számára, ami már akkoriban skálázható üzenetküldő rendszer volt. Napjainkban az Apache Kafka összefolyó sztrim platform része és napi események billióit látja el. Apache Kafka jó néhány megbízható vállalatnál tömte be ezt a piaci rést.

4.2. Mire használjuk a Kafka-t?

Röviden Kafka-t streamelési folyamatokra használják. Weboldalak az aktivitás nyomon követésére, metrikus adatok gyűjtésére, és felügyeleti rendszerek napló állományok összegyűjtésére. Valós időben és folyamatban lévő adatok elemzésekre. Komplex esemény folyamokra, adatok betöltésére Spark-ba, Hadoop rendszerre adatok betöltésére, parancssoros lekérdezésen alapuló szétválasztárásra (CQRS command query responsibility segregation), üzenetek visszajelzésére, hibák visszafejtésére, belépési elosztott napló memóriában történő számítására

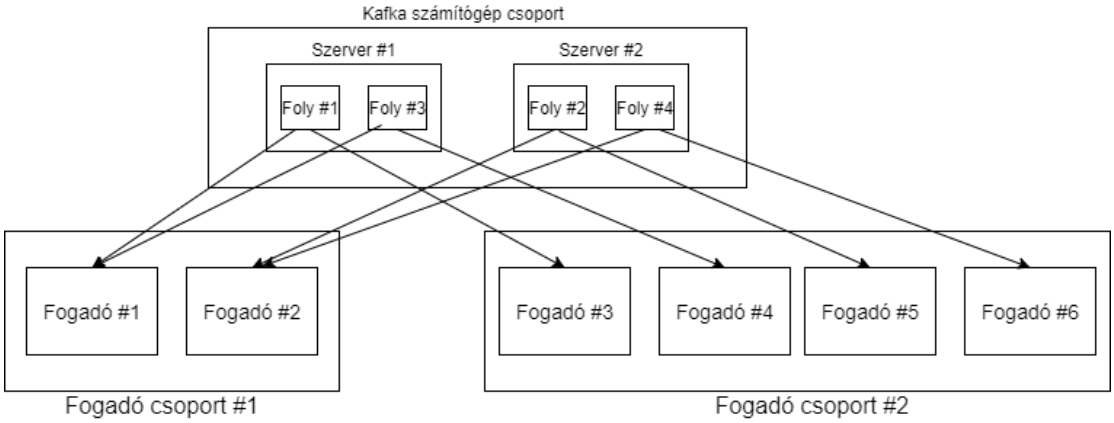
4.3. Kafka működése

4.3.1. Producer szempontból

Producerek az általuk választott adatokat avagy rekordokat megosztják, és ennek kiválasztásáért és megosztásáért a producer a felelős. Ezeket a kiválasztásokat meg lehet tenni egy rekord valamilyen kulcsa alapján.

4.3.2. Consumer szempontból

Consumer egy consumer csoportban vannak, akik bizonyos témájú üzenetekre vannak feliratkozva. Egy téma egy bizonyos csoporthoz továbbítódik, és egy témát több számítógép dolgoz fel, akkor az erőforrás megoszlik közöttük. Abban az esetben, ha minden fogadó különböző témájú csoportban van, akkor minden csoport számára az azokat a rekordokat továbbítani szükséges.



ábra 4.1Costumer kinézete

2 Kafka szerver csoport szolgáltat 4 (Foly #1 - #4) partíciót 2 consumer csoportnak. Consumer csoport #1-nek van 2 fogadó csoport#2-nek van 4 fogadó példánya. De általánosságban azt tapasztalhatjuk, hogy témánként kevesebb fogadó csoportok vannak., 1-1 témához csak 1-1 feliratkozó van. Minden egyes csoport sokféle féle példányból állhatnak össze, hogy hibatűrést, és skálázhatóságot prezentálnak. Ez nem más, mint a feliratkozóknak a szemantikájának a közététele, ahol egyetlen folyamatnak a leírása a fogadók helyett a fogyasztói csoport.

A felhasználás módja, hogy a Kafka egyenlő mennyiségre felosztja a napló állományokat, úgyhogy minden egyes példány egyenlően legyen szétosztva a fogadói csoportok és a fogadók között. Csoporttagság dinamikus fenntartását Kafka eljárás üzemelteti. Abban az esetben, ha új fogadók csatlakoznak be a csoportokba, akkor néhány fogadói példány területet átvesznek a többi csapattagtól. Ha egy fogadó kiesne, akkor a kiesett fogadó feladatait a maradék fogadó között osztják fel.

Kafka csak az összes rekord feletti kiosztásokkal foglalkozik, nem foglalkozik a témák közötti partíciókkal. A partíciónként megrendelés és a kulcsonként szétosztás elegendő a többi alkalmazás számára. Akkor, ha szükség van az összes rekordra, akkor ezeket tudja csatolni a témához egy partícióként, de ez csak egy felhasználói csoportnál csak egy felhasználói folyamatot eredményez.

4.4. Teljesítmény és skálázhatóság

4.4.1. Producer szempontjából

Feladó felelős a Kafka esetében, hogy az ügynök megkapja a feladótól az előállított adatokat, hogy az ügynök el tudja indítani a kézbesítés folyamatát. Különben a kézbesítési folyamat nem fog megtörténni. Feladókat lehet különféle módokon optimalizálni, úgyhogy az Apache Kafka-nak a beállítás igényeinek a kézbesítendő adatok megfeleljenek kézbesítés paramétereinek. A producer-i finom hangolással el lehet kerülni a gyakran előforduló hibákat, ennek érdekében meg kell győződnünk arról, hogy a beállítások megfelelőek-e az elvárásoknak, a producer beállításának ellenőrzésével.

Elismerési értékek vagy nyugtázási érték megmondja, hogy a consumer fél milyen típusú választ jelzett a feladó fél részére, erre legtöbbször 3 beállítást szoktak alkalmazni, melyek a következők.

- Elismerési-érték

Elismerés egy olyan visszajelzés, mely kommunikációk között átadódnak, ami azt jelenti, hogy a fogadó fél azaz Kafka producer megkapta az üzenetet. Elismerési érték egy producer-i konfigurációs paraméter érték, amelyek az Apache Kafka-ban az alábbi értékekre értéket vehet fel

- 0-ás érték esetén

A producer sose várja meg az broker-től a választ, abban az esetben, ha az elismerési érték nullára van beállítva. Ekkor nem lehet garantálni, hogy az broker ténylegesen megkapja a producer-től a kézbesítendő üzenetet. Ilyenkor a feladó nem küldi újra a kézbesítendő csomagot és ez esetben az üzenet elveszik, mert a feladó nem kap arról választ, hogy a csomagja elveszett-e vagy az broker megkapta-e a producer csomagját. Ez a típusú beállítás nagy kockázattal jár, mivel fennáll adatvesztés veszélye. A lehetséges adatvesztés független a sávszélességtől. Nagy sávszélességnél nagyobb adat mehet át, így nagyobb adat is veszhet el, azaz a sávszélesség nem garancia arra, hogy adatok biztosan eljutnak az broker-hez.

- 1-es érték esetén

Ha az elismerés értéke 1-esre van beállítva abban az esetben a feladó értesítést kap arról, hogy a fogadó ebben az esetben a broker megkapta az első csomagot, avagy a rekordot. Abban az esetben, ha producer nem kap visszajelzést az ügynöktől, akkor az átvitel megáll. Ha az első üzenet küldésére a feladó, az broker-től nem kap elismerési választ, hogy az broker fogadta az üzenetet. Ilyenkor az üzenet csak abban az esetben veszik el, ha az első csomagra azonnal nem kap visszajelzést a feladó. Ezt követően, hogy elismerte az broker, hogy megérkezett hozzá az elküldött üzenet, de ez még azelőtt, hogy a producer a következő csomagot elküldené. Ez a típusú beállítás egy késleltetést eredményez számunkra, ami egy lassabb áteresztő képességet tud okozni, de a kézbesített csomagok nagyobb valószínűséggel érkeznek meg a producer broker-ünkhöz. Ez egy lassabb folyamat, de tartósabb is egyben, mint 0 értékű elismerés esetében.

- Egész értesítési érték esetén

Elismerési érték egészre állítása azt jelenti, hogy producer kap egy elismerési visszaigazolást arról, hogy minden egyes rekord megérkezett a customer-hez, a mi esetünkben az broker-hez. A producer jelen állapot szerint az broker megvárja a teljes szinkronizált duplikációt, hogy vissza tudja igazolni, hogy a rekordokat megkapta. Ez azt tudja eredményezni, hogy hosszabb ideig tart a rekordok átvitele, de minden egyes üzenet megérkezik az broker-hez.

- Kafka elismerési értékének a beállítása

Hogy a legnagyobb áteresztő képességet tudjuk megkapni nullára javasolt az értéket beállítani. Hogyha nem szeretnék, hogy adatvesztés lépjen fel abban az esetben egészre vagy -1-re állítsuk be az értesítési értéket. Ebben az esetben nem lesz a legnagyobb az áteresztő képessége a rendszernek. Abban az esetben hogyha értesítési érték 1-re van állítva akkor gyorsabban megkapjuk az rekordokat, de fennállhat az a veszélyforrás, hogy nem érkezik meg az összes rekord, de ebben az esetben gyorsabban megérkeznek a rekordok, ha -1-t vagy egészre állítsunk be az értéket.

- Mit jelent az, hogy szinkronban.

Ha a Kafka úgy véli, hogy a rekord jóvá van hagyva, amikor minden egyes másolat bekerülnek a szinkronizálás másolatába, ezt szinkronizált másolat jóvá hagyja, és kikerülnek az elemek a lemezre. Az egész elismerési érték beállítása kéri, hogy egy elismerési választ küldjünk abban az esetben, ha a szinkronizált másolatnak minden egyes rekord megérkezett.

- Mi is az a szinkronizálási másolat

A szinkronizálási másolat egyszerűen egy területnek a teljes másolata, ami “szinkronban” van az elsővel. Alapjaiban a “szinkronizáció” függhet a területi beállításoktól, de alapértelmezetten, azt jelenti, hogy a másolat szinkronban maradt az első példánnyal az utolsó 10 másodpercben. Ezt a periódust “*replica.lag.time.max.ms*” alapjaiban a kiszolgáló ezt fel tudja írni bizonyos témánként.

A szinkronizálási másolat legalább az első tagból, és minden ezt követő tagból tud összeállni, amelyeket szintén szinkronban kell kezelni. A követő duplikált adatok az első tagtól kezdve az utolsó tagig, periodikusan kérnek egy frissítést, alapjáraton minden 500 milliszekundumonként.

Ha a követő elbukik, akkor abba maradnak a frissítések, és a szinkronizált másolatból 10 másodpercet követően törlődik. Hasonlóan, ha a követő másolatok lelassulnak, esetlegesen egy hálózati kimaradás vagy szerver oldali hiba esetén, amint elkezdtek lassulni első másolat mögött több mint 10 másodpercre már törlődik szinkronizált másolatból az adat. Szóval a célunk az, hogy képesek legyünk ezeket a szélső értékeket tolerálni, esetleg lehet egy-egy másolat elvesztése vagy belassulása.

- Mit tehetünk a szinkronizálási másolatért

A szinkronizálás másolat elismerési beállítási kompromisszum a biztonság és a késleltetés között.

Mint feladó, ha mi nem akarunk semmi féle körülmény között üzenetet elveszíteni, érdemes meggyőződnünk arról, hogy minden üzenet másolata megérkezett a szinkronizálási másolatba, azelőtt, hogy fogadnánk az elismerési választ. De a probléma ezzel, hogy egy másolatot elveszhetünk, avagy a partíció lassulás okozhat olyan esetet, hogy a partíció válasza extrém mértékben megnő rosszabb esetben elérhetetlenné válik.
Abban az esetben, mikor a producer-nek elismerési értéke egészre be van állítva. Azt mondja, hogy csak akkor adj nekem egy elismerési választ egyszer, amikor az összes szinkronizált másolatról megkapja az üzenetet. Ha a másolat elbukott vagy nagyon lassú, akkor ez az üzenet nem lesz a szinkronizálási másolatnak a része, és így nem okoz majd késedelmet rosszabb esetben elérhetetlenséget, és ilyen esetben legtöbbször ezeket az üzeneteket, avagy rekordokat eldobják.
Elmondhatjuk, hogy a szinkronizálási másolat egyensúlyba hozza a biztonságot, a rendelkezésre állást, és a késedelmeket. De ennek az egésznek van egy Achilles sarka. Ha az összes követő esetleg belassulna, akkor a szinkronizálási másolat csakis kizárólag az első másolatból tevődik össze. Tehát egy elismerés az egész, attól az üzenet nyugtázható, ha csak egyetlen másolat van az meg az első. Az az üzenet veszelyezteti azt, hogy az első üzenet utáni üzenetek elvesszenek. Ezt úgy lehet orvosolni, hogy a "*min-insync.replicas*" ügynök részén beállítjuk. Ha ezt például 2-re állítjuk, akkor, ha szinkronizálási másolat összetömörödik 1 másolattá, akkor a bejövő üzenetek elutasítottak lesznek. Ez egy biztonsági lépés arra, mikor, hogy az üzenetek vesztését minél inkább minimalizáljuk.

4.4.2. Broker szempontjából

Ez a fejezet arról fog szólni, hogy hogyan lehet beállítani az broker szemszögéből

- Több partíció nagyobb áteresztő képesség

Egy partíciót csak egy felhasználó tud kezelni, ebből eredően meg kell, hogy egyezzen a partíciók száma a felhasználóknak a számával. Több partíció több felhasználót is eredményez a párhuzamos olvasáshoz, különben nem beszélhetünk párhuzamos olvasásról. Ez több skálázható rendszert tud magába foglalni. Nagyobb területtel meg lehet oldani, hogy nagyobb áteresztő képességet kapjunk, mivel a fogadófelek párhuzamosan tudnak olvasni és nem szekvenciálisan.

- Ne állítsunk be túl sok partíciót

Partíciók kulcsa, azaz, hogy a Kafka skálázható legyen, de ez nem azt jelenti, hogy több partíciót érdemes létrehozni, mint amire szükségünk lenne vagy szükségünk van. Léteznek feladók, akiknek több partíciójuk van, mint amennyire szüksége lenne, ami meg teljes erőforrást fel tudja használni a szerverről. Vannak partíciók, amik nagyon sok memóriát emésztenek fel (ilyenek például a fájl leíró területek). Processzornak a terheltsége is megnő több partíció esetén, mivel a Kafka-nak nyomon kell követnie minden egyes partíció mozgását. Egy topiknak több mint 50 partíció ritkán szokott kelleni a legjobb gyakorlatban is

- Az egyensúly Kafka magok és a fogadó felek között

Egyes területek a Kafkában egy szálasak. Túl sok terület esetén nem lesz hatékony, ha kevés mag áll rendelkezésre a szerverekben. Ennek okául fontos a jó egyensúlyt meghatározni, és mellé beállítani a magok és a fogadok között. Nem jó stratégia, ha fogadó felek nincsen kihasználva azért, mert nincsen megfelelő erőforrás a ügynök számára.

- Kafka broker

Több Kafka broker egy számítógép csoportban, magasabb erőforrást eredményez, így ennek köszönhetően a terhelés is szétoszlik a rendelkezésre álló gépek között. Másolatkészítés a Kafka egyik erőssége, és a lényeges eleme, amíg eldönti, hogy hány ügynökre lesz szükség számítógépes csoportban addig minimum egy másolatot készít az adatokról.

- Minimum szinkronizálási másolatok

A minimum szinkronizálási másolatoknak a számát meg kell határozni, hogy hány másolatra van szükségünk, ahhoz, hogy a feladó el tudja küldeni sikeresen a területre a rekordokat. A témában szereplő adatok másolatát a beállító személy tudja a rendszerre szabni. Másolatok számát jövőre tekintve lehet utólag módosítani.
A legnagyobb minimum szinkronizálási másolatnak a száma magasabb perzisztenciát okoz, de a másik végről csökkentheti a rendelkezésre állást, mert még nem áll rendelkezésre a minimális másolatok száma, addig nem lehet megosztani az adatokat. Ha van 3 számítógép csoportunk és a minimális szinkronizálási másolatok száma be van állítva 3-ra, és az egyik csoport az megáll, akkor a másik 2 csoport is elérhetetlen lesz ebben az esetben, és nem fogja tudni fogadni az adatokat se. Oda kell figyelni, hogy minimum hány szinkronizációs másolatot készítünk, mikor számítógép csoport rendelkezésre áll és a megbízhatóság ezt tudja garantálni.
A minimum szinkronizálási másolatok száma nem befolyásolja az áteresztőképességet. Minimum szinkronizálási másolatok száma legalább 1, akkor jobb esetben kevesebb vagy egyáltalán nincsen adatvesztésünk, de továbbra az elismerési értéktől függ az átviteli sebesség mértéke.
Az alap minimum szinkronizálási másolatok száma az 1 CloudKarafka-nál. Ez azt jelenti, hogy ha minimum szinkronizálási másolatok száma 1 akkor legalább egyszer el kellett tudni küldeni sikeresen a partícióra rekordokat.

- Partíciók betöltése ügynökök között

A közös probléma az, hogy a betöltés nem szétosztott az broker-ek között. Ezen fontos rajta tartani a szemünket, hogy szétosztott partíciók legyenek szétosztottak és újraosztottak az új broker-ek között, ha szükséges, győződjünk meg arról, hogy közel minden broker egyforma terheltség alatt van.

4.4.3. Felhasználó szempontjából

Itt is lényeges, hogy a pontosan legyenek finomhangolva a beállítások ugyanúgy, mint a feladó esetében, mivel a jól hangolt finom beállításokkal elkerülhetőek a gyakran előforduló hibák.

Felhasználók napló eseményeket olvashatnak az broker-től, egy meghatározott kezdeti speciális eltolással. Felhasználó tud kapcsolódni egy csoporthoz, amit úgy neveznek, hogy felhasználói csoport. Egy felhasználói csoport tartalmazhat, egy felhasználói folyamatot, ami leírhat egy meghatározott témát vagy témakört. Ezt követően a felhasználók szétosztják maguk között a témaköröket, behatárolva azt, hogy minden témának egyetlen felhasználója legyen, és azt a témát csak egy felhasználó dolgozza fel. Például a csoportban minden egyes felhasználó fel van iratkozva egy folyamatra, amit Kafka garantál, hogy egyetlen feliratkozója tudja csak olvasni csoportból napló fájlt.

- Felhasználók kapcsolódása

Győződjünk meg róla, hogy a felhasználói csoportban minden felhasználónak megfelelő a kapcsolata az broker-rel. Felhasználók minden egyes alkalommal újra felosztják egymás között a feladatokat, mikor valamelyik felhasználó ki- vagy beszáll a csoportba. Ameddig a csoportban a feladatok újraosztása tart, addig a csoportban lévő felhasználók nem tudják olvasni a rekordokat. Abban az esetben, ha egy felhasználónak rossz kapcsolata van, ekkor a teljes csoport érintett és elérhetetlenné válik addig még a felhasználó újra nem kapcsolódik. A partíciók újraosztása megközelítőleg 2-3 másodpercet vagy akár ennél több időt igénybe vehet. Érdemes meggyőződni arról, hogy a beállítások gördülékenyen futnak, kifejezetten javasolt, hogy a felhasználók közötti kapcsolat 100%-osan megbízható legyen.

- Felhasználók száma

Ideális esetben, partíciók száma meg kellene, hogy egyezzen a fogadók számával, azonban ez eltérhet a felhasználási feltételeknek köszönhetően.
Nézzünk meg két esetet:

- Felhasználók száma az nagyobb, mint az brokerek száma

Abban az esetben, ha a felhasználó felek száma nagyobb, mint az broker-ek száma, akkor vannak olyan felhasználók, akik üresjáratban vannak, ami azt eredményezheti, hogy az erőforrás oldalon ebben az esetben a felhasználói oldalról pazarlás léphet fel.

- Felhasználók száma az kisebb, mint az broker-ek száma

Hasonló felhasználók tudnak egyszerre több broker-től is olvasni, abban az esetben hogyha az ügynökök száma nagyobb, mint a felhasználók száma.

4.5. Kafka elterjedése

Kafka egyszerű üzemeltetéssel rendelkezik. Kafka-t egyszerű beállítani, és egyszerű kitalálni hogyan működik. Azonban a Kafka fő szempontja az, hogy nagyon jó a teljesítménye. Stabil, megbízható, és tartós, rugalmas a megosztásokban és publikus feliratkozásokkal és várólistával rendelkezik, melyeket jól tud illeszteni N számú producer csoporthoz, komoly másolat készítővel rendelkezik, producer-ek számára beállítható biztos garanciát nyújt, és stabil elérést nyújt minden egyes rekord számára.

5. Data lake [6]

Data lake egy olyan tárhely, ahol központilag lehet tárolni nagy mennyiségű nyers adatot natív formában, addig a pontig ameddig nincsen analitikai alkalmazásokhoz, mint például Microsoft általi PowerBI eszközhöz, vagy egyéb analitikai eszközökhöz. Data lake szembe megy a hagyományos adattárház struktúrával, mivel adattárház strukturáltan tárolja le az adatokat, addig a data lake strukturálatlanul, úgymond ömlesztett formában.

Data lake adatforrásokat gyakran azonosítjuk Hadoop fájlrendszerrel. Ez az elosztott adatfeldolgozási keretrendszereken alapúló üzembe helyezéseknél az adatok Hadoop Distributed File System-be (más néven HDFS) töltődnek be, és Hadoop-fürtöket (más néven node-kat) különböző számítógépes csomópontokon lehet ezeket megtalálni. Hadoop helyett egyre inkább kezd elterjedni a felhős alapú szolgáltatás, mivel felhőben nem a tárolás okozza nagy költségeket, hanem az adatmozgatás tudja ezeket a költségeket generálni. Bizonyos NoSQL adatbázisok már data lake alapú architektúrában vannak már letárolva.

5.1. Data lake architektúra

Számos technika létezik data lake kialakításra, ezeket szolgáltatók különböző technikákkal kombinálják össze. Ez azt eredményezi, hogy egyik szolgáltató például Hadoop-t telepít, és ezt kombinálja össze Spark feldolgozó egységgel, és HBASE-zel, egy HDFS-n futó NoSQL adatbázissal. Egy másik esetben futhat egy rajta egy Spark az Amazon Simple Storage Service (S3)-ban tárolt adatokon. Egy harmadik esetben egy ismét eltérő eset léphet fel.

Ezenfelül nem minden data lake tárol csak is kizárólag forrásrendszer oldali nyers adatokat, hanem ezekben a data lake-kben előfordulhat, hogy ezeknek adatoknak a feldolgozása során kiszűrődhetnek adatkészletek, és későbbiekben ezeket fel lehet dolgozni. Ezeknek data lake-knek rendelkeznie kell valamilyen fajta analitikára alkalmas eszközzel is, és tesztkörnyezettel.

Mindazonáltal három fő architekturális elv különböztethető meg az adatforrásoktól a hagyományos adattáraktól:

- Semmilyen adatot nem kell visszautasítani. A forrásrendszerekből gyűjtött adatok mindegyike betölthető és megőrizhető egy data lake-ban, ha arra szükségünk van.
- Az adatok tárolhatók transzformálatlan vagy részben transzformálatlan állapotokban, ahogyan azok a forrásrendszerből érkeztek.
- Ezeket az adatokat később átalakítják, és szükség szerint egy strúkturába illesztik az adott elemzési követelmények alapján, ez a megközelítés struktúra olvasásakor ismert.

Bármilyen technológiát is használnak fel a data lake telepítésekor, vannak olyan más elemek is, amiket be kell építeni annak biztosítására, hogy a data lake működőképes

legyen, és a benne szereplő adatok biztonságban legyenek, és benne lévő adatok ne menjenek kárba, vagy rosszabb esetben vesszenek el. Ez a következőket tartalmazza:

- Általános mappastruktúra elnevezési konvenciókkal.
- Kereshető adatkatalógus, amely segít a felhasználóknak az adatok megtalálásában és azoknak a megértésében.
- Adatosztályozás rendszerezése az érzékeny adatoknak azonosítására, olyan információkkal, mint az adattípus, a tartalom, a használati forgatókönyvek és a lehetséges felhasználók csoportjai.
- Szabványosított adathozzáférési folyamat, amely segít ellenőrizni és nyomon követni, hogy ki férhet hozzá az adatokhoz.
- Adatvédelem, például adatmaszkolás, adattitkosítás és automatizált felhasználási megfigyelés.

Az data lake felhasználóinak adattudatossága szintén elengedhetetlen, különösen, ha olyan üzleti felhasználók is vannak köztük, akik állampolgári adattudósként tevékenykednek. Amellett, hogy a felhasználóknak kiképzést kapnak az data lake-ban való navigálásról, ismerniük kell a megfelelő adatkezelési és adatminőségi technikákat, valamint a szervezet adatkezelési és -használati szabályzatát.

6. Rendszer követelmény és telepítése

6.1. Rendszer követelmények [7]

Big Data rendszerek alapvetően Linux típusú operációs rendszerekre vannak tervezve és optimalizálva, így én is egy Ubuntu 20.04.2 ARM alapú operációs rendszert választottam. Kafka-nak szüksége van JAVA-ra, így minimum követelmény egy 2 magos és 4 gigabájt RAM-mal ellátott Linux operációs rendszerre van szükségünk. Én esetemben 6 gigabájt RAM-t használtam el, és 2 processzormagot. Kafka processzor architektúrától függetlenül működik, mert tud X64-s és ARM alapú architektúrán is elfutni. JAVA miatt érdemes 4 gigabájtnál több memóriát kiosztani, mert későbbiekben használhatatlanul lassú lesz a gép. Linux esetén csomagok kapcsán van csak különbség, és így lehet telepíteni mind kliens gépre és mind szerver típusú operációs rendszerre a Kafka-t, minden egyéb más Linux verzióra. Az alábbi parancsok eltérhetnek egyéb Linux disztribúció esetén.
OpenJDK 8-ra vagy újabb verzióra van szükség, és ennek a telepítésére Linux alapú operációs rendszeren. Kafka JAVA nyelven íródott, így JVM-t igényel. Kafka 2.1.1-s verziónál openJDK 8-ra van szükség. Én Kafka 2.12-3.0.0-t állítottam be ott működik újabb OpenJDK verzióval. Jelenleg nálam 11-s JAVA verzió van telepítve.

6.1.1. JAVA telepítése

JAVA legegyszerűbb telepítési formája az Ubuntu vagy egyéb linux csomag telepítésének a formája. Első körben a csomagok indexelését frissítsük meg.

```
sudo apt update
```

ábra 6.1Csomagok újra indexelése

Következő lépés, hogy megnézzük, hogy rendelkezzünk-e már telepített JAVA-val.

```
java –version
```

ábra 6.2JAVA verzió lekérdezése

Abban az esetben, ha a JAVA nincsen telepítve az operációs rendszerre a következő hibaüzenetet fogjuk kapni.

```
Command 'java' not found, but can be installed with:
sudo apt install default-jre
sudo apt install openjdk-11-jre-headless
sudo apt install openjdk-8-jre-headless
```

ábra 6.3 JAVA verzió üzenet

Ha a fenti hibaüzenetet megjelent a képernyőn, akkor a következő paranccsal tudjuk telepíteni a JRE-t az operációs rendszerre.

```
sudo apt install default-jre
```

ábra 6.4 JRE csomag telepítés

Ha a telepítés sikeres azt követően kérdezzük le a verzióját.

```
java –version
```

ábra 6.5JAVA verzió lekérdezése

Sikeres telepítés követően a következő üzenetet kell, hogy lássuk a képernyőn:

```
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.18.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.18.04, mixed
mode, sharing))
```

ábra 6.6 JAVA verzió üzenet

Előfordulhat, hogy a JRE mellett szükség lehet a JAVA fejlesztői környezetre is (JDK). Ezt a következő paranccsal lehet végrehajtani:

```
sudo apt install default-jdk
```

ábra 6.7JDK csomag telepítés

Sikeres telepítést követően nézzük meg, hogy milyen verzió települt fel:

```
javac –version
```

ábra 6.8 JDK verzió verzió lekézdezés

Abban az esetben, ha a következő üzenetet kapjuk a fenti parancs által akkor a JAVA-t sikeresen telepítettük:

```
javac 11.0.11
```

ábra 6.9 JDK verzió

Szükséges a gépnek kézzel beállított IP címet osztani. Ez az én esetemben 192.168.2.95 lett, azért szükséges kézzel beállított IP cím, más néven statikus IP cím, mivel egy szerver funkciót fogunk használni a későbbiekben.

6.1.2. Rendszer egyéb követelményei

Szükségünk van egy service felhasználóra, amit nevezzünk el “kafka”-nak, mivel a Kafka tudja kezelni a hálózati kéréseket. Ezzel minimálisra csökkenthetjük az operációs rendszer esetleges sérüléseket. Ezt a felhasználót később akár törölhetjük, de ezt nem javasolt, ha szeretnénk módosítania a beállításokon azt későbbiekben bonyolultabban tudjuk megtenni. Ezt az következő terminál paranccsal lehet létrehozni

```
sudo useradd kafka -m
```

ábra 6.10 kafka felhasználó létrehozás

Az -m kapcsoló biztosítja, hogy service felhasználónknak legyen dedikált saját könyvtára. Ez a következő lesz /home/kafka. Ez a kafka könyvtár későbbiekben munkakönyvtárra fog számunkra szolgálni.
Állítsunk be az fenti service felhasználónak jelszót. Ezt az alábbi paranccsal lehet végrehajtani:

sudo passwd <egyedileg kitalált jelszó>

ábra 6.11Jelszó létrehozás

Szükséges lesz ezt a felhasználót a sudo csoporthoz hozzáadni, hogy egyéb függőségek telepítéséhez rendelkeznie kell kellő mennyiségű jogosultsággal. Ezt a következő paranccsal tudjuk végrehajtani:

sudo adduser kafka sudo

ábra 6.12 kafka felhasználóhoz jogosultság osztás

Kafka service felhasználó elkészült, ezt követően az alábbi paranccsal lépjünk be a fiókba.

su -l kafka

ábra 6.13 kafka felhasználóval belépés

Parancssoros felület fogja ezt követően kérni a jelszót, abban az esetben, ha állítottunk be neki jelszót.

6.2. Kafka telepítése [7]

Töltsük le és csomagoljuk ki a Kafka binárist a kafka felhasználói fiók alatt egy számunkra megfelelő dedikált könyvtárba.

curl "https://www.apache.org/dist/kafka/3.0.0/kafka_2.13-3.0.0.tgz" -o ~/Downloads/kafka.tgz

ábra 6.14 Kafka csomag letöltése

Ezt követően csomagoljuk ki a .tgz állományunkat.

tar -xvzf ~/Downloads/kafka.tgz --strip 1

ábra 6.15 Kafka csomag kicsomagolás

A –strip1 kapcsoló által a letöltés mappába lesz majd kicsomagolva .tgz fájl tartalma, és nem egyéb könyvtárva mint például kafka_2.13_3.0.0.
Kafka telepítése esetén nem beszélhetünk hagyományos telepítésről. Itt egy szolgáltatást kell létrehoznunk ebben az esetben. Kafka alapvető viselkedése nem engedi, hogy töröljünk témát data pipeline-ból, így ezt először engedélyezni kell. A ./kafka/config/server.properties-t kell megnyitni egy terminál alapú szövegszerkesztővel ez lehet nano vagy vim. Következő beállítást adjuk hozzá a fájlhoz, és ez után mentsük el.

```
delete.topic.enable = true
```

ábra 6.16 topic törlésének engedélyezése

Ezek követően, hogy beállítottuk a kafka server.properties-ben, hogy engedélyezve legyen a téma törlés a data pipeline-ban, így tovább mehetünk a rendszerfájlokhoz, és azoknak a futtatásához, és az engedélyezéséhez a rendszerindításnál.

6.3. Kafka beállítása [7]

Ebben a fejezetben a rendszer egység fájlokat hozzunk létre, és állítjuk be Kafka szolgáltatás számára. Ez segít számunk olyan alapvető szolgáltatási műveletek végrehajtásában, mint például Kafka leállítása, elindítása vagy esetleg újraindítására, és az összes többi Linux szolgáltatással összhangban.
Ehhez szükségünk van Zookeeper-re, ami olyan szolgáltatást nyújt, ami a Kafka fürt (cluster) állapotának beállításainak kezelésére nyújt segítséget. Ezt legtöbbször elosztott rendszerekben használják egy integrált komponensként.

6.3.1. Zookeeper és annak beállítása [8]

Zookeeper egy nyílt forráskódú nagyteljesítményű központosított koordinációs rendszer egy elosztott alkalmazásokhoz. Elterjedt szolgáltatásokat, mint például az elnevezéseket, a konfigurációkezelést, az adatok szinkronizációját és a csoportszolgáltatásokat, egy egyszerű felületen teszi, így egy ilyet nem kell saját kezűleg elkészíteni. Úgy tervezték, hogy ez könnyen programozható legyen, és olyan adatmodellt használ, amely fájlrendszer szinten jól ismert könyvtárfa szerkezetet használ. JAVA nyelven fut és C kötésekkel lehet hozzá kapcsolódni. Koordinációs szolgáltatásokat nehéz karbantartani. Különösen hajlamosak “holtpontra” és “versenyfüggőségre”. Zookeeper lehetővé teszi, hogy az elosztott folyamatok koordinálják egymást egy megosztott hierarchikus névtéren keresztül, így nem alakul ki “versenyfüggőség” és “holtpont”, ami a hagyományos fájlrendszerhez hasonlóan szerveződik. Zookeeper szóhasználatban znode-oknak szokták nevezni. Zookeeper adatai memóriában vannak letárolva, ami azt eredményezi. Kis késleltetést eredményez, és nagy adatátviteli sebességet lehet vele elérni.

Ehhez annyi feladatunk van, hogy készítsünk egy szolgáltatást hozzá.
Hozzunk létre egy service fájlt a Zookeeper számára, ezt megtehetjük nano-val és vim-mel egyaránt.

```
sudo nano /etc/systemd/system/zookeeper.service
```

ábra 6.17Zookeeper szolgáltatás létrehozása

Illesszük be a következő kód részt:

```
[Unit]
Requires=network.target remote-fs.target
After=network.target remote-fs.target

[Service]
Type=simple
User=kafka
ExecStart=/home/kafka/kafka/bin/zookeeper-server-start.sh
/home/kafka/kafka/config/zookeeper.properties
ExecStop=/home/kafka/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

ábra 6.18 Zookeeper szolgáltatás kód

A[Unit] szakasz határozza meg, hogy a Zookeeper-nek hálózaton kell, hogy legyen csatlakoztatva, és a fájlrendszernek készen kell állnia, mielőtt elindulna.
A [Service] szakasz határozza meg, hogy systemd a zookeeper.server-start.sh zookeeper.server-stop.sh shell fájlokat kelljen használnia a szolgáltatás elindításához és a leállításához. Azt is meghatározza, hogy Zookeeper nem rendeltetésszerű leállása esetén automatikusan újrainduljon.
Következő lépésben készítsük el a kafka service-t, ezt megtehetjük nano-val és vim-mel is mint az előző lépésben.

```
sudo nano /etc/systemd/system/kafka.service
```

ábra 6.19 Kafka szolgáltatás létrehozása

Illesszük be a következő kód részt:

```
[Unit]
Requires=zookeeper.service
After=zookeeper.service

[Service]
Type=simple
User=kafka
ExecStart=/bin/sh -c '/home/kafka/kafka/bin/kafka-server-start.sh
/home/kafka/kafka/config/server.properties > /home/kafka/kafka/kafka.log 2>&1'
ExecStop=/home/kafka/kafka/bin/kafka-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

ábra 6.20Kafka szolgáltatás kód

A [Unit] szakasz határozza meg, hogy ez az egységfájl a zookeeper.service-től függjön. Ez biztosítja, hogy a zookeeper automatikusan induljon el a kafka szolgáltatással együtt.
A [service] szakasz határozza meg, hogy a systemd-nek a kafka-server-start.sh és kafka-server-stop.sh shell fájlokat kelljen használnia a szolgáltatás inditás és leállítása pillanatában. Azt is meghatározza, hogy Kafka-t automatikusan újraindítsa, ha nem rendeltetésszerűen lép ki.
Most, hogy a szolgáltatások meghatározása megtörtént, így van rá lehetőségünk elindítani a Kafka szolgáltatást a következő shell paranccsal:

```
sudo systemctl start kafka.service
```

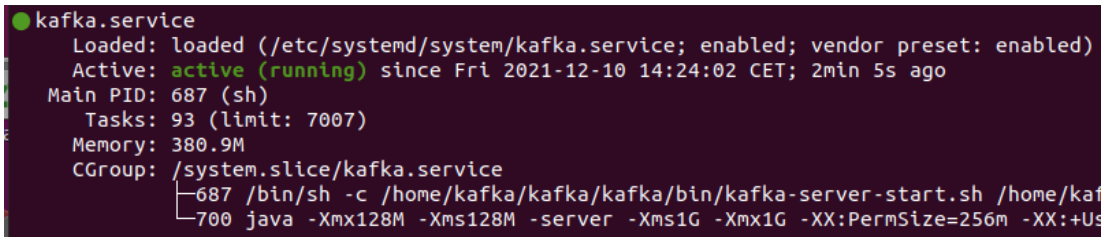
ábra 6.21Kafka szolgáltatás indítása

Sikeres Kafka szolgáltatás elindítása után győződjünk arról, hogy sikeresen elindult a Kafka szolgáltatás. Ezt a következő paranccsal lehet megtenni:

```
systemctl status kafka.service
```

ábra 6.22Kafka szolgáltatás állapot lekérése

Ha a következőt látjuk és az “active:” résznél látjuk, hogy “active (running)” ebben az esetben sikeresen elindult a szolgáltatásunk.



ábra 6.23 Kafka szolgáltatás állapota

Most már hogy sikeresen fut a Zookeeper és a Kafka szolgáltatás,így a Kafka 9092-s porton hallgatózik. Az operációs rendszer indításakor elinduljon a Kafka szolgáltatás ahhoz szükséges, hogy a következő parancsot kiadjuk:

```
sudo systemctl enable kafka.service
```

ábra 6.24 Kafka szolgáltatás engedélyezése

Jelenleg elkészültünk a Zookeeper és Kafka telepítésével. Most már meg tudjuk nézni, hogy működik-e a szolgáltatásaink.

6.3.2. Telepítés tesztelése

Bizonyosodjunk meg arról, hogy a kafka szerverünk megfelelően működik. Ehhez publikálni kell egy “Hello World” üzenetet egy témába.
Producer, ami lehetővé teszi a rekordok, adatok témakörökhöz való publikálást.
A Consumer más néven a fogyasztó, ami beolvassa az adott témában az üzeneteket vagy az adatokat.
Először hozzunk létre egy témát más néven egy “topic”-ot ezt a következő terminál paranccsal tudjuk végrehajtani:

```
./bin/kafka-topics.sh --create --topic test-topic --bootstrap-server 192.168.2.95:9092 --replication-factor 1 --partitions 1
```

Abban az esetben, ha a visszakapott terminál üzenet:

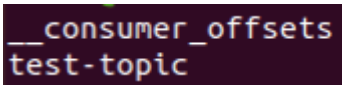
```
Created topic test-topic
```

ábra 6.25 topic létrehozás

biztosak lehetünk benne, hogy létrejött az adott című téma. Ezt meg tudjuk még nézni másik paranccsal is.
A következő paranccsal ki lehet listázni az adott szerveren lévő összes topic-okat.

```
./bin/kafka-topics.sh --list --bootstrap-server 192.168.2.95:9092
```

ábra 6.26 topic-ok listázása



ábra 6.27broker-nek topic-jai

Ezt követően, hogy elkészült adott című topic-unk, így ezt követően tudunk betölteni egyéb adatot, avagy adatokat:

```
echo “Hello World” | ./bin/kafka-console-producer.sh --broker-list 192.168.2.95:9092 --topic test-topic > /dev/null
```

ábra 6.28 topic-ba üzenet betöltés

Ahogy betöltöttük az adatot, avagy az adatokat a kiválasztott témába, ezt követően ki is tudjuk listázni a tartalmát, és megnézni, hogy topic-ban milyen adatok szerepelnek. A következő terminál paranccsal ki lehet listázni a test-topic-ban szereplő összes tartalmat az elejétől kezdve:

```
./bin/kafka-console-consumer.sh --bootstrap-server 192.168.2.95:9092 --topic test-topic --from-beginning
```

ábra 6.29 topic üzenetek listázása

A szkript várni fogja a további beérkező üzenetekre. Ebben az esetben nyissunk egy új terminál ablakot, és küldjünk be több üzenetet. Ezek az üzenetek sorfolytonosan meg fognak érkezni a másik terminál ablakba anélkül, hogy azt manuálisan kéne majd frissíteni. Mikor készen vagyunk a tesztelésével, hogy a szkript futása megálljon nyomjunk CTRL+C-t. Most már teszteltük, hogy sikeresen működik-e a kafka számunkra.

6.4. Python [9]

Lényeges, hogy ne keverjük össze a Python 2-t és a Python 3-t ebben az esetben, mert szintaxisban van különbség a 2 verzióban. Én Python 3-t használtam, későbbiekben csak Python-ként fogok rá hivatkozni. Python 3 könnyebben olvasható. Python 2 és Python 3 között drámaian nőtt a számítási sebesség. Python 3 javítja a Python 2 amúgy is hatalmas képességeit. Python 3 tartalmaz néhány matematikai számítási fejlesztést.

6.4.1. Python telepítés és kafka-python modul telepítés [10]

Ha 16.10-os Ubuntu-t vagy annál frissebb verziónál használunk kettő terminál parancsot szükséges kiadni, hogy feltelepítsük a Python 3-t. Először frissítsük meg a csomagok indexelését.

```
sudo apt update
```

ábra 6.30 csomagok újra indexelése

Ezt követően fel tudjuk telepíteni a Python 3-t a következő terminál parancs segítségével:

```
sudo apt-get install python3.8
```

ábra 6.31Python 3 csomag telepítése

Amennyiben sikeresen tudtuk telepíteni a python-t az operációs rendszerr, kezdjük el összekötni a python kódot és a Kafka-t. Ehhez telepítsük fel a python és kafka csomagot, így a már nem terminálból kell, hogy adatot töltsünk topic-okba, vagy terminálból hozzunk létre újabb és újabb topic-ot. Ismét lényeges, hogy pip vagy pip3-

t használunk majd csomag telepítés során mert pip-pel nem fog működni a későbbiekben, mivel Python 3-t telepítettük fel, így pip3-t kell, hogy használjunk.

```
pip3 install kafka-python
```

ábra 6.32kafka-paython csomagtelepítése python-hoz

Sikeres csomag telepítést követően hozzá állhatunk python kód írásának, mivel, hogy adatot tudjunk betölteni topic-ba minden rendelkezésre áll.

6.5. Adatok betöltése témákba

6.5.1. Forrásrendszer

Forrásrendszer lehet bármilyen forrásrendszer, ez lehet valamilyen ERP rendszer, CRM, KSH statisztikai adatok, árfolyam változással kapcsolatos adatok, vagy akár időjárás, tőzsdei adatok cégeknek. Én tőzsdei adatokat választottam, és 2 céget, egyik az Uber másik a Google.

6.5.2. Forrásrendszerek letöltése pythonnal

Be tudjuk szerezni a forrás állományt, és azokat képesek legyünk betölteni az adott topic-ba, ahhoz szükséges lesz egy API-ra, ahhoz, hogy rendelkezésre álljon számunkra bármiféle tőzsdei adat, ahhoz szükséges lesz újabb Python csomagokat telepíteni. Elengedhetetlen számunkra a numpy, pandas, yfinance, plot. Utolsó nem kötelező, de ha szeretnénk vele rajzolni akkor elengedhetetlen számunkra. Mi esetünkben nem grafikont létrehozni. Telepítés a következő 4 paranccsal tudjuk majd végre hajtani:

```
pip3 install pandas
pip3 install numpy
pip3 install yfinance
pip3 install plot
```

ábra 6.33 tőzsdei adatok lekérdezéséhez szükséges csomagok

Sikeres telepítést követően nano vagy vim segítségével nyissunk egy terminál alapú szöveg szerkesztőt, és ezeket az csomagokat importáljuk be a .py fájlba, amit később majd a python futtatása közben importálja.

```
import numpy as np
import pandas as pd
import yfinance as yf
```

ábra 6.34tőzsdei adatok lekérdezéséhez szükséges csomagok importálása

A 3 fenti csomagot importálás után, neki elkezdhethetjük letölteni a kedvünk szerint kiválasztott cégnek a tőzsdei adatait.

```
data = yf.download(tickers='Uber', period='60d', interval='5m').to_string()
```

ábra 6.35 Uber-nek a tőzsdei adatainak lekérdezése

Importálást követően a felső Python sorral van rá lehetőségünk eltölteni az adott cégnek a tőzsdei adatait. Tickers-be van rá lehetőségünk, hogy az adott cégnek a tőzsdén szereplő nevét adjuk meg. Perod-ban van rá lehetőség, hogy mennyi időre visszamenőleg kérjük le a tőzsdei adatait a cégnek. Intervalban lehetőségünk van meghatározni, hogy milyen bontásban kérjük le ezeket az nyers adatokat. Ezt későbbiek miatt szükséges szöveg típusú formázni, mivel jelenleg dataframe

formátumban szerepel, és ezzel későbbiekben nem fogunk tudni dolgozni.Kafka topic-ba csak bájt típusú változót van lehetőségünk betölteni.

6.5.3. Forrásrendszer tartalma betöltése topic-ba Pythonnal

Most már rendelkezzünk egy API-val, amivel le tudjuk tölteni az internetről tőzsdei nyers adatokat. Ezt követően töltsük be a forrásrendszer nyers adat tartalmát számunkra megfelelő topic-ba. Fentebb telepítve lett a python és a kafka csomag pip3 segítségével, így most már ez is rendelkezésre áll számunkra. Producer segítségével tudjuk már rendelkezésre álló adatokat betölteni számunkra megfelelő topic-ba. Importáljuk a Kafka csomagokat a python kódba, hogy tudjuk használni a kafka-python által nyújtott csomag tartalmat.
Következő Python kóddal tudjuk importálni a Kafka-hoz szükséges csomagokat.

```
from kafka.admin import KafkaAdminClient, NewTopic
from kafka import KafkaProducer
from kafka import KafkaConsumer
```

ábra 6.36Kafka csomagok importálása

Most már rendelkezésre áll számunkra minden ahhoz. hogy Python kódból töltsünk nyers tőzsdei adatot topic-ba. Figyelni kell arra, hogy a topic csakis kizárólag bájt típusú adatot képes fogadni, ha bármiféle más típusú adatot szeretnénk küldeni, futás időbeli hibát fogunk kapni. Ezért dataframe típusú adatból előbb szöveg típusú más néven string típusú adattá kell átalakítsuk, ezt egy to_string() függvény meghívással tudjuk megoldani. Ezt követően, el kell végezzünk egy enkódolást, hogy bájt alapú legyen a letöltött adatok, amit későbbiekben sorok vége mentén feltördelünk, egy .split(b‘\n’) segítségével, hogy sorról-sorra menjenek be az adatok. Kis b-nek jelentős szerepe van ebben az split függvény bemenetében, mert ebben az esetben bájt típusban tördeljük fel a sorok mentén az értékeket.
Ezeket követően hozzunk létre számunkra egy megfelelő névvel egy topic-t Pythonból, és töltsük be oda a már bájttá formált adatainkat. Ahhoz, hogy be tudjuk tölteni az adatokat újabb csomagokat kell importáljunk a Python kódba, ez következő a KafkaAdminClient és a NewTopic csomag, amiket kafka.admin csomag részből tudunk megtenni.

```
from kafka.admin import KafkaAdminClient, NewTopic
```

ábra 6.37Új topic-hoz szükséges csomagok importálása

Létre tudjunk majd hozni egy új topic-t, ahhoz deklarálni kell egy topic listát vagy tömböt.

```
topic_list = []
```

ábra 6.38 új topic lista

Ezt követően fűzzük hozzá, ehhez a listához az új topic-t szükséges paraméterekkel együtt, nálam a partíciók száma 1, replikációs faktorok száma 1. Ezt az append függvény meghívásával tudjuk megoldani.

```
topic_list.append(NewTopic(name=topicName, num_partitions=1, replication_factor=1))
```

Ezzel még csak hozzáadtuk az új “topic_list“ változóhoz az új topic-t még a Kafka szerveren nem lett létrehozva. Erre szükségünk van egy admin_client változó-ra, amit fel kell, hogy paraméterezzünk, meg kell adjuk a bootstrap kiszolgáló IP címét, avagy a kiszolgáló szerver nevét, azzal a port számmal, amin a kafka kiszolgálónk figyeli a bejövő adatokat.
Nálam ezek a következők lettek bootstrap kiszolgáló szervernek az IP címe és portszáma az 192.168.2.95:9092.

```
bootstrap_servers=['192.168.2.95:9092']
admin_client = KafkaAdminClient(bootstrap_servers=bootstrap_servers)
```

ábra 6.39 broker-hez kapcsolódás

Ezeket követően tudjuk hozzáadni az topic-t a változóban meghatározott bootstrap kiszolgálókhoz.

```
admin_client.create_topics(new_topics=topic_list)
```

ábra 6.40 új topic lista importálása broker-be

Fent szerepelt “topic_list” gyűjtemény típusú változóban szereplő topic-okat tudjuk hozzáfűzni a bootstrap kiszolgálón szereplő egyéb topic-okhoz.
Elkészült Python segítségével az új topic-unk, letöltöttük az API segítségével a számunkra szükséges tőzsdei adatokat, így ezeket az adatok jelenleg be tudjuk töltsük abba a topic-ba, amit létrehoztunk. Jelenleg deklarálni kell egy KafkaProducer-t, amivel tudunk kapcsolódni az adott Kafka kiszolgálóra, és aminek a segítségével tudjuk feltölteni a topic-t, ehhez ismét szükségünk a bootstrap szerverek gyűjteményére.

```
producer = KafkaProducer(bootstrap_servers=bootstrap_servers)
```

ábra 6.41Producer-hez kapcsolat kiépítés

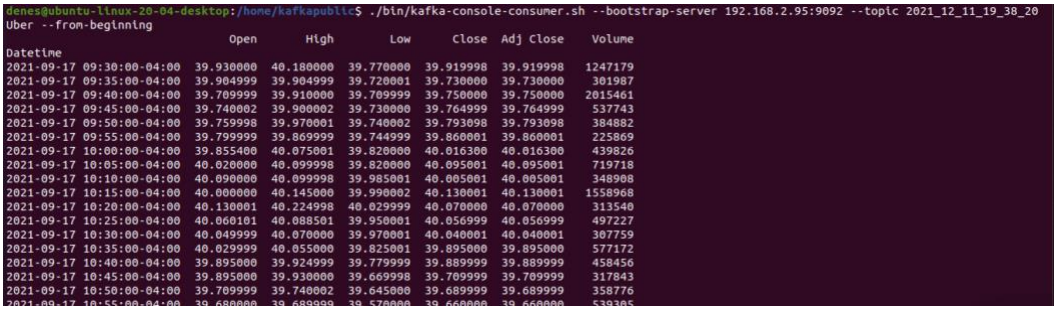
Az adatok bájt formátumban állnak számunkra rendelkezésre létre van hozva a topic a megadott névvel, kapcsolatot építettünk ki a kafka producer-hez, ezek után be tudjuk tölteni a nyers tőzsdei adatokat a számunkra kiválasztott cégről a létrehozott topicba. Ezt egy ciklus segítségével tudjuk megoldani.

```
for msg in data:
 producer.send(topicName, value=msg)
```

ábra 6.42 Producer-be és topic-ba adatok betöltése

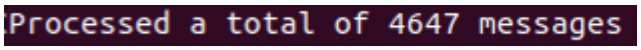
Producer-nek a send nevű függvényét kell meghívni, hogy be tudjuk küldeni a topic-ba az üzeneteket, “topicName” mint változó határozza meg hogy melyik topic-ba legyenek az üzenetek elküldve.

Jelenleg eljutottunk odáig, hogy API segítségével letöltöttük a számunkra releváns cégnek a tőzsdei adatait, létrehoztunk egy topic-t Python segítésével, és ezeket adatokat eljuttattuk egy topicba.



```
denes@ubuntu-linux-20-04-desktop:/home/kafkapublic$ ./bin/kafka-console-consumer.sh --bootstrap-server 192.168.2.95:9092 --topic 2021_12_11_19_38_20
Uber --from-beginning
                           Open       High        Low      Close  Adj Close    Volume
Datetime
2021-09-17 09:30:00-04:00  39.930000  40.180000  39.770000  39.919998  39.919998  1247179
2021-09-17 09:35:00-04:00  39.904999  39.904999  39.720001  39.730000  39.730000   301987
2021-09-17 09:40:00-04:00  39.709999  39.910000  39.709999  39.750000  39.750000  2015461
2021-09-17 09:45:00-04:00  39.740002  39.900002  39.730000  39.764999  39.764999   537743
2021-09-17 09:50:00-04:00  39.759998  39.970001  39.740002  39.793098  39.793098   384882
2021-09-17 09:55:00-04:00  39.799999  39.869999  39.744999  39.860001  39.860001   225869
2021-09-17 10:00:00-04:00  39.855400  40.075001  39.820000  40.016300  40.016300   439826
2021-09-17 10:05:00-04:00  40.020000  40.099998  39.820000  40.095001  40.095001   719718
2021-09-17 10:10:00-04:00  40.090000  40.099998  39.985001  40.005001  40.005001   348908
2021-09-17 10:15:00-04:00  40.000000  40.145000  39.990002  40.130001  40.130001  1558968
2021-09-17 10:20:00-04:00  40.130001  40.224998  40.029999  40.070000  40.070000   313540
2021-09-17 10:25:00-04:00  40.060101  40.088501  39.950001  40.056999  40.056999   497227
2021-09-17 10:30:00-04:00  40.049999  40.070000  39.970001  40.040001  40.040001   307759
2021-09-17 10:35:00-04:00  40.029999  40.055000  39.825001  39.895000  39.895000   577172
2021-09-17 10:40:00-04:00  39.895000  39.924999  39.779999  39.889999  39.889999   458456
2021-09-17 10:45:00-04:00  39.895000  39.930000  39.669998  39.709999  39.709999   317843
2021-09-17 10:50:00-04:00  39.709999  39.740002  39.645000  39.689999  39.689999   358776
2021-09-17 10:55:00-04:00  39.680000  39.689999  39.570000  39.660000  39.660000   539305
```

ábra 6.43 topic-ban lévő üzenetek



```
Processed a total of 4647 messages
```

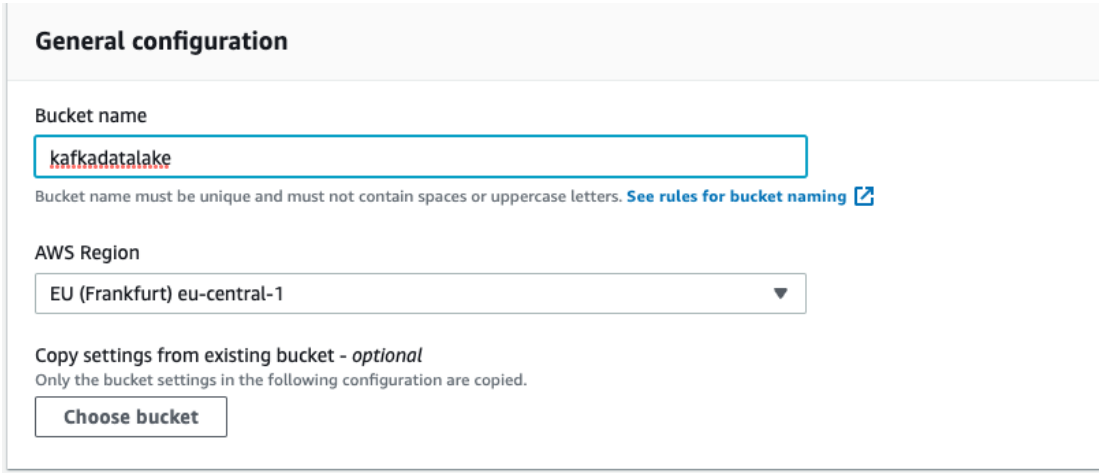
ábra 6.44üzenetek darabszáma

Ebben a topicban jelenleg 4647 üzenet van letárolva. Későbbiekben ezt a topicot, ha szeretnénk bővíteni természetesen van rá lehetőségünk.

6.6. Data lake

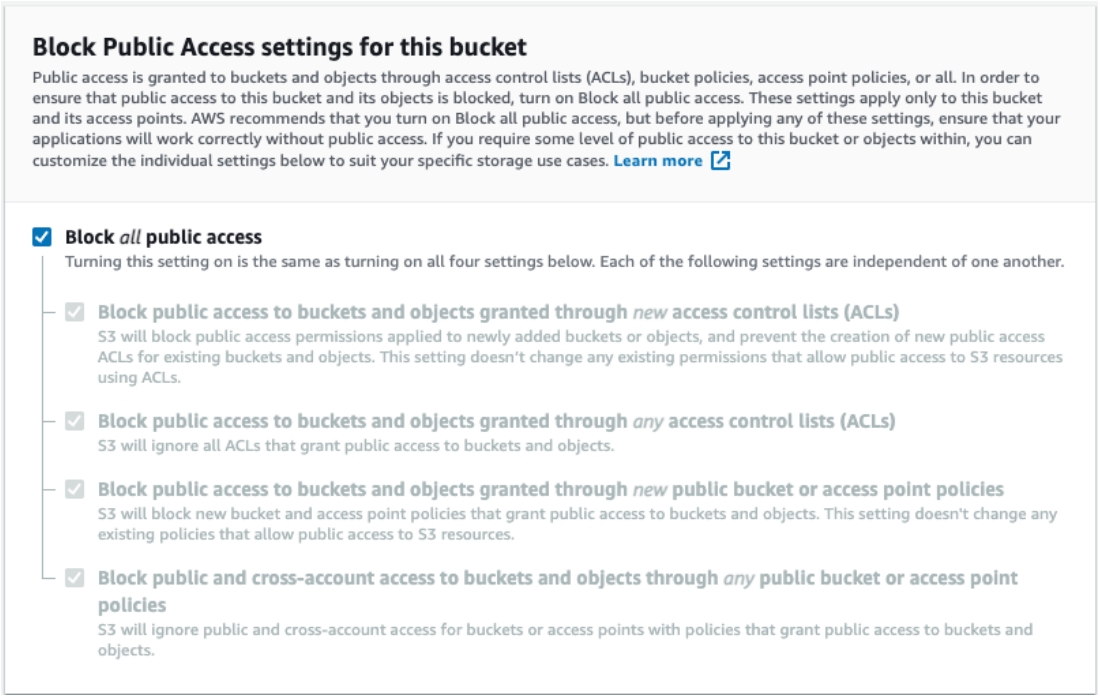
6.6.1. Data lake készítése

Én data lake-nek Amazon Web Service S3-t választottam, azért, mert ingyenes 5 gigabájt mértékig van lehetőség készíteni data lake-t. Amazon Web Service más néven AWS, ezt más néven hívja a data lake-t, ők ezt úgymond buckets-nek hívják.
Data lake nevét töltsük ki számunkra megfelelő névvel, én ebben az az esetben kafkadatalake nevet választottam. Érdemes az AWS Régiót a tartozkodási helyünkhöz a legközelebbí lehetőséget választani, mert adatok mozgása esetén költségek merülhetnek fel, abban az esetben, ha nem S3 licenszelésű data lake-t hozzunk létre és használnánk, és a sebesség is függ attól hogy a data lake hol helyezkedik el. Mert nem mindegy, ha Európában tartózkodunk, hogy esetleg Ameráka szeretnénk elhelyezni a data lake-t vagy esetleg itt Európában szeretnénk létrehozni.



ábra 6.45Data lake elnevezés

Tanácsos a “Block public access settings for this bucket” opciót aktiválni, mert későbbiekben mások is használhatják a data lake-ünket, ami nem számunkra nem biztos, hogy jó megoldás.

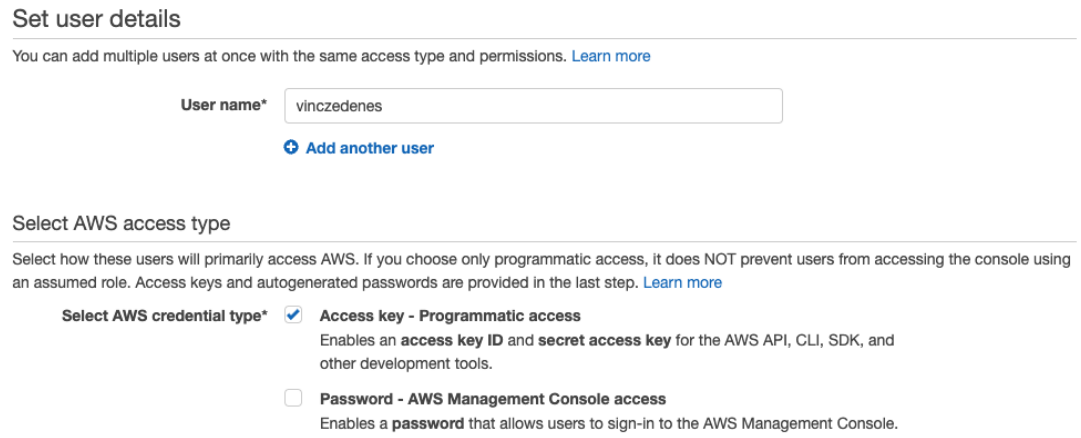


ábra 6.46 Publikus elérés letiltása

Összes többi beállítást hagyjuk AWS által beállított alapértelmezett beállításokon. Ezeket követően hozzuk létre a data lake-t az oldal alján elhelyezett “Create bucket” feliratú gombbal.

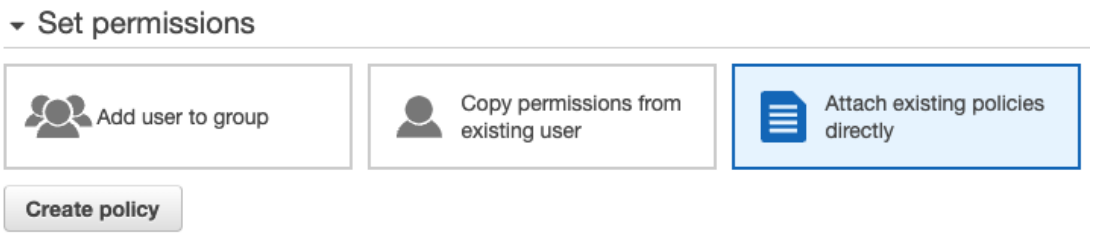
6.6.2. Data lake jogosultság osztás

A data lake felcsatlakoztatására szükségünk lesz plusz egy felhasználóra és egy policy-re magyar nevén egy házirendre, amiket AWS-n kell megcsinálnunk. Ezek után a keresőbe beírva IAM kulcs szót, ami az Identity and Access Management-nek felel meg.Hozzunk létre egy policy-t és egy felhasználót, amivel későbbiekben hozzáférést adunk majd a data lake használatára a létrehozott felhasználó számára. Töltsük ki számunkra megfelelő felhasználónévvel a User name mezőt, és pipáljuk be “access key” opciót, mert későbbiekben kapunk hozzáférést egy access key és egy secure access key-nek köszönhetően oldjuk meg, amiket későbbiekben használni fogunk.



ábra 6.47Data lake-hez felhasználó elnevezés

Amennyiben megvagyunk a felhasználónév beírásával és az access key bepipálásával ezek után haladhatunk tovább.
Válasszuk ki “Attach policies directly“ opciót, és ennek által hozzunk létre egy új policy-t, mert a data lake-ünkhöz specifikus policy-t nem fogunk találni a listában. “Create policy” feliratú gombbal létrehozhatjuk az új policy-nkat.



ábra 6.48 Jogosultság hozzácsatolás felhasználóhoz

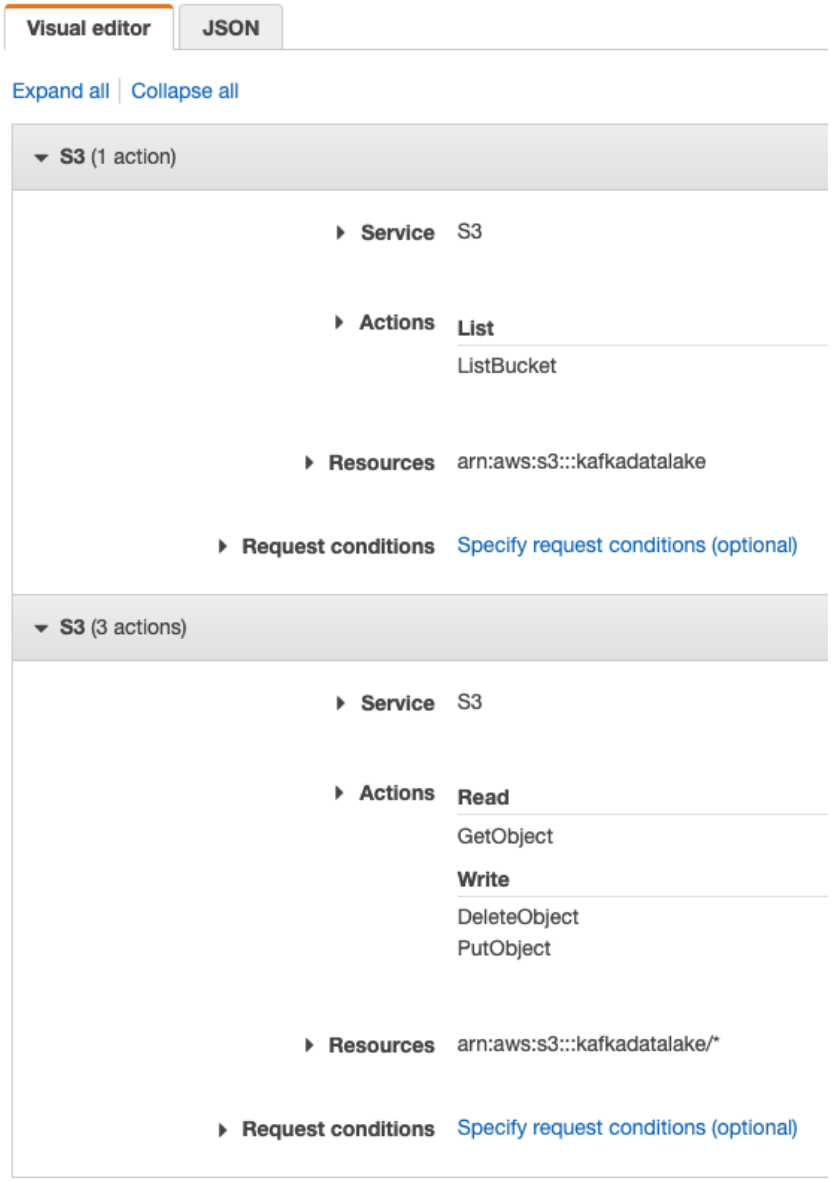
Ezeket követően válasszuk ki a JSON fület. és lenti JSON kódot illesszük be. Ügyeljünk arra, hogy a resource változónál data lake-ünk ARN kódját kell beilleszteni.

Visual editor | JSON

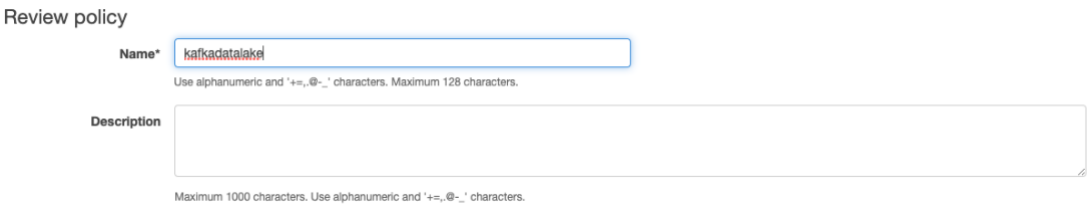
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::kafkadatalake"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": "arn:aws:s3:::kafkadatalake/*"
        }
    ]
}
```

ábra 6.49 Jogosultság implentálás

Ahogy a JSON kódot beillesztettük azt követően meg változni a Visual editor felülete a következő kinézetre, hogy milyen policy-t állítottunk be.

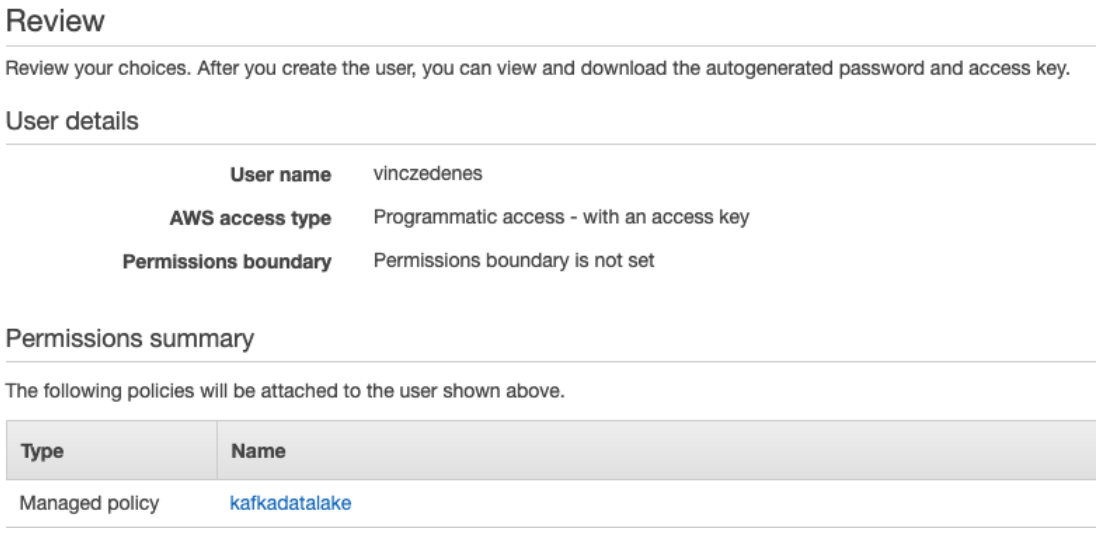


ábra 6.50jogosultság leírás



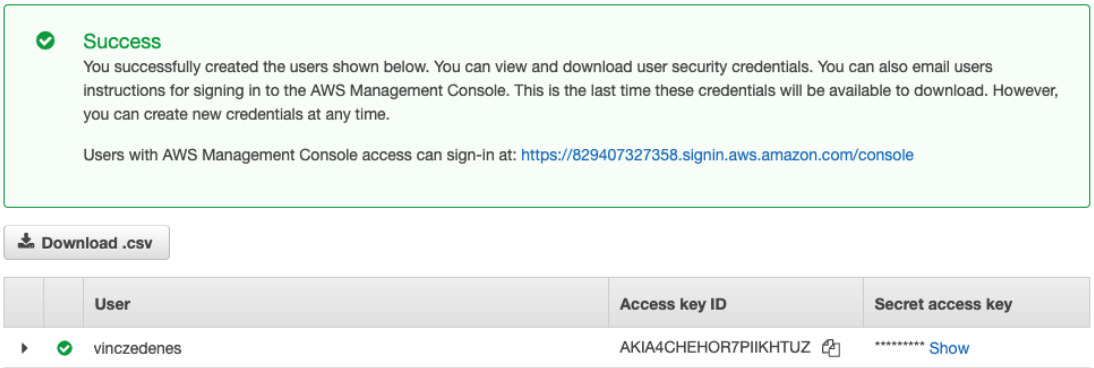
ábra 6.51Házirend áttekintés

Policy elnevezése után hozzuk létre a policy-t. Policy létrehozása után létre tudjuk hozni a felhasználót is. Felhasználó létrehozása előtt a következő oldalt kell, hogy lássuk az adott felhasználónak a leírásáról.



ábra 6.52 felhasználó áttekintés

Ha idáig eljutottunk, akkor felhasználót el tudjuk készíteni.



ábra 6.53 Sikeres felhasználó létrehozás

Ha zöld pipával “Success” feliratot látjuk, ebben az esetben sikeresen létrehoztuk a data lake-hez a felhasználót. Kaptunk hozzá egy Access key ID-t és egy Secret Access key-t. a Linux-n a csatlakozáshoz fel kell, hogy használjunk. Én nekem ebben az esetben már van egy létrehozott felhasználóm, én későbbiekben azt fogom majd használni, de ezzel ugyanúgy fog működni.

6.6.3. Data lake-re csatlakozás Linuxon

Ahhoz, hogy rendezettséget tudjuk tartani a gépen érdemes egy külön erre dedikált mappát létrehozni, ami alá fel tudjuk csatlakoztatni a data lake-t. Azt követően, hogy mappát létrehoztuk a gépen a data lake-hez, ezek után telepítsük fel a s3fs csomagot a gépre. Ezt követően a következő csomagokat kell feltelepíteni, hogy tudjuk csatolni a data lake-t.

```
sudo apt-get install automake autotools -dev fuse g++ git libcurl4-gnuts-dev
libfuse-dev libssl-dev libxml2-dev make pkg-config
```

ábra 6.54 Data lake-hez szükséges csomagok telepítése

Töltsük le a következő következő kódot a git clone segítségével,

```
git clone https://github.com/s3fs-fuse/s3fs-fuse.git
```

ábra 6.55 s3fs csomag git-ről letöltése

ahogy sikerült letöltenünk ezt a kódot, lépjünk be abba mappába hova letöltötöd, és futtassuk autogen.sh shell fájl futtatása után, ha ezt eredményt kaptuk akkor sikeresen lefutott.

```
--- Make commit hash file -------
--- Finished commit hash file ---
--- Start autotools -------------
--- Finished autotools ----------
```

ábra 6.56autogen.sh futtatási eredménye

Ezt követően futtassuk le a ./configure parancsot, ez beállítja számunkra a rendszerfájlokat, és létrehozza Makefile-t, hogy tudjunk make parancs segítségével telepíteni.

```
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating test/Makefile
config.status: creating doc/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

ábra 6.57configure.sh futtatási eredménye

Ezeket követően hívjuk meg a make parancsot, aminek a végén alábbi soroknak kell szerepelniük.

```
make[1]: Entering directory '/home/denes/s3fs-fuse'
Making all in src
make[2]: Entering directory '/home/denes/s3fs-fuse/src'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/denes/s3fs-fuse/src'
Making all in test
make[2]: Entering directory '/home/denes/s3fs-fuse/test'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/denes/s3fs-fuse/test'
Making all in doc
make[2]: Entering directory '/home/denes/s3fs-fuse/doc'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/denes/s3fs-fuse/doc'
make[2]: Entering directory '/home/denes/s3fs-fuse'
make[2]: Leaving directory '/home/denes/s3fs-fuse'
make[1]: Leaving directory '/home/denes/s3fs-fuse'
```

ábra 6.58 make parancs futtatásának eredménye

Ha le tudjuk kérdezni az s3fs-nek a verziószámát, abban az esetben sikeresen tudjuk telepíteni.

```
Amazon Simple Storage Service File System V1.90 (commit:5de92e9) with OpenSSL
Copyright (C) 2010 Randy Rizun <rrizun@gmail.com>
License GPL2: GNU GPL version 2 <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

ábra 6.59 s3fs verzió száma

Azokat követően, hogy sikeresen létrehoztuk a data lake-t és feltelepítettük a linux alapú operációs rendszerre s3fs-t, ezek után létrehozhatjuk a hitelesítési adatokat a data lake-hez és ezt a /etc/ könyvtárba fogjuk letárolni passwd-s3fs fájlban, ezt a fájlt nano vagy vim segítségével létre tudjuk hozni terminálból.

```
sudo /etc/passwd-s3fs
```

ábra 6.60 Data lake-hez hitelesítési adat létrehozása

Szükségünk van az access key-re és a secret access key-re, amit kettősponttal válasszuk el egymástól a fájlban.

```
AKIA4CHEHOR7L76BRXWZ:6xyL81mtwKD37BBAEg1pH20E49YfpTjw0x4g3wy4
```

ábra 6.61felhasználó hitelesítési adata

Ahogy elmentettük a hitelesítési adatokat a passwd-s3fs fájlba, ezt követően állítsunk rajta jogosultságot, tulajdonosának állítunk írás-olvasási jogosultságot csoportnak csak olvasási jogosultságot. Ezt a következő paranccsal tudjuk végre hajtani:

```
sudo chmod 640 /etc/passwd-s3fs
```

ábra 6.62hitelesítési fájlhoz jogosultság módosítás

Hitelesítési adatok megadását követően hozzunk létre számunkra olyan helyen egy szülő típusú mappát, ahova a data lake-t tudjuk csatlakoztatni. Könyvtár létrehozást az

```
mkdir -p <könyvtár neve>
```

ábra 6.63 szülő könyvtár létrehozás

paranccsal lehet létrehozni.
Csatlakoztatást a következő terminál paranccsal van lehetőségünk végrehajtani.

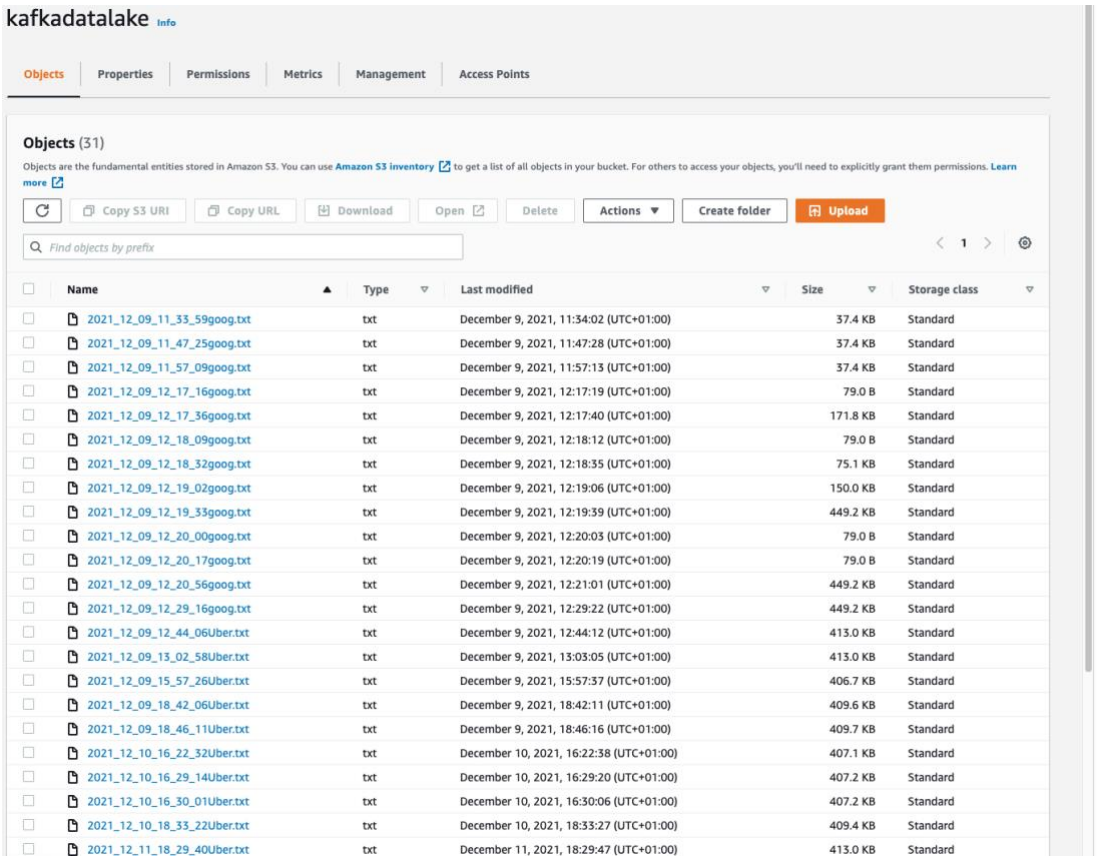
```
sudo s3fs -o allow_other kafkadatalake /home/kafkapublic/datalake/
```

ábra 6.64 Data lake felcsatolás mappa alá

Ha sikeresen megtörtént a felcsatolása a data lake-nek akkor, ahova csatlakoztattuk data lake-t ott üres mappának kell lennie, de ezt lehet azzal ellenőrizni, hogyha másolunk oda a gépről valamit és az fent AWS data lake-n megjelenik. Én esetemben már vannak itt lementett adatok.

```
denes@ubuntu-linux-20-04-desktop:~$ ll /home/kafkapublic/datalake/
total 9267
drwxrwxrwx  1 root  root       0 Jan  1  1970 ./
drwxrwxr-x 11 kafka kafka   4096 Dec  9 11:22 ../
-rw-rw-r--  1 denes denes  38313 Dec  9 11:34 2021_12_09_11_33_59goog.txt
-rw-rw-r--  1 denes denes  38313 Dec  9 11:47 2021_12_09_11_47_25goog.txt
-rw-rw-r--  1 denes denes  38313 Dec  9 11:57 2021_12_09_11_57_09goog.txt
-rw-rw-r--  1 denes denes     79 Dec  9 12:17 2021_12_09_12_17_16goog.txt
-rw-rw-r--  1 denes denes 175900 Dec  9 12:17 2021_12_09_12_17_36goog.txt
-rw-rw-r--  1 denes denes     79 Dec  9 12:18 2021_12_09_12_18_09goog.txt
-rw-rw-r--  1 denes denes  76923 Dec  9 12:18 2021_12_09_12_18_32goog.txt
-rw-rw-r--  1 denes denes 153549 Dec  9 12:19 2021_12_09_12_19_02goog.txt
-rw-rw-r--  1 denes denes 459954 Dec  9 12:19 2021_12_09_12_19_33goog.txt
-rw-rw-r--  1 denes denes     79 Dec  9 12:20 2021_12_09_12_20_00goog.txt
-rw-rw-r--  1 denes denes     79 Dec  9 12:20 2021_12_09_12_20_17goog.txt
-rw-rw-r--  1 denes denes 459954 Dec  9 12:21 2021_12_09_12_20_56goog.txt
-rw-rw-r--  1 denes denes 459954 Dec  9 12:29 2021_12_09_12_29_16goog.txt
-rw-rw-r--  1 denes denes 422877 Dec  9 12:44 2021_12_09_12_44_06Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec  9 13:03 2021_12_09_13_02_58Uber.txt
-rw-rw-r--  1 denes denes 416416 Dec  9 15:57 2021_12_09_15_57_26Uber.txt
-rw-rw-r--  1 denes denes 419419 Dec  9 18:42 2021_12_09_18_42_06Uber.txt
-rw-rw-r--  1 denes denes 419510 Dec  9 18:46 2021_12_09_18_46_11Uber.txt
-rw-rw-r--  1 denes denes 416871 Dec 10 16:22 2021_12_10_16_22_32Uber.txt
-rw-rw-r--  1 denes denes 416962 Dec 10 16:29 2021_12_10_16_29_14Uber.txt
-rw-rw-r--  1 denes denes 416962 Dec 10 16:30 2021_12_10_16_30_01Uber.txt
-rw-rw-r--  1 denes denes 419237 Dec 10 18:33 2021_12_10_18_33_22Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 18:29 2021_12_11_18_29_40Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 18:45 2021_12_11_18_45_29Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 18:56 2021_12_11_18_55_55Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 18:56 2021_12_11_18_56_23Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 18:56 2021_12_11_18_56_40Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 19:32 2021_12_11_19_32_50Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 19:37 2021_12_11_19_37_25Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 19:37 2021_12_11_19_37_50Uber.txt
-rw-rw-r--  1 denes denes 422877 Dec 11 19:38 2021_12_11_19_38_20Uber.txt
```

ábra 6.65Data lake-ben szerepló fájlok

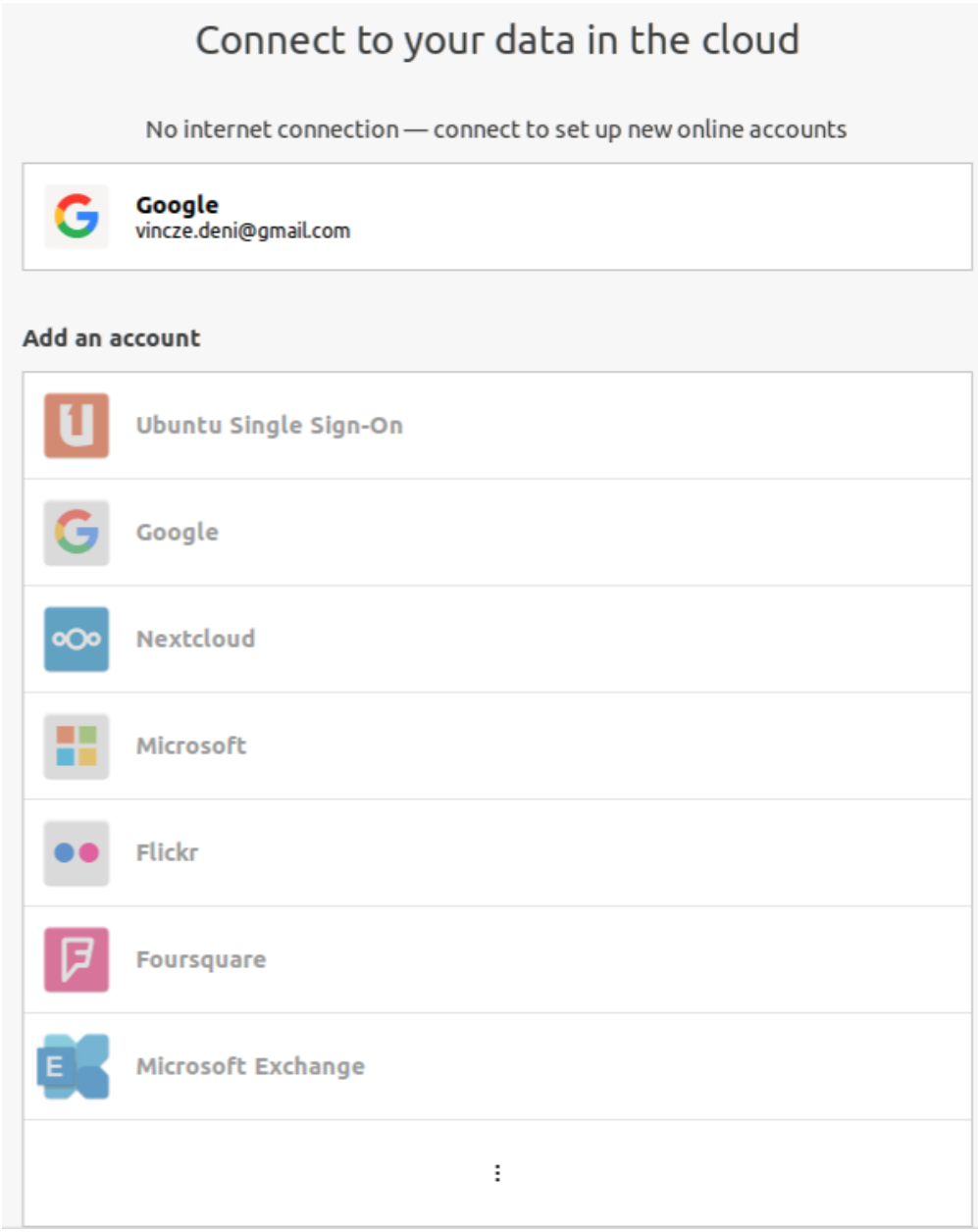


ábra 6.66AWS S3 data lake-ben szereplő fájlok

Ha mind a 2 helyen az adatok megegyeznek akkor sikeresnek tekinthető a data lake felcsatolása a megadott könyvtár alá.

6.7. Adatok betöltése központi rendszerekbe

Ha csak is kizárólag adattárolás szempontjából nézzük a data lake-t és a Google Drive-t, akkor ebben az esetben nincsen különbség a kettő között. Ahhoz, hogy Google Drive-ra tegyük fel topic-ban szereplő adatokat, ahhoz szükségünk van egy Google fiókra. Google fiókkal szükséges belépni a Lunixon, ha operációs rendszer rendelkezik grafikus felhasználó felülettel. Ezt a beállítások alatt az online fiókok alatt lehet megtalálni.



ábra 6.67 Google fiók csatlakoztatása

Ha inaktív a kezelőfelület, ebben az esetben hálózati beállítási gondok állnak fenn, amit az alábbi shell kóddal, lehet kiküszöbölni. Ezt a shell kódot a következő fájlba kell beilleszteni /etc/netplan/01-network-manager-all.yaml, ezt tudjuk tenni bármilyen terminálos

szövegszerkesztő segítségével, de hogy ezt el tudjuk végezni root jogosultságra van szükségünk mivel rendszer fájlt hozunk létre és módosítunk.

```
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
```

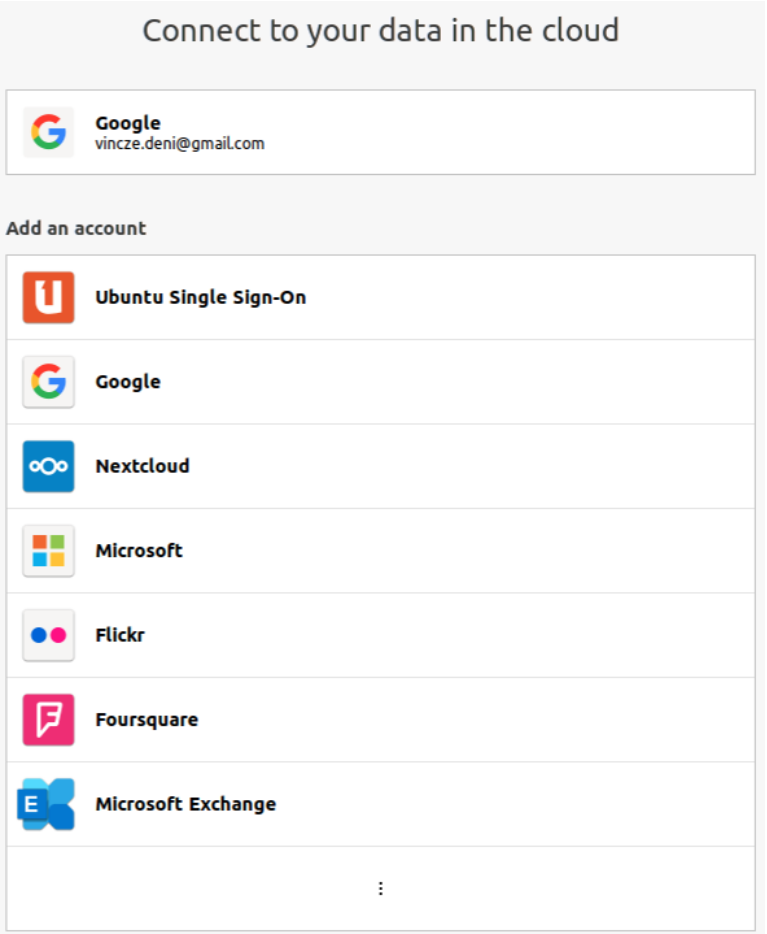
ábra 6.68 Hálózati kártya módosítás

Ezt követően ezeket változásokat aktiválunk kell, hogy életbe lépjenek a gépünkön, alábbi paranccsal tudjuk aktiválni:

```
sudo netplan apply
```

ábra 6.69 Hálózati kártya módosításának engedélyezése

aktiválást követően, aktív lesz az online fiók felvétele, így be tudunk lépni a Google fiókunkat csatolni tudjuk az operációs rendszerhez.



ábra 6.70 Google fiók csatlakoztatása

Miután sikeresen beléptunk a Google fiókunkkal, azt követően ezt vissza lehet tiltani, mivel ilyenkor automatikusan kap egy IP címet a gépünk, és ilyen esetben a Python kódban és a Kafka beállításainál kellene átírjuk az összes IP címet. Ezt követően hozzunk létre egy könyvtárat, ahova a Google Drive tartalmát felcsatoljuk, és akkor a

gépen lévő összes többi fájllal nem keverik a Google Drive-n tárolt fájljaink. Most csatoljuk fel a Google Drive-t, ehhez terminálból telepítenünk kell google-drive-ocamlfuse csomagot, amivel felcsatoljuk a Google Drive-unk tartalmát, mint egy plusz meghajtó. Ezt a csomag telepítést a következő terminál paranccsal tudjuk megtenni.

```
sudo apt update && sudo apt install google-drive-ocamlfuse
```

ábra 6.71Csomagok újraindexelése és Google Drive csomag telepítése

Hogyha sikeresen fel tudjuk tenni google-drive-ocamlfuse csomagot, ezek után tudjuk felcsatolni a Google Drive arra a mappára, amit kiválasztunk. Ezt a következő terminál paranccsal lehet megtenni.

```
mkdir <számunkra megfelelő könyvtár>
```

ábra 6.72 könyvtár létrehozás

Sikeres Google Drive felcsatolása után tudjuk elkezdeni topic-ból kiírni az adatokat fájlokba Python segítségével. Ahhoz, hogy kiírjuk az adatokat szükségünk lesz KafkaConsumer és TopicPartition csomagok importálására. Ezek után tudjuk lekérni a topicban szereplő adatokat. Ebben az esetben a bootstrap kiszolgáló nem változott.

```
GROUP = 'KafkaConsumer'
BOOTSTRAP_SERVERS =
['192.168.2.95:9092']
consumer = KafkaConsumer(
    bootstrap_servers=bootstrap_servers,
    group_id=GROUP,
    enable_auto_commit=True)
```

ábra 6.73 Consumer-hez csatlakozás

Az utolsó rekord megtalálásáért szükséges végigmenni a topic összes partícióján, mivel egy topic lehet több partíción is, és ez által keressük meg az utolsó topic-t, és ennek az offset-jét mentsük ki egy külső változóba. Mert későbbiekben még szerepe lesz számunkra.

```
#last_offset = 0
last_offset = 0
for p in consumer.partitions_for_topic(TOPIC):
   tp = TopicPartition(TOPIC, p)
   consumer.assign([tp])
   committed = consumer.committed(tp)
   if committed == None:
    committed = 0
   consumer.seek_to_end(tp)
   last_offset = consumer.position(tp)
   print("topic: %s partition: %s committed: %s last: %s lag: %s" % (TOPIC, p,
committed, last_offset, (last_offset - committed)))
```

ábra 6.74legnagyobb offset megkeresés

Ezt követően zárjuk be az adott topic figyelését.

```
consumer.close(autocommit=True)
```

ábra 6.75 Consumer bezárása

Jelenleg megtaláltuk az legnagyobb offset-tel rendelkező üzenetet az adott topic-ban, így el tudjuk kezdeni a topic tartalmának a kiírását.
Ehhez egy újabb consumer-re lesz szükségünk.

```
con = KafkaConsumer(topicName, bootstrap_servers=bootstrap_servers,
                    auto_offset_reset='earliest', enable_auto_commit=True,
                    auto_commit_interval_ms=1000,group_id=None)
```

ábra 6.76 Consumer-hez csatlakozás

Mikor a legnagyobb offset-tel rendelkező üzenetet kérdeztük le ott szükség volt group_id-ra, ebben az esetben nincsen. Most auto_offset_reset ami a lényeges számunkra, mert a legkorábbi üzenettől szeretnénk lekérdezni a üzeneteket, így ezt earliest-re kell hogy beállítsuk.

```
with
open('/home/kafkapublic/python3/googleDrive/StockNumbers/'+topicName+'.txt',
'w') as gd:
with open('/home/kafkapublic/datalake/'+topicName+'.txt','w') as d:
 for message in con:
  d.write(message.value.decode("utf-8"))
  d.write('\n')
  gd.write(message.value.decode("utf-8"))
  gd.write('\n')
  if last_offset-1 == message.offset:
   con.close()
```

ábra 6.77 topic-ban szereplő üzenetek kiírása fájlba

Esetleg, hogy hova mentjünk most adatot az csakis az elérési úttól függő jelenleg. Mivel mind a 2 esetben egy könyvtár alá van felcsatolva az erőforrás.

7. Tovább fejlesztés

Abban lehetne tovább fejleszteni, hogy a Kafka szolgáltatást valamilyen fajta cluster-be magyarul “fürtbe” helyeznénk. Ehhez szükséges kellő mennyiségű erőforrás, szerver oldali erőforrás. Ebben az esetben meg lenne a redundancia is a Kafka szolgáltatásnak, ha az egyik szerver elérhetetlenné válna bármiféle meghibásodás esetén, akkor lenne még egy vagy akár több egyéb erőforrás, ami át tudná venni ezt a fajta terhelését.

8. Összefoglaló

Data pipeline egy olyan technika, ami tudja sorfolytonosan fogadni az üzeneteket a forrásrendszer oldaláról, ez a forrás rendszer legyen akár ERP rendszer, CRM, KSH statisztikai adatok, árfolyam változással kapcsolatos adatok, vagy akár időjárás, tőzsdei adatok cégeknek, vagy bármi egyéb adatbázis oldali forrásrendszer. Ezeket az adatokat data pipeline le tudja tárolni, addig ameddig ezek az adatok a data pipeline-ból törlésre nem kerülnek. Ezeket adatokat későbbiekben valami központi oldalról le kell, hogy tudjuk tárolni, és esetleg későbbiekben riportokat tudjanak az elemzőink készíteni belőle. Ebben az esetben Google és az Uber tőzsdei adatait töltöttük le, és ezeket használtuk fel egy API kérés segítségével. Hoztunk létre topic-okat, hogy a letöltött adatokat el tudjuk tárolni. Ezeket az adatokat töltöttük be az előre meghatározott létrehozott topic-ba és data pipeline-ba. Ezeket az üzeneteket helyeztünk el végső helyükön, egy olyan típusú központi adattárban, amit a Big Data rendszerekben szokás használni, data lake-be, de ez lehetne valamilyen egyéb Hadoop tipusú elosztott rendszer is a végén. Végig lehetett követni, hogy akár tőzsdei adatok vagy egyéb logfájlok, hogyan tudnak eljutni egy központi adattárba egy data pipeline segítségével.

9. Summery

Data pipeline is a technique that can continuously receive messages from the source system, be it ERP system, CRM, CSO statistics, exchange rate data, or even weather, stock market data for companies, or any other database-side source system. . You can store this data in the data pipeline as long as this data is not deleted from the data pipeline. We need to be able to store this data from a central site later, and possibly report it back to our analysts. In this case, we downloaded Google and Uber stock data and used it using an API request. We have created topics so that we can store the downloaded data. We loaded this data into the predefined created topic and data pipeline. We placed these messages in their final location, in a central repository of the type commonly used in Big Data systems, in a data lake, but it could be some other Hadoop-type distributed system in the end. It was possible to keep track of how even stock data or other log files could be accessed to a central repository using a data pipeline.

10. Ábrajegyzék

11. Források

[1]
https://dih.telekom.net/en/author/sven-loeffler/ (Olvasva 2020.05.10)
[2]
https://hazelcast.com/glossary/data-pipeline/ (Olvasva 2020.05.07)
[3]
https://medium.com/@mrashish/design-strategies-for-building-big-data-pipelines-4c11affd47f3 (Olvasva: 2020.05.07)
[4]
https://www.alooma.com/blog/what-is-a-data-pipeline
Megjelenés dátuma: 2018 11 26
Író: Garrett Alley
[5]
Apache Kafka for Beginners Guide part 1-2
https://www.cloudkarafka.com/files/PtgDYAyuBn3fcJBZV/apache-kafka-beginner-guide.pdf (Olvasva: 2020.05.18)
[6]
https://searchdatamanagement.techtarget.com/definition/data-lake (Olvasva: 2021.12.10)
[7]
https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-ubuntu-18-04 (Olvasva: 2021.12.11)
[8]
https://zookeeper.apache.org/doc/r3.5.4-beta/zookeeperOver.html (Olvasva: 2021.12.11)
[9]
https://learn.onemonth.com/python-2-vs-python-3/ (Olvasva: 2021.12.09)
[10]
https://docs.python-guide.org/starting/install3/linux/ (Olvasva: 2021.12.11)
[11]
https://www.zuar.com/blog/what-is-a-data-pipeline-and-how-to-build-one/