Količina podataka koja danas putuje internetom je velika, ali i kompleksna. Svi ti podaci koji kolaju internetom se koriste za stvaranje izvještaja, donošenje zaključaka, kao i za razne analize kod mnogih tvrtki, institucija i sličnog. Pravi izazov je, zapravo, manipulirati tokom tih podataka te njihovom analizom.

Velike tvrtke razvile su, tijekom proteklih godina, zahtjevne metode koje im dozvoljavaju da prate razvoj tržišta, njihovu poziciju na tržištu, efikasnost usluga koje nude i slično. Kako bi to sve bilo moguće, potreban im je pristup velikim količinama aktualnih podataka.

S takvim količinama podataka, gotovo je nemoguće raditi na vlastitom računalu. „Streaming“ podataka u stvarnom vremenu je postao izazov za mnogo tvrtke.

[1] Big Dana Challanges, <http://www.dbjournal.ro/archive/13/13_4.pdf>

Ljudi skupljaju podatke od početka svog postojanja. Teško je pojasniti sve razloge za ovakav skok u količini podataka. Objašnjenje se može pronaći u nastanku interneta. Ljudi svakodnevno koriste Internet i njihova aktivnost se koristi za komercijalne i znanstvene svrhe. Posebno, pokušavaju se personalizirati prostori na internetu pojedincima korištenjem prikupljenih podataka. U globalu, nada se da, ako bi se svi ti podaci mogli ispravno i efektivno koristiti, znanost bi dosegnula svoj vrhunac, a tehnologija bi postala adaptivna, personalizirana i robustna. To bi, na primjer, mogli uočiti posebno u medicini, gdje bi se onda podaci jednog pacijenta mogli kombnirati i uspoređivati s podacima drugih te bi svatko mogao dobivati personalizirane tretmane, upravo onakve kakvi najviše odgovaraju pojedincu.

U radu s velikim količinama podataka postoje mnogi izazovi koji kreću od prikupljanja pa sve do analize i korištenja podataka. Neki od njih su rad s distribuiranim izvorima podataka, validacija podataka, rad s različitim formatima i strukturama, osiguravanja integriteta i sigurnosti, razvijanje metoda za vizualizaciju velikih skupova podataka i sl.

[2] Frontiers in Massive Data Analysis, [https://nap.nationalacademies.org/catalog/18374/frontiers-in-massive-data-analysis#](https://nap.nationalacademies.org/catalog/18374/frontiers-in-massive-data-analysis)

Što su opće veliki skupovi podataka? Postoje mnoge definicije. Jedna od njih povezuje velike skupove podataka s tri svojstva na slovo „V“ prema engleskim nazivima: volumen (engl. *volume*), brzina (engl. *velocity*) i raznolikost (engl. *variety*). Konkretno se to odnosi na povećanje u količini podataka, povećanje brzine kojom se oni stvaraju te povećanje broja formata i oblika u kojima se pojavljuju. Kasnije se u ovu definiciju dodalo i četvrto „V“, a o to je istinitost (engl. *veracity*). Istinitost uvodi pitanja ispravnosti i nesigurnosti u sadržaj podataka kao i rezultate analize podataka.

[3] <https://arxiv.org/pdf/1309.5821.pdf>

Danas se koriste mnoge tehnologije za rad s velikim skupovima podataka. Kako bih odabrao valjanu tehnologiju, važno je znati njena svojstva. Tvrtke koriste tehnologije za rad s velikim skupovima podataka kako bi dobile bolje uvide u svoj rad i donosili profitabilnije odluke. Tehnologije koje se koriste za rad s velikim skupovima podataka su zapravo softveri koji se primarno koriste za analizu, procese i izvlačenje informacija iz velikih skupova podataka složenih struktura kojima se ne može upravljati koristeći tradicionalne tehnologije za procesiranje podataka (kao npr. RDBMS, datotečni sustavi i sl.).

Tipovi tehnologija za rad s velikim skupovima podataka:

Operativne tehnologije

* To se odnosi na podatke koji se stvaraju iz svakodnevnih aktivnosti, poput Internet transakcija, društvenih platformi i sličnog. Takvi podaci su neobrađeni. Takve tehnologije koriste na primjer za online kupovine, za društvene mreže ili podatke po zaposlenicima.

Analitičke tehnologije

* Mogu se vidjeti kao modificirane varijante koje su kompleksnije od operativnih. Analitičke se tipično koriste kada se trebaju mjeriti performanse i kada se trebaju donijeti kritične poslovne odluke temeljene na izvještaju dobivenom analize. Koriste se za prognozu vremena, medicinska izvješća i slično.

Tehnologije se mogu podijeliti i na:

* Spremanje podataka
* Rudarenje podataka
* Analiza podataka
* Vizualizacija podataka

[4] <https://www.interviewbit.com/blog/big-data-technologies/>

Spremanje podataka

Zbog svojih veličina, velikim skupovima podataka se rukuje koristeći drugačija spremišta od tradicionalnih. Tvrtke poput Googlea i Amazona imaju velike podatkovne centre koji mogu spremati i obrađivati podatke s minimalnim kašnjenjem. To nam pokazuje da tradicionalni USB-ovi i vanjski diskovi nisu korisni za rad s velikim skupovima podataka.

The data warehouse

Skladište podataka (engl. *Data warehouse*) je postupak skupljanja i upravljanja podacima iz različitih izvora za sakupljanje informacija. Tipično se koriste za spajanje i analizu podataka iz različitih izvora.

The data lake

Jezero podataka (engl. *Data Lake*) je središnji spremišni repozitorij koji sprema podatke iz raznih izvora. Može spremati podatke u strukturiranom, polu-strukturiranom ili nestrukturiranom obliku.

Za razliku od jezera podataka, skladište podataka je repozitorij strukturiranih i filtriranih podataka koji se već oblikovao za posebnu svrhu.

NAS

NAS (engl. *Network Attached Storage*) je uređaj za spremanje podataka kojem se može pristupiti spajanjem na Internet. Sadrži procesore i operacijske sustave koji mu dozvoljavaju da vrti aplikacije i jednostavno dijeli datoteke između autoriziranih korisnika.

The Cloud

Pomoću oblaka, podaci se spremaju online i mogu se dohvatiti od bilo kuda, bez potrebe za direktnim pristupom računalu.

Spremište objekata

Svi podaci se spremaju u veliki repozitorij koji se može distribuirati preko više fizičkih spremišnih uređaja.

[5] <https://www.bocasay.com/how-to-store-big-data/>

Velike količine podataka neprestano okružuju ljude, počevši od korištenja društvenih mreža, poput Facebooka ili Instragrama, pa sve do zdravstvenih usluga, vremenskih prognoza, GPS-a. Kod svih njih neprestano teku podaci bez kojih bi njihov rad bio nemoguć. Aplikacije danas, prikupljanjem podataka, pokušavaju što više personalizirati svoje sadržaje korisnicima. Kako bi to bilo moguće, prikuplja se gotovo sve što korisnik radi dok koristi aplikaciju te se analizom i obradom odredi kakav će se sadržaj prikazati.

U nastavku ovog rada otkrivaju se problematike prikupljanja, analize i vizualizacije podataka te se prikazuje primjer toka podatak. Detaljnije prikazuje sljedeća slika:

SLIKA!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Slika na kojoj se prikazuje tekst, snimka zaslona, dijagram, crta

Opis je automatski generiran

Kao izvor podataka koristit će se Spotify API (<https://developer.spotify.com/documentation/web-api>). Spotify je aplikacija za slušanje glazbe koja prikupljanjem podataka o korisniku može preporučiti pjesme ili popise pjesama za reprodukciju. Podatke prikuplja Apache Kafka, distribuirana platforma otvorenog koda za strujanje događaja koju koriste tisuće tvrtki za stvaranje visoko učinkovitih podatkovnih cjevovoda [6]. Pomoću Confluent Platform platforme, stvara se Kafka poveznica (engl. *Kafka Connection*) s Microsoft Azure Cloudom. Confluent Platform je platforma koja omogućuje organiziranje i upravljanje podacima iz mnogo različitih izvora s jednim pouzdanim sustavom visokih performansi [7]. Podaci putuju i spremaju se u *json* obliku unutar Azure Blob Storagea, koji se koristi za masivnu i sigurnu pohranu objekata [8].

Azure Databricks se kasnije koristi za transformaciju podataka pomoću Apache Sparka i Apache Iceberga. Unutar Databricksa kreira se radni prostor u kojem se pišu naredbe, u ovom slučaju u PySparku. PySpark je Pythonov API namijenjen za Apache Spark. Za samu transformaciju podataka koristi se Spark. Podaci se prepisuju u Iceberg tablice te se u *parquet* obliku zapisuju natrag u Azure Blob Storage. Nakon prikupljanja, transformacije i pohrane podataka, podaci se vizualiziraju unutar Power BI-ja.

[6] <https://kafka.apache.org/>

[7] <https://docs.confluent.io/platform/current/overview.html>

[8] <https://azure.microsoft.com/en-us/products/storage/blobs>

Izvor podataka predstavlja početnu lokaciju na kojoj se podaci stvaraju. Konkretno, izvor podataka može biti baza podataka, neprekinuta datoteka (engl. *flat file*), mjerenja dobivena s uređaja, podaci prikupljeni s web izvora ili bilo koja od bezbroj statičkih ili tokovnih podatkovnih usluga kojih ima u izobilju na internetu [9].

Izvori podataka trebaju pomoći korisnicima i aplikacijama da se povežu i premjeste podatke tamo gdje trebaju biti. Oni prikupljaju relevantne tehničke informacije na jednom mjestu i skrivaju ih kako bi se potrošači podataka mogli usredotočiti na obradu i identificirati kako najbolje iskoristiti svoje podatke [9].

[9] <https://www.talend.com/resources/data-source/#:~:text=A%20data%20source%20is%20the,process%20accesses%20and%20utilizes%20it>.

Većina izvora podataka može se podijeliti u dvije glavne kategorije: strojni izvor podataka i datotečni izvor podataka[10].

Kod strojnog izvora podataka, podaci se stvaraju na korisnikovom uređaju. Taj uređaj može biti njegovo računalo, mobilni telefon, IoT ili neki drugi uređaj. Podaci su dostupni korisniku koji je trenutno prijavljen u sustav i ne mogu se dijeliti s drugim uređajima. Dalje se mogu dijeliti na korisnikove izvore podataka, kod kojeg su podaci dostupni samo određenom korisniku, i sistemske izvore podataka, kod kojeg su podaci dostupni svim sistemskim korisnicima. Neki primjeri su zapisi u mrežnom prometu, zapisi sustava i aplikacija, izlazi iz senzora, podaci o događajima na IoT uređajima, rezultati upisa na bazama podataka itd.[10].

Datotečni izvori podataka nisu dodijeljeni određenim strojevima, aplikacijama ili korisnicima. Mogu se dijeliti između uređaja, a obično su pohranjeni u zasebnim tekstualnim datotekama. Takvi izvori podataka uključuju tablice, tekstualne datoteke, PDF datoteke, slike, audio i video datoteke [10].

Podaci se prenose uz pomoć postojećih mrežnih protokola. Protokol za prijenos datoteka (engl. *File Transfer Protocol – FTP*) i protokol za prijenos hiperteksta (engl. *HyperText Transfer Protocol – HTTP*) najčešće su korišteni. Ostali protokoli za dohvaćanje podataka između sustava, posebice na webu, uključuju NFS (engl. *Network File Sharing*), SMB (engl. *Server Message Block*), SOAP (engl. *Simple Object Access Protocol*), REST (engl. *REpresentational State Transfer*) i WebDAV (engl. *Web-based Distributed Authoring and Versioning*) [10].

Druga metoda prijenosa podataka od izvora do odredišta je korištenje aplikacijskih programskih sučelja (engl. *application programming interface - API*) koje pružaju web stranice, umrežene aplikacije i druge usluge [10].

[10] <https://www.alphaservesp.com/blog/what-is-a-data-source-definitions-types-examples>

API je kod koji omogućuje komunikaciju dvaju softverskih programa. API definira kako razvojni programer treba zatražiti usluge od operacijskog sustava (OS) ili druge aplikacije i izložiti podatke unutar različitih konteksta i na više kanala.

API se sastoji od:

* Specifikacije koja opisuje kako se informacije razmjenjuju između programa u obliku zahtjeva i povratu potrebnih podataka
* Softverskog sučelja koji je napisan prema tim specifikacija i objavljen za korištenje

Za softver koji želi pristupiti značajkama i mogućnostima API-ja kaže se da ga poziva, a za softver koji stvara API kaže se da ga objavljuje[11].

[11] <https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API>

Spotify API je REST API koji se može koristiti za povlačenje Spotify podataka. Što točno znači da je API RESTful? Ukratko, ako se API shvati kao skup pravila koja dopuštaju jednom dijelu softvera da komunicira s drugim, tada se REST može shvatiti kao standardni skup pravila kojeg programeri slijede za stvaranje API-ja. Kako bi se API smatrao RESTful, mora ispoštovati određena pravila:

* Treba omogućavati **klijent-poslužitelj** arhitekturu sa zahtjevima kojima se upravlja putem HTTP-a.
* Svi zahtjevi trebaju biti **neovisni** jedni o drugima.
* API bi trebao podatke spremiti u **predmemoriju** (engl. *cache*).
* Treba postojati **uniformno sučelje** između komponenti tako da se informacije prenose u standardnom obliku.
* Klijent koji podnosi zahtjev ne mora znati komunicira li sa stvarnim poslužiteljem ili s posrednikom, tj. arhitektura je **slojevita**.
* Treba omogućiti slanje izvršnog koda s poslužitelja klijentu na njegov zahtjev.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver, Font

Opis je automatski generiran

Pravilo klijent-poslužitelj RESTful API-ja znači da bi trebalo biti moguće primiti dio podataka (resurs – engl. *resource*) kada se pristupi unaprijed definiranom URL-u [12].

Zahtjevom na Spotify API, mogu se dobiti podaci o glazbenicima, njihovim albumima, pjesmama, popisima za reprodukciju itd..

Jedan primjer unaprijed definiranog URL-a kojem se može pristupiti je sljedeći zahtjev za popisom za reprodukciju:

<https://api.spotify.com/v1/playlists/3cEYpjA9oz9GiPac4AsH4n>. Pristupanjem tom URL-u u nekom od preglednika dobije se *json* odgovor koji vjerojatno glasi:

{

"error": {

"status": 401,

"message": "No token provided"

}

}

Ono što se zapravo dogodilo je da je zahtjev odbijen zbog nevaljane autentifikacije. Kako bi se uspješno autentificirao, korisnik treba predati tzv. „*access token*“ ili pristupni token [13].

Autorizacija se odnosi na postupak davanja dopuštenja pristupa korisniku ili aplikaciji Spotify podacima i značajkama (npr. aplikacija treba dopuštenje korisnika za pristup njihovim popisima za reprodukciju) [15].

Prije početka rada sa Spotify API-jem potrebno je imati Spotify račun za razvojne programere. Taj je račun isti kao račun za Spotify aplikaciju i ne treba Spotify Premium verziju računa. Sljedeći korak je stvoriti aplikaciju unutar Spotify API-ja ( LINKKKK NA APIIII) klikom na *create an app* gumb. Kada se to učini moguće je dohvatiti javni i privatni ključ koji su potrebni kako bi se API mogao koristiti [14].

Kada se posjeti aplikacija koja se prethodno stvorila, tamo se mogu dohvatiti dva parametra: *client ID* i *client secret*. Oba su potrebna za autentifikaciju i dohvat pristupnog tokena. Nakon dohvaćanja tih dvaju parametara, može ih se spremiti lokalno u zasebnu datoteku, recimo *.env*.

Slika na kojoj se prikazuje tekst, snimka zaslona, crta, Font

Opis je automatski generiran

Pristup zaštićenim resursima određen je jednim ili više opsega (engl. *scope*). Opsezi omogućuju vašoj aplikaciji pristup određenim funkcijama (npr. čitanje popisa za reprodukciju ili samo strujanje podataka) u ime korisnika. Skup opsega koje ste postavili tijekom autorizacije određuje dopuštenja pristupa koja se od korisnika traži da odobri. Detaljne informacije o opsegima možete pronaći u dokumentaciji opsega (<https://developer.spotify.com/documentation/web-api/concepts/scopes>) [15].

Jedna je vrsta opsega, koji će se koristiti u primjeru, da se autentificira bez posebnog korisnika. Na taj način moguće je koristiti općenite značajke Spotifyja, a ne mogu se vidjeti statistike vezane uz posebnog korisnika, kao što su njegovi popisi za reprodukciju. Za takvu autentifikaciju potrebni su *client ID*  i *client secret*[14].

Nakon prijave u Spotify API sučelje, izrade aplikacije i dohvaćanja potrebnih vjerodajnica, slijedi pisanje programskog koda za stvaranje pristupnog tokena. Kod je pisan u *pythonu* te je ideja da taj kod bude dijeljen između nekoliko budućih programskih kodova kojim će se dohvaćati određeni podaci. Zbog toga je odabrano ime datoteke *common\_functions.py* što zapravo predstavlja dijeljene funkcionalnosti.

Slika na kojoj se prikazuje tekst, Font, snimka zaslona

Opis je automatski generiran

Prethodni isječak koda prikacuje učitavanje .env datoteke i spremanje *client ID* i *client secret* vjerojadajnica u dvije varijable. Sljedeće što želimo je dobiti pristupni token pomoću te dvije varijable.

Definiramo metodu *get\_token*:

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, softver

Opis je automatski generiran

Prethodni isječak koda prikazuje dohvaćanje pristupnog tokena slanjem POST zahtjeva na Spotify API. Slični primjeri dohvaćanja tokena vidljivi su pretraživanjem dokumentacije Spotify API-ja.

Pristupni token niz je koji sadrži vjerodajnice i dopuštenja koja se mogu koristiti za pristup određenom resursu (npr. izvođačima, albumima ili pjesmama) ili korisničkim podacima (npr. vašem profilu ili vašim popisima za reprodukciju) [15]. Korištenjem tokena jedan cURL zahtjev za dohvaćanjem zapisa GET metodom bi izgledao nešto poput:

curl --request GET \

'https://api.spotify.com/v1/tracks/2TpxZ7JUBn3uw46aR7qd6V' \

--header "Authorization: Bearer NgCXRK...MzYjw"

Ako se pozove *get\_token* metoda s ispravnim vjerdajnicama dobije se pristupni token koji vrijedi 60 minuta. Pri isteku 60 minuta dobije se poruka o nevaljaloj autorizaciji. Tada je potrebno ponovno zatražiti token.

Taj primjer se može isprobati u Postmanu. Postman je API platforma za izradu i korištenje API-ja. Postman pojednostavljuje svaki korak životnog ciklusa API-ja i usmjerava suradnju tako da možete brže stvarati bolje API-je [17].

Postman aplikacija može se preuzeti lokalno na računalo. Nakon pokretanja otvorit će se jednostavno i intuitivno sučelje koje će se koristiti za slanje prethodnog zahtjeva na Spotify API. Na slici **BR SLIKE** zaokružene su opcije na koje treba pripaziti, kao npr. odabir GET metode. Treba obratiti pozornost i na ispravno postavljanje tokena. Token koji je unesen rezultat je izvršavanja spomenute *get\_token* metode.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, broj

Opis je automatski generiran

Ako je zahtjev ispravno poslan, kao odgovor se dobije *json* zapis. Svaki zapis (engl. *track*) ima svoj ID, u ovom primjeru to je vrijednost 2TpxZ7JUBn3uw46aR7qd6V. Isto tako, vlastiti ID imaju i izvođači, albumi itd.. U sljedećem *json* odgovoru uočavamo da je prvo naveden album u kojem se zapis pojavljuje, njegov tip, ID, tržište na kojem se pojavljuje u obliku oznake (npr. oznaka za Španjolsku je ES). Nakon albuma vidi se i izvođač (engl. *artist*) koji izvodi taj glazbeni zapis te neki ključni podaci o njemu. Zatim slijede informacije vezane uz sam zapis, kao što su trajanje u milisekundama – *duration\_ms*, popularnost, tip, naziv i sl..

Slika na kojoj se prikazuje tekst, Font, snimka zaslona, dokument

Opis je automatski generiran

***PRIMJER JEDNOG ZAHTJEVA U POSTMANU? OPIS JEDNOG ODGOVORA?***

[12] možda prvo pričat što je REST api? <https://www.newline.co/courses/build-a-spotify-connected-app/okay-but-what-is-a-rest-api>

[13] <https://engineering.atspotify.com/2015/03/understanding-spotify-web-api/>

[14] <https://towardsdatascience.com/extracting-song-data-from-the-spotify-api-using-python-b1e79388d50>

[15] <https://developer.spotify.com/documentation/web-api/concepts/authorization>

<https://jit.ndhu.edu.tw/article/view/2261>

<https://www.hindawi.com/journals/tswj/2014/712826/>

<https://computools.com/how-is-big-data-collected/>

<https://codeit.us/blog/how-is-big-data-collected>

https://www.sciencedirect.com/science/article/pii/S1319157817300034

Prikupljanje velikih skupova podataka

Koliko se podataka dnevno stvori? U 2021. godini ljudi su stvarali 2.5 kvintilijuna bajtova podataka svaki dan. 650 milijuna novih Tweetova se dnevno objavljivalo u 2022. godini. Pretpostavlja se da će do 2030. godine broj IoT uređaja dostići 25.44 milijardi. Svaku sekundu se dogodi 99 000 pretraživanja na Googleu [17]. Svi ovi brojevi zapravo pokušavaju dočarati koliko podataka neprestano teče i koliko njih se koristi, a da ljudi toga nisu ni svjesni.

Svi ti podaci nisu uzaludni i zapravo su danas prijeko potrebni za rad mnogih tvrtki, ali i čovjeku za obavljanje svakodnevnih aktivnosti. Kako bi se podatke uopće moglo iskoristiti, prvo ih je potrebno prikupiti.

Prikupljanje velikih skupova podataka je metodički pristup skupljanja i mjerenja velikih količina informacija iz različitih izvora. Vrijednost tih podataka ne leži u njihovoj količini, već u važnosti koju nose za današnja poduzeća i poslove. Kolekcije velikih skupova podataka sadrže strukturirane, polu-strukturirane i nestrukturirane podatke koje su stvorili ljudi i računala [16].

Strukturirani podaci su čuvani u unaprijed definiranom formatu, dok nestrukturirani podaci postoje u onom obliku u kojem su stvoreni, kao na primjer u obliku objave na društvenim mrežama. Polu-strukturirani podaci kombinacija su strukturiranih i nestrukturiranih podataka [18].

Jedan primjer polu-strukturiranih podataka su *json* podaci jer ne zahtijevaju shemu. JSON (engl. *JavaScript Object Notation*) format je podataka koji se temelji na tekstu i mogu ga čitati ljudi i strojevi [19]. Široko je korišten te nudi dodatnu fleksibilnost za pohranjivanje podataka i postavljanje upita koji se ne pridržavaju uvijek fiksnih shema i vrsta podataka [20]. Primjer *json*-a prikazan je u prethodnom poglavlju.

[16] <https://www.techtarget.com/searchdatamanagement/feature/Big-data-collection-processes-challenges-and-best-practices#:~:text=Big%20data%20collection%20is%20the,and%20make%20critical%20business%20decisions>.

[17] <https://techjury.net/blog/how-much-data-is-created-every-day/>

[18] <https://k21academy.com/microsoft-azure/dp-900/structured-data-vs-unstructured-data-vs-semi-structured-data/#:~:text=Structured%20data%20is%20stored%20is,databases%20or%20other%20data%20table>

[19] <https://www.snowflake.com/guides/what-is-json>

[20] <https://cloud.google.com/blog/products/data-analytics/bigquery-now-natively-supports-semi-structured-data>

Dohvaćanje podataka od bilo gdje do neke lokacije na kojoj se planiraju koristiti, započinje procesom unosa podataka (engl. *data ingestion*). Alati koji se koriste za unos podataka su softverski alati koji korisnicima omogućuju prijenos podataka iz sustava na drugu lokaciju, kao što je baza podataka. Cilj im je da automatiziraju proces dohvaćanja podataka iz različitih izvora prije slanja tih podataka u željenu lokaciju. Nužni su za brži prijenos velikih količina nestrukturiranih podataka, bez potrebe ručnog unosa ključnih informacija [21]. Kada bi se prijenos podataka radio ručno, troškovi u obliku vremena i novca bi znatno skočili.

Neke prednosti korištenja alata su:

* Različiti izvori: Alati mogu sakupljati podatke iz različitih izvora, uključujući baze podataka, web usluge, datoteke i redove poruka (engl. *message queues*). Podaci se mogu prikupljati u raznim formatima, kao što su JSON, CSV ili XML.
* Automatizacija i raspoređivanje zadataka: Automatizacija i raspoređivanje važna je značajka koja omogućuje automatizaciju i postavljanje vremena obavljanja zadataka, bez potrebe za ručnom intervencijom.
* Rješavanje i oporavak od pogrešaka: Pogreške se lako mogu dogoditi prilikom prikupljanja velikih količina podataka iz više različitih izvora. Mnogi alati nude opcije kao što su ponovno pokretanje nakon pogreške prilikom pokušaja dohvata podataka [21].

Podaci se mogu unositi u obliku toka podataka (engl. *stream*) ili u serijama podataka (engl. *batch*). Kada se podaci šalju u serijama unose se skupina podataka za skupinom, a svaka slijedi nakon određenog intervala. Kod unosa u toku koji se izvršava u stvarnom vremenu, svaki podatak se unosi nakon što ga je izvor emitirao [22].

MYB SLIKA BATCH VS STREAM

Tipovi

* Web pretraživači (engl. *web crawlers*): To su alati koji sakupljaju podatke iz web stranica skeniranjem i povlačenjem dostupnih informacija.
* Aplikacije za strujanje podataka: Takvi alati dozvoljavaju prijenos podataka u stvarnom vremenu pomoću toka zapisa.
* Skripte ili ručno izrađen kod: Alati dozvoljavaju korisnicima pisanje vlastitih skripti za sakupljanje podataka.
* ETL (engl. *extract, transform, load*) alati: Oni povlače podatke iz različitih izvora, transformiraju ih u korisne formate i na kraju učitavaju u središnju bazu podataka ili repozitorij.
* Alati za repliciranje baza podataka: Oni repoliciraju strukture i sadržaj jedne baze u drugi primjerak te baze podataka.
* Alati za rad s oblacima: Oni na siguran način i čuvajući integritet podataka, unose podatke u okruženja u oblaku [21].

Neki od poznatijih alata su Improvado, Apache Kafka, Hevo, Apache NiFi, Apache Flume, Rivery, Precisely Connect, Apache Storm i drugi. Improvado je alat koji omogućuje prikupljanje iz različitih izvora i obavljanje ETL-a nad podacima te konačno spremanje u skladište podataka (engl. *data warehouse*). Sučelje mu se jednostavno koristi, a uglavnom se koristi u tvrtkama za analize prodaje i oglašavanja prikupljanjem svih podataka na jedno mjesto [23]. Apache NiFi je snažan i skalabilan alat za preusmjeravanje i transformaciju podataka. Dizajniran za automatiziranje toka između dvaju softverskih sustava. Za rad mu nije potrebna shema, što znači da Nifi procesor je sam odgovoran za interpretiranje sadržaja podataka koje primi [24]. Apache Flume je distribuirana i otporna usluga za učinkovito prikupljanje, agregiranje i premještanje velikih količina „logiranih“ podataka. Za nastavak primjera koji se prolazi u ovom radu, koristi se Apache Kafka. Apache Kafka je poznat po svojim visokim performansama, koje mu omogućuju rad s tisućama poruka u sekundi [24]. Više o Kafki u nastavku rada.

Kako bi se odabrao odgovarajući alat, potrebno je znati njegove mogućnosti i znati što je potrebno u vlastitom slučaju. Prije odabira potrebno je razmisliti koju vrstu podataka želimo dohvaćati i kako ih nakon toga želimo transformirati. Također je važno razmisliti i budžetu i vremenskim ograničenjima.

[21] <https://sourceforge.net/software/data-ingestion/>

[22] <https://hevodata.com/learn/data-ingestion-tools/>

[23] <https://datavid.com/blog/data-ingestion-tools>

[24] <https://www.integrate.io/blog/top-data-ingestion-tools/>

APACHE KAFKA

Prije objašnjavanja što je Apache Kafka, kako radi i kako se koristi u ovom radu, važno je razumjeti koncept slanja poruka korištenjem objavi-pretplati sustava (engl. *publish/subscibe*). Objavi-pretplati ili prema engleskim nazivima skraćeno pub/sub slanje poruka je proces slanja poruka u kojem pošiljatelj (engl. *publisher*) podatka (tj. poruke) ne šalje poruku određenom primatelju, nego pošiljatelj nekako klasificira podatak i primatelj se pretplaćuje na primanje određenih klasa podataka. Takvi sustavi često imaju posrednika (engl. *broker*), tj. središnju točku na koju se poruke objavljuju.

Mnogi slučajevi pub/sub sustava započnu s jednostavnim redovima poruka (engl. *message queue*). Na primjer, stvoriš aplikaciju koja treba slati informacije negdje, pa otvoriš direktnu vezu od svoje aplikacije na aplikaciju koja prikazuje te informacije. Nakon nekog vremena, odlučiš da bih te informacije, osim prikaza, htio i analizirati. Iz tog razloga, pronađeš drugu aplikaciju koja će to obavljati i sada tvoja aplikacija šalje podatke u druge dvije. Recimo, da sada poželiš dodatno proširiti svoj sustav, on već postaje kompleksan. Kako bi riješio taj problem, odlučiš između dodati pub/sub sustav.

Što ako sada odlučiš slati neku drugu vrstu informacija, recimo da si prethodno slao mjerenja, a sada želiš slati logove, tj. zapise. Ali, ne želiš ih slati svima ili možda ne u istom obliku. Rješenje s pub/sub sustavom je u redu, ali dolazi do pojave dupliciranja. Kao rješenje ovog problema dolazi Apache Kafka [27].

ODE DODAT SVE SLIKE I PRIMJERE

**[27] Knjiga kao izvor**

Apache Kafka je originalno stvoren u LinkedInu kako bi obrađivao zapise u stvarnom vremenu s kašnjenjem ne više od nekoliko sekundi. Dizajniran je da bude distribuiran, skalabilan, izdržljiv i sustav bez pogrešaka s visokom propusnošću. LinkedIn se jako oslanja na skalabilnost i pouzdanost Kafke za slučajeve poput nadgledanja mjerenja sustava, tradicionalnih slanja poruka (engl. *traditional messaging*) ili praćenju rada stranice. Trenutno, zbog svoje jake izdržljivosti i visoke propusnosti s niskim kašnjenjem, Apache Kafka je široko primijenjena u Internet kompanijama [25].

Kafka je rješenje stvarnim softverskih problema današnjice, kao što su, nošenje s velikim količinama informacija i njihovo brzo preusmjeravanje prema potrošačima (engl. *consumer*). Kafka se ponaša kao posrednik između dvaju sustava: proizvođača (engl. *producer*) i potrošača (engl. *consumer*) [26]. Jedinica podatka u Kafki se zove poruka. Poruka se može zamisliti kao jedan zapis u bazi podataka, a Kafka su vidi kao polje bajtova pa podaci sadržani unutra ne moraju biti nekog određenog formata ili značenja [27].

Kafka čuva poruke u kategorijama koje se zovu „teme“ (engl. *topic*)*.* Tema je dio Kafke koji sprema i objavljuje poruke. Sve Kafka poruke se organiziraju u teme. Proizvođači su procesi koji objavljuju poruke na Kafka teme, a potrošači procesi koji se pretplaćuju na Kafka teme te dohvaćaju te poruke. Kafka tema se dijeli na više particija (engl. *partitions*) [28]. Kafka se vrti na klasteru (engl. *cluster*) koji se sastoji od jednog ili više poslužitelja od kojih se svaki zove broker, tj. posrednik [26]. Kafka brokeri su osmišljani da rade kao dio klastera. Unutar klastera, jedan broker je upravitelj klastera te je odgovoran za zadatke kao što su dodjeljivanje particija brokerima i nadgledanje pogrešaka drugih brokera.

Za svaku temu, Kafka klaster održava particiju (engl. *partition*) za skaliranje, paralelizam i provjeru pogrešaka. Svaka particija je uređeni, neizmjenjiv slijed poruka. Kafka se obično sastoji od nekoliko klastera. Kako bi se balansirao teret, teme se dijele na nekoliko particija i svaki broker čuva jednu ili više od tih particija. Također, više proizvođača i potrošača mogu objavljivati, odnosno povlačiti poruke u isto vrijeme [26]. Kafka koristi particije kako bi se teme skalirale na nekoliko poslužitelja te da proizvođači mogu slati poruke paralelno, tj. da potrošači mogu paralelno čitati [25].

Svaka poruka u particiji ima svoj sekvencijalni broj koji se zove pomak (engl. *offset*). [26]. Potrošač se pretplaćuje na jednu ili više tema i čira poruke onim redoslijedom kojim su došli u particiju. U isto vrijeme, prati koje je poruke već primio na način da pamti pomak poruke. Pomak je broj koji se konstantno povećava i to je još jedan metapodatak kojeg Kafka nadoda na poruku kada se prenosi u particiju. Zbog toga što pamti pomak, potrošač može stati i nakon nekog vremena nastaviti bez da izgubi mjesto na kojem je stao.

Recimo sada nešto više o proizvođačima i potrošačima.

Već je rečeno da proizvođači objavljuju podatke na teme po svom izboru. Oni su odgovorni za biranje koje poruke se dodjeljuju kojoj particiji jedne teme [26]. Pretpostavljeno je da proizvođač balansira poruke jednako na sve particije teme. U nekim slučajevima, proizvođač će usmjeriti poruke na određenu particiju. To se obično događa kada se kod poruka koristi ključ koji se mapira na točno specifičnu particiju [27].

Poruke se obično šalju na dva načina: u redovima poruka (engl. *queue*)ili u pub/sub modelu. Kada se koristi redovi, potrošači mogu čitati s poslužitelja i svaka poruka ide jednom od potrošača. Kod pub/sub načina rada, poruka se šalje svima. Kafka nudi apstrakciju koja generalizira oba načina rada uvođenjem potrošačkih grupa (engl. *consumer group*). Potrošači se označe potrošačkom grupom i svaka poruka koja se objavi na temu se dostavlja jednom potrošaču unutar pretplaćene potrošačke grupe. Ako su svi potrošači unutar iste potrošačke grupe, onda Kafka radi kao da se radi o redovima poruka te se teret raspoređuje na potrošače. Ako su svi potrošači u zasebnim potrošačkim grupama, onda Kafka radi kao pub/sub model i sve su poruke odašiljane svim potrošačima [26]. Dakle, kada se potrošačka grupa pretplati na temu, svaki potrošač unutar grupe će dohvaćati poruke paralelno iz različitih podskupova particija u toj temi. Potrošač može dohvaćati poruke iz više particija, dok jednu particiju može konzumirati samo jedna instanca potrošača iz jedne potrošačke grupe. Ipak, različite potrošačke grupe mogu nezavisno dohvaćati iste poruke i nije potreba nikakva koordinacija među potrošačkim grupama. Rezultat toga je da broj particija je zapravo maksimalni broj potrošačkih grupa i očito je da broj potrošača u potrošačkoj grupi ne bi trebao biti veći od broja particija u temi za koju se pretplatila [25].

Kafka brokeri su bez stanja (engl. *stateless*) p akoriste Zookeeper za održavanje stanja njihovog klastera. Zookeeper se koristi za upravljanje i koordiniranje Kafka brokera. Zookeeper usluge se uglavnom koriste da obavijeste proizvođača i potrošača o prisutnosti novog brokera ili o pogreški u brokeru [28]. Kafka osigurava pretpostavljeni i jednostavni konfiguracijski dokument Zookeepera koji se koristi za pokretanje njegove instance [26].

Značajke

Glavna značajka Kafke je **čuvanje** podataka određeni period vremena. Kafka brokeri imaju postavljene vrijednosti čuvanja podataka unutar tema na određeno vrijeme, na primjer na 4 dana, ili dok se ne dosegne određena veličina u bajtovima. Jednom kada se taj limit dosegne, poruke su istekle i one se brišu [27].

Pored toga, Kafka može primati podatke **iz više proizvođača**, bez obzira koriste li se oni istim ili različitim temama. To svojstvo je korisno za sustave koji žele ujediniti podatke iz mnogih frontend sustava i učiniti ih konzistentnim [27]. Osim toga može slati podatke **većem broju potrošača**. Uz to, potrošači, zbog svojstva čuvanja podataka, **ne moraju uvijek raditi u stvarnom vremenu**. Mogu dohvatiti podatke i kasnije koristeći se pomakom [27].

Kafka je **skalabilna**. Može se početi raditi s jednim brokerom i razviti veliki klaster s više stotina brokera koji rastu preko vremena dok se podaci skaliraju. Ako se dogodi da proizvođač proizvodi znatno više nego što potrošač može primiti, podaci će se akumulirati u Kafki sve dok ih potrošač ne uspije primiti [27].

Još jedna važna značajka podrška za **različite formate i tipove podataka**. Tipovi podataka koji su podržani variraju između različitih baza podataka i drugih skladišnih sustava [27].

Može raditi s **velikim skupovima podataka**. I poznat je po svom **niskom kašnjenju [28]**.

MOŽEŠ PISATI I O ETL I ELT??

S Kafkom je moguće stvoriti i ETL sustav, obavljanjem transformacija unutar Kafke. Ipak, mana ovog pristupa je što ne žele svi potrošači transformirati podatke na isti način. Zato se može koristiti drugi pristup – ELT (engl. *extract, load, transform*). ELT znači da podaci u podatkovnom cjevovodu prolaze minimalne transformacije s ciljem da se osigura da podacci stignu na cilj što sličiniji onom obliku u kakvom su primljeni. Tako potrošači mogu primati nestrukturirane podatke i kasnije ih transformirati po želji [27].

Primjeri korištenja Kafke:

Twitter koristi kafku kao infrastukturu za obradu toka. U LinkedInu se koristi za strujanje podataja. Ti podaci se koriste za različite proizvode kao što su offline analitički sustavi. Yahoo! Koristi kafku za analitiku u stvarnom vremenu. Netflix, kojim dnevno prođe miljarde podataka, koristi Kafku za prikupljanje podataka[28].

Loše strane /mane

Strma krivulja učenja i tehnološka složenost – Kafka je kompleksna za postaviti, održavati i dizajnirati njene cjevovode. Kako bi ju se savladalo treba vremena, čak i osobama s tehnološkom predznanjem.

Potreba za tehnološkim stručnjacima – Iz sličnih razloga kao što je već navedeno, Kafku je potrebno poznavati kako bi se s njom moglo upravljati. Iz tog razloga su potrebni stručnjaci, što predstavlja dodatne troškove, ako takvi već ne postoje u timu.

Nema upravljačkih i nadzornih alata – Nema svoje sučelje za nadzor rada. Tehničko osoblje onda treba provesti puno vremena u analizi i provjeri ispravnog slanja poruka. Postoje rješenja koja se mogu koristiti zajedno s Kafkom kako bi se izbjegao ručni pregled rada.

Problemi rada Zookeepera – Ovisnost o Zookeeperu može biti mana samim time što nadodaje složenosti već složenoj Kafki. Također, ne dozvoljava Kafki da ima više od 200 000 particija i usporava rad brokera kada se priključuje ili kada napušta klaster. Trenutno se radi na novom rješenju bez Zookeepera [29].

[25] <https://ieeexplore.ieee.org/abstract/document/8855525> -28

[26] <http://ijsetr.com/uploads/436215IJSETR3636-621.pdf> - 29

[28]<https://ieeexplore.ieee.org/abstract/document/8533771?casa_token=EC59Qiqqb6MAAAAA:NkcCf_KH8r8J3xlHFniTZCYzzxPzjobv_yFHDodBbLFQ-voLbAbQTX56sa0NoGMEevlADG9M1n8> IMA DOBREEE SLIKEEEEE -30

[29] <https://www.altexsoft.com/blog/apache-kafka-pros-cons/> -31

**„How we move the data becomes nearly as important as the data itself.“ [27]**

1. **Opisat kako se kafka može koristiti s pythonom, onda što se dodalo u kod od prije (producer)  
   kako to radi?**
2. **Što je confluent platforma, kako radi i kako se pokreće**
3. **Pokretanje pyhton skripti**
4. **Prikaz platforme i prikaz stizanja poruka**
5. **Kafka connect na azur**

Apache Kafka može se koristiti samostalno ili s dodatnom tehnologijom iz Confluenta [31]. Apache Kafka i Confluent Kafka se oboje temelje na istoj Apache Kafka tehnologiji, ali se razlikuju u nekim značajkama [30]. Confluent su razvili isti inženjeri kao i Apache Kafku. Cofnluent pruža upravljane usluge izgrađene na Kafki koje je timovima lakše implementirati, održavati i nadzirati [32]. Apache Kafka je streaming platforma otvorenog koda koja se koristi za strujanje i obradu podataka. Confluent Kafka je platforma koja pruža komercijalnu verziju Apache Kafke s dodatnim značajkama. Confluent kafka pruža širok raspon značajki kao što su skalabilnost, tolerancija grešaka, visoka propusnost i niska razina kašnjenja. Osim toga, Confluent Kafka pruža dodatne značajke kao što su replikacija, sigurnost i nadzor.

I Apache Kafka i Confluent Kafka vrlo su skalabilne i učinkovite platforme koje mogu obraditi velike količine podataka. Apache Kafka ne pruža nikakve sigurnosne ili nadzorne značajke, dok Confluent Kafka pruža oboje [30]. Confluent ima dio ponude koji je besplatan samo onima s licencom, a neki dijelovi su dostupni besplatno svima. Većina tvrtki koriste Confluent verziju Kafke [32]. Ti dijelovi, koji su besplatni, ne uključuju Confluent Cloud opciju. Uključuju ugrađeno sučelje naredbenog retka (engl. *command line interface* – *CLI*) u više programskih jezika i više Kafka konektora. Ipak, iako koristi Kafkine temeljne funkcije, Confluent je ograničen u smislu prilagodbe i personalizacije. Platforma također nema najbolju reputaciju kada je u pitanju tolerancija na pogreške. A budući da je Confluent upravljana verzija Apache Kafke, većina Kafkinih nedostataka vrijedi i ovdje. Kafka je primarni alat koji većina tvrtki koristi pri postavljanju ETL cjevovoda u stvarnom vremenu. Kafka nije nužno ETL alat, ali se može koristiti za njihovo stvaranje. Confluent je sjajan primjer kako se Kafka može koristiti za stvaranje platfomrne koja na neki način sliči ETL-u, ali je sveukupno još uvijek širi okvir [32].

[30] <https://www.alibabacloud.com/tech-news/kafka/34u-what-is-the-difference-between-apache-kafka-and-confluent-kafka#:~:text=Both%20Apache%20Kafka%20and%20Confluent%20Kafka%20are%20highly%20scalable%20and,provides%20security%20and%20monitoring%20features>. - 32

[31] <https://dattell.com/data-architecture-blog/comparing-confluent-kafka-and-apache-kafka/>

[32] <https://estuary.dev/confluent-kafka-vs-apache-kafka/#what-is-apache-kafka> -33

Confluentna platforma i Confluent Kafka povezani su, ali različiti koncepti unutar ekosustava Apache Kafka. Evo pregleda svakog od njih:

Confluent Platform: Confluent Platform je komercijalna ponuda izgrađena oko Apache Kafke. Pruža dodatne značajke i alate na razini poduzeća za poboljšanje Kafkinog ekosustava. Confluent Platforma uključuje Apache Kafku kao svoju temeljnu komponentu, ali također dodaje nekoliko drugih komponenti kako bi pružila potpunu platformu za strujanje. Ove dodatne komponente uključuju:

Spomenimo prvo ZooKeepera kao tehnologiju pomoću koje Apache Kafka radi. Platforma se brine za njezino postavljanje i pokretanje. Pored toga, platforma pokreće i ksqlDB komponentu. To je SQL mehanizam koji nudi jednostavno i interaktivno SQL sučelje za obradu toka na Kafki, bez potrebe pisanja koda u programskom jeziku kao što su Java ili Python.

Kako bi se podaci mogli povlačiti ili spremati u baze podataka poput MySQL ili u skladišta na oblacima, koriste se Kafka konektori. Platforma za to nudi korištenje Kafka Connect API-ja za povezivanje Kafke s drugim sustavima. Confluent nudi korištenje i komercijalnih i licenciranih konektora [34].

Jedan od izazova povezivanja dvaju sustava je osiguravanje kompatibilnosti podataka. To se može ostvariti dogovorom oko zajedničkog formata za poruke koji se naziva shema.

Schema Registry komponenta pruža spremišta za upravljanje i provjeru ispravnosti shema. Proizvođači i potrošači Kafkinih tema mogu koristiti sheme kako bi osigurali dosljednost i kompatibilnost podataka kako se sheme razvijaju. Schema Registry održava bazu podataka svih shema koje su upisane u teme u klasteru za koji je odgovoran. Kada je proizvođač konfiguriran za korištenje Schema Registry sheme, on Schema Registryju predstavlja shemu nove poruke. Ako je ista kao posljednje proizvedena poruka, slanje je dozvoljeno. Ako se razlikuje od posljednje poruke, ali odgovara pravilima kompatibilnosti definiram za temu, produciranje i dalje može uspjeti. Ali, ako krši pravila kompatibilnosti, takva proizvodnja se odbacuje.

Isto tako na strani potrošača. Ako potrošač pročita poruku koja ima nekompatibilnu shemu od verzije koju potrošač očekuje, Schema Registry će mu reći da ne konzumira poruku [35].

Confluent REST Proxy pruža RESTful sučelje za Apache Kafka klaster, olakšavajući proizvodnju i konzumiranje poruka, pregled stanja klastera i izvođenje administrativnih radnji bez korištenja izvornog Kafka protokola ili klijenata.

Neki primjeri korištenja su:

* Izvještavanje Kafki o podacima s bilo koje frontend aplikacije koja je izgrađena na jeziku kojeg ne podržavaju služeni klijenti Confluenta
* Ubacivanje poruka u okvir za obradu toka koji još ne podržava Kafku
* Skriptiranje administrativnih radnji [36]

Za kraj spomenimo Control Centar. To sustav temeljen na GUI-u za upravljanje i nadzor Kafke. Omogućuje jednostavno upravljanje Kafka Connectom, stvaranje, uređivanje i upravljanje vezama s drugim sustavima. Također, omogućuje praćenje toka podataka od proizvođača do potrošača, osiguravajući da je svaka poruka isporučena i mjeri koliko je vremena potrebno za isporuku poruka [34].

Za potrebe ovog rada odabrana je Confluent Kafka tehnologija. Kako bi se koristila lokalno, potrebno je instalirati i postaviti Confluent Platform platformu. Instalacija je moguća koristeći se sljedećim koracima. Potrebno je preuzeti paket prateći upute na sljedećoj poveznici: POVEZNICAAAAAA i raspakirati.   
  
Trebali bi se stvoriti sljedeći direktoriji:

/bin/ => Driver skripte na pokretanje i zaustavljanje usluga

/etc/ konfiguracijski dokumenti

/lib/ systemd usluge

/libexec/ više-platformske CLI binaries???

/share/ jars i dozvole

/src/ izvorni dokumenti koji zahtjevaju platformski nezavisnu izgradnju

Postavite varijablu okruženja za početni direktorij Confluent Platforme:

export CONFLUENT\_HOME=<The directory where Confluent is installed>

Dodajte direktorij bin Confluent Platform u svoj PATH:

export PATH=$PATH:$CONFLUENT\_HOME/bin

Provjerite jeste li ispravno postavili varijablu CONFLUENT\_HOME izvođenjem naredbe confluent:

confluent –help

Ako želite pokrenuti Confluent Platformu u svrhu testiranja i istraživanja, to možete učiniti tako da konfigurirate varijablu CONFLUENT\_HOME, a zatim koristite naredbu za pokretanje konfluentnih lokalnih usluga, koja će pokrenuti Confluent Platformu lokalno u ZooKeeper modu.

Ako želite koristiti konfluentne lokalne naredbe, morate imati instaliranu Javu 11 ili 8 (nizovi verzije 1.11 ili 1.8).

confluent local services start

Starting Zookeeper

Zookeeper is [UP]

Starting Kafka

Kafka is [UP]

Starting Schema Registry

Schema Registry is [UP]

Starting Kafka REST

Kafka REST is [UP]

Starting Connect

Connect is [UP]

Starting KSQL Server

KSQL Server is [UP]

Starting Control Center

Control Center is [UP]

Tijekom pokretanja u ZooKeeper modu, brokeri se registriraju u ZooKeeperu kako bi postali član klastera.

Idite do Kontrolnog centra na http://localhost:9021. Može proći minuta ili dvije dok se Control Center ne pokrene i učita [33].

[33] <https://docs.confluent.io/platform/current/installation/installing_cp/zip-tar.html#prod-kafka-cli-install>

[34] <https://docs.confluent.io/platform/current/platform.html>

[35] <https://developer.confluent.io/learn-kafka/apache-kafka/schema-registry/>

[36] <https://docs.confluent.io/platform/current/kafka-rest/index.html>