Kővári Attila

CSŐ

Real time Big Data architektúrák építése Azure-ban

Minta (Csak az 1-3 fejezetet tartalmazza)

Írta: Kővári Attila

Szerkesztés lezárva: 2016. augusztus

Második verzió

Tartalom

1 Bevezetés

Jelen fejezet célja, hogy meghatározza e könyv terjedelmét (scope-ját). A scope meghatározásakor szűkítjük a vállalatok körét, az általuk használt adatokat, majd azok feldolgozási módját. Kezdjünk is hozzá.

1.1 Scope

1.1.1 A vállalat

A könyv arra a kérdésre keresi a választ, hogy hogyan profitálhat ma egy vállalat az eddig nem, vagy csak korlátozottan elemzett logokból.

Mire lehet a logokat használni, hol, és hogyan érdemes a logokat tárolni ahhoz, hogy aztán be lehessen őket csatornázni a vállalat BI architektúrájába, hogy az adatvagyont gyarapítva hasznot hajtson.

A könyv nem a Google, Twitter, LinkedIn, Facebook, Prezi méretű és típusú internetes cégekre fókuszál, hanem azokra a „hagyományos" vállalatokra, akiknek vannak logjaik, méréseik, eseményeik és keresik azok alkalmazási lehetőségeit, a bennük rejlő üzleti potenciált.

Nagyon sokat tanultunk a Big Datát kitaláló és először használó úttörő cégektől. Teljesen más szintre helyezték a döntéstámogatást, kitolták a kezelhető adatmennyiségek határait, lecsökkentették a költségeket és fricskát adtak az összes híres adatbázis- és szoftvergyártó orrára. De a vállalatok többsége a mai napig hagyományos adatbázisokkal, strukturált adatokkal dolgozik. Nincs 150 millió szenzoruk és nem

keletkezik 40 millió adatuk másodpercenként, mint mondjuk a CERN-nek. De vannak logjaik amik a saját mércéjükkel nézve nagyok, nehezen kezelhetők és ezért a mai napig parlagon hevernek. Nem termelnek hasznot, talán 15 percnél hosszabb időre nem is tárolják őket. De vannak, és érték lehet bennük. Ha ez így van, akkor e könyv segítségével reményeim szerint képesek lesznek egy olyan olcsó (havi párszázezer forintból fenntartható) és relatíve egyszerű architektúrát összeállítani, amely a bennük rejlő információkból értéket teremt.

A „hagyományos" - jellemzően a termelő, szállítmányozó, telekom, IT szektorban tevékenykedő - vállalatok többsége nem azért kezdi el használni a Big Datát, mert hagyományos eszközökkel nem tudná elemezni az adatait, hanem azért mert hagyományos eszközökkel nem tudná költséghatékonyan elemezni őket. A kezelhetetlen adatmennyiség számukra nem azonos a Google, LinkedIn, … által kezelt adatmennyiséggel. Lehet, hogy csak egy gyártósor másodpercenkénti 10 üzenetét kell feldolgozniuk és real time elemezniük, ami éves szinten sem több mint párszázmillió sor. Nem Big Data méret – gondolhatnánk – de nekik az. Nekik ezeket az adatokat egész egyszerűen nem éri meg a hagyományos technológiákkal tárolni.

A könyvben nem fogunk olcsó hardverekből, nyílt forráskódú szoftverek felhasználásával bonyolult hardver- és szoftverarchitektúrákat tervezni, mert a mai vállalatok jelentős részénél nincs ilyen irányú kompetencia és nem is biztos hogy megéri ilyen irányú kompetenciát építeni és fenntartani. Bérelhető hardver- és

szoftverkomponensekből fogunk összekattintgatni egy olyan jól skálázható architektúrát, amely költséghatékonyan, egyszerűen biztosít lehetőséget a vállalatok számára a logjaikban rejlő értékek termőre fordítására.

A könyv elsődleges célja tehát, hogy utat nyisson a vállalatok számára a logjaikban rejlő értékek kiaknázásához.

1.1.2 A vállalati adatvagyon

1.1.2.1 A vállalati adatvagyon értéke

A vállalati adatvagyont kategorizálhatjuk az adat értéke szerint. Ezek alapján megkülönböztetünk:

- Üzletileg kritikus adatokat
- Redundáns, elöregedett adatokat
- és szürke adatokat

Az üzletileg kritikus adatok szükségesek a vállalat működéséhez. Ezek azok az adatok, amelyek nélkül nem lehetne olyan sikeres a vállalat, mint amilyen sikeres az adat használatával.

A redundáns vagy elöregedett adatok képzik a vállalati adatvagyon azon részét, amelyet biztonsági okokból tárolnak, de üzleti értékük már nincs. (pl.: backup) Ezek azok az adatok, amelyek döntéstámogatás szempontjából értéktelenek így nem képezik részét a könyvben bemutatásra kerülő architektúrának.

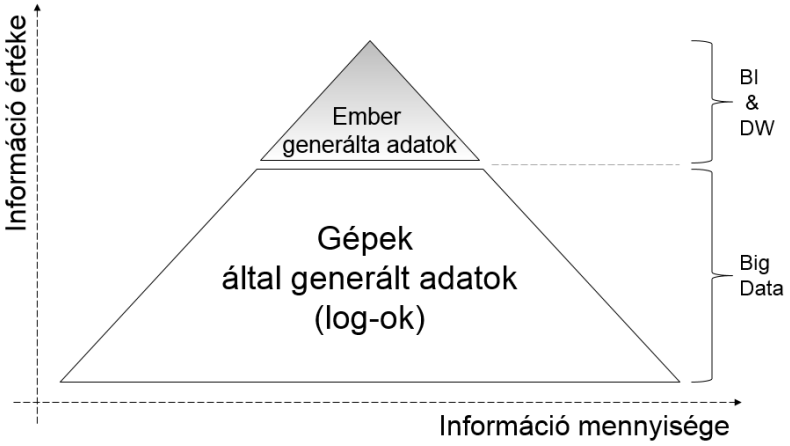
A szürke adatok azok az adatok, amiről még vajmi keveset tudunk. Nem tudjuk hogy mennyi érték lehet bennük, és azt sem hogy tárolásuknak van-e/lehetne-e üzleti haszna.

Ezek azok az adatok, amelyek fontos részét képezik e könyv terjedelmének (scope)

1.1.2.2 A vállalati adatvagyon származása

Egy vállalat adatvagyonát származása szerint két részre oszthatjuk:

1) Emberek által generált adatok (számlák, megrendelések, szállítólevelek, stb.)
2) Gépek/szoftverek által generált adatok (Logok)



1. ábra: A vállalati adatvagyon szerkezete

Az ember által generált adatok jellemzője, hogy relatíve kis méretéhez képest ennek az adathalmaznak van a legnagyobb értéke a vállalat

számára. Ezek azok az adatok, amelyekkel hagyományos BI és adattárház technológiák évek óta foglalkoznak.

Nekünk e könyv szempontjából azonban sokkal fontosabb a gépek/szoftverek által termelt adatok, az úgynevezett logok és a könyv hátralevő részében jellemzően ezekkel az adatokkal fogunk foglalkozni.

De még mielőtt rátérnék, hagy meséljek egy példát, amely szemlélteti az adatok közti különbséget. Egy vállalat szempontjából nagyon fontos látni, hogy ki mit vásárolt. Ezek az adatok az emberek által generált adatok közé tartoznak még akkor is, ha fizikailag nem ember készíti el a számlát. Ezek azok az adatok, amelyek óriási jelentőséggel bírnak a vállalat számára és ezek azok az adatok, amelyeket 20 éve elemzünk BI és adattárház rendszerek segítségével.

De ezen kívül érdekelhet minket az is, hogy ki, mit nem vásárolt. Ezek tartoznak a gépek által generált adatok közé. Sokkal többen vannak azok, akik nem vásároltak semmit, mint azok akik vásároltak és sokkal kevesebb értéket képvisel a vállalat számára az az információ, hogy ki mit nem vett, mint az, hogy ki mit vett. Korábban a ki mit nem vett kérdésre nem tudtunk válaszolni, mert nem állt rendelkezésre olyan technológia, amellyel költséghatékonyan fel tudtuk volna dolgozni azokat a logokat, amelyekből ezek a fontos információk kinyerhetők.

1.1.3 A log

Logok olyan gépek, szoftverek által készített üzenetek sorozata, amely alapján rekonstruálni lehet a gép/szoftver által végrehajtott folyamatokat. A logokat rendszerint hibák, működési rendellenességek utólagos elemzésre készítik.

Logokat gyárthatnak szenzorok, mérőműszerek, egy gyártósor, egy alkalmazás, egy webszájt vagy bármi, ami képes belső folyamatainak eseményeit szöveges üzenet formájában tárolni vagy csak megosztani valamilyen protokollon keresztül.

A feldolgozás során ezeket a rendszerint strukturálatlan üzeneteket alakítjuk át olyan idősorosan is összehasonlítható, aggregálható, elemezhető formátumúra, hogy azokat már gépek/szoftverek is tudják értelmezni és e szoftverek segítségükkel olyan következtetéseket tudjunk levonni, amelyek messze túlmutatnak a logok eredeti funkcióján.

2. ábra: Logok feldolgozása

A könyvben bemutatásra kerülő módszertan ezekkel a logokkal és feldolgozásukkal, elemzésükkel fog foglalkozni. Először azt fogjuk bemutatni, hogy milyen architektúrák a legalkalmasabbak logok feldolgozására és tárolására.

1.1.4 A betöltés gyakorisága

A logok feldolgozását lehetővé tevő architektúrák felépítését alapvetően befolyásolja, hogy milyen gyakran kell azokat betölteni az architektúrába. A gyakoriság lehet

1) **Eseti.** Az eseti betöltések során esetenként, jellemzően kézzel mozgatjuk át az adatokat az input oldalról az architektúrába. Az eseti betöltéseket olyan technológiák támogatják a Microsoft Azure-os világban, mint az Azure Import/Export Service, az AZCopy, az Azure PowerShell, vagy ha adatbázis forrásról van szó, akkor az Azure SQL Database Tool
2) **Periodikusam ismétlődő** (tipikusan napi) betöltések során az adat jellemzően napi egyszeri rendszerességgel jut el a forrásból az architektúrába. A napi rendszeres áttöltésre jelenleg a legjobb eszközök a hagyományos ETL szoftverek, de – mint azt a könyvben látni fogjuk – a real-time feldolgozó eszközöket is használhatjuk napi egyszeri, kötegelt adatfeldolgozásra.
3) **Real time** (kis késleltetésű) betöltések során az adatok betöltése folyamatosan történik. A folyamatosan érkező adatok fogadására az Event Hub, a feldolgozására a Stream Analytics a legjobb technológia az Azure világában.

A könyv a továbbiakban elsősorban olyan architektúrákkal foglalkozik, amelyek támogatják a folyamatosan érkező adatok feldolgozását és real time elemzését, de ezzel párhuzamosan mutatunk majd utat az

egyszer (ősfeltöltés) és a napi egyszeri kötegelt (batch) jellegű feldolgozás megvalósítására is.

Összefoglalva: E könyvben azokra a vállalatokra fókuszálunk, akik nem tudnak/akarnak saját környezetükben felépíteni egy real-time Big Data architektúrát, de vannak logjaik, és lehetőségük van ezek real time betöltésére.

Mielőtt rátérünk az architektúrák ismertetésére, essen még néhány szó magáról a real time adatról (stream-ekről) illetve azok viselkedéséről.

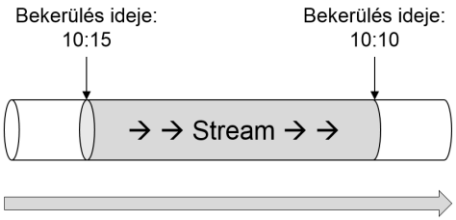
1.2 A Stream-ek világa

A Stream-ek világban az adatok folyamatosan áramlanak a csövön és ez teljesen más gondolkodást követel, mint a táblák által jól körülhatárolt adatok világa. A Stream-nek nincs eleje és vége, gyakorlatilag egy csövet látunk, amibe akárhányszor belenézünk mindig mást és mást látunk.

Ahogy egyetlen ember sem léphet kétszer ugyanabba a folyóba, úgy a csőből sem lehet kétszer ugyanazt az adatot kivenni, mert az már nem lesz ugyanaz.

Nem lehet a csövet zárolni (lock), mint egy táblát, hogy „hé, most ne nyúlj a táblához mert én kérdezem le”. Az adatokat sem `Select * From` formában kérdezzük le, hanem inkább a `Select * From Elmúlt 2 perc`, `Select * From Dátumtól-ig` módszereket használjuk.

A streamek világában aggregálni és join-olni is csak úgy tudunk, hogy a cső tartalmát szűkítjük az esemény keletkezésének/vagy bekerülésének dátuma alapján egy időintervallumra, és utána ezen időintervallum által meghatározott szeletét aggregáljuk, kötjük:



3. ábra: Stream-ek zárolása két időpont között

Mekkora ez az időintervallum? Jellemzően percek. 5 perc, 10 perc, stb. A stream-ek világában ilyen 5-10 perces ablakokkal dolgozunk. 5 perces ablakra aggregálunk, két ötperces ablakot kapcsolunk egymáshoz.

Egy stream-ek világában tehát úgy tudunk adatokkal dolgozni, adatokat aggregálni, adatokat egymással összekötni, ha a stream-et leszűkítjük (zároljuk) egy olyan időablakra, amelynek adatai még megtalálhatóak a csőben. Ez így elmondva teljesen logikus, de a rutinszerű használatához egy táblákon vagy kockákon szocializálódott embernek idő kell ☺

2 Real time architektúrák

Az előző fejezetben megbeszéltük, hogy e könyv terjedelmét a real time logok képzik, a most következő fejezetben pedig megnézzük, hogy milyen architektúrával tudjuk feldolgozni és tárolni ezeket az adatokat.

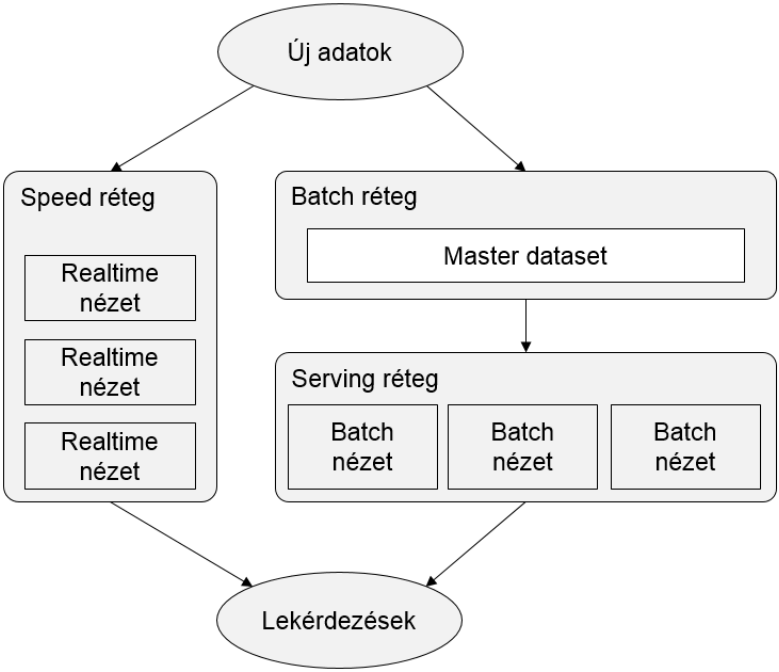
A real time Big Data architektúrák tárgyalását két architektúra bemutatásával kezdjük. Ezek:

1) Lambda architektúra
2) Vortex architektúra

2.1 Lambda architektúra

Aki a real time Big Data megoldások felé kacsintgat, elkerülhetetlen, hogy ne találkozna Nathan Marz Lambda architektúrájával.

A Lambda architektúra lényege, hogy a bejövő adatokat két külön szálon továbbítja a real time igényeket kielégítő „speed” rétegbe és a hosszabb időtávot átfogó batch rétegbe, majd a két réteget összekapcsolva szolgálja ki a historikus és a friss (real time) adatokat is igénylő lekérdezéseket:



4. ábra: Lambda architektúra

A következő fejezetekben megismerkedünk a Lambda architektúrával és annak komponenseivel.

2.1.1 Batch réteg

A batch réteg egy olyan komponense az architektúrának amely:

1) változás nélkül tárolja a folyamatosan érkező eseményeket és
2) periodikusan tovább küldi az előaggregált, átalakított adatokat a serving layer adatbázisába.

A batch réteg tehát nem csak az adatok tárolásáért felel, hanem azok előfeldogozásáért és továbbküldéséért is.

2.1.2 Serving réteg

A Serving réteg egy elosztott, batch frissítésekre és véletlenszerű olvasásokra optimalizált adatbázis, amely gondoskodik a batch réteg által előfeldolgozottan átküldött adatok tárolásáról és azok véletlenszerű lekérdezhetőségének biztosításáról. Gyakorlatilag egy cache, amely a batch réteg hosszú feldolgozási idői miatt mindig késni fog néhány órával a real time adatokhoz képest.

És itt jön képbe a speed réteg…

2.1.3 Speed réteg

A Serving réteg tartalmaz minden forrásadatot, amit a batch feldolgozás kezdetekor látott, ugyanakkor azokat az adatokat, amelyek a feldolgozás óta keletkeztek, nem. A speed réteg elsődleges feladata, hogy a serving rétegben meg nem található friss adatokból állítsa össze a lekérdezéshez szükséges nézeteket.

A Serving réteg batch nézeteit és a Speed réteg real time nézeteit összekötve kialakítható egy olyan architektúra, amely real time válaszidőkkel képes kiszolgálni a felhasználók igényeit. Mindezt óriási mennyiségű strukturálatlan adtahalmazból.

2.1.4 Példa

Az architektúra komponenseinek feladata talán úgy magyarázható el a legkönnyebben, ha mondok egy példát.

Tegyük fel, hogy azt akarjuk real time látni, hogy a mai napon eddig hányan látogatták meg a weboldalainkat. A webszerver folyamatosan, minden egyes látogatás után küldi egy eseményt az architektúrába. Ezután a

1) batch rétegben megjelenik az esemény és ott letárolásra kerül. (Master dataset)
2) A batch réteg ezt követően periodikusan, mondjuk óránként elindít egy jobot, amely felaggregálja az aznapi látogatások számát és ha ezzel elkészül, akkor azt tovább küldi a serving layernek.
3) A serving layeren megjelenik a job induláskori időpontjában ismert látogatások száma, és az letárolásra kerül.
4) Mindezzel párhuzamosan a speed rétegben gyűlnek az utolsó egy óra látogatásai. Ezt összeadva a serving rétegben tárolt t-1 óráig aggregált látogatások számához megkapjuk a pillanatnyi látogatások számát.

Megj.: Az adattárházas világban hasonlóan építettünk ki nagy adatmennyiséget feldolgozó kis késleltetés idejű rendszereket: A speed réteg volt az ODS (Operational Data Store), a batch réteg szerepét egy normalizált vagy Data Vault adattárház töltötte be és a serving réteget csillagsémás adatpiacokból vagy OLAP kockákból építettük fel.

A Lambda architektúrának van számos előnye (pl.: adatok auditálható megőrzése egy forrástól független rétegben, a batch nézetek újratölthetősége, …) de van egy hátránya is: két komplex rendszerben is meg kell írni azokat a kódokat amelyek az outputot előállítják (speed réteg, batch réteg), két rendszeren kell majd lekövetni a változásukat, stb.

A könyv célcsoportját képező vállalatok igényeinek kielégítéséhez a Lambda architektúra sokszor túl nagy, túl komplex. „Ágyúval verébre" szoktuk mondani, amikor egy probléma megoldásához a legjobb, de nem a leghatékonyabb módszert választjuk. A Lambda architektúra tökéletes választás lehet egy Twitternek, egy LinkedInnek vagy egy Facebooknak. De a klasszikus vállalatoknak akik nem az interneten élnek sokszor sok.

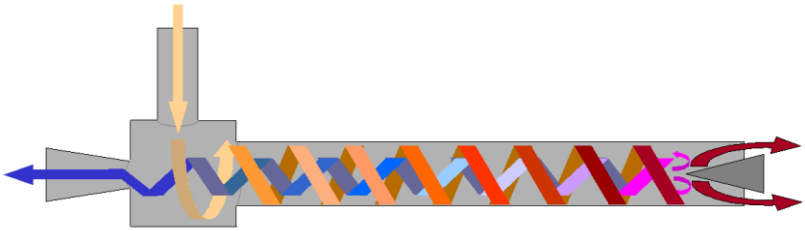
Mindezek miatt a könny további részében nem foglalkozunk a Lambda architektúrával. Helyette egy olyan architektúrát javaslok, amely egy real time eseményfeldolgozó rendszerrel és annak párhuzamosított outputjaival oldja meg a problémát. Mindazonáltal Nathan Marz Lambda architektúráját bemutató Big Data című könyv kötelező olvasmány mindazoknak, akik a real time Big Data architektúra tervezésére készülnek.

2.2 Vortex architektúra

A könyvben részletesen is bemutatásra kerülő Vortex architektúra működését tekintve nagyon sok hasonlóságot mutat az

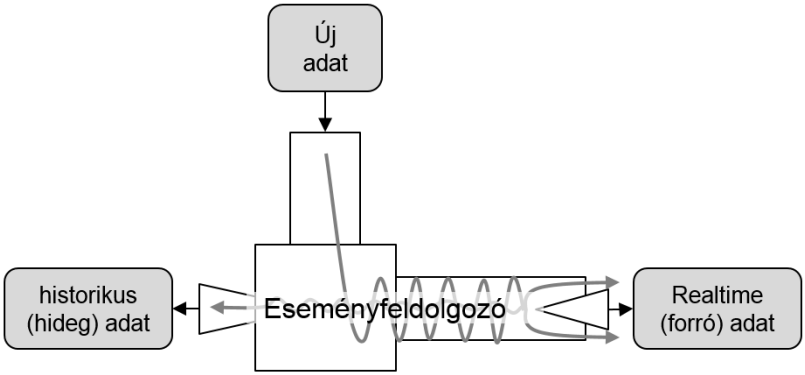
örvénycsövekhez. (Innen a név: Örvénycső = Vortex Tube és innen e könyv neve is ☺)

Az örvénycső egy olyan mechanikus eszköz, amely az örvénycsőbe nagy sebességgel beáramló levegőt megpörgetve hideg és meleg levegőre választja szét:



5. ábra: Az örvénycső (Vortex Tube) működés közben: A fentről beáramló levegőt az örvénycső hideg levegőre (bal oldal) és forró levegőre (jobb oldal) alakítja. Forrás: https://en.wikipedia.org/wiki/Vortex_tube

Ezen az elven működik a Vortex architektúra is: A nagy sebességgel beáramló eseményeket a real time eseményfeldolgozó (örvénycső) két irányba küldi: A forró irány képviseli a real time réteget, ahol nagyon nagy jelentősége van az azonnaliságnak, a hideg pedig a historikus réteget, ahol az adatok hosszú távú elérhetősége a fontos:



6. ábra: A Vortex architektúra működése

A könyv hátralévő részében csak a Vortex architektúrával fogunk foglalkozni, de mielőtt még belevágnánk a Vortex architektúra részletes ismertetésébe megvizsgáljuk a Vortex és a Lambda architektúra közti fő különbségeket.

A Vortex architektúra a Lambda architektúránál lényegesen egyszerűbb. Értem ezalatt, hogy:

1) A Vortex architektúrában nincs batch réteg, azaz hiányzik belőle az a réteg, amely az adatokat forrásrendszer változásaitól függetlenül őrzi (Helyette az adatokat transzformáció mentesen (eredeti formájukban) tárolja az archívumban)
2) A Vortex architektúrában egy motor (real time eseményfeldolgozó) szolgál ki minden outputot. A Lambda architektúrában erre két külön rendszer is szükséges (Egyik a speed réteget, a másik a batch nézeteket szolgálja ki)

Nézzük a különbségeket részletesen is:

2.2.1 Nincs batch layer

A Batch réteg egy olyan eleme a Lambda architektúrának, amely hosszútávon őrzi az adatokat egy olyan adatmodellben, amely független az input adatok szerkezetétől. (Adattárházas világban a Batch réteget leginkább az Inmon féle normalizált, vagy a Linstedt féle Data Vault adattárház réteg szerepéhez lehetne hasonlítani)

A batch rétegre a Lambda architektúrában a következők miatt van szükség:

1) Újratölthetőség: A batch réteg biztosítja az újratölthetőséget: Ha valamit elrontunk a serving réteg nézeteinek töltésekor, akkor a batch rétegben megtalálható adatokból újra tudjuk építeni a serving réteget
2) História: Legyenek meg az adatok egy helyen, ahonnan el lehet érni őket a későbbiekben is.
3) Forrásfüggetlenség: Legyen egy olyan rétege az architektúrának, amely tartalmazza az összes szükséges adatot, de olyan struktúrában, amely függetleníthető a forrás változásától.

A Vortex architektúrából a batch réteg teljesen hiányzik. Ez nem azt jelenti, hogy a fenti funkciók (újratölthetőség, história, forrásfüggetlenség) nem fontosak. Nagyon is fontosak és a Vortex architektúra is eleget tesz ezeknek a követelményeknek, csak erre, mint a későbbiekben látni fogjuk nem épít külön réteget: Egy réteg segítségével oldja meg a fenti feladatokat:

1) Újratölthetőség: Az újratölthetőség a Vortex architektúrában az archívumban őrzött események felgyorsított újratöltésével valósul meg.
2) Megőrzés: A megőrzés feladatát az archívum oldja meg.
3) Függetlenség a forrásrendszertől: A Vortex architektúrának nincs olyan komponense, amely független a forrásrendszer adatszerkezetétől. Később, az adatmodell építéssel foglalkozó fejezetben átbeszéljük majd részletesen is, hogy miért nincs. Elöljáróban legyen elég annyi, hogy a logok esetében nagyon nehéz – és nem is biztos, hogy előnyös – egy forrásfüggetlenséget biztosító adatmodell felépítése.

2.2.2 Két rendszer helyett egy

A Lambda architektúra legtöbbször hangoztatott kritikája, hogy két külön kódot kell írni és karbantartani ugyanannak az outputnak az előállításához (Speed réteg, Serving réteg batch nézetei) Ráadásul nem elég hogy két kódot kell megírni és karbantartani, két komplex rendszerben (Hadoop, Storm) kell megírni azokat a kódokat, amelyek ugyanolyan szerkezetű (csak más aggregáltságú) outputot generálnak.

A Vortex architektúra ezzel szemben két komplex rendszer helyett egy esemény feldolgozó rendszerrel (Stream Analytics) és két különböző outputtal állítja elő a forró és hideg adatokat.

2.2.3 Kételyek, kritikák

2.2.3.1 Nem lehet újratölteni

A Vortex architektúrával szemben megfogalmazott egyik kétely az, hogy vajon képes lesz-e a real time üzenetek feldolgozására optimalizált eseményfeldolgozó egy esetleges újratöltés alkalmával megbirkózni a hatalmas mennyiségű archív adat betöltésével?

A válasz igen. Szolgáltatásként bérelt architektúrák egyik legnagyobb előnye, hogy nagyon jól és egyszerűen skálázhatók. Ha nagyobb áteresztőképességre (Throughput) van szükségünk az újratöltéshez, akkor feljebb tekerjük a potmétert és több kapacitást használunk. Aztán ha az „ősfeltöltés" sikeresen megvalósult és visszaállunk a normális kerékvágásba, akkor visszatekerjük a potmétert is.

2.2.3.2 Nagyobb tárolókapacitást igényel

A másik gyakran hangoztatott kritika a Vortex architektúrával szemben, hogy sokkal több tárkapacitásra van szüksége mint a Lambda architektúrának, hiszen a batch réteg csak az elemzéshez szükséges adatokat tárolja szemben a Vortex architektúrával, amely minden adatot változtatás nélkül tárol az architektúrában.

Ez igaz. De

1) Az archívum óriási biztonságot ad azzal, hogy minden forrásadatot megőriz. Ha valamit el is rontunk a későbbi transzformációk során, az archívumhoz akkor is vissza tudunk térni és onnan újra tudjuk tölteni akár az egész

architektúrát. És ez a biztonság különösen fontos amikor az első lépéseinket tesszük a real time Big Data világa felé.

2) A tárkapacitás ma már olcsó. A könyv célcsoportját képező vállalatoknál a tárolási többletköltség elenyésző.

2.2.3.3 A hibás transzformációval előállított adatok utólag nem módosíthatók

Ez így igaz. A Vortex architektúra inkrementális töltésű. Egy hibás transzformáció, egy hibás üzleti definíció miatti hiba csak az adatok újratöltésével korrigálható. A Lambda architektúrának van olyan (adattárház szerű) rétege (Master dataset), amelyből a származtatott mutatók újraszámolhatók, de a Vortex architektúrának nincs ilyen. Mindent az archívumokból újratöltve lehet csak lépésről lépésre korrigálni.

2.2.3.4 A Vortex architektúra inkrementális töltésű

Ez így igaz, és megmondom őszintén, hogy ez rejt is magában némi kockázatot. Eddig még nem futottunk bele olyan problémába amit ne tudtunk volna megoldani, de a Lambda architektúra batch rétege és batch karakterisztikájú jobjai sokkal nagyobb fejlesztői szabadságot nyújtanak mint a mindent röptében számoló Vortex architektúra.

2.3 Melyiket válasszam?

Mindkét architektúrának megvannak a maga előnyei és hátrányai.

A Vortex architektúra lényegesen egyszerűbb, és sokkal gyorsabb. Kockázatmentesebb bevezetést tesz lehetővé, mint a Lambda architektúra, hiszen nem kell kiépíteni a Batch réteget, egy eszközzel megoldható a real time és a serving réteg töltése. Ugyanakkor a Vortex architektúra Serving rétege csak inkrementálisan tölthető, ami kevésbé tolerálja a fejlesztési hibákat.

A választást ahhoz tudnám hasonlítani, mint amikor arról kell döntenünk, hogy építsünk-e normalizált adattárházat a csillagsémás adatpiacok alá, vagy inkább a csillagsémás adatpiacokból építsük fel az egész adattárházat. (Kimball-Inmon dilemma) E hasonlatban a Lambda architektúra képviseli az Inmon féle „normalizált adattárház tetején csillagsémás adatpiacok" szemléletet, a Vortex architektúra képviseli a Kimball féle csillagsémás adattárház szemléletet.

Mindig egyedileg kell mérlegelni az egyik vagy a másik architektúra alkalmazását, de mint a bevezetőben említettük, a könyv célcsoportját képező vállalatok logelemzési problémája hatékonyabban oldható meg a Vortex architektúrával mint a Lambda architektúrával. Ezért ettől a ponttól kezdve csak a Vortex architektúrával fogunk részletesen foglalkozni.

3 A Vortex architektúra bemutatása

Fontos kiemelnünk rögtön az elején, hogy a Vortex architektúra futhat a felhőben is és futhat a vállalatok saját szerverein is, azonban az alábbi előnyök miatt érdemes eleve felhőben felépíteni:

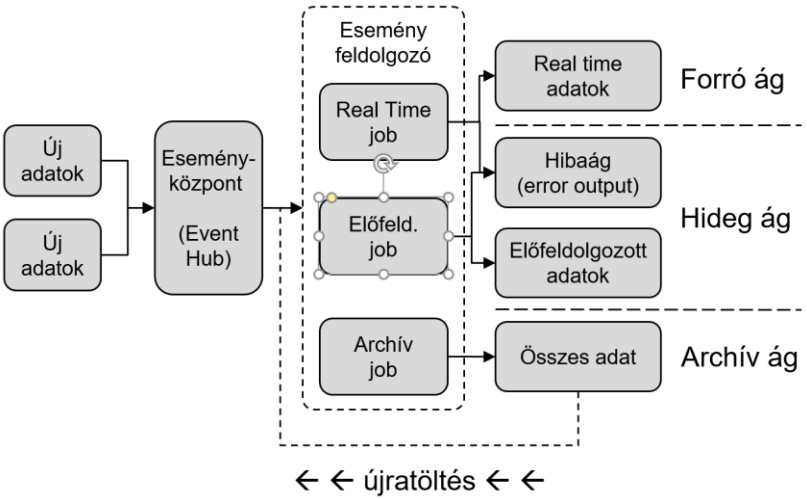
1) A felhő nagyon jól skálázható. Egy szülinapi akció miatti webszájt forgalomnövekedés, egy csúcsra járatás, egy ősfeltöltés okozta terhelésnövekedés esetén azonnal új kapacitások vonhatók be a feldolgozási folyamatba.
2) Time to Market: A felhő nagyon gyors (napok alatt megvalósítható) bevezetést tesz lehetővé. Két cégnél is felépítettem már a Vortex architektúrát és mind a két esetben 25 nap alatt maradt a megvalósítás teljes időszükséglete. Ugyanezt az architektúrát kiépíteni földi körülmények között valószínűleg hónapokba telne.
3) A felhő relatíve olcsó. Csak annyi kapacitásért fizetünk, amennyit használunk.
4) Nem kell hozzá nagyon mély, nagyon speciális üzemeltetési szaktudás. A könyv célcsoportját képező vállalatok nagy részénél komoly problémát jelentene egy kikukázott hardverekből összeállított architektúra felépítése és főleg az üzemeltetése. A felhő architektúrákban ezt elvégzi helyettünk más.
5) Ma már minden számunkra fontos funkció elérhető szolgáltatás formájában. Tehát nem kell virtuális masinákból összeépíteni az architektúrát, az egész architektúra

(önkiszolgáló módon) felépíthető mikroszolgáltatások okos összekapcsolásával. És talán ez az, ami a leginkább szükséges volt ahhoz, hogy hazai méretekben is érdemes legyen Big Data architektúrákban gondolkodni.

Ez után csak felhőben futó architektúrával fogunk foglalkozni, és bemutatom majd, hogy az architektúra felépítéséhez milyen felhőszolgáltatásokat választhatunk a Microsoft felhőszolgáltatásai közül. Az architektúra részletes tárgyalásakor meg fogjom mutatni, hogy melyek ezek a felhőszolgáltatások, milyen szinten állnak, mit lehet velük megvalósítani és mit nem.

Kezdjünk is bele és nézzük meg részletesen hogyan épül fel a Vortex architektúra.

A Vortex architektúra a felépítését a következő ábra szemlélteti:



7. ábra: A Vortex architektúra felépítése

Az események, üzenetek az Event Hub-on keresztül kerülnek az architektúrába, ahol az eseményfeldolgozók (stream processzor) átalakítják a kívánt formátumra és elágaztatják a kívánt outputra.

A Vortex architektúra 3 outputra küldi az input adatokat:

1) Forró ág: Pár óra adatát tartalmazó, real time lekérdezésre optimalizált réteg
2) Hideg ág: Hónapok, évek adatát tartalmazó, idősoros lekérdezésre optimalizált réteg
3) Archív ág: Az input adatok transzformáció mentes archívuma.

Nézzük a Vortex architektúra egyes elemeit részletesen:

3.1 Eseményközpont (Event Hub)

Az eseményközpont (Event Hub) egy jól skálázható eseménygyűjtő felhőszolgáltatás, amely másodperceként több milliónyi esemény gyűjtésére és ideiglenes tárolására képes.

A Vortex architektúrában az Event Hubok biztosítják az aszinkronitást az eseményt küldő szenzorok, eszközök, webalkalmazások és az eseményeket feldolgozó szolgáltatások között. Egyfajta puffer terület, ahol addig gyűjthetők és állomásoztathatók az üzenetek, ameddig azok feldolgozása be nem fejeződik.

Ha egy hiba miatt leáll, vagy egy tervszerű karbantartás miatt leállítjuk az események feldolgozását, akkor az Event Hub gondoskodik az események tárolásáról addig, amíg a hibát ki nem javítjuk, vagy a tervszerű karbantartást be nem fejezzük.

Az Event Hub-ok adattárházas megfelelője leginkább a transient (állandóan felülírt) stage terület lenne.

3.2 Real time eseményfeldolgozó (Stream Analytics)

A real-time eseményfeldolgozó másodpercenként többmillió eseményt képes feldolgozni és különböző formára transzformálni.

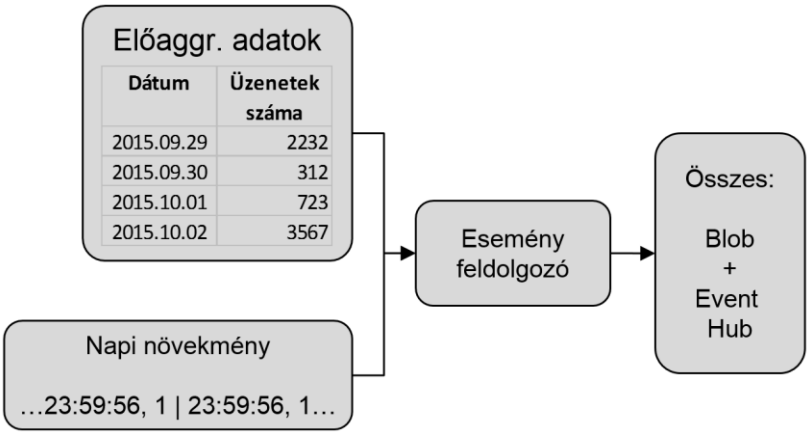
A real-time eseményfeldolgozó a Vortex architektúra lelke. Ez maga az örvénycső (Vortex-tube) amely a nagysebességgel bejövő adatokat hideg (historikus) és forró (real time) outputokra küldi.

A real time eseményfeldolgozó végzi el azokat a transzformációkat, amely eredményeképp a nyers input eseményekből üzleti jelentéssel bíró adatok keletkeznek és kerülnek letárolásra az architektúra forró (real time) és hideg (historikus) rétegeiben.

Fontos, hogy a Vortex architektúrában minden transzformációt a real time eseményfeldolgozó végez. Nem csak a real-time feldolgozásokat, hanem a batch jellegű (óránkénti, éjszakánkénti) feldolgozásokat is. A stream processzor számolja ki, hogy éppen hány bejelentkezett felhasználója van egy webszájtnak és azt is, hogy hány különböző felhasználója volt egy nap az oldalnak. De a napon túli aggregációk egy részét, (mint például egy hónap különböző felhasználóinak száma) – mint a későbbiekben látni fogjuk – már más eszközök fogják kiszámítani.

A real time eseményfeldolgozó ugyanis csak 1-2 napnyi adatot lát. (Ennyi adatot tárol az Event Hub), ennyiből tud aggregálni, eltérők darabszámát (Distinct Count) számolni.

Megj.: Az idő mentén aggregálható (inkrementálisan kiszámolható) mutatókat ki tudja számolni a stream processzor akár egy évre is, ha stream-be bevezetjük a már előaggregált adatokat és ahhoz hozzáadjuk a napi aktuális növekményt:



8. ábra: Real time kalkuláció a teljes adathalmazon

Az inkrementálisan nem számolható mutatókkal, (mint pl. Distinct Count) már nem birkózik meg ilyen könnyen a real time eseményfeldolgozó, de erről majd később, a hideg ág töltésénél részletesen is beszélünk.

Adattárházas terminológia szerint a real time eseményfeldolgozó funkcióját tekintve leginkább az ETL (Extract Transform Load) eszközökhöz hasonlít: Adatokat olvas be, azokat átalakítja, majd átadja a fogadó rétegnek.

A real time eseményfeldolgozó feladatát a Stream Analytics látja el a Microsoft Azure kínálatában.

3.3 Forró ág (Real Time réteg)

A forró ág (vagy más néven a Real Time réteg) felhasználási területét tekintve kettős célt szolgál:

A real time rétegből elégítjük ki azon felhasználók igényét, akik

1) visszamenőlegesen csak 10 perc, 1 óra, 1 nap, 1 hét adatait akarják real time elemezni
2) és innen elégítjük majd ki vállalat teljes log adatvagyonának real time elemzéséhez szükséges azon részét, amely az utolsó 1,2 órában keletkezett és az architektúra historikus ágában még nem jelentek meg.

3.3.1 Kik lesznek a real time réteg fő felhasználói?

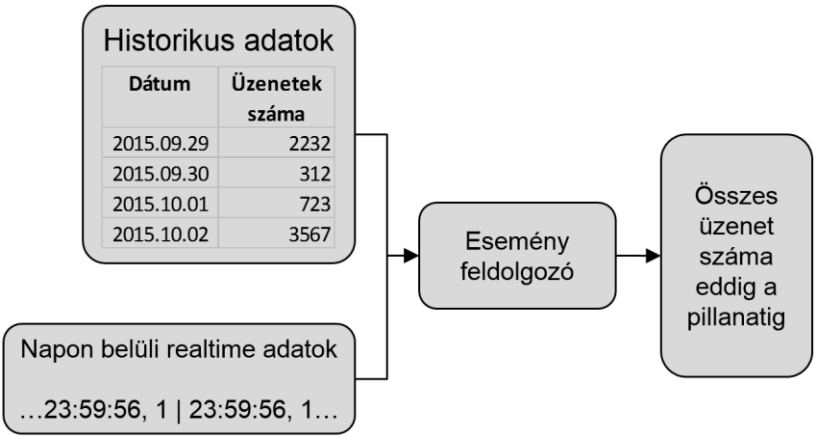
Azok a felhasználók akik csak az utolsó 1 nap, 1 hét, stb. adataira kíváncsiak általában

1) kiugró értékeket akarnak azonnal látni, anomáliákat detektálni, szokásostól eltérő mozgásokról riasztást kapni és
2) egy rövid időperiódus adatait akarják real-time elemezni.

Ők azok, akik egyből tudni akarják, ha valami letér az ideális pályáról, vagy egy hiba, hacker támadás esetén azonnal hozzá akarnak férni

egy olyan adatbázishoz, amely segítségével a hibát rekonstruálhatják, a hiba/sebezhetőség okát minél hamarabb megtalálják. Fő felhasználói jellemzően fejlesztők, üzemeltetők, minőségbiztosítók.

A real time réteg felhasználói közé tartoznak közvetve azok a felhasználók is, akik az adatvagyon teljes egészét akarják real time elemezni. Ezeket a felhasználókat a Vortex architektúra hideg (historikus) és meleg (real-time) ágából közösen szolgáljuk ki:



9. ábra: real time elemzésigények kiszolgálása

A real time réteg másik feladata tehát a historikus réteg utolsó töltése óta keletkezett új adatok tárolása. Adattartalmát tekintve csak töredékét tartalmazza a vállalat teljes log adatvagyonának.

A real time réteg adatainak tárolására olyan technológiák jöhetnek csak szóba, amelyek támogatják a gyors (eseményenkénti) beszúrásokat és lekérdezéseket. A hagyományos fájl alapú Big Data technológiák ezt a szerepet nem tudják betölteni. Ebben a memóriaalapú adatbázis kezelők a profik.

Microsoft Azure szolgáltatások közül ilyen lehet – késleltetési szinttől függően - a DocumentDB vagy az SQL Azure, korlátozottan a Power BI illetve Event Hub is.

3.4 Hideg ág (Historikus réteg)

A hideg ág, vagy más néven historikus réteg elsődleges feladata az elemzői-, adattárház és egyéb rendszerigények kiszolgálása.

Hideg ágat azért építünk, hogy legyen egy olyan területe az architektúrának, amely már előfeldolgozottan, a későbbi fogyasztást megkönnyítő struktúrában és aggregáltsági szinten tárolja az adatokat.

Előaggregálhatjuk az adatokat például azért, mert az adattárháznak csak percenként egy mérésre, vagy negyedórás átlagra, vagy óránkénti szummára van szüksége.

Előaggregálhatjuk az adatokat azért is, hogy támogassuk a real time lekérdezések kiszolgálását. Az üzenetek számának idősorost alakulását például úgy tudjuk meghatározni, hogy minden nap végén kiszámítjuk és letároljuk a historikus rétegben az adott napi üzenetek számát és ehhez hozzáadjuk a real time rétegben található mai üzenetek számát.

A historikus réteg karakterisztikáját tekintve egyszeri írásra, folyamatosan növő adatmennyiségre és többszöri tömeges olvasásra lett kitalálva, ezért olyan technológiát érdemes választani, amely ennek a karakterisztikának megfelel.

A historikus réteg igényeit Microsoft felhőszolgáltatásai közül leggyakrabban a Blob Storage-dzsel, a NoSQL DocumentDB-vel vagy az relációs SQL Azure-ral szolgáljuk ki, de nagy potenciál van a most még bétában lévő Azure SQL Data Warehouse, vagy az Azure Data Lake.

3.5 Archív réteg

Az archív rétegnek két szerep jut a Vortex architektúrában: Lehetőséget biztosítson

1) az adatok hosszú távú megőrzésére és
2) újratölthetőségére.

Hívhatnánk időkapszulának, vagy használhatnánk a még divatosabb adattenger (Data Lake) kifejezést is, de a célját legjobban az archívum szó írja le, ezért azt fogjuk használni.

Az archív rétegben az adatok addig vannak tárolva, amíg az adatok újratölthetősége vagy megőrzése megkívánja. Üzleti igénytől függően ez lehet egy hónap, egy év, vagy akár a történelem kezdete.

Az archív rétegben az adatokat átalakítás nélkül tároljuk és az archív réteg igényeit minden esetben Azure Blob Storage-dzsel elégítjük ki.

Megjegyzés: Az archív réteg megvalósítására nagyon ígéretes eszköznek néz ki az Azure Data Lake Store, de ez jelen pillanatban még csak preview állapotban van. Ha elérhetővé válik a végleges szolgáltatás, akkor mindenképpen érdemes lesz kiértékelni, mint a Vortex Architektúra archív ága.

A könyvnek itt még nincs vége. Az eddig olvasottak után rátérünk a tényleges megvalósítás lépéseire. Kezdünk azzal hogy hogyan érdemes elindítani egy real time big data projektet majd beszélünk a tervezés kérdéseiről és végül megnézzük, hogy hogyan lehet mindezt megvalósítani Microsoft Azure-ban.

Ha tetszett amit eddig olvasott, akkor tetszeni fog az is ami ezután következik ☺ Igaz ehhez meg kell vásárolni a könyvet ☹

Köszönöm hogy letöltötte ez a mintát. Remélem hasznosnak találta az olvasottakat és remélem, hogy hasznosnak találja majd a teljes könyvet is.

A könyv megvásárolható itt: http://www.biprojekt.hu/Cso-Real-time-Big-Data-architekturak-epitese-Azure-ban.html