UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Aleksandar Ignjatijević

E2 12/2021

APACHE FLINK

Profesor:

Dr Vladimir Dimitrieski

Novi Sad, 2022.

Sadržaj

Uvod

Predmet ovog seminarskog rada je alat Apache Flink, alat za obradu ograničenih i neograničenih skupova podataka.

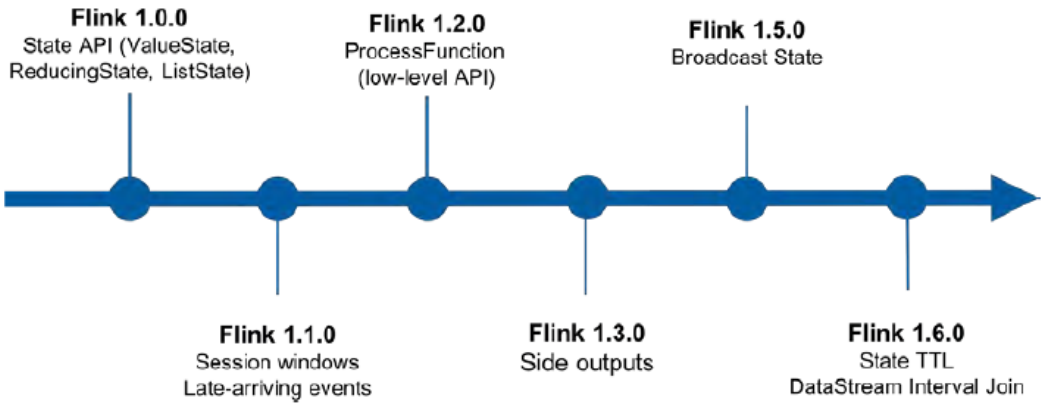
Kako se veliki broj aplikacija danas oslanja na konstantan tok podataka, logično je za zaključiti da je vid obrade koji obrađuje takvu vrstu podataka veoma bitan. IoT, pametni automobili, finansijske transakcije, kao i mnoge druge stvari se oslanjaju na ovakav vid podataka. Nameće se zaključak da ukoliko već podaci dolaze u vidu toka, da je želja tako ih i obrađivati. Tu na scenu stupa Apache Flink.

Shodno tome, u radu ćebiti prikazane karakteristike, sposobnosti, funkcionalnosti i slučajeve korišćenja Apache Flink alata, kao jednog od moćnijih alata u tom domenu. Apache Flink predstavlja *open-source* platformu napravljenu od strane Apache Flink zajednice nastao 2011. godine i od tad, zahvaljujući neprestanom radu *open-source* zajednice, izašlo je nekoliko verzija, a trenutna stabilana verzija je 1.14.2. Nažalost, nedostaje motivacije za razvoj ovog, a i sličnih alata.

U prvom poglavlju biće reči o samom istorijatu Apache Flink-a, kao i o funkcionalnostima koje su implementirane u prvih šest verzija. Drugo poglavlje daće detaljnije objašnjenje arhitekture Apache Flink-a, kao i objašnjenje osnovnih pojmova kojima Flink barata, kao i sa njegovim prednostima, poput skaliranja. Treće poglavlje se bavi pojmovima, bitnim za Apache Flink aplikacije, poput: toka podataka (eng. *stream*), stanja (eng. *state*), vremena i *Layered API-em.* Četvrto poglavlje daje primere slučajeva korišćenja ovog alata na primerima *event-driven, data-analytics* i *data pipeline* aplikacija. Sam zaključak je posvećen rekapitulaciji rada.

Istorijat Apache Flink-a

Kao moćan alat za *batch* i *stream* procesiranje, Apache Flink je dobio na popularnosti u zadnje vreme. Upravo *streaming* procesiranje se može koristiti za razvoj *event-driven* aplikacija, o kojima će biti reči u nastavku rada. Na slici 1 prikazane su izmene u verzijama Apache Flink API-a, zaključno sa 2019. godinom.



Slika 1. Razvoj Apache Flik-a (od verzije 1.0.0 do 1.6.0) [1]

U prvoj 1.0.0 verziji Flink je uveo StateAPI, što je predstavljalo najveću inovaciju u samom Flinku. Ovo unapređenje je omogućilo korisnicima da se služe Flink-ovim stanjem (eng. *State*) kao sa *Java Collection-om*, bez straha od gubitaka, usled otkaza. Nakon toga, u verziji 1.1.0, započeta je podrška *Session Window-u* i tačnijim procesiranjem nesortiranih podataka, kao i onih koji su kasnili. U narednoj, 1.2.0 verziji, implementiran je Flink-ov *ProfessFunctionAPI*. *ProcessFunctionAPI* je API nižeg nivoa za implementiranje kompleksnijih funkcija. Takođe, implementirana je podrška za *EventTime* i *ProcessingTime*, time su otvorena vrata za razvoj *event-based* i *time-based* aplikacija. Nakon verzije 1.2.0, usledila je verzija 1.3.0 u kojoj su implementirane *Side Output* funkcije. Kako je ponekad poželjno na nekom drugom izlazu prikazivati abnormalne podatke i podatke koji su stigli kasnije u vidu nekih *side stream-ova*, ove funkcionalnosti su se pokazale veoma korisne. BroadcastState je usledio kao nadogradnja *StateAPI-a* u verziji 1.5.0. Uz pomoć ove ekstenzije, ubrzalo se izvršavanje nejednakih *join* scenarija u *SQL* upitima. Verzija 1.6.0 je donela *StateTTL* i *DataStream Interval. StateTTL* je doneo sa sobom specifične *time-to-live* parametre uz pomoć kojih je moguće određene podatke brisati iz memorije kada im istekne vreme. Pre toga, korisnici su morali da uz pomoć *ProcessFunction* da registruju neki tajmer i

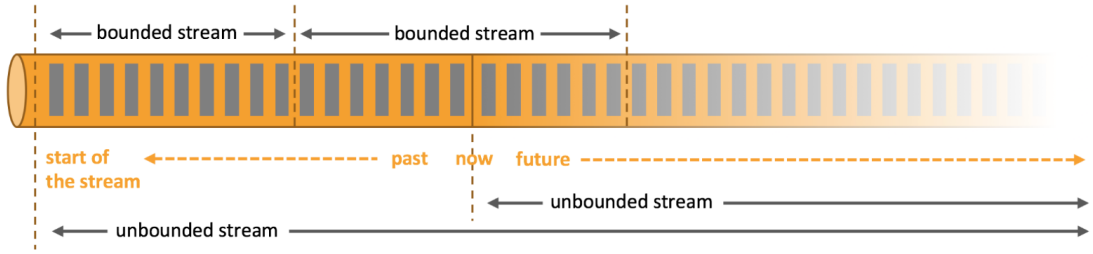
da ručno brišu *state* nakon tajmerove *callback* funkcije. Sa druge strane, *DataStream* intervali su doneli mogućnost *interval join-a*.[1]

Osnove arhitekture Apache Flink-a

Apache Flink je radni okvir (eng. *framework*) i alat za distribuiranu obradu ograničenih i neograničcenih tokova podataka (eng. *data streams*). Napravljen je sa idejom da radi u svim poznatim klaster okruženjima i da obavlja izračunavanja *in-memory* brzinom, na bilo kom nivou. Takođe, Flink koristi istu arhitekturu koja podržava i *batch* i *stream* obradu podataka.

Procesiranje ograničenih i neograničenih skupova podataka

Za početak je bitno razumeti razliku između ograničenih (eng. *bounded*) i neograničenih (eng. un*bounded*) skupova podataka. Na slici 2 se može videti razlika između ograničenih i neograničenih skupova podataka.



Slika 2. Prikaz ograničenih i neograničenih skupova podataka [2]

Karakteristike ograničenih skupova podataka: [2]

- Imaju definisani početak i definisani kraj
- Moguće je procesuiranje nakon što se učitaju svi podaci
- Učitavanje po redosledu nije obavezno
- Moguće je sortirati u bilo kom trenutku

Takođe, procesuiranje ovih skupova podataka se svodi na batch obradu.

Karakteristike neograničenih skupova podataka: [2]

- Definisan je početak, ali ne i kraj
- Moraju da budu obrađivani redom, tj u onom trenutku u kom se učita podatak
- Nije moguće čekati da se ceo skup podataka učita, pa da se nakon toga radi obrada, jer nema definisani kraj, pa je čekanje neograničeno
- Procesiranje ovih skupova podataka zahteva da se podaci obrađuju u onom redosledu u kom su pristigli

Apache Flink uspešno realizuje procesuiranje obe vrste skupova podataka.

Podizanje aplikacija i skaliranje

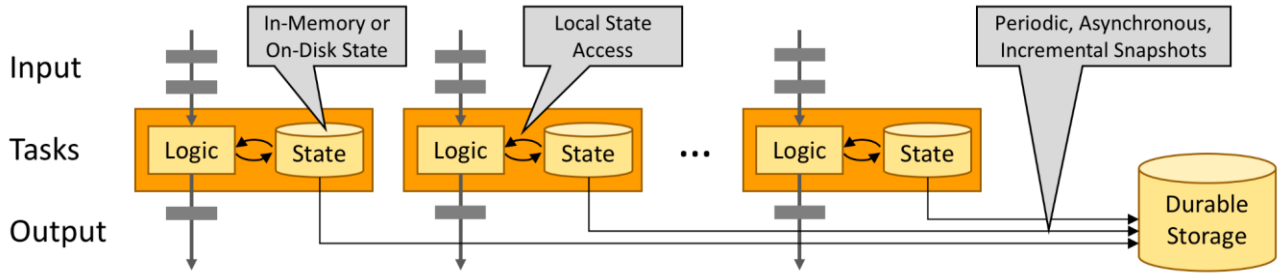
Kako je Apache Flink distribuirani sistem, potrebna mu je komputaciona snaga da bi izvršavao aplikacije. Prednosti Apache Flink-a su to da dobro funkcioniše sa popularnim *cluster resource managers,* poput Kubernetes-a, Apache Mesos-a i YARN-a, ali je isto tako moguće podesiti ga da radi na pojedinačnom klasteru. Pri samom podizanju arhitekture, Flink će sam prepoznati koliko resursa je potrebno za uspešno izvršavanje aplikacije, na osnovu toga kako je konfigurisana aplikacija, sa aspekta paralelizacije. Nakon što prepozna potrebne resurse, zahtevaće to od *resource manager-a,* a u slučaju otkaza, automatski će zahtevati podizanje novih kontejnera (eng. *container*). Sva ova komunikacija sa Apache Flink aplikacijom se odvija putem REST poziva, što dodatno olakšava izvršavanje aplikacija u bilo kom okruženju. [2]

Skaliranje sa Apache Flink-om je veoma zahvalno, i otprilike neograničeno. Razlog za to je sama sposobnost Flink-a da aplikacije paralelizuje u veliki broj zadataka koji su distribuirani i koji se konkurentno izvršavaju u klasterima. Takođe, Flink ima sposobnost da održava veoma veliki *application state*. Uz pomoć asinhronog i inkrementalnog *checkpoint* algoritma (eng. *asynchronous and incremental checkpointing algorithm)*, Apache Flink osigurava minimalni uticaj kašnjenja obrade. [2]

Flink podržava skalranje do nivoa od: [2]

- više triliona događaja (eng. *event*) po danu
- više terabajta stanja (eng. *state*)
- pokretanje aplikacije na hiljadama jezgara (eng. *core*)

Flink je predviđen da pokreće *stateful streaming* aplikacije i optimizovan je za pristup lokalnom stanju (eng. *local state access*). Kako je *task state* čuvan ili u memoriji ili u strukturama podatka jednostavnim za pristup, ukoliko prevazilazi veličinu slobodne memorije, sva izračunavanja se vrše pristupanjem lokalnoj memoriji i zahvaljujući tome vreme kašnjenja obrade je veoma malo. Dodatna pogodnost Flink-a je da garantuje konzistentnost u vidu tačno jednog stanja, u slučaju otkaza. Ovo je omogućeno upravo zahvaljujući asinhornom *checkpoint-ovanju* lokalnog stanja.[2] To je prikazano na slici 3.



Slika 3. Prikaz funkcionisanja asinhronog inkrementalnog *checkpoint* algoritma[2]

Na slici 3 je prikazano funkcionisanje asinhronog inkrementalnog *checkpoint* algoritma. Da bi se omogućila konzistentnost u vidu tačno jednog stanja i otpornost na otkaze, svaka promena stanja se piše u *storage.* Naravno, u slučaju velikog *state-a* takve akcije uzimaju puno resursa i vremena, te je zbog toga uvedena inkrementacija. Svaki *checkpoint* predstavlja razliku između trenutnog stanja i prethodnog *checkpointa,* te je na taj način znatno ubrzan algoritam.

Apache Flink aplikacije

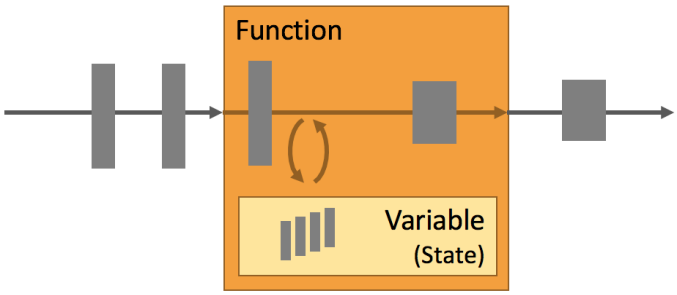
Da bi se razumela struktura aplikacija, potrebno je razmotriti sledeće pojmove:[3]

- Tok podataka (eng. *stream*)
- Stanje (eng. *state*)
- Vreme
- *Layered APIs: ProcessFunctions, DataStream API i SQL & Table API*

Kako su sami tokovi podatak osnovni za *stream* procesiranje, potrebno je razumeti razlike između samih vidova tokova koje Apache Flink podržava.

Apache Flink podržava dva vida tokova: ograničeni/neograničeni vid tokova i *real-time/recorded* tokove. Ograničeni, odnosno neograničeni tokovi imaju, odnosno nemaju svoj kraj, a Flink uspešno izvršava izračunavanja. Kod *real-time* tokova postoje dva načina obrade: obrada u trenutku u kom stižu podaci ili čuvanje u fajl sistemu i obrada kasnije. Flink uspešno barata sa obe tehnike.[3]

Svaka aplikacija koja izvršava neku *business* logiku mora da pamti stanje u kom se nalazila pre, da bi u trenutku sledećeg događaja mogla da nastavi odatle. Flink veoma uspešno izvršava ovaj zadatak. Neke od prednosti koje Flink nudi su: veoma veliko stanje (nekoliko terabajta), konzistentnost "tačno jednog" stanja, skalabilne aplikacije i mnoge druge. Takođe, Flink omogućava developerima da odaberu strukturu samog stanja, u smislu, mape, liste ili neke atomične vrednosti. Sama skalabilnost se svodi na redistribuciju stanja u više *worker-a.* Prikaz funkcionisanja stanja dat je na slici 4.[3]

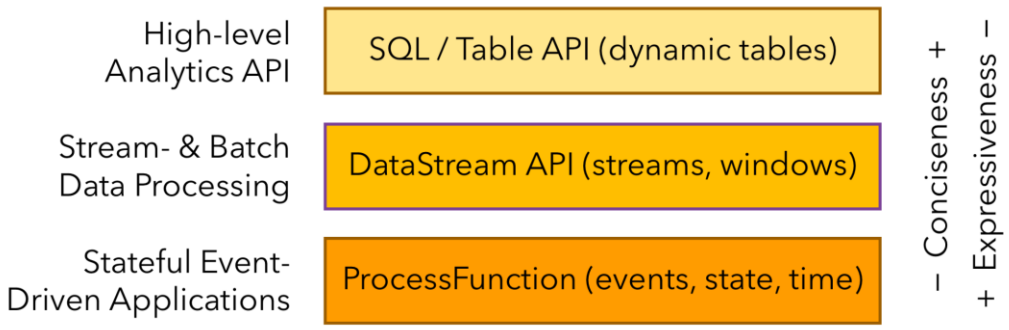


Slika 4. Prikaz funkcionisanja stanja [3]

Kako je vreme veoma bitna stavka prilikom streaming obrade, Flink nudi veoma širok spektar funkcionalnosti, poput: [3]

- *Event-time Mode* - sam *event-time processing* omogućava tačno i konzistentno procesiranje na osnovu *timestamp-ova,* bez obzira da li su podaci *recorded* ili kao *real-time.*
- *Watermark Support* - Flink implementira ovaj vid funkcionalnosti zarad poimanja vremena u *event-time* aplikacijama.
- *Late Data Handling* - ukoliko je u aplikaciji implementiran *watermark* mehanizam, te dođe do toga da neki od događaja stignu nakon izvršenog izračunavanja, flink ovom funkcionalnosću nudi ili preusmerenje na neki sporedni izlaz ili vrši *update* prethodnog izračunavanja.
- *Processing-time Mode* - ova funkcionalnost je pogodna za aplikacije koje mogu da prihvataju aproksimiranje rezultate, a imaju nizak stepen tolerancije na kašnjenje.

Slika 5 predstavlja odnos konciznosti i ekspresivnosti *LayeredAPI*-a.



Slika 5. Odnos konzistentnosti i ekspresivnosti u LayeredAPI [3]

Kao što je i moguće videti na slici 5, *ProcessFunction* nudi najveći stepen ekspresivnosti. Takođe, Flink nudi mogućnost procesiranja pojedinačnih događaja sa jednog ili dva izvorna toka podataka ili podatke koji su grupisani. *ProcessFunction* ima sposobnost da svojevoljno promeni stanje i registruje tajmere koji će okinuti *callback* funkciju u budućnosti, te zahvaljujući tome nudi veoma bitne mogućnost izvršavanja logički kompleksnih zadataka koje zahtevaju *stateful event-driven* aplikacije.[3]

Kompromis između konciznosti i ekspresivnosti predstavlja *DataStreamAPI*. On nudi veliki broj uobičajeniih operacija za *stream* procesiranje. Dostupan je za *Scalu* i *Javu* i zasnovan je na funkcijama poput: *map(), reduce(), aggregate()*.[3]

Najbolji API po pitanju konciznosti su relacioni API-i *SQL* i *Table API*, a nastali su sa željom da se olakša analitika podataka, *data pipelining* i nastanak *ETL* aplikacija, o kojima će

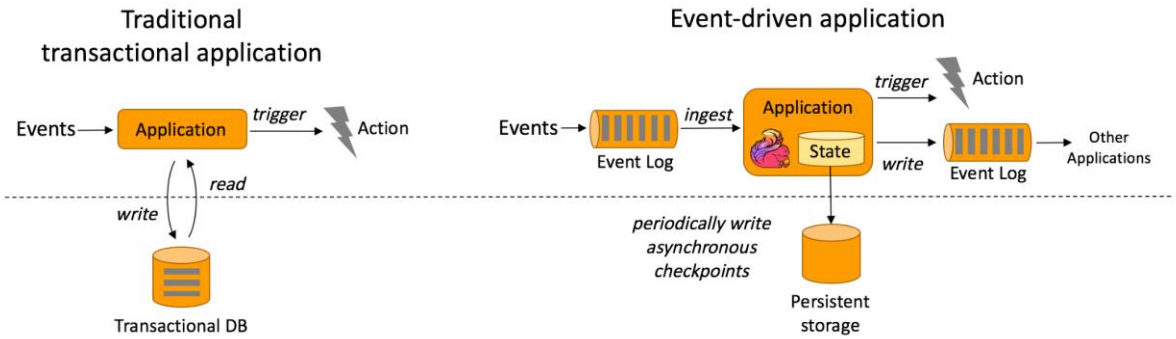
biti reči u nastavku. Prednost ovih API-a je u tome da se sami upiti isto pišu i za *batch* i za *streaming* obradu. Takođe, mogu da se integrišu sa *DataStreamAPi*-em i podržavaju korisničko definisanje skaliranja, agregiranja i ostalih *table-valued* funkcija. [3]

Slučajevi korišćenja

Flink je veoma pouzdan i uspešan u svim scenariima obrade podataka i postoji nekoliko karakterističnih primera upotrebe Apache Flinka, poput *event-driven* aplikacija, *data analytics* aplikacija i *data pipeline* aplikacija.

Event-driven aplikacije

Ovakve aplikacije koriste određene događaje iz nekih tokova koji aktiviraju izračunavanja ili promene stanja. U arhitekturi standardnih aplikacija, aplikacije čitaju podatke i pišu u *remote* transakcionu bazu podataka. Sa druge strane, *event-driven* aplikacije se baziraju na *stateful stream processingu*. Podaci i izračunavanja se skladište na istom mestu, što olakšava pristup. Otpornost na greške se postiže upisivanjem *checkpoint-a* u odvojeno perzistentno skladište. Slika 6 prikazuje odnos standardnih i *event-driven* aplikacija. Takođe, dodatna prednost *event-driven* aplikacija je upotreba sopstvenih baza podataka, tako da samo skaliranje ili promena baze ne predstavlja veliki problem i ne zahteva koordinaciju velikog broja servisa. [4]



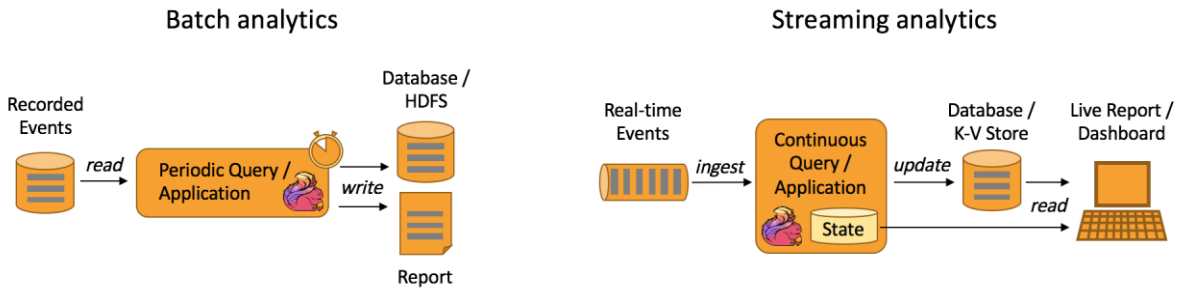
Slika 6. Odnos tansakcionih i *event-driven* aplikacija [4]

Ograničenja kod ovakvih aplikacija se svode na to koliko dobro tokovi mogu da rukuju vremenom i stanjima, a kako Flink uspešno rukje ovim stvarima, predstavlja veoma dobar

izbor. Neke od čestih implementacija u Flinku su: sistemi za detekciju anomalija, prevara, sistemi za monitoring poslovnih procesa ili društvene mreže. [4]

Data Analytics aplikacije

Ovakve aplikacije izvlače informacije iz sirovih podataka. Obično se ovo radi batch obradom nad ograničenim skupom podataka. Ali kako je potrebno uključiti i nove podatke koji stižu, potrebno je dodati te podatke u već obrađeni set ili ponovo pokretati aplikaciju. Flink ovaj problem rešava tako što omogućava korisnicima da koriste *streaming* obradu prilikom analize podataka. Ovaj mehanizam prikazan je na slici 7. [4]



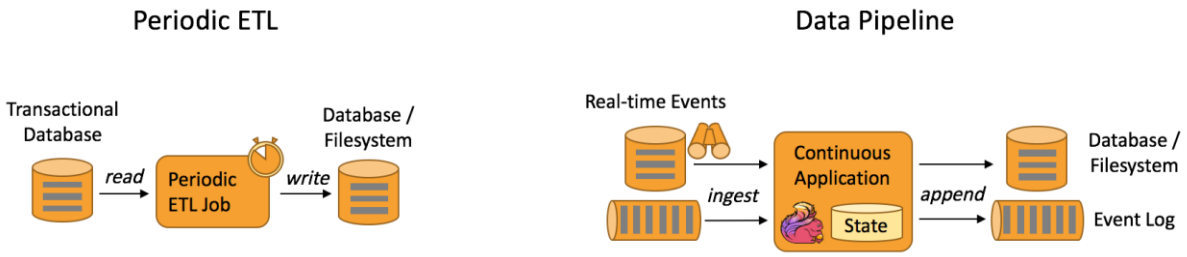
Slika 7. Odnos *batch* i *streaming* analize [4]

Prednosti ovakvog vida obrade se ne završavaju samo na kraćem vremenu. Upiti ne moraju da barataju sa ograničenim skupovima podataka, arhitektura je jednostavnija, a i aplikacije se oslanjaju na Flinkov mehanizam oporavka od greške. Tipične aplikacije ovog tipa su: monitoring kvaliteta, analiza *update-ova* i eksperimentalna evaluacija u mobilnim aplikacijama, kao i ad-hoc analiza podataka u konzumerskim tehnologijama. [4]

Data Pipeline aplikacije

Extract-transform-load (ETL) je čest pristup transformacijama i premeštanju podatak između različitih sistema za skladištenje. Često su ovakve akcije pokrenute za periodično premeštanje podatak iz transakcione u analitičku bazu podataka. Data pipeline-ovi rade poslove slične ETL-u. Razlika je u tome što su njihove akcije kontinualne, a ne periodične, pa se samim time koriste za izvore podatak koji konstantno izbacuju nove podatke. Primer ovkve

aplikacije može biti aplikacija koja će konstantno pisati podatke koji stižu sa *event stream-a* u bazu. Slika 8 prikazuje razliku između *ETL*-a i *data pipeline*-a. [4]



Slika 8. Periodični *ETL* i *Data Pipeline* [4]

Zaključak

Apache Flink nude veliki broj pogodnosti korisnicima i predstavlja veoma korisnu tehnologiju u *big data* svetu. Kroz veliki broj update-ova, Apache Flink je oblikovan da bude jedan od najraznovrsnijih alata za obradu podataka danas.

Uspešno izvršava *batch* i *streaming* obradu podatka sa minimalnim kašnjenjem obrade. Takođe, Apache Flink nudi veoma moćne opcije skaliranja sistema, po potrebi, a i sam određuje potrebne resurse za izvršavanje, tako da ne koristi više nego što mu je potrebno.

Apache Flink nudi veoma širok spektar slučajeva korišćenja, poput *event-driven* aplikacija, aplikacija za analizu podataka ili *data pipeline* aplikacija i sve zadatake i izazove koji dolaze sa tim aplikacijama izvršava veoma uspešno.

Literatura

[1] A Brief History of Flink: Tracing the Big Data Engine's Open-source Development, https://hackernoon.com/a-brief-history-of-flink-tracing-the-big-data-engines-open-source-development-87464fd19e0f (pristupljeno: 03.1.2022.)

[2] Apache Flink, Flink arhitektura, https://flink.apache.org/flink-architecture.html (pristupljeno: 28.12.2021.)

[3] Apache Flink, Flink aplikacije, https://flink.apache.org/flink-applications.html (pristupljeno: 03.01.2021.)

[4] Apache Flink, Flink slučajevi korišćenja, https://flink.apache.org/usecases.html (pristupljeno: 03.01.2021.)