Kutatási jelentés

Adatok tárolását, feldolgozását, megjelentését támogató informatikai rendszer

feladat 1. fázisához kapcsolódóan

TARTALOM

1 A FELADAT MEGHATÁROZÁSA

Feladat megnevezése: adatok tárolását, feldolgozását, megjelentését támogató informatikai rendszer

Fázis 1:

Az elektroszmog szakterület backend moduljának elkészítése mely megvalósítja az adatok fogadását, tárolását, elemzését, lekérdezhetővé tételét, push értesítést és felhasználókezelést. A fejlesztés során mikroszolgáltatásokra épülő szerver oldali architektúrát alkalmazunk, melynek révén biztosított lesz az egyes szerver oldali komponensek egységbe zárása, skálázhatósága. Adattárolásra open-source relációs és noSQL adatbázisokat alkalmazunk.

Fázis 2:

Az elektroszmog tématerület webes kliensének fejlesztése, mely lehetővé teszi az összegyűjtött és elemzett adatok megjelenítését, monitorozást, nyomonkövetést modern webes felületen. A kliens felületen kereshetővé válnak az egyes területek térképes nézetben. Jól használható térképek készülnek az összegyűjtött adatokra alapozottan, melyek szemléltetik az adott területen mért elektroszmog mértékét.

Fázis 3:

Az elektroszmog szakterület backend moduljának második változata: elemzés, lekérdezhetővé tétel, push értesítés és felhasználókezelés.

Az elektroszmog tématerület webes kliensének második változata: monitorozás, nyomonkövetés modern webes felületen.

Jelen dokumentum az első fázisban történt kutatási-fejlesztési feladatokat összegzi.

2 BEVEZETÉS

Az utóbbi években mind a Big Data rendszerek, valamint a dolgok internete (Internet of Things; IoT) alapú megoldások nagy fejlődésen mentek keresztül.

A digitalizációnak, valamint a nyílt platformoknak köszönhetően az internetezők számára egyre több hasznos szolgáltatás és adatforrás érhető el, melyek nélkül mindennapi tevékenységeink már szinte elképzelhetetlenek lennének.

Ilyen szolgáltatás lehet egy egyszerű útvonaltervezés két helyszín között gyalog vagy autóval, vagy például különböző hasznos helyek, érdekes pontok (Points Of Interest; POI) adatbázisa. Természetesen napjainkra ezek a szolgáltatások már általában egymásba integrálva használhatók, különböző nagyvállalatok szolgáltatásaiként vagy akár közösségi összefogások eredményeként.

Ezeket az eredményeket terjesztik ki az okoseszközök és a hozzájuk kapcsolódó szolgáltatások, melyek egyre jelentősebb szereppel bírnak a legtöbb iparág számára is.

Az elmúlt évtizedekben nagyban nőtt az internetre kapcsolt eszközök száma. Ez főként az okostelefonok és a hordozható „kütyük" fejlődésének, növekedésének és elterjedésének köszönhető. Néhány tényadat az elmúlt évek mobilvilágával kapcsolatban:

- 4,5 milliárd telefon van a világban, ennek mintegy a fele okostelefon.
- A mobil-előfizetők száma közel 7 milliárd.
- Az eszközök kétharmadán van kamera, amelyet a felhasználók 72%-a használ is.
- A mobiltulajdonosok 91%-a 7×24 órában a telefonja közelében van.
- Minden harmadik ember olvas híreket a telefonján.
- Több mint egymilliárd ember tölt le alkalmazásokat, használ közösségi hálót.

A tendencia tartja magát, sőt a következő évekre további növekedés várható, ahogy újabb megoldási lehetőségek, technológiák kerülnek előtérbe, többek között a kommunikációhoz elengedhetetlen ötödik generációs vezeték nélküli hálózat (5G), valamint az új IPv6 internetes protokoll. Így 2020-ban várhatóan már 30 milliárd IP-szenzor lesz működésben.

Ezek adathalmazait az interneten keresztül összegyűjtve egy intelligens környezet, egy világméretű kép alakulhat ki az okoseszközöket használók tulajdonságairól, szokásairól. Az adatok segítségével akár a jövőbeli tevékenységeiket, szándékaikat is meg lehet jósolni.

Az IP alapú szenzorok működéséhez a nevéből következően szükség van IP címekre is. 2011. február 3-án az internet központi adminisztrációjáért felelős IANA (Internet Assigned Numbers Authority) kiosztotta az utolsó blokkokat a közel 4 milliárd IPv4-es címek közül, ezzel kifogyasztva az összes globális szabad tartalékot. Ez nem volt váratlan esemény, már az 1980-as években is elkezdtek kialakítani további technológiákat a korlátos tartomány kiküszöbölésére, így 1981-ben a címosztályokat használó hálózatkezelést, 1983-ban az osztály nélküli forgalomirányítást és a hálózati címfordítást, 1999-ban pedig megszületett az új szabvány, az IPv6.

Összehasonlításképpen az IPv4-es címek 32 bitesek, így az elméleti maximálisan kioszthető címek száma $4*10^9$ darab, az IPv6-os címek 128 bitesek, így az elméleti maximálisan kioszthető címek száma $3*10^{38}$ darab cím. (Az elméleti szó fontos, mert mindkét esetben nem számítanak a privát tartományok és egyéb különleges címek.)

Ha összevetjük az emberiség létszámát ($7,4*10^9$ fő) **Hiba! A hivatkozási forrás nem található.** a kioszthető IP-címekkel látható, hogyha mindenkinek egyetlen internetre csatlakozó eszköze lenne, akkor sem lenne elég az IPv4-es címtartomány. (Persze itt is éltem mindenféle tényezők elhanyagolásával: egyidejűség, belső hálózat, fejlődő országok, stb.) Ha pedig vesszük napjaink eszközeit melyek az internetre csatlakoznak, így kiindulásképpen először a hálózati kártya az asztali számítógépünkben, aztán a notebookunkban, a tabletünkben, a telefonunkban, a másik telefonunkban, a nyomtatónkban, az IP-kameránkban, a riasztónkban, a hűtőnkben, a mindenféle szenzorunkban, a további háztartási eszközeinkben, járműveinkben, akkor a szükséges IP címek mennyisége robbanásszerűen meg tud növekedni.

Az IPv6-os címtartományra történő átállás sem zökkenőmentes folyamat, az internetszolgáltatók próbálták húzni, halasztani az átálláshoz szükséges infrastruktúra kiépítéseket és átszervezéseket, ameddig csak lehetett. 2008-ban indultak meg az első tesztelések, azóta egyre több és több helyen vált elérhetővé az IPv6-os címzés és a hozzákapcsolódó további szolgáltatások.

Az IPv6 globális elterjedésével pedig lehetővé vált, hogy az eddigi „emberek internete" mellé a fogalmaink közé, valamint az internet világába becsatlakozzon a „tárgyak internete" is, mely stabil alapot nyújthat egy globális internet alapú társadalomhoz és a ráépülő élő, intelligens alkalmazásokhoz (város, közlekedés, otthon, e-közigazgatás, energiahasznosítás, oktatás, 3D média stb. mind-mind az okos- előtaggal).

Ha egy okostelefonra nemcsak telefonként tekintünk, hanem szenzorok halmazaként, akkor könnyen beláthatjuk, hogy ezen adatok felhasználásával hihetetlen mennyiségű információ nyerhető ki. A térinformatikát tekintve pedig gondoljunk a földrajzi koordinátákra (mind GPS, mind a bázisállomások alapján), a mikrofonra, a kamerára, a gyorsulásmérőre, a fényérzékelőre, lépésszámlálóra stb. És ezek csak a legalapvetőbb, szinte minden okostelefonban és okosórában elérhető mérőegységek.

Azonban amikor ennyi eszköz csatlakozik a világhálóra, nem egyszerű feladat a különböző adatfolyamok és a kommunikáció megvalósítása, egységes kezelése. Főleg, hogy már régebb ideje ad-hoc módon kezdtek el különböző eszközöket, különböző protokollokkal, különböző módokon csatlakoztatni a világhálóra. Másik feladat pedig a hatalmas mennyiségű adat tárolása, és későbbi hasznosítása. Ezen adatokkal kapcsolatos feladatokra került bevezetésre a Big Data fogalma, a kapcsolódó elméleti koncepciók és gyakorlati megvalósítások. Amíg az adatok feldolgozása egy szabvány vagy szervezet fennhatósága alatt van, addig az adott terület adatai, protokolljai egységesnek gondolhatók, azonban ha már például két különböző szervezet infrastruktúrája között kell adatátvitelt megvalósítani, felmerülhetnek problémák. Ennek a megvalósítására egyelőre nem nagyon várható globális protokoll, inkább a nyílt alkalmazásprogramozási interfészek (API) felé mozdulunk el, azonban a feladatok hasonló kezelése miatt lehetőség van egy egységes felület létrehozására, ahol a feladatok logikáját csak egyszer kell implementálni és utána az adott szakterületre való leképezés és finomhangolás már könnyen, apró módosításokkal megvalósítható.

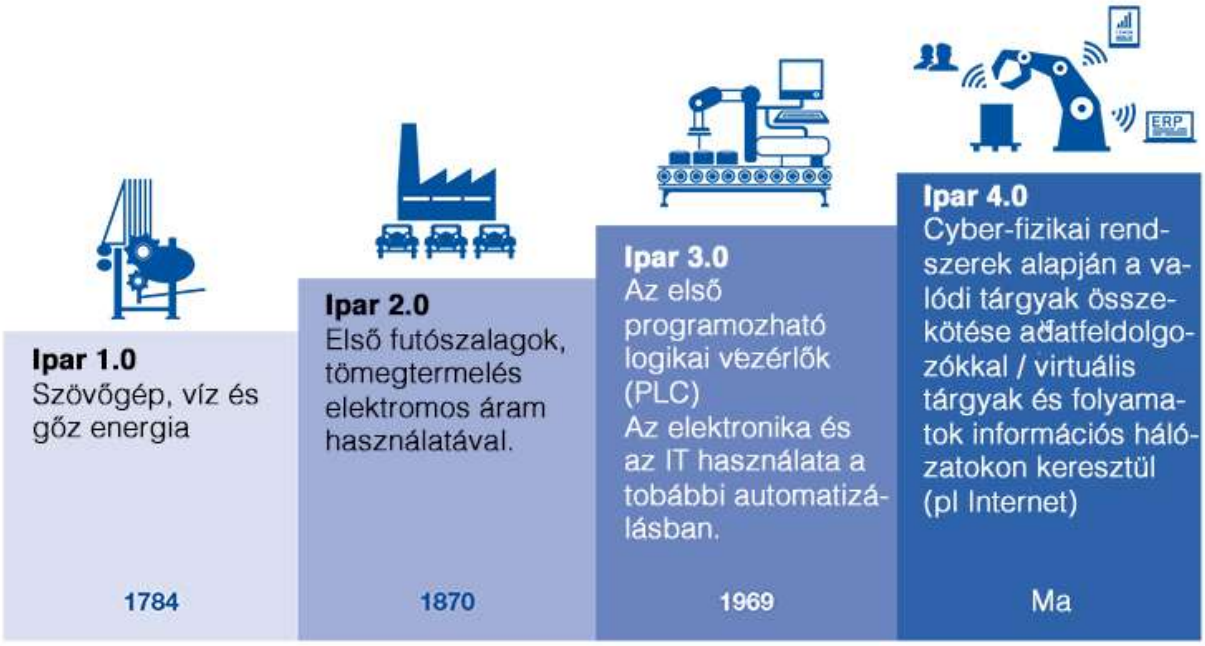
Mindebben a fejlődésben és a megjelenő új eszközök és technológiák sokaságában érthető, hogy a környezetre való hatás, az emberekre és a természetre mért hatások kérdése felmerül. Ezen a ponton van jelentősége a különböző sugárzások mérésének, ábrázolásuknak, majd az adatok feldolgozásával az együttes erejüknek és hatásuknak az elemzésének. A projekt céljai ezen területhez kapcsolódnak. Eszközként az IoT szakterületet használva, úgynevezett IoT-jellegű módszer kidolgozásával és IoT keretrendszer felkészítésével történik.

Napjaink IoT platformjainak első lépése az adatok gyűjtése (érzékelés), különböző adatforrásokból, melyek a megfelelő átjárókon eljutnak a platform Big Data környezetébe, ahol megtörténik a második lépés, vagyis a feldolgozás. A feldolgozott adatokból kinyert információk, riportok és összefüggések adnak értéket az adat tulajdonosának a kezébe, melyek generálják a harmadik lépést, a beavatkozások vezérlését.

Saját IoT koncepciónk és keretrendszerünk biztosítja az alapot az IoT világban bevett „érzékelés-feldolgozás-beavatkozás” szemlélethez, a gyakorlati alapot pedig a szerver oldali, Big Data alapú elemzéshez és üzleti intelligencia riportok elkészítéséhez.

3 IPAR 4.0

Ez az irány lesz az alapja a napjainkra fogalmaink közé épülő Ipar 4.0 megoldások megvalósíthatóságának, mely szemléletmód a negyedik ipari forradalom elérkezését takarja. Az első ipari forradalom, mely egy az átfogó társadalmi, gazdasági és technológiai változást jelent, a XVIII. század második felében és a XIX. század első felében zajlott le, először Nagy-Britanniában majd Európában végül Amerikában. Alapját az 1769-ben Watt által elkészített gőzgép és a Cartwright által 1784-ben elkészített szövőgép jelentette, mely gépesítéseket ezután egyre több és több ipari felhasználásba építettek be. A nagyhatású fejlődések és fejlesztések majdnem 100 év múlva elhozták a második ipari forradalmat, mely a futószalagos gyár, újabb találmányok, elektromosság bevezetésével segítették a gazdasági és társadalmi fejlődést. Megint 100 évnek kellett eltelnie, mire az elektromos ipar odáig fejlődött, a harmadik ipari forradalomnak nevezett korszakban, hogy elkészültek az első számítógépek, mikrovezérlők, 1969-ben pedig megjelent az internet, vagyis a hálózatba kapcsolt számítógépek első realizálása. Eltelt 40-50 év és eljutottunk napjainkra a negyedik ipari forradalomhoz, melynek alapját az internet, a hálózatba kapcsolt számítógépek és egyéb eszközök, valamint a mesterséges intelligencia jelenti.

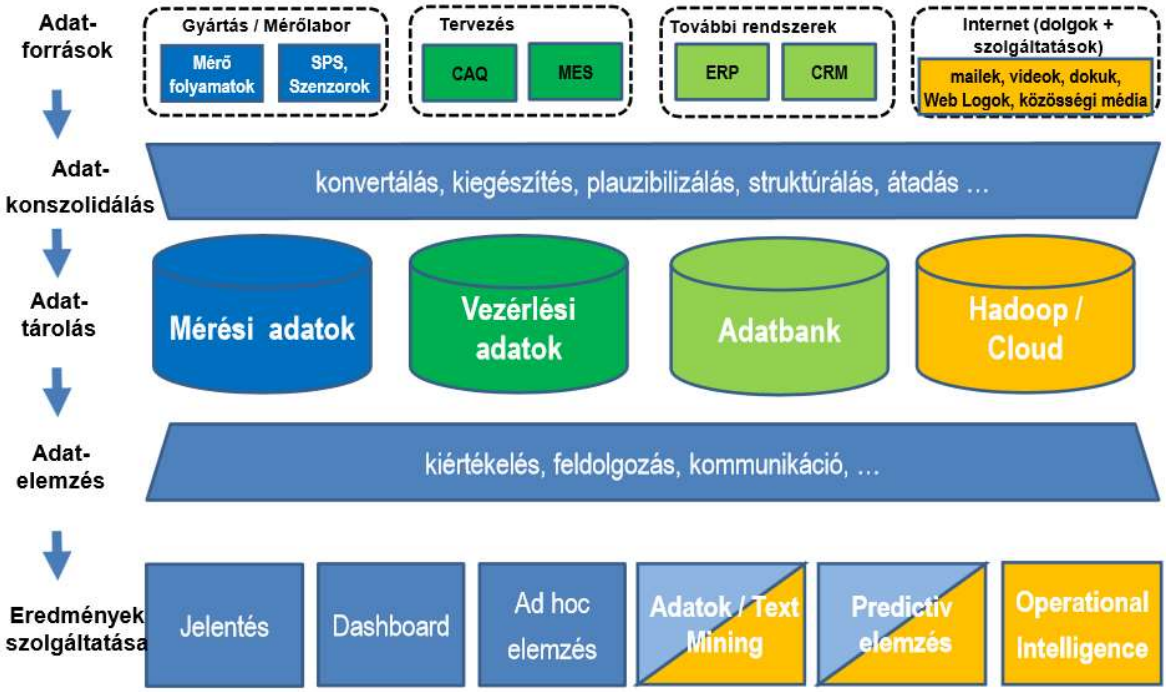


Ábra: Ipari forradalmak

Az Ipar 4.0 koncepciója az internet, IP-alapú, hálózatba kapcsolt számítógépek, eszközök és szenzorok bekapcsolása a gyártási folyamatokba, Big Data alapú adattárolással és gépi tanulási algoritmusokkal

kibővítve (kiber-fizikai rendszerek és a dolgok internete). Vagyis a koncepció célkitűzése az önkonfigurálhatóság, önoptimalizálás, öndiagnózis és megítélés szemlélete mellett a kompakt és intelligens termelés megvalósítása a rendelkezésre álló technológiák kihasználásával.

A Big Data elveket követve az ipari és gyártási folyamatok adatforrásai felől érkező adatok először adatkonszolidáláson mennek keresztül, mely a konvertálás, kiegészítés, plauzibilizálás, strukturálás, átadás stb. műveleteket jelentheti. Az így elkészült tisztított, transzformált adatok kerülnek eltárolásra a megfelelő adatbázisokban. Az eltárolt adatokból az eredmények, mérőszámok, jelentések pedig a megfelelő adatelemzések lefuttatása után állnak elő, mely lehet adatszelektálás, statisztikai kiértékelés, feldolgozás, kommunikáció stb.



Ábra: Hagyományos és új elemek az ipari termelésben

Ebbe a koncepcióba kapcsolódhat majd bele a saját IoT platformunk is, mely a megfelelő adminisztrációs és menedzsment komponensekkel kiegészítve rendelkezésre tud állni az Ipar 4.0-ra átálló ipari partnerek rendszereinek alapjául, és megfelelő módon biztosítva lesz a használt szolgáltatások és adatmennyiségek monitorozása, melyek felhasználásával, valamint pénzügyi metódusok integrálásával a teljesítmény és kihasználás alapú számlázás is elérhetővé válik a felhasznált adatmennyiségek alapján.

Jelen állás szerint a saját IoT platformunkat (továbbiakban IKT platform) életciklusát két irányba tervezzük és fejlesztjük. Az egyik különböző projektek, kutatások és fejlesztések alapjául szolgáló IoT keretrendszer, a másik pedig bekapcsolódás a piacra és a platform szolgáltatásainak és erőforrásainak kiajánlása az ipari, vállalati partnerek felé.

Az IKT Platformhoz kapcsolódó feladatok:

- A különböző szakterületeken gyűjtött adatok szakterületi igények szerinti feldolgozása.

- Multi-domain környezetet támogató Big Data módszerek, új összefüggések feltárása.
- Valós idejű feldolgozási módszerek multi-domain környezetben.
- IoT fejlesztések, biztonságot támogató algoritmusok kidolgozása.
- 5G alkalmazások és szolgáltatások fejlesztési módszertanok fejlesztése.
- Ipar 4.0 megoldások serkentése.

4 IoT PLATFORM SPECIFIKÁCIÓ

4.1 Szenzoradatok tárolásához szükséges szoftverkörnyezet

A szenzoroktól és egyéb IoT eszközöktől (pl. nagy sebességű és felbontású IP kamera forgalmi szituációk és teszt esetek elemzéséhez) származó adatok tárolásához megfelelő, moduláris adattároló platform kialakítása szükséges. Ezen IoT platform szoftverkörnyezete Hadoop-os/HDFS-es 'big data' alapokra – mint a Hortonworks HDP platform vagy ezzel ekvivalens – épül, és alkalmas kell, hogy legyen nagy mennyiségű, heterogén adatforrásokból származó adatok hatékony fogadására, kezelésére, tárolására és az adatokhoz való hozzáférés szabályozására.

A kialakítandó IoT platformnak skálázhatónak, hibatűrőnek, gyorsnak és kellően rugalmasnak kell lennie. Így a rendszer komponenseinek mind fizikai, mind virtuális csomópontokra telepíthetőnek kell lenniük elosztott módon, biztosítva ezáltal a terheléselosztást is. A különböző funkciókat ellátó modulok egy lazán csatolt rendszerben kell, hogy kapcsolódjanak egymáshoz. Az egyes modulokat rugalmas módon – célszerűen Docker konténerekben Kubernetes menedzsment rendszerrel kezelve – kell implementálni, hogy könnyen migrálhatók és lecserélhetők legyenek.

4.2 Adatok fogadása

Az adatok fogadására egyrészt szabványos MQTT IoT protokoll használatával (tipikusan szenzor adatok esetén) egy MQTT átjáró modulon keresztül, másrészt HTTP protokoll használatával (tipikusan nem szenzoroktól származó, vagy historikus adatok esetén) egy API átjáró modulon keresztül kell lehetőséget biztosítani. Az MQTT átjáró modul egy jól skálázható MQTT bróker – mint a VerneMQ vagy azzal ekvivalens –, amely különböző témákra (topic) feliratkozott és TLS alapú jogosítványok segítségével azonosított szenzorok adatait fogadja. A témákhoz való hozzáférés ACL-ek használatával dinamikusan szabályozható kell, hogy legyen. Az API átjáró modul egy egységes interfészt nyújt a külvilág felé az IoT platform szolgáltatásainak az igénybevételéhez a megfelelő autentikáció és jogosultság ellenőrzés után.

Az IoT platformhoz csatlakozó szenzorok/eszközök nyilvántartása egy eszköznyilvántartó modul segítségével kell, hogy megvalósuljon. Ezen modul kezeli az eszközök regisztrációját, és egyúttal előállítja a megfelelő hozzáférést biztosító jogosítványokat, amelyekkel az eszközök az MQTT átjáró modulnál azonosíthatják magukat. Ugyanakkor az eszköznyilvántartó modul hozzáadja a beregisztrált eszközöket az MQTT átjáró modul hozzáférésszabályzási listaiba (ACL), hogy engedélyezze a megfelelő témákra való adatküldést.

4.3 Adatok kezelése

A beérkező adatok kezelése és az IoT platformon belüli hatékony, gyors és hibatűrő irányítása egy megfelelő üzenetkezelő rendszer segítségével – mint az Apache Kafka vagy azzal ekvivalens – kell, hogy történjen. Ebbe az üzenetkezelőbe a beérkező adatokat egy-egy megfelelő modulnak kell átemelnie mind az MQTT átjáró, mind pedig az API átjáró modulból a korábban alkalmazott azonosítás és jogosultság ellenőrzés figyelembevételével. Az üzenetkezelő rendszerből közvetlenül, vagy ha előprocesszálás szükséges, akkor egy valós idejű esemény feldolgozási rendszeren – mint az Apache Nifi vagy azzal ekvivalens – keresztül kerülnek az adatok letárolásra.

Mivel a szenzoroktól és az adott szcenáriótól függően az adatok rendszerint különböző formátumban érkeznek az IoT platformhoz, ezért a letárolás előtt a beérkező szenzoradatokat egy adatkonverziós modul segítségével egységes – célszerűen JSON – formátumra kell alakítani a hatékony kezelhetőség végett. Az adatkonverzióhoz a beérkező adatokra vonatkozó – célszerűen valamilyen szkript nyelv alapú – adatséma leírás megadása és a konverziót végző modul esetleges felparaméterezése után a bemeneti adatokat a modul átkonvertálja az egységes kimeneti formátumba. A modul biztosítja továbbá a különböző adatséma leírók megfelelő tárolását és kezelését, valamint a bemeneti adatokhoz a megfelelő adatséma leíró kiválasztásához szükséges logikát. A megoldásnak kellően általánosnak kell lennie, így alkalmazható kell, hogy legyen a tesztpálya vonatkozásában megvalósítandó összes IoT szcenárió esetén.

4.4 Adatok tárolása

Az adattárolást hatékonyan, az aktuális igényeknek megfelelően többféle módon és formátumban is biztosítani kell mind relációs (SQL alapú), mind pedig nem relációs (NoSQL) megközelítés szerint, így MySQL, Apache Hive, illetve Apache HBase, MangoDB, vagy ezekkel ekvivalens adattárolási/adatbázis megoldások használatával.

4.5 Adatokhoz való hozzáférés szabályozása

Az IoT platformban kezelt és letárolt adatokhoz való, jogosultság alapú hozzáférés szabályozásáról megfelelő módon – célszerűen az Apache Ranger vagy azzal ekvivalens rendszer alkalmazásával – gondoskodni kell. Így a rendszernek több szintű – mint például tulajdonos, csoport, egyéb – hozzáférés szabályozást kell biztosítania mind az IoT platformon éppen kezelés/előfeldolgozás alatt álló, mind pedig a letárolt adatok vonatkozásában.

4.6 Egyéb követelmények

Az IoT platformnak lehetőséget kell biztosítania igény esetén a letárolt adatokon való feldolgozási, analitikai és vizualizációs feladatokat megvalósító modulok – mint az Apache Superset, Apache Zeppelin, Apache Spark vagy ezekkel ekvivalens – integrálására. Viszont a jelen fázisban ezen feladatok/funkciók megvalósítása nem része a kialakítandó IoT platformnak.

5 AZ IOT KÖRNYEZET ÁLTALÁNOS JELLEMZŐI

A következő fejezetekben az IoT rendszerekben alkalmazható adattárolási és adatfeldolgozási módszereket tekintjük. A második fejezet az IoT környezet általános jellemzőit ismerteti. A harmadik fejezetben az IoT környezetben használt adatok életciklusát tekintjük át. A negyedik fejezet az IoT adattovábbítás módszereit foglalja össze. Az ötödik fejezetben a valósidejű feldolgozást és a NoSQL adatbázisokat ismerjük meg. A hatodik fejezet a MapReduce programozási modellel történő utólagos adatfeldolgozást tárgyalja.

IoT alkalmazásokon általában olyan alkalmazásokat értünk, amelyek változatos hálózatra csatlakozó eszközökön futnak. A Gartner definíciója egész pontosan a következőket mondja [1]:

> Az Internet of Things (IoT) olyan fizikai objektumok hálózata, amelyek beágyazott technológiákat használnak a kommunikációhoz és az érzékeléshez, vagy pedig interakcióba lépjenek a belső állapotaikkal vagy a külvilággal.

A fejezetben áttekintjük az IoT környezet általános jellemzőit.

5.1 Felhasználási területek

A felhasználási területek sokasága miatt nem lehetséges pusztán egy példával szemléltetni az IoT eszközök jellemzőit. Az IoT eszközök használata egymástól nagyon különböző területeken is elterjedt, mégis indokolt ezeket az alkalmazásokat egy közös kategória alatt említeni, hiszen számos közös sajátossággal rendelkeznek. Hogy megérthessük ezeket a sajátosságokat, célszerű először áttekinteni a felhasználási területeket, és ezeken belül egy-egy alkalmazási példát. Ez segít átlátni a további alfejezetekben összefoglalt közös vonásokat.

A Beecham Research az IoT piacon kilenc különböző szolgáltatási szektort azonosított be [2]. A továbbiakban áttekintjük ezeket a szolgáltatási szektorokat. A szektorokon belül további kategóriákat is azonosítottak, ez azonban túlmutat a jelentés terjedelmi keretein.

Az első azonosított szektorba az **épületekhez kapcsolódó** rendszerek tartoznak. Idesoroljuk a fűtést, hűtést és légkondicionálást ellátó berendezéseket. Ezeknek a rendszereknek a hatékony működéséhez elengedhetetlenek a szenzorok, amelyek visszacsatolást nyújtanak a rendszernek, és képesek szabályozni a rendszert. Szintén ebbe a szektorba sorolhatók a biztonsági beléptetőrendszerek vagy a világítás. Például, a mozgásérzékelővel vagy fényerőszenzorral támogatott világítási rendszer hatékonyabb energiafelhasználást tesz lehetővé, mint egy hagyományos, programozott ideig világító társasházi lépcsőház-világítás. Az épületekben alkalmazott IoT rendszerek további példái a személyi biztonságért felelős rendszerek, pl. a szénmonoxidérzékelő vagy a sprinklerrel vagy oltógázos rendszerrel kombinált tűzjelző.

A második szektor az **energiatermelés.** Ebben a szektorban el kell látni az energia iránti kereslet és kínálat összehangolását. Ehhez hagyományos (fosszilis, víz- vagy nukleáris-) és megújuló energiaforrások működését szükséges összehangolni, valamint járulékos feladatokat is el kell látni, pl. a szükséges nyersanyagok (kőolaj, földgáz) szállítását.

A harmadik szektor a **háztartás és fogyasztás.** Ide változatos fogyasztói szolgáltatások tartoznak, amelyek egy része átfedést mutat az épületen belüli IoT rendszerekkel, azonban a pontos igények általában eltérőek. Ilyenek a fűtést, hűtést, légkondicionálást végző eszközök. Manapság egyre terjednek az okos termosztátok, amelyek fejlett szoftvere kifinomult hangolást tesz lehetővé a rendszerhez, és képes távolról is vezérelni a rendszert, hogy a hazaérkezésünkre kellemes hőmérséklet fogadjon. Szintén átfedést mutat a világítás, illetve a személyi biztonságért felelő rendszerek. Gyakoriak a vagyonvédelmi szolgáltatóhoz csatlakozó riasztók. A szektoron belül egy újabb csoportba sorolhatók a háztartási munkák elvégzését segítő eszközök, pl. robotporszívók, robotfelmosók, mosogatógépek, mosógépek stb. Ezek az eszközök is mind szenzorokkal és fejlett szoftverrel teszik kényelmesebbé és hatékonyabbá a házimunkát. Egy újabb csoportot képeznek a szórakoztatóeszközök. Idetartoznak a házimozirendszerek képi megjelenítői és hangrendszerei, valamint a játékkonzolok és a hozzájuk készült VR kiegészítők.

A negyedik szektort az **egészségügy** képviseli. Idetartoznak a kórházon belül vagy akár otthoni kezeléshez alkalmazott diagnosztikai és monitorozó eszközök, pl. a telecare rendszerek, amelyek egy mobileszközön futnak, és a mobileszközzel kommunikáló műszerektől (pl. mérleg, vérnyomásmérő) érkező adatok alapján monitorozzák a beteg egészségét.

Az ötödik szektorba az **iparban használt** eszközök sorolhatók. Ebben a szektorban pl. folyadékok, gázok továbbítását, csomagolást, gyártási folyamatokat, ellátási lánc monitorozását és biztosítását is vezérelni kell. Szintén idesorolhatók a mezőgazdasághoz kapcsolódó eszközök, pl. az öntözés és betakarítás.

A hatodik szektort a **szállítás** alkotja. Idetartoznak a járművökön belüli navigációs, monitorozó és biztonsági rendszerek, az utasinformációs rendszerek és az utasok szórakoztatását szolgáló berendezések is. Egy másik csoportot alkotnak a nem járművön belüli rendszerek, pl. az úthasználati díjat ellenőrző vagy torlódást detektáló rendszerek.

A hetedik szektort az **árusítás** eszközei képviselik. Ezek pl. a POS-terminálok, kávé- és édességautomaták, elektronikus pénztárgépek stb., de idesorolhatók az ellátási láncot vagy a vásárlói adatokat nyilvántartó rendszerek és az RFID-s termékmegjelölés is.

A nyolcadik szektorba a **közbiztonságot ellátó** eszközök sorolhatók, pl. a hatóságok és az orvosi segítségnyújtás mobil eszközei vagy a fixen felszerelt megfigyelő eszközök, mint a járműforgalom monitorozása.

A kilencedik szektort az **IT és hálózat eszközei** alkotják. Ebbe a szektorba tartoznak az irodai eszközök, mint nyomtatók, fénymásolók, szkennerek vagy a hálózati kommunikációhoz használt routerek, tűzfalak. Szintén a szektorba sorolhatók a magas rendelkezésre állású szerverszámítógépek mellett használt speciális fűtő-, hűtő-, szellőző- vagy tűzoltórendszerek.

5.2 Beágyazott technológiák

A fenti példákból megállapítható, hogy az IoT eszközök nagy része korlátozott erőforrással rendelkező beágyazott rendszer. Ennek oka gyakran a mobilitás szükségessége, pl. egy futár praktikusan egy tableten vagy egy okostelefonon tudja a csomag felvételét és a kézbesítés tényét rögzíteni, de egy teljes laptop kezelése már jelentős kényelmetlenséget és többletmunkát jelentene számára. A beágyazott technológiák alkalmazásának másik tipikus oka, hogy az eszközök méretét és komplexitását korlátozni

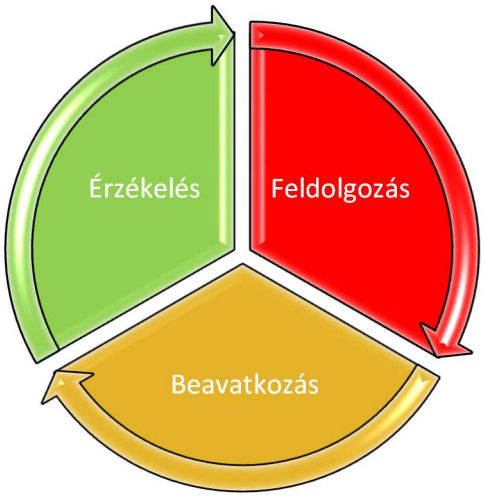
akarjuk. Pl. egy biztonsági kamera vagy egy kihelyezett hőmérő praktikusan nem lehet túl nagy, nem fogyaszthat túl sokat, ugyanakkor valamilyen célszoftvert kell futtatnia, amellyel az adatokat a megfelelő helyre továbbítja. Ezek a sajátosságok ahhoz vezetnek, hogy az IoT alkalmazásainkat korlátozott erőforrású környezetben futtatjuk, ezért különösen ügyelni kell a hatékony szoftverek kifejlesztésére. Másrészt, ezekre a rendszerekre jellemzően valamilyen hardverközeli programozási nyelv, tipikusan C segítségével fejlesztünk, bár esetenként (pl. Android alatt), rendelkezésünkre állnak magasabb szintű programozási nyelvek is.

5.3 Hálózati kommunikáció

Az IoT alkalmazásokban kiemelt szerepe van az eszközök kommunikációjának. Az adatok gyakran több szenzor együttes használatával állnak össze, és ezek a szenzorok sokszor fizikailag önálló eszközök. Például egy telecare rendszer használata során a páciens több műszerrel is rendszeresen végez méréseket (pl. vérnyomásmérő, vércukorszintmérő, mérleg stb.), de az adatok pontos értelmezéséhez az értékeket együtt kell elemezni. Egy másik példa lehet a forgalomszabályozás. A jelzőlámpák ideális hangolásához, és a megengedett sebesség megállapításához az útszakaszon sok tényezőt szükséges figyelembe venni. Pl. a gépjárművek haladása mellett figyelni kell azt is, hogy a gyalogosok mennyire torlódnak fel a gyalogátkelőhelyen, vagy a tömegközlekedési eszközök a megállók miatt mennyivel maradnak le a többi gépjármű mellett. Ha a gyalogosok feltorlódnak a zebra előtt, az balesetveszélyes lehet, de a gépjárművek túlzott feltorlódása sem ideális, mert több károsanyag-kibocsájtással jár. Egy összehangolt kompromisszumot csak az összes tényező megfelelő elemzésével lehet kijelölni.

5.4 Érzékelés, feldolgozás és beavatkozás

Az IoT alkalmazásokban a szenzorokkal vagy esetenként felhasználói inputtal bevitt adatokat felhasználjuk, eredményekre jutunk belőlük. Pl. egy telecare rendszerben a hosszabb ideje fennálló magas vérnyomás krónikus betegségre utalhat. Vagy egy tűzjelző rendszerben a füstszenzor jelzése tűzesetet jelenthet. Ilyen esetekben a szoftvernek fel kell dolgoznia az adatokat, és ha indokolt, be kell avatkoznia. Ez a beavatkozás a példákban lehet a kezelőorvos értesítése vagy a sprinklerek beindítása. Általánosan megfigyelhetjük, hogy a legtöbb IoT alkalmazás a fenti három lépés – érzékelés, feldolgozás, beavatkozás – ismétlésével működik. Az alábbi ábrán látható a háromlépéses folyamat.



5.5 Nagy mennyiségű adatok

A nagy mennyiségű és változatos adatokkal dolgozó rendszereket BigData [3] alkalmazásoknak hívjuk. Bár nem mindre, számos IoT alkalmazásra ez a jellemző, ezért BigData alkalmazásnak tekinthetők. A BigData alkalmazásokat az „5V”-vel szokták jellemezni, amelyek 5 V betűvel kezdődő jellemzőt értenek. A konkrét jellemzőket különböző vállalatok más-más módon definiálják. Általában az első három V mindenhol megyezik, a másik pedig változhat:

- Volume: nagy mennyiség
- Velocity: nagy sebességgel
- Variety: változatos formában
- (Variability: akár ellentmondó, inkonzisztens adatok)
- (Veracity: pontosság kérdéses)
- (Value: ezek az adatok értékesek, üzleti értéket teremtenek)

6 Adatok IoT környezetben

Ebben a fejezetben az adatok életciklusát vizsgáljuk az IoT alkalmazásokban. Az adatokat érintő események illeszkednek az előző fejezeteben bemutatott háromlépcsős folyamatba. Az adatok előállítása az érzékelésnek felel meg. Utána azonban az adatokat gyakran továbbítani is szükséges, hiszen a példák során láttuk, hogy az IoT alkalmazások gyakran elosztott rendszerek, és az eszközöknek kommunikálniuk kell egymással. Az adatokat ezután általában tároljuk is, kivéve, ha azonnal feldolgozzuk őket, és többé már nincs rájuk szükség. Végül pedig az adatokat fel kell dolgoznunk, hogy a megfelelő beavatkozást el lehessen végezni.

6.1 Az adatok előállítása

Az adatokat tehát nagyrészt a szenzorok állítják elő. Természetesen felhasználói input is paraméterezheti az IoT alkalmazások működését, de ezek volumene általában elhanyagolható a szenzoradatokhoz képest. Ha megvizsgáljuk a példákban elképzelhető szenzorokat, arra jutunk, hogy

elég sokféle szenzort használunk IoT alkalmazásokban. Ebből következően az adatok igen sokfélék lehetnek. Két fő típusukat tudjuk megkülönböztetni. Ezeket az alábbi táblázatban foglaljuk össze.

Példa szenzor	 **Biztonsági kamera**	 **Légszennyezettség-mérő**
Adategység mérete	Nagy fájlok (pl. óránkénti, napi bontás)	Kis csomagok (néhány paraméter aktuális értéke)
Gyakoriság	Kevés fájl	Nagyon sok fájl
Elérés	Szekvenciális (megnézünk egy részt egyben)	Véletlenszerű

Az előállított adatokra általánosan jellemző, hogy nem reprodukálhatók. Az adatok védelme ezért kiemelt szereppel bír, és a hagyományos backup megoldások a folyamatosan érkező adatok miatt nem is alkalmasak a teljes védelemre. Ezért inkább az integrált megoldások kerülnek előtérbe, pl. a tükrözött lemezek, amelyek ezen kívül az olvasási teljesítményt is javítják.

A hardver terén szintén korlátozást jelent a hagyományos hardveres megoldás. Ugyan az SSD (Solid-State Drive) lemezek gyorsak, kevés energiát fogyasztanak, illetve a mozgó alkatrészek hiánya miatt tartósak is, ezért jó választásnak tűnhetnek. A tárolandó adatmennyiség azonban gyakran meghaladja az SSD lemezekből ésszerűen kiszolgálható limitet. Jellemzőbb ezért a felhőalapú tárhelyszolgáltatások használata, amelynek segítségével szoftveresen konfigurálhatók a használni kívánt virtuális lemezek (Software-Defined Storage).

A IoT alkalmazások adatának további jellemzője, hogy nem mindig tudjuk, hogy később mely adatok lesznek értékesek. A követelmények és az ipari trendek ugyanis változhatnak. Ezért érdemes lehet olyan méréseket és statisztikákat is tárolni, amelyeket jelenleg nem használunk fel, de esetleg a jövőben hasznosak lehetnek. Ez viszont további erőforrásokat igényel.

6.2 Az adatok továbbítása

Az adatokat a szenzorok vagy egyéb egymással kommunikáló komponensek közt továbbítani szükséges. A továbbításnak korlátot szab az IoT rendszerekre jellemző erőforráskorlátozott beágyazott környezet. Ezért a kis overheadet jelentő pehelysúlyú, kapcsolatnélküli protokollok ideálisak. Az adatküldés az erőforrásokkal – főleg az energiafelhasználással – történő spórolás érdekében történhet periodikus, batch megközelítéssel is. A protokollválasztásban az is szempont lehet, hogy a kommunikáló felek egyenrangúak-e, és kölcsönösen megszólíthatják-e egymást, vagy pedig mindig egy kliens kezdeményezi a kommunikációt. Az IoT környezetben használt protokollokat a 4. fejezet tárgyalja.

6.3 Az adatok tárolása és feldolgozása

Az adatok tárolása és feldolgozása nem választható el élesen egymástól, ugyanis a tárolás ideális struktúrája már a későbbi hatékony feldolgozást kell szolgálja. A feldolgozás módjától függően elképzelhető hagyományos fájlalapú tárolás, vagy használhatunk a fájlrendszeren újabb rétegként futó adatbáziskezelő-rendszert is. Az adatbáziskezelő-rendszerek az adatok strukturálásával és különböző indexek létrehozásával és karbantartásával növelik a kívánt adatok kinyerésének hatékonyságát. A feldolgozás pedig kétféle lehet: valósidejű és utólagos. A valósidejű feldolgozásnál a kéréseket gyorsan, belátható időn belül meg kell válaszolnunk. Ilyen például annak az eldöntése egy telecare rendszerben, hogy szükséges-e orvost hívni. Más problémáknál, ahol nem kritikus az azonnali beavatkozás, elegendő az adatokat utólagosan feldolgozni. Ilyen például a forgalomszabályozás rednszeres felülvizsgálata és optimizálása. A kevésbé hatékony forgalomszabályozás ugyan eredményezhet forgalmi dugókat, de nem jelent azonnal elhárítandó veszélyt. Az alábbi táblázat összehasonlítja a feldolgozás két módszerét.

	Valós idejű	**Utólagos**
Időkeret	1-100 ms	1-100 óra
Kérések	1.000-100.000	1-10
Elérési mód	Írás-olvasás	Olvasás
Lefedett adatok	Működéshez szükséges	Utólagosan elemzendő
Végfelhasználó	Megrendelő	Data scientist
Technológia	NoSQL	MapReduce, Hadoop

A valósidejű feldolgozást az 5 fejezetben, az utólag feldolgozást pedig a 6. fejezetben tekintjük át.

7 AZ ADATOK TOVÁBBÍTÁSA

A fejezetben az IoT alkalmazásokon belül elterjedt adattovábbítási lehetőségeket vizsgáljuk meg.

7.1 Adattovábbítás REST elven

A HTTP (HyperText Transfer Protocol) [4] az egyik legelterjedtebb alkalmazásszintű protokoll, ezért minden jelentős platformon több library is támogatja a használatát. Így kézenfekvő választásnak tűnik az IoT alkalmazásokhoz is. A web eléréséhez is HTTP protokollt használunk, de az általánostól egyszerűbb formában. Tipikusan GET kérésekkel HTML oldalakat, képeket és egyéb erőforrásokat (CSS, JavaScript)

töltünk le, és ezeket a böngészőnk számunkra megjeleníti. Ha adatot szeretnénk a webalkalmazásnak beküldeni, – pl. egy megrendelés véglegesítése – azt egy form kitöltésével tehetjük meg. Ez többnyire POST kérést küld a szervernek.

A HTTP hagyományos webes felhasználása ember-gép kommunikációra alapszik. A HTTP azonban használható gép-gép közti kommunikációra is. Ekkor az elért erőforrások nem HTML nyelvű oldalak, hanem erőforrások reprezentációja egy programozottan jól feldolgozható formátumban, pl. XML vagy JSON. Ezt a kommunikációs elvet REST-nek (Representation State Transfer) [5] nevezzük. A REST kommunikációban a HTTP-kérést definiáló két legfontosabb paraméter az URL, ahova küldjük, illetve a kérés típusa. Az URL az erőforrást tükrözi, amelyet el akarunk érni, a kérés típusával pedig kifejezhető az erőforrással kapcsolatos szándékunk. A leggyakoribb kéréstípusok az alábbiak:

- GET: erőforrás lekérdezése
- POST: erőforrás létrehozása
- PUT: erőforrás frissítése
- DELETE: erőforrás törlése

A kéréseknek törzse is lehet, ez tartalmazhatja a létrehozandó vagy frissítendő erőforrást. A kérések paramétereket is továbbíthatnak a szerver felé az URL részeként, vagy a kérésben található fejlécben. Ezekkel pl. kijelölhető, hogy a választ milyen formátumban várjuk.

A szerver által küldött válasz tartalmaz egy válaszkódot, amely röviden összefoglalja a kérés kimenetelét, hogy sikeres volt-e, vagy ha nem, a kkor milyen hibával hiúsult meg. A válasz is tartalmazhat törzset, ebben visszaadható a lekérdezett erőforrás reprezentációja. Fejlécek is megadhatók a válaszban, pl. ezekkel adható meg, hogy a törzs milyen formátumban van.

A HTTP protokoll közismert és jól támogatott volta miatt egyszerű megoldást jelent IoT kommunikációhoz, azonban nem erre a célra tervezték, és ezért számos hátránnyal is bír. Először is, a protokoll üzenetei nem elég tömörek, és nagy hálózati forgalmat generál. A korlátozott erőforrású környezetben ez igen előnytelen. Másrészt, a protokoll kliens-szerver megközelítésű, és a kommunikációt mindig a kliens által küldött kérés kezdeményezi. A szerver nem képes önmagától üzenetet küldeni a kliensnek. Ezért események bekövetkezését csak periodikus lekérdezéssel – pollozással – lehet megállapítani. Ez további felesleges hálózati forgalomhoz vezet.

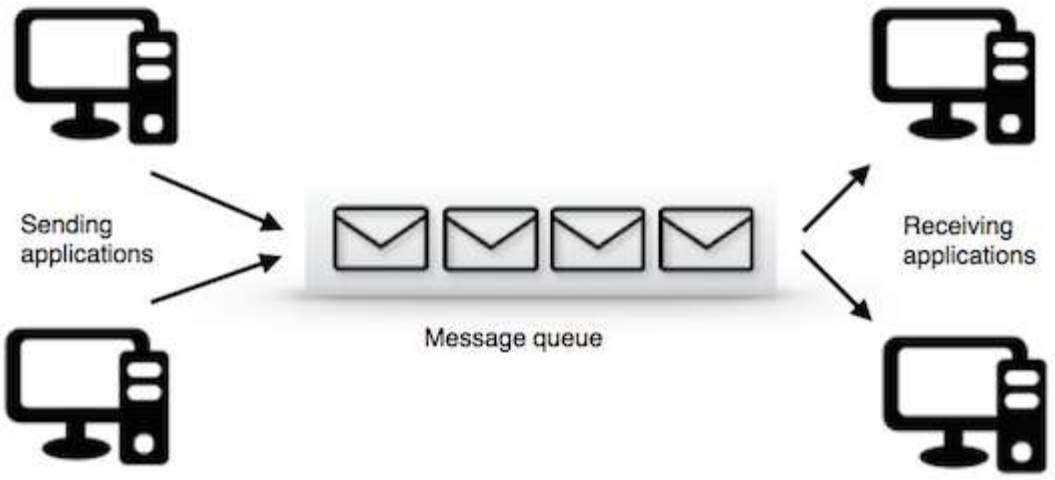
7.2 Adattovábbítás CoAP protokollal

A CoAP (Constrained Application Protocol) [6] a REST elv adaptációja erőforráskorlátozott környezetre. A protokollt az RFC 7252 dokumentum definiálja, és alacsony fogyasztású eszközökhöz, magas késleltetésű, nem megbízható hálózatokhoz készült. A CoAP tulajdonképpen a hálózati forgalmat csökkenti a HTTP feletti REST megvalósításokhoz képest, és a TCP helyett a kevesebb erőforrást igénylő, de kevésbé megbízható UDP protokollt használja. Az üzenetek fejléce bináris formátumú és tömörebb a HTTP fejléceinél. Az üzenetek könnyen lefordíthatók HTTP kérésekre, így a hagyományos HTTP feletti REST elven és a CoAP protokollal működő rendszerek jól integrálódnak. A protokoll azonban továbbra is kliens-szerver architektúrájú és kérés-válasz elven működik, tehát a pollozás problémáját nem orvosolja.

7.3 Adattovábbítás MQTT protokollal

A HTTP-vel ellentétben az MQTT (MQ Telemetry Transport) [7] kimondottan IoT környezethez készült. 1999-ben Dr. Andy Stanford-Clark (IBM) és Arlen Nipper (Arcom, most Eurotech) hozta létre, majd 2013-ban az OASIS kezdte meg a szabványosítását. A jelenlegi szabvány verziószáma 3.1.1. A protokoll definícióján kívül a hozzá készült könyvtárakat is megnyitották, és ezek nyílt forrású szoftverként elérhetők az Eclipse Paho [8] projekt keretén belül.

Az MQTT protokoll mögött egy üzenetsor (MQ – message queue) húzódik meg. Üzenetsorokat tágabb körben is alkalmaznak, nemcsak IoT környezetben. Az üzenetsorok két fél kommunikációjában látnak el közvetítő szerepet. Egyrészt könnyebbé teszik az integrációt, mert a két kommunikáló fél eltérő technológiákkal is készülhet, nincs szükség „natív" kommunikációs technológia támogatására a két platformon. Mindkét félnek csupán az üzenetsorral kell tudniuk kommunikálni, és az üzenetsorok gyakran több kommunikációs protokollt is támogatnak az üzenetek fogadásához. Másrészt, ugyan az üzenet is rendelkezik kötött formátummal, ez a megkötés mégis kisebb, mintha közvetlen a másik fél API-ját hívnánk. Ezáltal a kommunikáló felek szabadabban fejlődhetnek, változhatnak, mert lazul a köztük levő csatolás. Harmadrészt, az üzenetsor a rendelkezésre állási problémákat is át tudja hidalni, hiszen az üzenet tárolható, és később is kézbesíthető. Az alábbi két szemlélteti az üzenetsorral kialakított architektúrát.



Forrás: https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html

Az MQTT protokoll ilyen üzentsorokra épül. Publish-subscribe elven működik. Topic-okba tudunk üzenetet küldeni, és az erre feliratkozott kliensek az üzenetet megkapják. A topic-ok névtere hierarchikus, és a feliratkozás nemcsak konkrét topic-ra, hanem topic-ok csoportjára is könnyen megadható helyettesítő karakterekkel. A + pontosan egy komponenst jelöl a névtérben, a # pedig az összes további komponenst lefedi. Pl. ha /sensors/device_4/temp formában vannak a topic-ok, a következő módon használhatók a helyettesítő karakterek:

- /sensors/+/temp: bárki által beküldött hőmérséklet
- /sensors/device_4/+: négyes eszköz mérései

- /sensors/#: összes szenzor összes mérése

A protokoll egyszerű és pehelysúlyú, ideális alacsony sávszélességű, nagy késleltetésű, megbízhatatlan hálózatokhoz. TCP/IP protokoll felett az 1883-as és a 8883-as (SSL) porton működik alapértelmezésben. A garantált kézbesítés foka hangolható három szinten:

- QoS 0: Egyszeri kézbesítés, nincs nyugta
- QoS 1: Legalább egyszeri kézbesítés, egyszerű nyugta
- QoS 2: Pontosan egyszeri kézbesítés, összetett nyugtázási folyamat

A QoS szinttel a késleltetés és a sávszélesség-igény növekszik. Opcionálisan használhatunk tartós üzeneteket. Ez azt jelenti, hogy a topic-ra újonnan feliratkozó fél utólag is megkapja a legutolsó üzenetet. Lehetséges a kapcsolatbontások közt is megtartani a feliratkozást, és utólag megkapni az üzeneteket. Az ún. will üzenetek pedig támogatják, hogy a feliratkozók értesüljenek a publikáló elérhetetlenségéről. Ha sok ideig nem érkezik üzenet, akkor nem tudhatnánk, hogy csupán nem történik mérés, vagy a publikáló szenzor épp nem érhető el. A will üzenetet a szenzor állítja be az üzenetsorhoz történő kapcsolódáskor, és a kapcsolat megszakadásakor az üzenetsor a feliratkozóknak továbbítja azt.

Az MQTT protokollnak két további változata is elterjedt. Az MQTT-SN még korlátozottabb erőforrású környezetekhez készült. Az MQTT protokoll TCP felett működik, és ez a kapcsolat folyamatos fenntartását igényli. Az MQTT-SN lehetővé teszi a protokoll használatát datagramalapú (pl. UDP) protokollok felett is. Az MQTT-SN továbbá a kapcsolat felépítését is három fázisra bontja, hogy kis méretű üzenetekkel is felépíthető legyen a kapcsolat. A kapcsolat felépítésének első fázisa kötelező, a második kettőben az opcionális will topic és üzenet adható meg. További újdonság, hogy a topic-ok regisztrálhatók, és kétbájtos indexszet kaphatnak, amelyet a hosszú név helyett használhatunk. Ezzel tovább csökken a hálózati forgalom. Az MQTT-SN megengedi a perzisztens will topic-ot és üzenetet is, valamit alvó módot és hálózati felderítést is támogat.

Az MQTT over WebSocket azért jött létre, hogy böngészőben futó kliensoldali programok is küldhessenek MQTT üzeneteket. Mivel a hagyományos MQTT kommunikáció TCP protokoll felett történik, böngészőből nem használható, ugyanis JavaScript kódból nincs mód TCP kapcsolat létrehozására. A WebSocket protokollt a HTTP korábban említett korláta miatt vezették be, vagyis a csak kliens által kezdeményezhető kommunikáció miatt. A WebSocket segítségével a JavaScript programunkból kétirányú kommunikációt tudunk megvalósítani a szerverrel, nincs szükség pollozásra. A WebSocketet minden HTML 5-kompatibilis böngészőnek támogatnia kell, ezért kézenfekvő, hogy TCP helyett ezt használjuk az MQTT kommunikációhoz.

8 VALÓSIDEJŰ ADATFELDOLGOZÁS

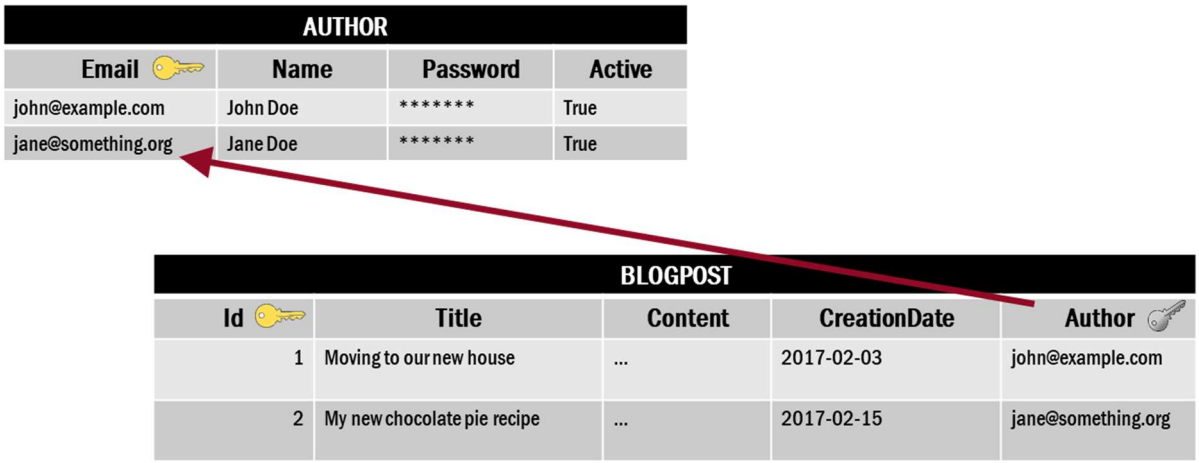
A valósidejű adatfeldolgozáshoz használt megoldások azon az elven alapulnak, hogy olyan adatbázisstruktúrát alkalmazunk, amelyekből később az adat hatékonyan kinyerhető. Tipikusan NoSQL [9] adatbázisokat alkalmazunk BigData alkalamzásokban. A NoSQL a „Not only SQL” rövidítése, azonban az elnevezés félrevezető, mert a NoSQL megoldások nem az SQL lekérdezőnyelvet, hanem a relációs

sémát hivatottak leváltani. Ebben a fejezetben áttekintjük a relációs adatbázisok hátrányait, majd megvizsgáljuk, hogyan reagálnak ezekre a NoSQL adatbázisok.

8.1 A relációs adatbázisok hátrányai

A relációs sémával kapcsolatban két problémát azonosíthatunk. Az első probléma a relációs adatbázisok túl merev és kötött sémája. Ezekben az adatbázisokban a tábla definíciója rögzíti a sémát, és ettől eltérni nem lehet. Az utólagos változtatás vagy bővítés bonyolult. Az IoT alkalmazásoknál jellemző sokszínű és változatos adatokkal ez a megközelítés nehezen használható, mert vagy sok táblát kell létrehoznunk a különböző adatokhoz, vagy egy nagy táblában próbálunk sok különböző dolgot eltárolni, és ekkor sok nem értelmezhető adat helyén NULL kell álljon. A kötött sémának ugyanakkor előnye is van, ugyanis kikényszeríti az adatok helyes formátumát.

A relációs adatbázisokkal kapcsolatban felmerülő második probléma a kapcsolódó adatokat érinti. A relációs modellben a kapcsolat nem elemi fogalom, hanem az adatok közti kapcsolódást kulcshivatkozással valósítjuk meg. Több-több jellegű kapcsolatnál ráadásul külön kapcsolótáblára is szükség van, amely az összetartozó sorok kulcspárjait tárolja. Az alábbi ábra egy blogalkalmazás bejegyzései és szerzői közti hivatkozásokat szemlélteti.



AUTHOR

Email	Name	Password	Active
john@example.com	John Doe	*******	True
jane@something.org	Jane Doe	*******	True

BLOGPOST

Id	Title	Content	CreationDate	Author
1	Moving to our new house	...	2017-02-03	john@example.com
2	My new chocolate pie recipe	...	2017-02-15	jane@something.org

A hivatkozások mentén a kapcsolódó sorok egyesíthetők a join művelet segítségével. A join művelet azonban igen költséges, ugyanis az összekapcsolandó táblák soraiból először Descartes-szorzat képződik, majd ezen az eredményhalmazon végzi el az adatbáziskezelő a kulcshivatkozás szerinti szűrést. Ha kettőnél több táblát kell összekapcsolni, akkor pedig többszörösen érvényesül ez a jelenség, és ez rontja a teljesítményt és a skálázódást.

8.2 A NoSQL adatbázisok

A NoSQL adatbázisok igen sokfélék lehetnek, de általánosan a következő célkitűzésekkel rendelkeznek:

- Szolgálják ki a BigData alkalmazásokat
    - Volume: nagy mennyiségű adatot tudjanak tárolni
    - Velocity: képesek legyenek a gyorsan érkező adatokat elmenteni
    - Variety: támogassák az adatok változatosságát (vö. kötött séma)
- Tipikusan nincs kötött sémájuk, vagy opcionális

- Elkerülik a join jelenséget

Minden adatbáziskezelő-rendszer a CAP-elmélet szerinti korlátokon belül tud csak működni. A CAP elmélet kimondja, hogy az alábbi három követelmény közül legfeljebb csak kettőt tudunk garantálni:

- Consisteny: Konzisztencia, vagyis minden olvasás a legfrissebb adatot adja, vagy hibát jelez.
- Availability: Rendelkezésre állás, vagyis minden kérésre hibamentes válasz érkezik.
- Partitioning: Partícionálhatóság, vagyis a rendszer akkor is működik, ha a csomópontok közt elvesznek (késnek) az üzenetek.

Az elosztott működés a skálázhatóság miatt elengedhetetlen, ezért a konzisztencia és a rendelkezésre állást közt kell választanunk. Az ACID működésű adatbázisok a konzisztenciát választják. A konkurenciát tranzakciókkal kezelik, és ha ütközés lép fel, akkor visszagörgetik a tranzakciót. Ezzel biztosítható a konzisztencia. Idesorolhatók a hagyományos relációs adatbázisok, de néhány NoSQL adatbázis is. A NoSQL adatbázisok azonban döntően BASE elvűek (Basically Available, Soft State, Eventual Consistency). Ezeknél a csomópontokon az adatok nem minden pillanatban konzisztensek, de idővel azok lesznek. A NoSQL adatbázisoknál gyakran alkalmazzák a sharding technikát. Ez azt jelenti, hogy az adatok valamilyen kulcs szerint szét vannak válogatva a csomópontok közt. Így ha mindig a kijelölt csomópontot szólítjuk meg, akkor a legfrissebb adatot kapjuk. Ezzel a technikával jól eloszthető a működés, és a MapReduce modell is jól működik vele.

A NoSQL adatbázisok négy nagy csoportba oszthatók:

- Kulcs-érték tárak
- Dokumentumtárolók
- Oszlopcsaládok
- Gráfadatbázisok

A kulcs-érték tárak egy nagy asszociatív tömbhöz hasonlíthatók, vagyis kulcsokhoz tartozó értékeket tartunk bennük. Az érték általában összetett adattípus is lehet, pl. további asszociatív tömb, lista, halmaz, rendezett halmaz stb. Ennek a típusnak az egyik legelterjedtebb képviselője a Redis.

A dokumentumtárak tárolási egysége a dokumentum, amely kulcs-érték párok sorozata. Az érték maga is újabb dokumentum lehet. A dokumentum vizualizálható XML, JSON, YAML vagy ezekhez hasonló formátumban, azonban a dokumentumtárak nem rendelkeznek kötött dokumentumsémával. A kulcsok tetszőlegesek lehetnek. Így az objektumorientált adatok könnyen leképezhetők. A dokumentumtárak tulajdonképpen tekinthetők specializált kulcs-érték táraknak, amelyek gazdagabb API-val rendelkeznek, és jobban támogatják a dokumentumon belüli navigálást, lekérdezést. Ennek az adatbázistípusnak a legjelentősebb képviselője a MongoDB és a CouchDB.

Az oszlopcsaládok első ránézésre nagyon hasonlítanak a relációs adatbázisokhoz, de a táblákban soronként változhat az alkalmazott oszlopok száma. Tulajdonképpen egy sor egy kulcshoz rendelt kulcs-érték párok sorozatának felel meg. Ebben a tekintetben a kulcs-érték tárakhoz és a dokumentumtárak is hasonlítanak. Néhány oszlopcsalád „szupersorokat” (super column) is támogat. Ezekben az osztlopokban maguk az adatok is további kulcs-érték párok. Az oszlopcsaládok lényegi különbsége a relációs adatbázisokhoz képest a tárolás módjában rejlik. A relációs adatbázisok kötött sémája meghatározza az

egyes sorok méretét a táblában, és a sorok tárolása is egymás után, azonos méretű egységekben történik. Ez megnehezíti a sorok későbbi felvitelét vagy törlését, hiszen az eddigi adatokat is mozgatni szükséges. Az oszlopcsaládok olyan struktúrát alkalmaznak, hogy az oszlopok hatékonyan variálhatók legyenek. Relációs adatbázisokban csak úgy tudnánk változatos oszlopokat használni a táblában levő sorokhoz, ha NULL értékekkel töltjük ki a nem értelmezett helyeket. Ez viszont felesleges helyet foglal, és lekérdezéskor rontja az olvashatóságot. Az oszlopcsaládok kiküszöbölik ezt a nehézséget, és minden sorra csak a rájuk értelmezett oszlopok jelennek meg. A legimsertebb oszlopcsaládok a Cassandra és az HBase.

A gráfadabázisok nem annyira a relációs adatbázisok kötött sémájának lazítására, hanem a kapcsolatok támogatására fókuszálnak. Gráfokban elemi fogalom az él, és konstans lépéssel bejárható. Általában csomópontokra és élekre is tehetünk tulajdonságokat, amelyek kulcs-érték párok sorozataként jelennek meg. Konkrétan ezt a megközelítést property graph modellnek nevezzük. Ilyen elven működik az egyik legelterjedtebb gráfadatbázis, a Neo4j is. Előfordulnak olyan gráfadatbázisok is, amelyek hipergráfokat is tudnak tárolni. Hipergráfokban egy él kettőnél több csomópontot is összeköthet.

8.3 Az adatmodell kiválasztása

Az adatbázis típusát és a konkrét adatbázist mindig a feladathoz szükséges megválasztani. Ennek ellenére megállapíthatók bizonyos „best practices”-ek, pl.:

- Hagyományos pénzügyi adatok, ahol a kötött séma és a konzisztencia fontos → relációs
- Termékkatalógushoz, ahol az egyes termékek konkrét jellemzői változatosak →dokumentumtár
- Ajánlórendszer sok kapcsolattal → gráf
- Változatos szenzoradatok → oszlopcsalád
- Bevásárlókocsi → kulcs-érték

Egyre elterjedtebb az ún. polyglot perzisztenciát alkalmazó megközelítés is. Ez azt jelenti, hogy ugyanazon alkalmazáson belül különböző feladatokra más-más adatbázist használunk. Így lehetséges az egyes részfeladatokhoz praktikusabb technológiát használni. Ez azonban növeli a komplexitást, mert minden technológiához külön library szükséges, és ha kereszthivatkozások vannak a különböző adatbázisok közt, ott kézzel kell biztosítanunk az integritást. A polyglot perzisztenciát elősegítendő, ún. multi-model adatbázisok is megjelentek, amelyek több adatmodellt is támogatnak. Ilyen pl. az ArangoDB.

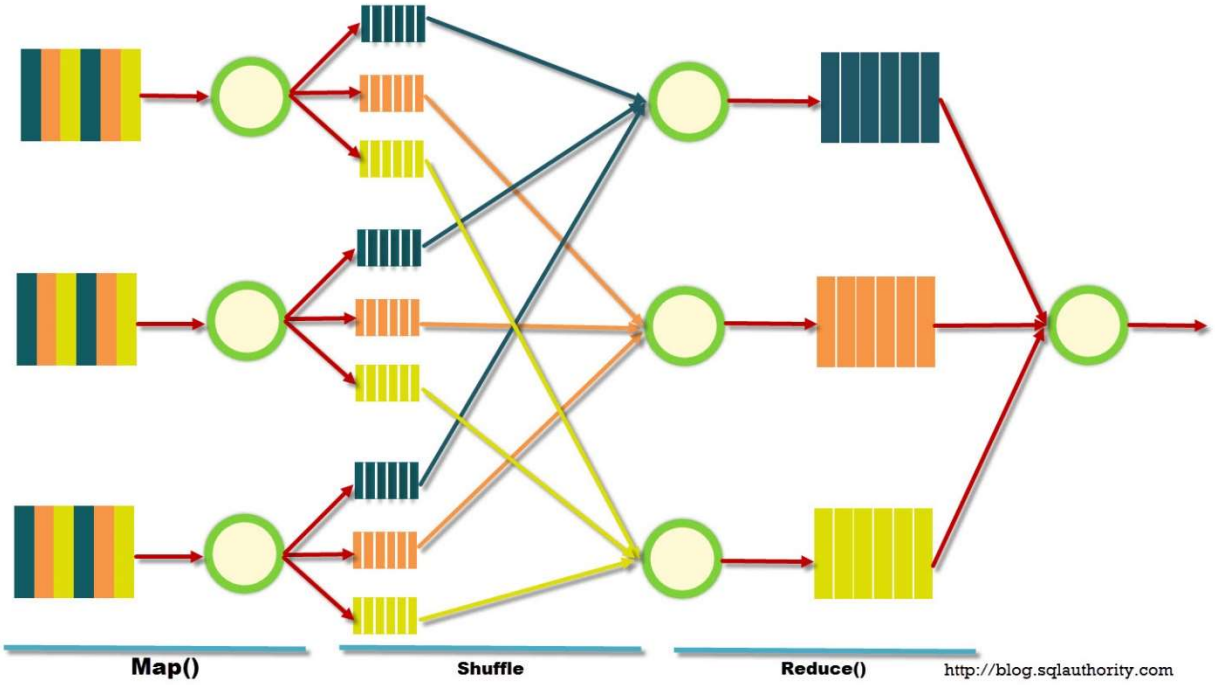
9 UTÓLAGOS ADATFELDOLGOZÁS

Utólagos adatfeldolgozáshoz elterjedt megoldás MapReduce [10] modell használata. A MapReduce egy programozási modell, amelyet a Google fejlesztett ki kimondottan párhuzamosításra. A Google a webről készült teljes indexét újraépítette ezen az elven. A MapReduce modell kulcs-érték párok feldolgozását támogatja három lépésen keresztül:

1. Map: feltételek szerinti szűrés
2. Shuffle: kulcsok szerinti csoportosítás

3. Reduce: csoportok összegzése

A lépésekre szekvenciálisan gondolunk, de valójában átlapolódhatnak. A következő ábra szemlélteti a feldolgozás menetét:



Forrás: https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/

A MapReduce modell a következő előnyökkel rendelkezik:

1. Nem szükséges hozzá célhardver, hétköznapi hardveren is futhat.
2. Csak a map és a reduce programszéket szükséges megírnunk a feldolgozáshoz, a többit a keretrendszer nyújtja.
3. A map művelet bemenetenként független számítható, így párhuzamosítható.
4. A reduce művelet csoportonként független számítható, így párhuzamosítható.
5. Ha kiesik egy feldolgozási csomópont, akkor csak az ő munkáját kell pótolni, ezért a rendszer hibatűrő. Ehhez viszont elosztott fájlrendszeren szükséges tárolni a részeredményeket.

A MapReduce számos hátránnyal is rendelkezik:

1. Alacsony szintű modell, nem használ ki semmilyen adatsémát, sem a séma használatából adódó előnyöket.
2. Nem illeszkedik az emberi gondolkodásmódhoz. A lekérdezéseinket át kell fogalmazni a MapReduce modelljére, és ez a modellben jártasságot, szakértelmet igényel.
3. Ugyan a shuffle fázist nem kell megírnunk, a tervezés során figyelembe kell vennünk, mert a hatékonysága az adatok jellegétől függően jelentősen változik. A reduce művelethez ugyanis rendezés szükséges, és ez elosztott környezetben nemlineáris komplexitású.

4. A hibatűréshez a részeredményeket elosztott tárba kell tenni, ez pedig jelentős overheadet jelent.
5. Nem támogat iteratív, ismétlődő feldolgozást, csak teljes lekérdezések futtatását.

Összegzésképp azt mondhatjuk, hogy a MapReduce önmagában nem hatékony, és nem is intuitív és kényelmes, de ereje a párhuzamosíthatóságban rejlik. A párhuzamosítás segítségével igen nagy adathalmazok is kezelhetők, ráadásul célhardver sem szükséges hozzá. A MapReduce modellt gyakran alkalmazzák mintaillesztésre, rendezésre, logok elemzésére, indexépítésre, gépi tanulásra, valamint statisztikaalapú gépi fordításra.

A MapReduce elterjedt és nyílt forrású megvalósítása az Apache Hadoop [11]. A Hadoop modularizált szoftver, négy fő komponense van:

1. Hadoop Common: a közös részek, amelyek a többi modulnak alapszolgáltatásokat nyújtanak
2. Hadoop Distributed File System (HDFS): a Hadoop hibatűrő, elosztott fájlrendszere
3. Hadoop YARN: munkaütemező és erőforrásmenedzser
4. Hadoop MapReduce: a MapReduce modell megvalósítása

A Hadoophoz további keretrendszerek és modulok érhetők el, amelyek többletszolgáltatásokat nyújtanak. A Hadoop Javában íródott, de az algoritmusaink map és reduce részeit több programozási nyelven is elkészíthetjük.

Az alábbi példakódokkal egy Hadooppal elkészített lekérdezést szemléltetünk. Feltesszük, hogy személyek születési dátumait és béreit kapjuk meg soronként a következő formátumban:

```
1945/7/29 269571
```

Forráskód: A bemenet formátuma

A feladatunk korcsoportonként (<20, 20-30, 30-40, 40-50 és >50 éves) kiszámítani az átlagfizetést. A megoldás elve, hogy a map fázisban a korcsoport szerint kulcsot rendelünk a beolvasott fizetésekhez, majd a shuffle az azonos korcsoportok bejegyzéseit azonos reducer processznek adja át. A reduce fázisban ezért a kapott bemenetek közt csupán átlagot számítunk.

Az alábbi kód szemléleti a map fázis megvalósítását. Látható, hogy a Hadoop Mapper osztályától kell örökölnünk, és a map megvalósítása a map metódusba kerül.

```
import java.io.IOException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeFormatterBuilder;
import java.time.temporal.ChronoField;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MapClass extends Mapper<Text, Text, Text, LongWritable> {

     private static DateTimeFormatter formatter;
```

```
	static {
		DateTimeFormatterBuilder dfb = new DateTimeFormatterBuilder();
		dfb.appendValue(ChronoField.YEAR).appendLiteral('/')
			.appendValue(ChronoField.MONTH_OF_YEAR)
			.appendLiteral('/')
			.appendValue(ChronoField.DAY_OF_MONTH);
		formatter = dfb.toFormatter();
	}

	private String yearsToKeyLabel(int y) {
		if (y < 20)
			return "<20";
		else if (y < 30)
			return "20-30";
		else if (y < 40)
			return "30-40";
		else if (y < 50)
			return "40-50";
		else
			return ">50";
	}

	@Override
	protected void map(Text key, Text value, Context context)
			throws IOException, InterruptedException {
		String[] fields = key.toString().split(" ");
		if (fields.length == 2) {
			LocalDate date = LocalDate.parse(fields[0], formatter);
			int years = date.until(LocalDate.now()).getYears();
			context.write(new Text(yearsToKeyLabel(years)),
				new LongWritable(Long.parseLong(fields[1])));
		}
	}
```

Forráskód: A korcsoportonkénti átlagfizetést számító mapper

Ezután a reduce fázisban az átlagszámítás már szinte triviális feladat. A Reducer osztálytól kell örökölnünk, és a reduce metódus megvalósítása szükséges.

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReduceClass extends Reducer<Text, LongWritable, Text, LongWritable> {

	@Override
	protected void reduce(Text key, Iterable<LongWritable> values, Context
context)
			throws IOException, InterruptedException {

		int sum = 0;
		int count = 0;
		for (LongWritable val : values) {
```

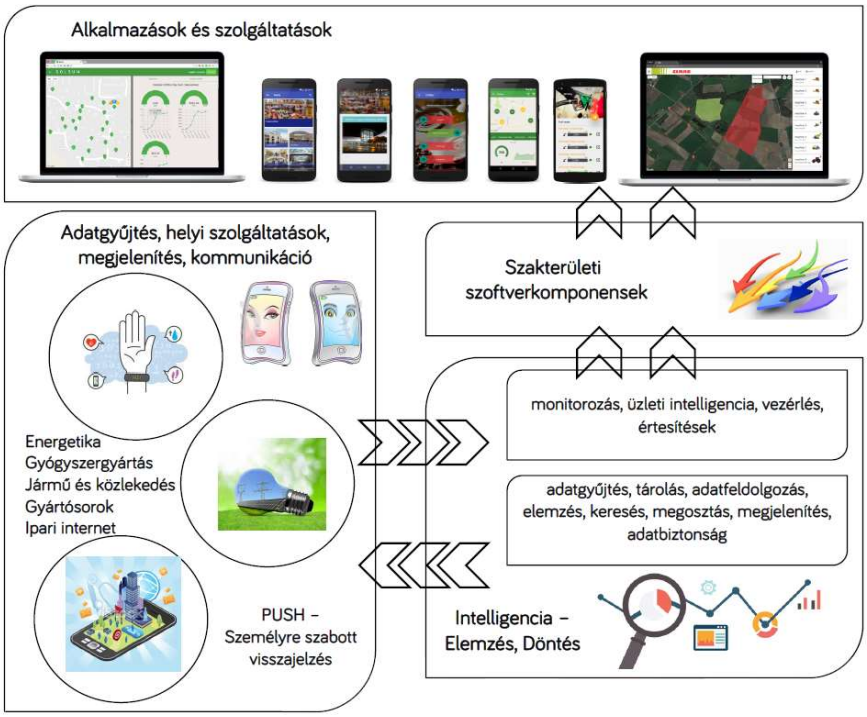
```
                    sum += val.get();
                    count++;
                }
                long avg = (long)(sum / (double)count);

                context.write(key, new LongWritable(avg));
        }
}
```

Forráskód: A korcsoportonkénti átlagfizetést számító reduce

10 Saját IoT Platform

A saját IoT platformunk a SensorHUB keretrendszer. A SensorHUB képességeinek is számos alkalmazási területe lehet, például az egészségügy, okos városok, térinformatika, járműipar, gyártósorok stb. Ezek tipikusan olyan területek, ahol nagy mennyiségű (szenzor)adatok feldolgozása, tárolása, elemzése és felhasználása a feladat.



Ábra: A SensorHUB koncepció

A SensorHUB koncepció elemei:

- Szenzorok, adatok gyűjtése, helyi szolgáltatások, megjelenítés és adatátvitel.
- Felhő alapú háttérrendszer, Big Data menedzsment funkciókkal.
- Szakterület specifikus szoftver komponensek.

- Alkalmazások, szolgáltatások, üzleti intelligencia jelentések és elemzések.

A SensorHUB koncepciónak előnye, hogy a gyakran előforduló, adatokkal kapcsolatos műveleteket csak egyszer kell implementálni és ezek után megfelelő dokumentációkkal támogatva bárki tud olyan alkalmazást készíteni, ami felhasználva a SensorHUB keretrendszert, képes lesz adatai megjelenítésre, feltöltésére, elemzésére.

A SensorHUB keretrendszer a SensorHUB koncepció realizálása. Az implementálás során cél volt a koncepcióban definiáltak megvalósítása és a koncepció minél több szakterületre történő kiterjeszthetősége.

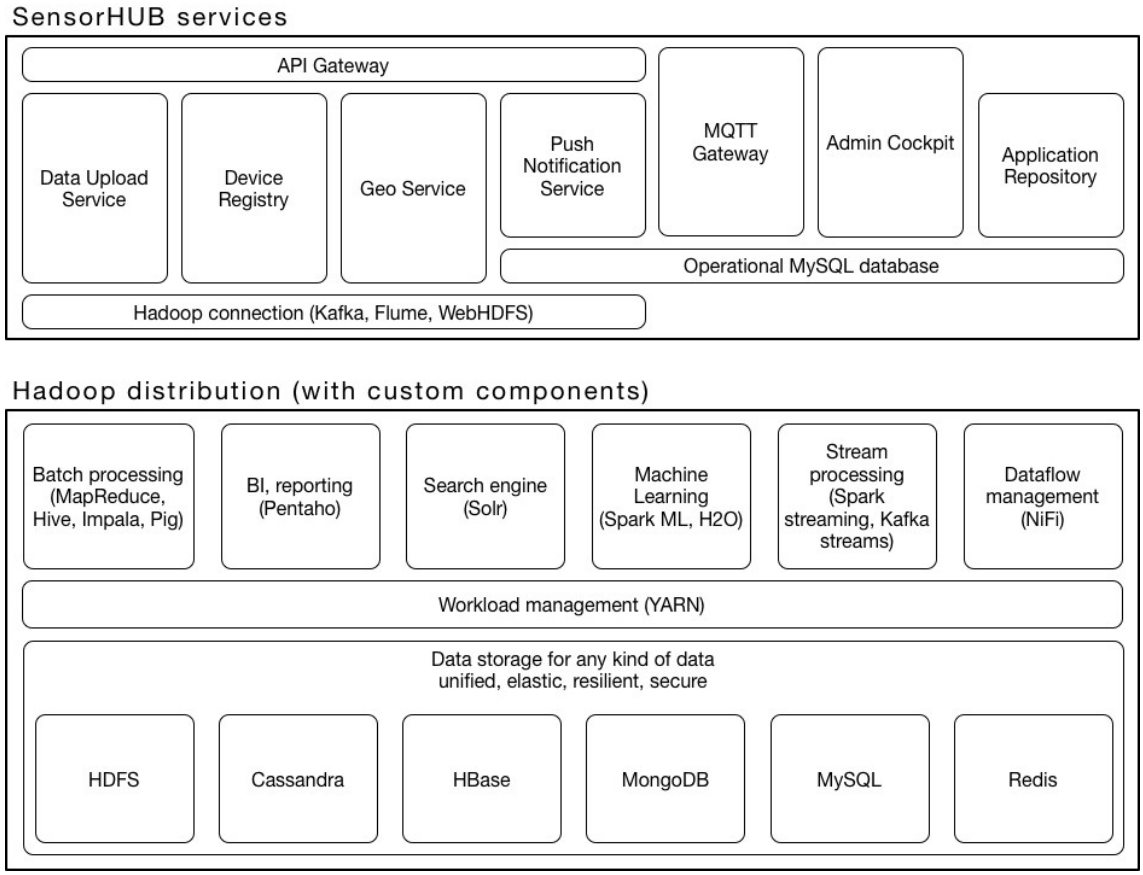
A keretrendszer alapja a nagy mennyiségű adat kezelésére képes Big Data elosztott fájlrendszer, tudni kell ebbe betölteni és ebből kiolvasni az adatokat, minél több interfészt támogatva. A cél ugyanaz, minél több, hasznos funkciót és technikát egyetlen egyszer jól megírni úgy, hogy ezek minél több alkalmazásba beintegrálhatóak legyenek a későbbiekben.

Az utólag SensorHUB 1.0 névre keresztelt változata a keretrendszernek, még egy egyszerűbb, szűkebb infrastruktúrát tartalmazott. Ebben a változatban a fő irányvonal az okostelefonos alkalmazások integrációja volt a keretrendszerbe és a kommunikáció megvalósítása a szerver oldali Hadoop alapú Big Data adattárházzal.

A SensorHUB 1.0 egyik eleme egy osztálykönyvtár volt, melyben a kommunikációval, megjelenítéssel és adatkonverzióval kapcsolatos metódusok és hálózati hívások kerültek megvalósításra. Ezek a funkciói az osztálykönyvtárnak, például egy Android alkalmazás projektjébe történt beimportálást követően elérhetővé válnak a kódból és hasznosíthatóak a megírt kódrészletek. Másik fő eleme pedig a mobil alkalmazás és a Hadoop oldal közti kapcsolatot megteremtő szerver oldali komponens (mediator agent), mely a Hadoop felé adat előfeldolgozást, az okostelefonos kliens felé pedig egy puffert és adatlekérési felületet biztosít.

A SensorHUB 2.0 az előző verzióhoz képest egy teljesen újragondolt változatot jelent a megvalósítás során. Az eddigi Cloudera Hadoop Big Data alapú megvalósítás mellett nagyobb hangsúlyt kapott a szerver oldali komponens, ugyanis bevezetésre került egy Java és Node.js alapú mikroszolgáltatás réteg, mely összekapcsolja a kliens oldali alkalmazásokat a Hadoop klaszterrel, valamint adminisztrációs, menedzsment, terheléselosztó (replikációs) és kommunikációs feladatokat is ellát.

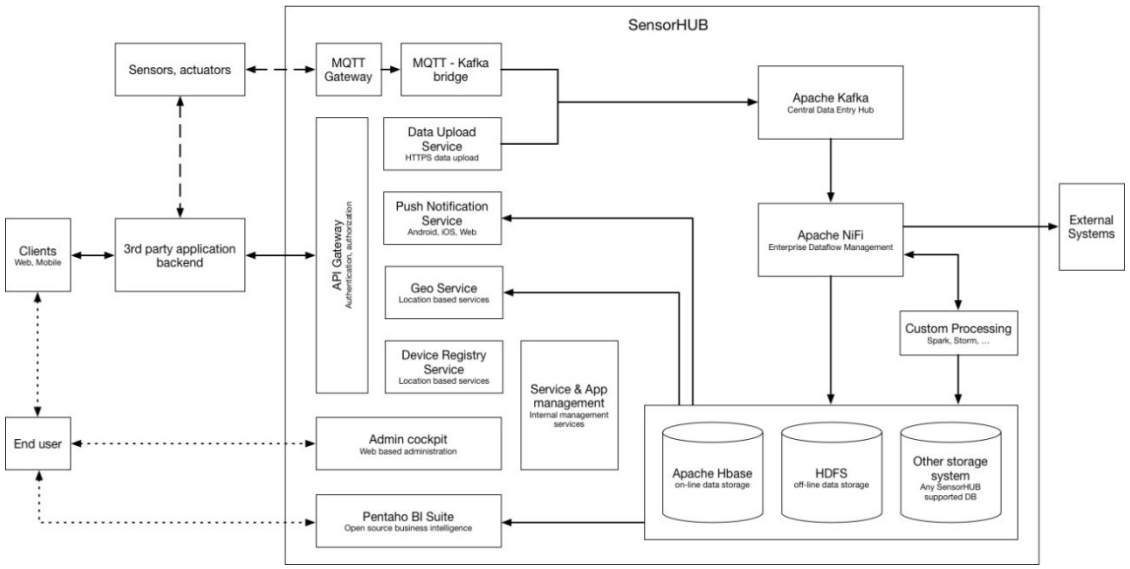
A SensorHUB 3.0-ás verziójában lecserélésre került az eddigi Hadoop disztribúció a Hortonworks által készített platformra, valamint számos további egyedi mikroszolgáltatás szintű, illetve adminisztrációs fejlesztés került bele. **Hiba! A hivatkozási forrás nem található.** Az új verzióban nagyobb szerepet kapott az időközben kurrenssé vált további Big Data implementációk kihasználása is, valamint az egyedi programcsomagok integrálása az adatfolyamba.



Ábra: A SensorHUB technológiai felépítése

A technológiai felépítésből látható, hogy a SensorHUB két fő komponensből áll:

- Hadoop réteg: ami a Hortonworks Hadoop disztribúcióját és a kapcsolódó további Big Data implementációkat takarja.
- SensorHUB szolgáltatás réteg: Mikroszolgáltatás komponensek, amik a SensorHUB egyediségét adják, itt találhatóak a gyakran használt szolgáltatások implementációi, menedzsment funkciók, adatkommunikációs interfészek.



Ábra: SensorHUB architektúra

A SensorHUB architektúrájából látható, hogy itt már az okostelefonos klienseken kívül webes kliensek, illetve „natív" szenzor komponensek is kapcsolódhatnak a rendszerhez. Az architektúra ábra SensorHUB dobozon kívüli része az alkalmazásfüggő rész, vagyis maguk a kliensek, és a hozzájuk kapcsolódó szerver oldali komponensek. Az ábra SensorHUB dobozon belüli része pedig a mikroszolgáltatás rész, illetve a Hadoop rész (a három tárolót szemléltető „henger ábrával": Apache HBase, HDFS, egyéb tárolók).

A szenzorok tehát akár egy saját szerver komponensen keresztül, akár közvetlenül, egy erre felkészített MQTT (Message Queuing Telemetry Transport) alapú közzétesz-feliratkozik (publish-subscribe) típusú interfészhez kapcsolódnak a SensorHUB bemeneti felületén. Ez egy egyszerű, hat metódussal rendelkező hálózati protokoll, mely minimális feszültség és minimális hálózati erőforrást vesz igénybe a kommunikációhoz. Emiatt gyakran használják IoT alapú szenzor adatok átviteléhez.

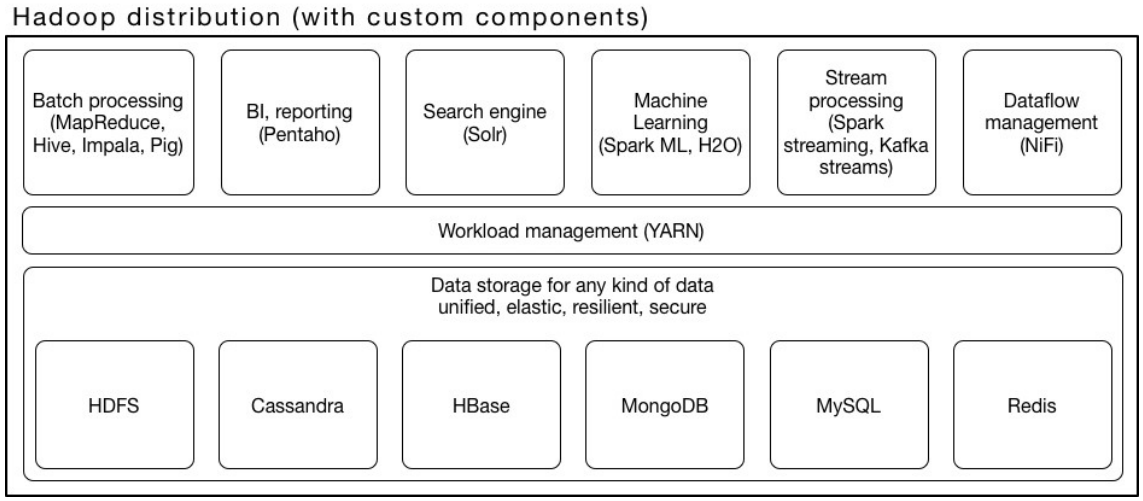
A felhasználók pedig elérik a webes, illetve mobil klienst, melyek szintén saját szerver komponensen keresztül kapcsolódnak a SensorHUB keretrendszerhez. Hozzáférnek továbbá a SensorHUB webes adminisztrációs felületéhez (Admin cockpit), ahol tájékozódni lehet a felhasználók, alkalmazások és a Hadoop fájlrendszer állapotáról, metrikákról, lehetőség van figyelni az erőforrás kihasználtságokat. További opció van az alkalmazások adatáramlásának letiltására vagy engedélyezésére.

Valamint hozzáférnek az üzleti intelligencia és riportkészítő funkciókat támogató rendszerhez (Pentaho BI Suite), ami a különböző adattárolókból tudja feldogozni az ott található adathalmazt ETL (Extract-Transform-Load) folyamatokkal, mely során konvertálhatjuk, szűrhetjük, tisztíthatjuk az adatainkat, hogy a megjelenítés vagy további felhasználás ellentmondás-mentes legyen. Ezt megtehetjük például valamelyik SQL-szerű lekérdezőnyelv segítségével. Így készülhetnek el azok a részadathalmazok, melyekre a riportnak, vagy a felhasználónak szüksége van.

A további komponensek bemutatására a következő alfejezetekben kerül sorra.

10.1 Hadoop réteg

Mint említettem, a SensorHUB 3.0 már a Hortonworks Hadoop disztribúcióját és a hozzá kapcsolódó további Big Data implementációkat használja. Maga a disztribúció két külön rendszert takar a Data Platform-ot (HDP), illetve a Data Flow-t (HDF). Miután a SensorHUB támogatja a valós idejű és a kötegelt szolgáltatásokat is, ezért mindkét rendszerre szükség van.

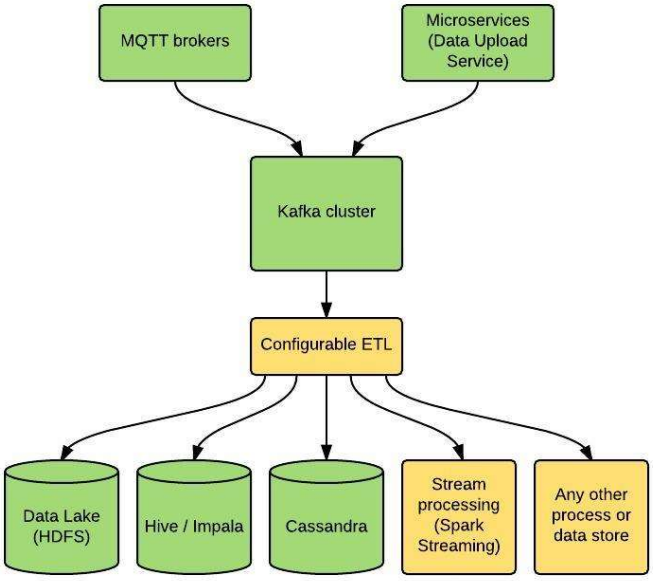


Ábra: SensorHUB Hadoop réteg

A SensorHUB-ban használt és eddig még nem ismertetett komponensek:

- Hive: A Facebook által készített SQL alapú lekérdező nyelv a Hadoop rendszerben, segítségével az SQL-ből ismert kulcsszavak felhasználásával nyerhetünk ki adatokat.
- Impala: A Hive-hoz hasonló, de annál gyorsabb, a Cloudera által készített C++ alapú lekérdező nyelv.
- Pig: SQL-re hasonlító szkriptnyelv, MapReduce feladatok létrehozására, nevét onnan kapta, hogy szinte bármit leírhatunk benne, mert bármit „megeszik”.
- Solr: Elosztott architektúrájú kereső szolgáltatás, támogatja a szöveges adatok különböző formátumok szerinti indexelését és kereshetőségét.
- Spark ML, H2O: Gépi tanulás algoritmusok megírását és futtatását támogató komponensek.
- Spark Streaming: A Spark valósidejű, adatfolyam alapú komponense.
- Cassandra: Elosztott, skálázható, hibatűrő, nyílt forráskódú oszlop (oszlopcsalád) alapú tárolást megvalósító NoSQL adatbázis.
- HBase: Elosztott, hibatűrő, nyílt forráskódú, NoSQL adatbázis.
- MongoDB: Elosztott, skálázható, hibatűrő, nyílt forráskódú, dokumentum alapú tárolást megvalósító NoSQL adatbázis.
- MySQL: Klasszikus SQL-alapú relációs adatbázis kezelő rendszer.

- Redis: Nyílt forráskódú, memória alapú, hibatűrő, kulcs-értékpárokat tároló NoSQL adatbázis.

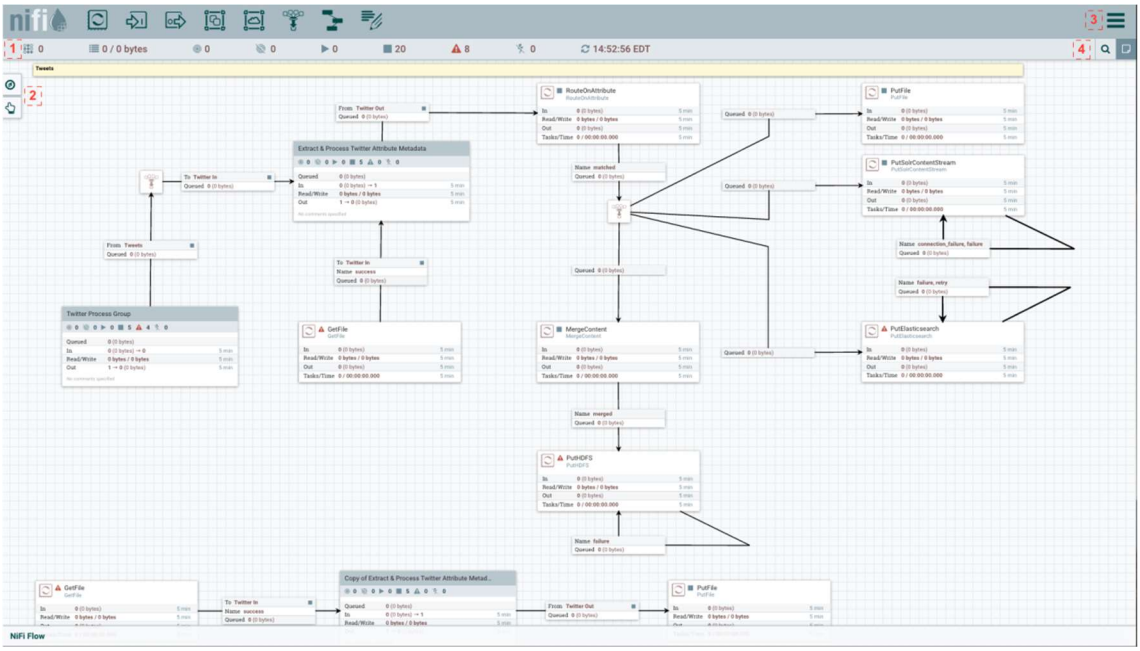


Ábra: Kafka komponens kapcsolódásai

Ahogy a SensorHUB architektúrája mutatja a két legfőbb komponens az Apache Kafka és az Apache NiFi, melyeken minden bemenő adat áthalad.

A terhelhetőséget biztosítva a Kafka komponens több példányban, skálázható módon fut, konténer alapú architektúrában Docker alatt, Kubernetes üzemeltető rendszerrel. Ennek köszönhetően automatikusan tud skálázódni a futó példányok száma attól függően, hogy a jelenlegi terhelés függvényében mennyi erőforrásra van szükség.

Az előző ábrán szereplő Configurable ETL pedig a már említett Apache NiFi adatfolyam menedzselő rendszer.



Ábra: Példa Apache NiFi adafolyam a felhasználói felületen

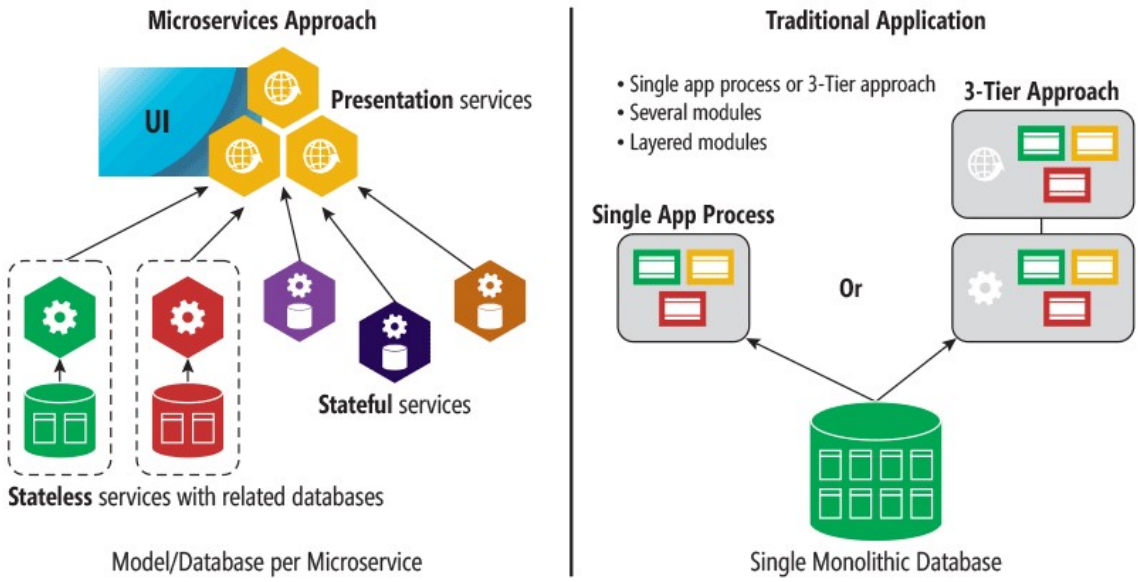
10.2 SensorHUB szolgáltatás réteg

Következik a réteg, amitől SensorHUB a SensorHUB, vagyis a szolgáltatás réteg bemutatása a technológiai felépítésből. Ebben a rétegben kerültek megvalósításra a közös, gyakran használt funkciók, melyek beépíthetőek az alkalmazásokba, valamint ez a réteg működik közre az adatáramlásban a Hadoop és az alkalmazások között.

Már többször hivatkoztam a mikroszolgáltatásra, illetve a mikroszolgáltatás architektúrára, de még nem fejtettem ki a jelentésüket.

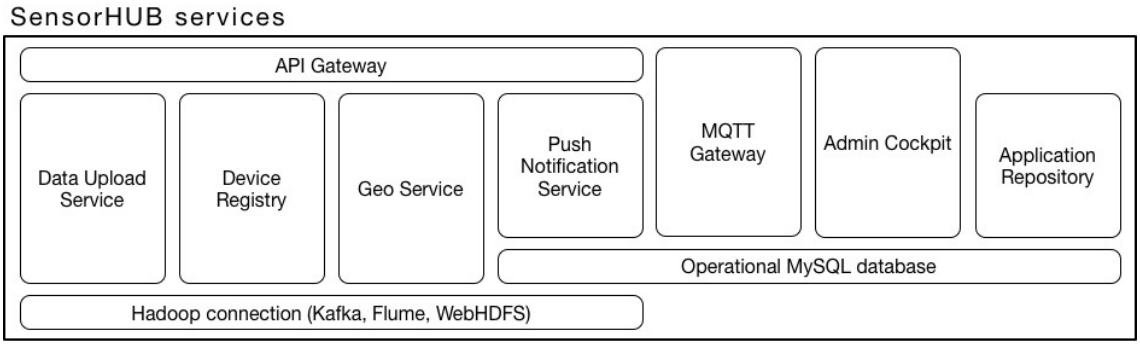
Egy mikroszolgáltatás egy konkrét üzleti vagy felhasználói funkciót valósít meg, saját verziózása van, önmagában telepíthető, frissíthető. Jól meghatározott interfésze van, műveletei szabványos protokollokon keresztül elérhetők. Saját egyedi címével címezhető, hiba esetén is konzisztens és elérhető marad. Folyamatosan adatot szolgáltat diagnosztikával és a működőképességével kapcsolatban (heartbeat). Az egyes szolgáltatások saját adatbázist használhatnak, de nem minden mikroszolgáltatás tárol adatot. Ennek tekintetében megkülönböztetünk állapotot tároló és állapotmentes mikroszolgáltatásokat is.

A mikroszolgáltatás architektúra ilyen mikroszolgáltatások egymással kommunikáló, skálázható hálózata, ahol az egyes komponensek különböző technológiákat, eszközöket használhatnak, és különböző platformokon futhatnak. A komponensek lazán csatoltsága miatt könnyű egy újabbat elkészíteni, és elérhetővé tenni a rendszerben, ezáltal, a szükséges komponensek építőkockaként tehetőek össze az alkalmazásunkhoz.



Ábra: A mikroszolgáltatás és a monolitikus architektúra összehasonlítása

A rétegben található komponensek tehát mind-mind önálló, autonóm szerver komponensek, dedikált feladattal futnak, és dedikált célt látnak el, egymással és a kapcsolódó kliensekkel REST-alapú (REpresentational State Transfer) **Hiba! A hivatkozási forrás nem található.** hálózati hívásokkal tudnak kommunikálni, például GET (adat lekérés a szervertől), illetve POST (adatküldés a szervernek) metódusokkal **Hiba! A hivatkozási forrás nem található.**.



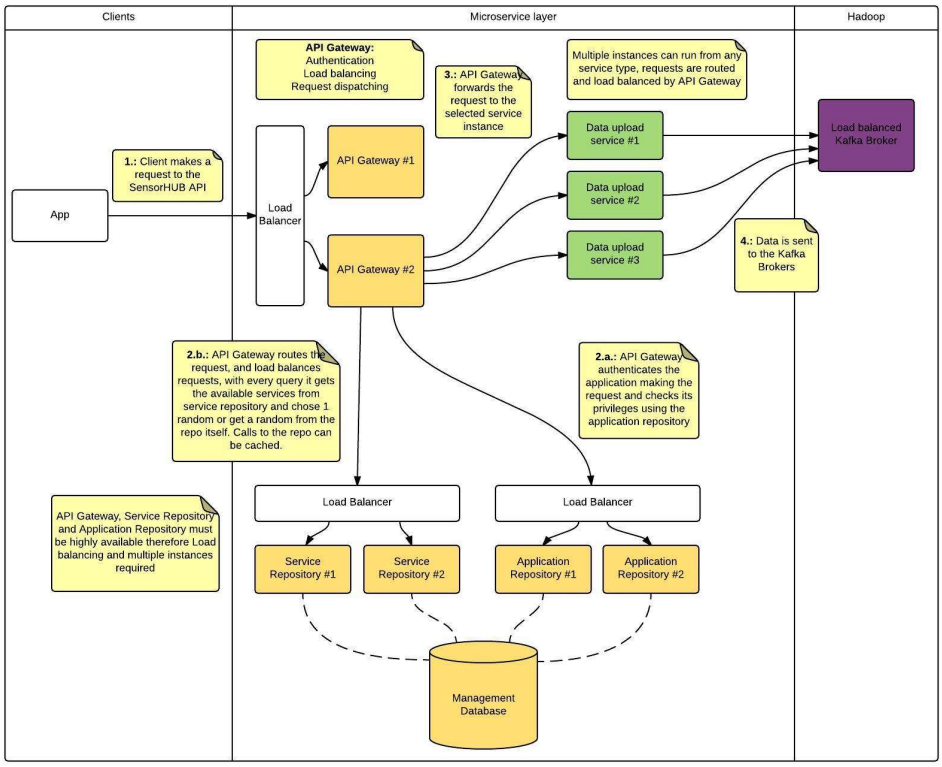
Ábra: SensorHUB szolgáltatás réteg

A SensorHUB-ban használt komponensek:

- API Gateway: A SensorHUB-ot használó alkalmazások belépési pontja a keretrendszerhez. Validálja a beérkező kéréseket és továbbítja a megfelelő komponenshez. Terheléselosztóval rendelkező, hibatűrő szolgáltatás.
- Data Upload Service: Adatfeltöltő szolgáltatás HTTP protokollon keresztül REST használatával. Továbbítja az adatokat a Kafka komponenshez a Hadoop rétegben.

- Device Registry: A keretrendszer belső szolgáltatása. Nyilvántartja a futó mikroszolgáltatások példányait, fogadja az API Gateway-en keresztül érkező kéréseket, és visszaadja az egyes szolgáltatások elérhetőségeit. Terheléselosztóval rendelkező, hibatűrő szolgáltatás.
- Geo Service (Proximity): A szolgáltatás segítségével a geofencing-et valósíthatjuk meg, vagyis körbekeríthetünk a térképen különböző területeket, és a szolgáltatás meg tudja mondani az éppen aktuális koordinátáink alapján, hogy benne vagyunk-e és ha igen, akkor melyik területben vagyunk benne.
- Push Notification Service: A Google Firebase használatával Android, iOS és webes értesítések küldését valósítja meg.
- Hadoop connection: Az adat kommunikációt valósítja meg a Hadoop réteggel, a már ismertetett technológiák (Kafka, NiFi, és további opcionális feldolgozók) segítségével.
- MQTT Gateway: A MQTT protokoll alapú üzenetek belépési pontja.
- Admin Cockpit: A keretrendszer szolgáltatásaihoz és alkalmazásaihoz kapcsolódó webes adminisztrációs felület. Továbbá az egyes szolgáltatásokkal kapcsolatos metrikák mérését, statisztikák számítását támogató szolgáltatás.
- Application Repository: A keretrendszer belső szolgáltatása. Nyilvántartja a keretrendszert használó alkalmazásokat. Végpontot biztosít az alkalmazások beregisztrálásához, autentikálásához és a lekérdezésekhez. Terheléselosztóval rendelkező, hibatűrő szolgáltatás.
- MySQL (Management Database): A SensorHUB belső működését segítő opcionálisan használható adatbázis. A keretrendszer itt tárolhatja el a beregisztrált szolgáltatásokat és alkalmazásokat, menedzselheti azok példány adatait, frissítheti az kihasználtsági adatokat.

Az ábrán még nem szerepel az általam fejlesztett, diplomatervemben részletesen bemutatott CV4SensorHUB mikroszolgáltatás, ami gépi látás projektek menedzselését segítő, illetve térinformatikai adatmenedzselést támogató komponens.



Ábra: Kérésfeldolgozás a SensorHUB keretrendszerben

Kérésfeldolgozás során a kliens alkalmazás kérést indít a SensorHUB API felé (1). Az API Gateway-ek terheléselosztójának az IP címe a belépési pont. A terheléselosztó továbbítja a kérést az egyik API Gateway példánynak. Az API Gateway az Application Repository terheléselosztóján keresztül eljut valamelyik konkrét példányhoz, ami a Management Database adatbázist használva autentikálja az alkalmazást és ellenőrzi a jogosultságait (2.a).

Ennek sikeressége után az API Gateway a Device Registry (Service Repository) terheléselosztóján keresztül eljut valamelyik konkrét Repository-hoz, és megkeresi a kérésnek megfelelő szolgáltatást az adatbázisban (2.b).

Ezek után az API Gateway továbbítja a kérést az így kinyert szolgáltatás címére (3). Innen pedig a kérés tartalmán és a szolgáltatás típusán múlik a folytatás, például adatfeltöltés esetén a kéréssel érkező adat továbbításra kerül a Hadoop rétegbe (4).

A SensorHUB-ban minden kérés HTTP protokollon keresztül, REST-es alapokon történik. Minden szolgáltatás külön IP címeken, illetve eltérő portokon fut, a Repository-k ezeket a címeket tárolják, és ezen információ kinyerésével fog a kérés a megfelelő szolgáltatáshoz eljutni. Tárolásra kerül továbbá az adott szolgáltatás állapota is, ez 10 másodpercenkénti állapotjelzéssel (heartbeat) történik a Device Registry felé. 30 másodperc kimaradás után a szolgáltatás törlődik az aktív komponensek listájáról.

IRODALOMJEGYZÉK

[1] Gartner, „Internet of Things Defined,” [Online]. Available: https://www.gartner.com/it-glossary/internet-of-things/.

[2] Beecham Research, „M2M/IoT Sector Map,” [Online]. Available: http://www.beechamresearch.com/article.aspx?id=4.

[3] L. Arthur, „What Is Big Data?,” [Online]. Available: https://www.forbes.com/sites/lisaarthur/2013/08/15/what-is-big-data/#30676de75c85.

[4] „RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1,” Internet Engineering Task Force.

[5] R. T. Fielding, „Chapter 5: Representational State Transfer (REST),” in *Architectural Styles and the Design of Network-based Software Architectures (Ph.D.)*.

[6] „RFC 7252 - The Constrained Application Protocol (CoAP),” Internet Engineering Task Force.

[7] „MQTT Version 3.1.1,” OASIS.

[8] „Eclipse Paho Website,” [Online]. Available: https://www.eclipse.org/paho/.

[9] P. J. Sadalage és M. Fowler, NoSQL Distilled, Addison-Wesley, 2009.

[10] J. Dean és S. Ghemawat, „MapReduce: Simplified Data Processing on Large Cluster,” in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, 2004.

[11] „Apache Hadoop Website,” [Online]. Available: http://hadoop.apache.org/.