

ZAHVALA

Zahvaljujem mentoru, izv.prof.dr.sc. Borisu Milašinoviću, na podršci i mentorstvu tijekom izrade mog diplomskog rada. Posebno se želim zahvaliti svojoj obitelji i prijateljima koji su mi pružali neprestanu motivaciju i potporu tijekom cijelog mog studiranja. Veliko hvala svima.

Sadržaj

ZAHVALA.....	1
Uvod	1
1. Veliki skupovi podataka	2
1.1. Općenito o velikim skupovima podataka	2
1.2. Uvod u praktični primjer	3
2. Izvor podataka	5
2.1. Tipovi izvora podataka	5
2.2. Prijenos podataka pomoću API-ja	6
2.3. Spotify API.....	6
2.3.1. Spotify API – autorizacija korisnika	7
2.3.2. Spotify API – primjer slanja zahtjeva u Postmanu.....	9
3. Dohvaćanje velikih skupova podataka	19
3.1. Prikupljanje i tipovi podataka.....	20
3.2. Alati za dohvaćanje podataka	20
3.2.1. Tipovi i stvarni primjeri alata za dohvaćanje podataka.....	21
3.3. Apache Kafka	22
3.3.1. Model objavi-pretplati	22
3.3.2. Uvod u rad Apache Kafke	23
3.4. Dohvaćanje podataka sa Spotify API-ja pomoću Kafke	26
3.4.1. Pokretanje Confluent Platforme	27
3.4.2. Pokretanje Spotify API Kafka proizvođača.....	30
3.4.3. Postavljanje Azure Blob Storage Sink Connectora	33
3.5. Prednosti i mane korištenja Apache Kafka tehnologije.....	35
4. Skladištenje velikih skupova podataka.....	37
4.1. Različite vrste sustava za pohranu.....	37

4.2.	Skladištenje u oblacima	38
4.3.	Azure Blob Storage	38
4.4.	Stvaranje potrebnih resursa u Azure Cloudu	39
5.	Transformacija velikih skupova podataka	43
5.1.	Općenito o transformaciji velikih skupova podataka	43
5.2.	Azure Databricks	44
5.3.	Uvod u Apache Spark.....	46
5.4.	Primjena Apache Sparka	47
5.4.1.	Povezivanje s Azure Blob Storageom	47
5.4.2.	Strukture podataka u Apache Sparku	48
5.4.3.	Čitanje podataka iz Azure Blob Storagea u Spark DataFrame.....	49
5.4.4.	Lijena evaluacija u Apache Sparku	52
5.4.5.	Obavljanje transformacija nad podacima	52
5.5.	Apache Iceberg	56
5.5.1.	Struktura Apache Iceberga	56
5.5.2.	Svojstva Apache Iceberga	58
5.5.3.	Primjena Apache Iceberga.....	60
6.	Vizualizacija velikih skupova podataka	63
6.1.	Općenito o vizualizaciji velikih skupova podataka	63
6.2.	Primjeri primjene vizualizacije i primjeri poznatih alata	64
6.3.	Microsoft Power BI	65
6.3.1.	Postavljanje i korištenje Microsoft Power BI-ja.....	66
	Zaključak	75
	Literatura	76
	Sažetak.....	80
	Summary.....	81

Uvod

Svijet se danas sve više orijentira korištenju novih tehnologija kojima je cilj čovjeku olakšati život čineći svakodnevne aktivnosti jednostavnijima. Ta priča se rasteže od najjednostavnije poruke poslane preko društvene mreže, kao što je Instagram, pa sve do sakupljanja podataka sa satelita koji okružuju Zemljinu sferu. Tehnologija je tako dotakla sve, od ruku velikih stručnjaka do običnog đaka te je broj ljudi koji su povezani i komuniciraju u svakom trenutku naglo porastao. Informacije neprestano kruže komunikacijskim mrežama tolikom brzinom i količinom da su se počele prilagođavati pojedincu kako ga ne bi zamarale onim informacijama koje ga možda uopće neće zanimati.

Zbog tog naglog razvitka tehnologije i njene povećane pristupačnosti velikom broju ljudi, ovo stoljeće se upoznalo s novim problemom, a to je problem pojave velikih količina podataka. Taj se problem nije mogao ignorirati jer se prikupljanjem velikih količina podataka olakšao daljnji razvoj znanosti kao što su medicina, astronomija, meteorologija i slične. Stručnjaci su odjednom trebali pronaći rješenja koja će se moći nositi s radom s velikim skupovima podataka. Počeli su se razvijati alati za prikupljanje, skladištenje, analizu, dijeljenje, transformaciju i vizualizaciju velikih skupova podataka.

U nastavku ovog rada pratit će se primjer prikupljanja podataka iz *Spotify API*-ja, njihovo skladištenje, transformacija i na kraju vizualizacija. Svi koraci rada s velikim skupovima podataka i korištene tehnologije detaljnije su pojašnjene u zasebnim poglavljima. U prvom poglavlju uvodi se pojam velikih skupova podataka kao i detaljniji opis primjera koji se proteže kroz rad. Nakon toga, u drugom poglavlju, pojašnjavaju se izvori podataka, a detaljnije se opisuje *Spotify API* kao izvor podataka. U trećem poglavlju iznose se načini dohvaćanja podataka. Kao alat za prikupljanje podataka iz *Spotify API*-ja odabran je *Apache Kafka* pa su u ovom poglavlju objašnjeni detalji njegovog rada. Četvrto poglavlje pojašnjava različite mogućnosti skladištenja velikih skupova podataka s naglaskom na skladištenje u *Azure Blob Storage*. Transformacija velikih skupova podataka objašnjena je u petom poglavlju koje, osim općenitih tehnologija, opisuje način rada *Apache Sparka* unutar platforme *Azure Databricks*. Posljednje poglavlje iznosi važnost vizualizacije velikih skupova podataka, a detaljnije objašnjava korištenje vizualizacijskog alata *Microsoft Power BI*.

1. Veliki skupovi podataka

Ljudi sakupljaju podatke od početka svog postojanja. Jedan takav primjer su zapisi u obliku špiljskih crteža, a najstariji su stari i preko 40000 godina [1]. Danas je uočljiv ogroman skok u količini podataka, a objašnjenje za to je sve veće korištenje interneta. Ljudi ga svakodnevno koriste, a njihove aktivnosti se prate te se prikupljeni podaci koriste za komercijalne, znanstvene i razne druge svrhe.

1.1. Općenito o velikim skupovima podataka

Postoje mnoge definicije koje se povezuju uz velike skupove podataka. Jedna od njih velikim skupovima podataka dodjeljuje tri svojstva na slovo „V“ vodeći se engleskim riječima. Prvo svojstvo je volumen (engl. *volume*) koji nam naglašava veličine tih podataka u bajtovima [2]. Zbog svojih veličina, velikim skupovima podataka rukuje se koristeći spremišta koja mogu podnijeti tolike količine podataka. Tvrtke poput *Googlea* i *Amazona* imaju velike podatkovne centre koji mogu spremati i obrađivati podatke s minimalnim kašnjenjem. Drugo svojstvo je svojstvo brzine (engl. *velocity*). Pod tim se misli na brzinu kojom podaci dolaze. Neki podaci dolaze u stvarnom vremenu, dok drugi mogu dolaziti u serijama [3]. Posljednje svojstvo je raznolikost (engl. *variety*) podataka. Prije su se podaci sakupljali iz jednog mjesta i dostavljali u jednom formatu. Danas podaci putuju u raznim formatima poput videa, teksta ili slika, a sakupljaju se iz mnoštva različitih izvora. Sve to doprinosi njihovoj raznolikosti.

Količina podataka koja svakim trenutkom putuje internetom je velika, ali i kompleksna. Temeljeno na prethodno navedenim svojstvima velikih skupova podataka, mogu se pretpostaviti mogući izazovi u radu s njima. Već je spomenuto da je potrebno novo skladištenje takvih podataka, ali izazovi kreću već od samog prikupljanja, preko analize, korištenja i konačno njihovog skladištenja [4].

Kao rješenja za sve probleme s kojima se može susresti, razvile su se mnoge nove tehnologije specificirane za rad s velikim skupovima podataka. Primjerice, tvrtke koje u svom poslovanju proizvode ili koriste mnogo podataka, sve češće primjenjuju usluge oblaka (engl. *cloud*) i mnogih tehnologija koje oblaci nude. Rad s velikim skupovima

podataka može pridonijeti i u donošenju odluka unutar tvrtke time što se provjerom podataka dobiva uvid u profite, rad radnika, napredak i slično. Danas se teži što efikasnijem radu sa što većim količinama podataka te se smatra da bi se u radu s njima mogao doseći vrhunac u znanosti, a svakodnevne aktivnosti i tehnologije koje se njima koriste bile bi sve više prilagodljive i personalizirane [5].

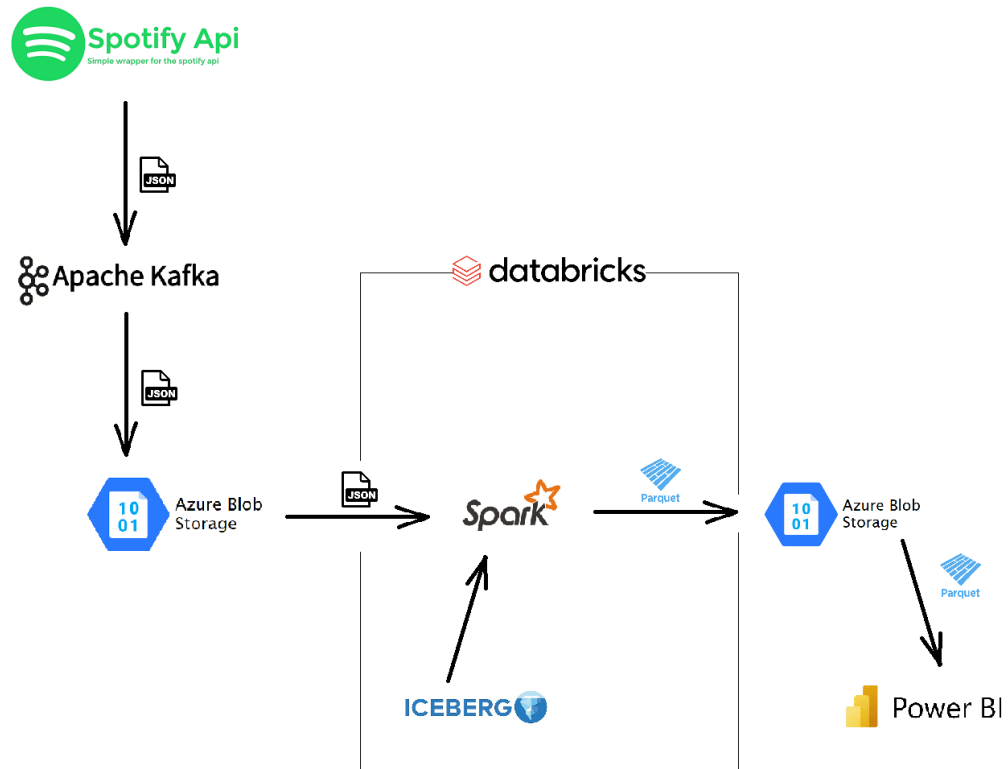
1.2. Uvod u praktični primjer

Podaci neprestano i u velikim količinama struje oko ljudi, počevši od korištenja društvenih mreža, poput Facebooka ili Instagrama, pa sve do primjena u zdravstvu, prognoziranju vremena, GPS-u i slično. Za sve navedeno potrebni su podaci koji neprestano teku. Aplikacije danas, prikupljanjem podataka, pokušavaju što više personalizirati svoje sadržaje prema korisnicima. Kako bi to bilo moguće, prikuplja se što korisnik radi dok koristi aplikaciju, što prati ili što ga više zanima te se analizom i obradom odredi kakav će se sadržaj prikazivati.

U nastavku ovog rada otkrivaju se problematike prikupljanja, analize i vizualizacije podataka te se prikazuje primjer toka podataka. Detaljnije o primjeru koji se prati kroz rad prikazuje slika 1.1.

Kao izvor podataka koristi se *Spotify API* [6]. *Spotify* je aplikacija za slušanje glazbe koja prikupljanjem podataka o korisniku preporučuje pjesme ili popise pjesama za reprodukciju. Podatke prikuplja *Apache Kafka*, distribuirana platforma otvorenog koda za strujanje (engl. *streaming*) podataka koju koriste tisuće tvrtki za stvaranje učinkovitih podatkovnih cjevovoda (engl. *data pipelines*) [7]. Pomoću platforme *Confluent Platform*, koja je na slici 1.1 sažeta u *Apache Kafku*, stvara se *Kafkina* poveznica (engl. *Kafka Connect*) s *Microsoft Azure Cloudom*. *Confluent Platform* je platforma koja omogućuje organiziranje i upravljanje podacima iz mnogo različitih izvora s jednim pouzdanim sustavom visokih performansi [8]. Kao što je vidljivo na slici 1.1 podaci putuju i spremaju se kao JSON unutar *Azure Blob Storaga*, koji se koristi za masivnu i sigurnu pohranu objekata [9]. Iduća strelica koja nosi podatke odlazi u *Azure Databricks*, koji se koristi za transformaciju podataka pomoću *Apache Sparka* i *Apache Iceberga*. Unutar *Databricksa* stvori se radni prostor u kojem se mogu pisati naredbe u *SQL-u*, *Pythonu* ili *Scala*. U ovom primjeru, koristi se *PySpark*. To je *Pythonov* API namijenjen za *Apache Spark*. Za obavljanje transformacija nad podacima koristi se *Apache Spark*. Transformirani

(„očišćeni“) podaci prepisuju se u *Iceberg*ove tablice te se zapisuju natrag u *Azure Blob Storage*. Nakon prikupljanja, transformacije i pohrane podataka, slijedi vizualizacija. Podaci se čitaju s *Microsoft Azure Clouda* u *Power BI*, gdje se konačno vizualiziraju.



Slika 1.1 Pregled tehnologija korištenih u praktičnom primjeru toka podataka

2. Izvor podataka

Izvor podataka predstavlja početnu lokaciju na kojoj se podaci stvaraju. 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 podatkovnih usluga kojih ima u izobilju na internetu [10]. Ovo poglavlje detaljnije opisuje kako se iz *Spotify API*-ja mogu dobiti podaci i koji su to podaci (Slika 2.1).



Slika 2.1 Prvi korak primjera: *Spotify API* kao izvor podataka

2.1. Tipovi izvora podataka

Većina izvora podataka može se podijeliti u dvije glavne kategorije:

1. Strojni izvor podataka (engl. *machine data source*).
2. Datotečni izvor podataka (engl. *file data source*).

Kod strojnih izvora podataka, podaci se stvaraju na korisnikovom uređaju. Taj uređaj može biti njegovo računalo, mobilni telefon, IoT ili neki drugi uređaj. Podaci se ne mogu dijeliti s drugim uređajima, a dostupni su onom korisniku koji je trenutno prijavljen u sustav. Strojne izvore podataka dalje se može podijeliti na korisnikove izvore podataka i systemske izvore podataka. Kod prvih su podaci dostupni samo određenom korisniku, a kod drugih svim korisnicima. Neki primjeri su zapisi o mrežnom prometu, zapisi sustava i aplikacija, izlazi iz senzora, podaci o događajima na IoT uređajima, rezultati upita na bazama podataka i tako dalje [11].

Datotečni izvori podataka nisu dodijeljeni određenim strojevima, aplikacijama ili korisnicima. Kod njih se podaci mogu dijeliti između uređaja te su obično pohranjeni u zasebnim tekstualnim datotekama. U takve izvore spadaju tablice, tekstualne datoteke, PDF datoteke, slike, audio i video datoteke [11].

2.2. Prijenos podataka pomoću API-ja

Jedan način prijenosa podataka od izvora do odredišta je korištenje aplikacijskih programskih sučelja (engl. *application programming interface* – *API*) [11]. API je programsko sučelje koje omogućuje komunikaciju softverskih programa na način da definira kako razvojni programer treba zatražiti usluge od operacijskog sustava ili od druge aplikacije, a sastoji se od:

- specifikacije koja opisuje kako se informacija razmjenjuje između programa u obliku zahtjeva i odgovora i
- programskog sučelja koje je napisano prema tim specifikacijama i objavljeno za korištenje.

Za softver koji želi pristupiti sadržaju kojeg API nudi kaže se da ga poziva, a za softver koji stvara sadržaj kaže se da ga objavljuje [12].

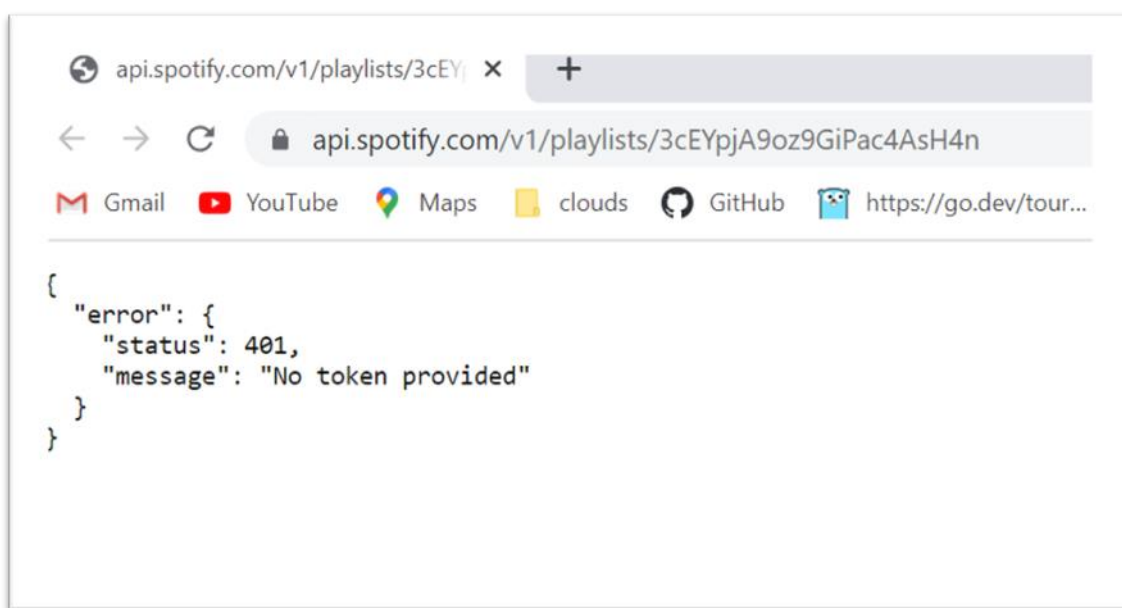
2.3. Spotify API

Kao što je već spomenuto, *Spotify API* je korišten kao izvor podataka u primjeru koji će pratiti ovaj rad. *Spotify API* je API koji se može koristiti za dohvaćanje *Spotify* podataka. Kod izgradnje API programskog sučelja potrebno je 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 odgovora** s poslužitelja klijentu na njegov zahtjev.

Zahtjevom na *Spotify API*, mogu se dobiti podaci o glazbenicima, njihovim albumima, pjesmama, popisima za reprodukciju i tako dalje. Pravilo *klijent-poslužitelj* znači da bi trebalo biti moguće primiti dio podataka kada se pristupi unaprijed definiranom URL-u [13]. Jedan primjer unaprijed definiranog URL-a kojem se može pristupiti je sljedeći zahtjev za popisom za reprodukciju (engl. *playlist*) od izvođača s ID-jem 3cEYpjA9oz9GiPac4AsH4n prikazan na slici 2.2.



Slika 2.2 Pristupanje *Spotify API* URL-u

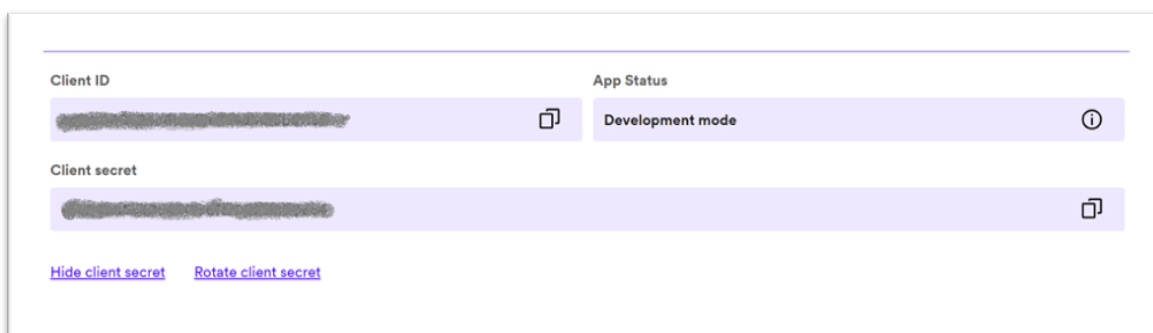
Pristupanje URL-u sa slike 2.2 će se uspješno izvršiti, ali će se dobiti odgovor koji otkriva da je došlo do pogreške. Razlog dobivanja takvog odgovora je odbijanje zahtjeva zbog neispravne autorizacije [14].

2.3.1. Spotify API – autorizacija korisnika

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 nije potrebna *Spotify Premium* verzija računa. Sljedeći korak kojeg je

potrebno napraviti je izrada aplikacije unutar *Spotify API* sučelja klikom na *create an app* gumb (<https://developer.spotify.com/dashboard>). Kada se to učini, moguće je dohvatiti vjerodajnice koje su potrebne kako bi se API mogao koristiti [16].

Kada se posjete postavke definirane aplikacije koja se prethodno stvorila, tamo se mogu pronaći dva parametra: *client ID* i *client secret* (Slike 2.3). Oba su potrebna za dohvat pristupnog tokena (engl. *access token*). Više o pristupnom tokenu bit će rečeno u nastavku poglavlja. Nakon dohvaćanja tih dvaju parametara, može ih se spremiti lokalno u zasebnu datoteku, recimo *.env*.



Slika 2.3 *Client ID* i *client secret*

Pristup zaštićenim resursima *Spotify API*-ja određen je s jednim ili više opsega (engl. *scope*). Opsezi omogućuju aplikaciji pristup određenim funkcijama (npr. čitanje popisa za reprodukciju) u ime korisnika. Skup opsega koji se postave tijekom autorizacije određuje dopuštenja pristupa. Detaljnije o opsezima može se pronaći u dokumentaciji opsega (<https://developer.spotify.com/documentation/web-api/concepts/scopes>) [15].

U nastavku primjera koristit će se opseg koji omogućuje autorizaciju bez posebnog korisnika. Na taj način moguće je koristiti općenite značajke *Spotifyja*, tj. dobiti informacije o izvođačima, pjesmama i slično, ali nije moguće vidjeti statistike vezane uz posebnog korisnika kao što su njegovi popisi za reprodukciju [16].

Nakon prijave u *Spotify API* sučelje, izrade aplikacije, odabira opsega i dohvaćanja potrebnih vjerodajnica, slijedi pisanje programskog koda za stvaranje pristupnog tokena. Pristupni token je niz koji sadrži vjerodajnice i dopuštenja koja se mogu koristiti za pristup određenom resursu (npr. izvođačima, albumima ili pjesmama) ili korisničkim podacima (npr. vlastitom profilu ili vlastitim popisima za reprodukciju) [15]. Kod za dohvaćanje tokena pisan je u *Pythonu* te je ideja da taj kod bude dijeljeni između nekoliko budućih

programskih kodova kojima će se dohvaćati željeni podaci. Zbog toga je kao ime datoteke odabran naziv *common_functions.py*.

Sljedeći isječak koda prikazuje učitavanje *.env* datoteke i spremanje *client ID* i *client_secret* vjerodajnica u dvije varijable:

```
load_dotenv()
client_id = os.getenv("CLIENT_ID")
client_secret = os.getenv("CLIENT_SECRET")
```

Kod 2.1 – Kreiranje *client_id* i *client_secret* varijabli

Nakon toga ono što želimo je dobiti pristupni token pomoću te dvije varijable pa definiramo metodu *get_token*:

```
def get_token():
    auth_string = client_id + ":" + client_secret
    auth_bytes = auth_string.encode("utf-8")
    auth_base64 = str(base64.b64encode(auth_bytes), "utf-8")
    url = "https://accounts.spotify.com/api/token"
    headers = {
        "Authorization": "Basic " + auth_base64,
        "Content-Type": "application/x-www-form-urlencoded"
    }
    data = {"grant_type": "client_credentials"}
    result = post(url, headers=headers, data=data)

    json_result = json.loads(result.content)
    token = json_result["access_token"]
    return token
```

Kod 2.2 – Metoda za dohvaćanje pristupnog tokena

Prethodni isječak koda prikazuje dohvaćanje pristupnog tokena slanjem POST zahtjeva na *Spotify API*. Slični primjeri dohvaćanja tokena vidljivi su pretraživanjem dokumentacije *Spotify API*-ja. Ako se pozove *get_token* metoda s ispravnim vjerodajnicama dobije se pristupni token koji vrijedi 60 minuta. Pri isteku 60 minuta, dobije se poruka o nevaljaloj autorizaciji. Tada je potrebno ponovno zatražiti token.

2.3.2. Spotify API – primjer slanja zahtjeva u Postmanu

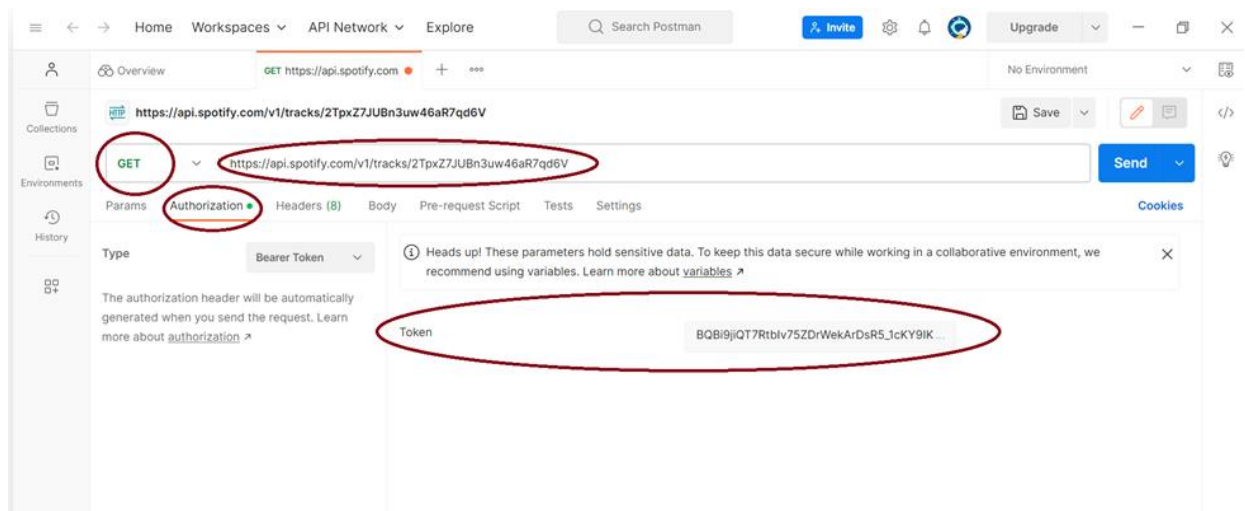
Jedan zahtjev za dohvaćanjem zapisa GET metodom uz korištenje tokena izgledao bi nešto poput:

```
curl --request GET
'https://api.spotify.com/v1/tracks/2TpxZ7JUBn3uw46aR7qd6V' --header
'Authorization: Bearer NgCXRK...MzYjw'
```

Kod 2.3 – Primjer slanja zahtjeva

Primjer se može isprobati u naredbenom retku pozivanjem naredbe *curl*. Kako bi se jasnije vidjeli rezultati poziva, 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 2.4 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* funkcije.



Slika 2.4 Slanje zahtjeva pomoću *Postmana*

Ako je zahtjev ispravno poslan, kao odgovor se dobije *json* kao na slici 2.5. Svaki zapis (engl. *track*) ima svoj ID. U ovom primjeru to je vrijednost 2TpxZ7JUBn3uw46aR7qd6V. Isto tako, vlastiti ID imaju i izvođači, albumi, popisi za reprodukciju i tako dalje. U *json* odgovoru sa slike 2.5 uočavamo da je prvo naveden album u kojem se zapis pojavljuje, njegov tip, ID, tržište na kojem se pojavljuje u obliku oznake (npr. za Španjolsku je oznaka ES). Nakon albuma vidi se 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 slično.

```

{
  "album": {
    "album_type": "album",
    "artists": [
      {
        "external_urls": {
          "spotify": "https://open.spotify.com/artist/08td7MxkoHQkXnWAYD8d6Q"
        },
        "href": "https://api.spotify.com/v1/artists/08td7MxkoHQkXnWAYD8d6Q",
        "id": "08td7MxkoHQkXnWAYD8d6Q",
        "name": "Tania Bowra",
        "type": "artist",
        "uri": "spotify:artist:08td7MxkoHQkXnWAYD8d6Q"
      }
    ],
    "available_markets": [
      "AD",
      ...
      "ZW"
    ],
    "external_urls": {
      "spotify": "https://open.spotify.com/album/6akEvsyscLGftJxYudPjmqK"
    },
    "href": "https://api.spotify.com/v1/albums/6akEvsyscLGftJxYudPjmqK",
    "id": "6akEvsyscLGftJxYudPjmqK",
    "images": [
      ...
    ],
    "name": "Place In The Sun",
    "release_date": "2004-02-02",
    "release_date_precision": "day",
    "total_tracks": 11,
    "type": "album",
    "uri": "spotify:album:6akEvsyscLGftJxYudPjmqK"
  },
  |
  "artists": [
    ...
  ],
  "available_markets": [
    "AR",
    ...
    "XK"
  ],
  "disc_number": 1,
  "duration_ms": 276773,
  "explicit": false,
  "external_ids": {
    "isrc": "AUCR10410001"
  },
  "external_urls": {
    "spotify": "https://open.spotify.com/track/2TpxZ7JUBn3uw46aR7qd6V"
  },
  "href": "https://api.spotify.com/v1/tracks/2TpxZ7JUBn3uw46aR7qd6V",
  "id": "2TpxZ7JUBn3uw46aR7qd6V",
  "is_local": false,
  "name": "All I Want",
  "popularity": 1,
  "preview_url": "https://p.scdn.co/mp3-preview/2cc385470731bc540a08caef5eab31dec7",
  "track_number": 1,
  "type": "track",
  "uri": "spotify:track:2TpxZ7JUBn3uw46aR7qd6V"
}

```

Slika 2.5 Primjer *json* odgovora

U nastavku ovog rada slat će se zahtjevi na tri krajnje točke:

1) zahtjev za dohvaćanjem podataka o izvođačima

- Mogući primjer URL zahtjeva za izvođačem može glasiti <https://api.spotify.com/v1/artists/0TnOYISbd1XYRBk9myaseg>. To je zahtjev za izvođačem s ID-jem 0TnOYISbd1XYRBk9myaseg. Za slanje zahtjeva za izvođačem potrebno je navesti ID traženog izvođača.

2) zahtjev za dohvaćanje podataka o albumima

- Mogući primjer URL zahtjeva za albumom ili albumima može glasiti <https://api.spotify.com/v1/artists/0TnOYISbd1XYRBk9myaseg/albums?limit=5>.

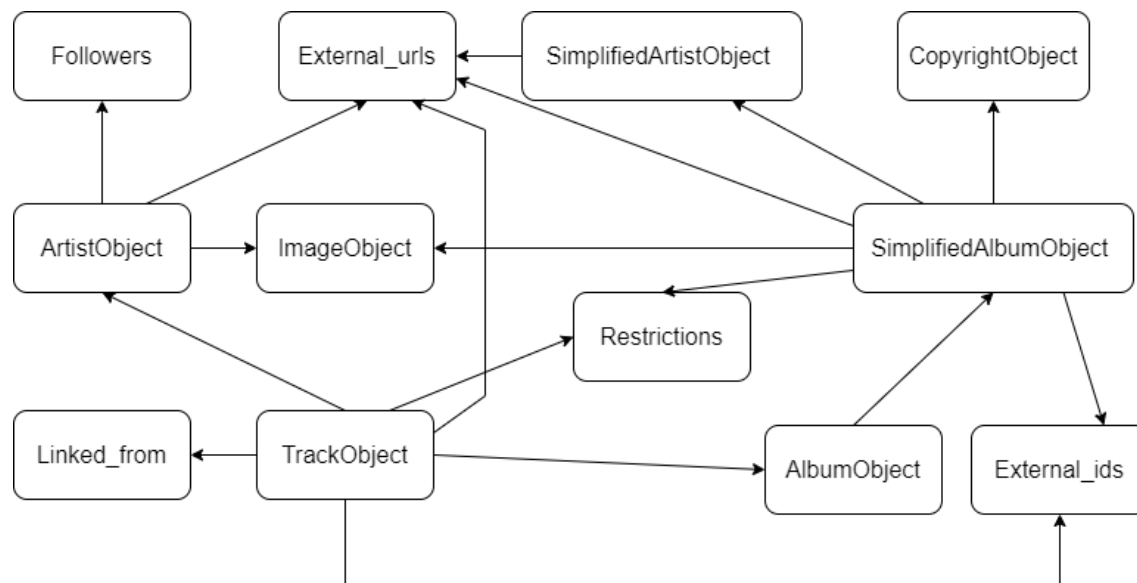
Potrebno je navesti ID izvođača čiji se albumi žele dohvatiti. Moguće je postaviti i dodatne parametre: *include_groups* (parametar koji služi za filtriranje albuma, a može poprimiti vrijednosti *album*, *single*, *appears_on* i/ili *compilation*), *market* (poprima vrijednost oznake zemlje koja govori da se dohvate samo albumi koji se mogu slušati u zemlji s tom oznakom, npr. ES je oznaka za Španjolsku), *limit* (broj čija je najmanja vrijednost jedan, a najveća pedeset, a govori koliko će se albuma dohvatiti), *offset* (pomak koji govori koji po redu album treba preuzeti).

3) zahtjev za dohvaćanje podataka o glazbenim zapisima, tj. pjesmama

- Mogući primjer URL zahtjeva za pjesmama može glasiti <https://api.spotify.com/v1/artists/0TnOYISbd1XYRBk9myaseg/top-tracks?market=ES>. Potrebno je navesti ID izvođača čije se pjesme žele dohvatiti. Moguće je postaviti i dodatan parametar: *market* (poprima vrijednost oznake zemlje koja govori da se dohvate samo pjesme koje se mogu slušati u zemlji s tom oznakom, npr. ES je oznaka za Španjolsku)

Sljedeća slika prikazuje pojednostavljeni model objekata koji se dobiju kao odgovor na slanje prethodno navedenih zahtjeva (Slika 2.6). Konkretno kao odgovor na zahtjev za dohvaćanjem izvođača dobije se objekt *ArtistObject* koji sadrži nekoliko drugih objekata. To su objekti *Followers*, *External_urls* i *ImageObject*. Kao odgovor na zahtjev za dohvaćanje podataka o albumima dobije se *AlbumObject* koji u sebi sadrži jednostavniji oblik objekta koji predstavlja album, tj. *SimplifiedAlbumObject*. *SimplifiedAlbumObject* sastoji se od *External_urls*, *ImageObject*, *Restrictions*, *CopyrightObject*, *External_ids* i *SimplifiedArtistObject*. *SimplifiedArtistObject* pojednostavljeni je oblik objekta *ArtistObject*. *TrackObject* dobije se kao odgovor na zahtjev za dohvaćanjem podataka o

glazbenim zapisima. *TrackObject* u sebi sadrži objekte *AlbumObject*, *ArtistObject*, *External_ids*, *External_urls*, *Linked_from* i *Restrictions*.



Slika 2.6 Pojednostavljeni model objekata sadržanih u odgovoru na slanje zahtjeva za dohvaćanjem izvođača, albuma i/ili glazbenih zapisa

Slika 2.6 prikazuje samo nazive objekata i odnose među njima. Slijedi nekoliko tablica koje detaljnije opisuju od kojih se vrijednosti sastoje objekti koje je nužno razumjeti za nastavak ovog rada. To su redom objekti: *ArtistObject*, *AlbumObject*, *TrackObject*, *SimplifiedAlbumObject* i *SimplifiedArtistObject*.

ArtistObject		
Vrijednost	Tip	Opis
<i>external_urls</i>	objekt <i>External_urls</i>	vanjski URL-ovi izvođača
<i>followers</i>	objekt <i>Followers</i>	informacije o pratiteljima izvođača
<i>genres</i>	lista elemenata tipa <i>string</i>	lista žanrova s kojima je izvođač povezan
<i>href</i>	<i>string</i>	poveznica na Web API krajnju točku koja nudi detaljne informacije o izvođaču
<i>id</i>	<i>string</i>	<i>Spotify</i> ID izvođača

<i>images</i>	lista objekata tipa <i>ImageObject</i>	slike izvođača u različitim veličinama
<i>name</i>	<i>string</i>	ime izvođača
<i>popularity</i>	<i>integer</i>	broj između 0 i 100 koji označava popularnost izvođača; 100 označava najveću popularnost; izračunava se iz popularnosti pjesama izvođača
<i>type</i>	<i>string</i>	tip izvođača; jedina dozvoljena vrijednost je <i>artist</i>
<i>uri</i>	<i>string</i>	<i>Spotify</i> URI izvođača

Tablica 2.1 Detaljniji opis objekta *ArtistObject*

AlbumObject		
Vrijednost	Tip	Opis
<i>href</i>	<i>string</i>	poveznica na Web API krajnju točku koja nudi detaljne informacije o albumu
<i>limit</i>	<i>integer</i>	definira maksimalni broj albuma koji se vraćaju slanjem zahtjeva
<i>next</i>	<i>string</i>	URL na albume koji slijede
<i>offset</i>	<i>integer</i>	pomak koji govori koji po redu album treba preuzeti
<i>previous</i>	<i>string</i>	URL na albume koji prethode
<i>total</i>	<i>integer</i>	broj koji govori koliko je maksimalno moguće dohvatiti albuma
<i>items</i>	lista objekata tipa <i>SimplifiedAlbumObject</i>	detaljnije informacije o albumima sadržane unutar objekta

		<i>SimplifiedAlbumObject</i>
--	--	------------------------------

Tablica 2.2 Detaljniji opis objekta *AlbumObject*

TrackObject		
Vrijednost	Tip	Opis
album	objekt <i>AlbumObject</i>	album na kojem se pjesma nalazi
artists	lista objekata tipa <i>ArtistObject</i>	popis izvođača koji izvode pjesmu
available_markets	lista elemenata tipa <i>string</i>	popis zemalja u kojima se pjesma može izvoditi
disc_number	<i>integer</i>	broj diska (album se može sastojati od više diskova); uobičajena vrijednost je jedan
duration_ms	<i>integer</i>	duljina pjesme u milisekundama
explicit	<i>boolean</i>	ako je vrijednost istinita (engl. <i>true</i>) onda pjesma sadrži eksplicitne izraze, inače je vrijednost lažna (engl. <i>false</i>)
external_ids	objekt <i>External_ids</i>	vanjski ID-jevi pjesme (koji nisu Spotify ID)
external_urls	objekt <i>External_urls</i>	vanjski URL-ovi pjesama
href	<i>string</i>	poveznica na Web API krajnju točku koja nudi detaljne informacije o pjesmi
id	<i>string</i>	<i>Spotify</i> ID pjesme
is_playable	<i>boolean</i>	ako je vrijednost istinita (engl. <i>true</i>), onda je pjesmu moguće slušati u zemlji koja je postavljena pod opcionalnim parametrom

		<i>market</i> , inače je vrijednost lažna (engl. <i>false</i>)
<i>linked_from</i>	objekt <i>Linked_from</i>	ako se pjesma koja se dohvaća ikada zamijenila s drugom pjesmom, ovaj objekt sadrži informacije o originalnoj pjesmi
<i>restrictions</i>	objekt <i>Restrictions</i>	uključen u odgovor kada se primjenjuje zabrana sadržaja
<i>name</i>	<i>string</i>	ime pjesme
<i>popularity</i>	<i>integer</i>	broj između 0 i 100 koji označava popularnost pjesme; 100 označava najveću popularnost; računa se pomoću ukupnog broja preslušavanja pjesme i informacije koliko nedavno su bila ta slušanja
<i>preview_url</i>	<i>string</i>	poveznica na pretpregled pjesme u trajanju od 30 sekundi
<i>track_number</i>	<i>integer</i>	redni broj pjesme na disku
<i>type</i>	<i>string</i>	tip pjesme; jedina dozvoljena vrijednost je <i>track</i>
<i>uri</i>	<i>string</i>	<i>Spotify</i> URI pjesme
<i>is_local</i>	<i>boolean</i>	je li pjesma s lokalnog dokumenta

Tablica 2.3 Detaljniji opis objekta *TrackObject*

SimplifiedAlbumObject		
Vrijednost	Tip	Opis
<i>album_type</i>	<i>string</i>	tip albuma

<i>total_tracks</i>	<i>integer</i>	broj zapisa u albumu
<i>available_markets</i>	lista elemenata tipa <i>string</i>	popis zemalja u kojima se album može izvoditi
<i>external_urls</i>	objekt <i>Exteranal_urls</i>	vanjski URL-ovi albuma
<i>href</i>	<i>string</i>	poveznica na Web API krajnju točku koja nudi detaljne informacije o albumu
<i>id</i>	<i>string</i>	<i>Spotify</i> ID albuma
<i>images</i>	lista objekata tipa <i>ImageObject</i>	slike albuma u različitim veličinama
<i>name</i>	<i>string</i>	ime albuma
<i>release_date</i>	<i>string</i>	datum izdavanja albuma
<i>release_date_precision</i>	<i>string</i>	preciznost vrijednosti <i>release_date</i> (npr. <i>year</i> , <i>month</i> , <i>day</i>)
<i>restrictions</i>	objekt <i>Restrictions</i>	uključen u odgovor kada se primjenjuje zabrana sadržaja
<i>type</i>	<i>string</i>	tip albuma; jedina dozvoljena vrijednost je <i>album</i>
<i>uri</i>	<i>string</i>	<i>Spotify</i> URI albuma
<i>copyrights</i>	lista objekata tipa <i>CopyrightObject</i>	sadrži izjave o autorskim pravima albuma
<i>external_ids</i>	objekt <i>External_ids</i>	vanjski ID-jevi albuma (koji nisu <i>Spotify</i> ID)
<i>genres</i>	lista elemenata tipa <i>string</i>	lista žanrova s kojima je album povezan

<i>label</i>	<i>string</i>	oznaka povezana s albumom
<i>popularity</i>	<i>integer</i>	broj između 0 i 100 koji označava popularnost albuma; 100 označava najveću popularnost
<i>album_group</i>	<i>string</i>	poprima vrijednosti <i>album</i> , <i>single</i> , <i>compilation</i> ili <i>appears_on</i>
<i>artists</i>	lista objekata tipa <i>SimplifiedArtistObject</i>	izvođači albuma

Tablica 2.4 Detaljniji opis objekta *SimplifiedAlbumObject*

SimplifiedArtistObject		
Vrijednost	Tip	Opis
<i>external_urls</i>	objekt <i>Exteranal_urls</i>	vanjski URL-ovi izvođača
<i>href</i>	<i>string</i>	poveznica na Web API krajnju točku koja nudi detaljne informacije o izvođaču
<i>id</i>	<i>string</i>	<i>Spotify</i> ID izvođača
<i>name</i>	<i>string</i>	ime izvođača
<i>type</i>	<i>string</i>	tip izvođača; jedina dozvoljena vrijednost je <i>artist</i>
<i>uri</i>	<i>string</i>	<i>Spotify</i> URI izvođača

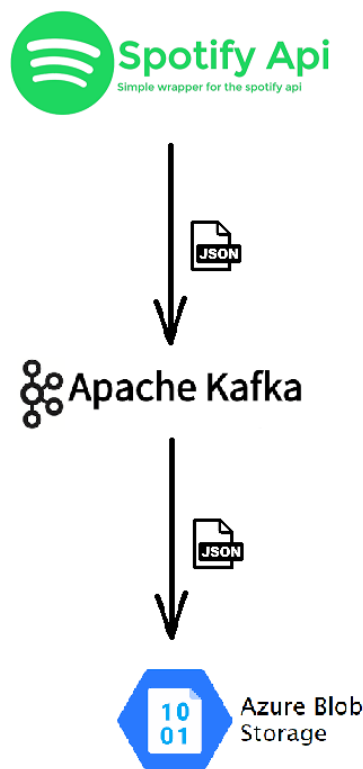
Tablica 2.5 Detaljniji opis objekta *SimplifiedArtistObject*

U ovom poglavlju pojašnjeno je kako se podaci mogu dohvatiti iz *Spotify API*-ja te je dan pregled koji podaci će se dohvaćati, a iduće poglavlje pojašnjava na koji način i kojim alatima će se obavljati dohvaćanje podataka.

3. Dohvaćanje velikih skupova podataka

Koliko se dnevno stvori podataka? U 2021. godini ljudi su stvarali 2,5 kvintilijuna ($2,5 * 10^{30}$) bajtova podataka svaki dan. U 2022. godini dnevno se objavljivalo 650 milijuna novih tvitova (engl. *tweet*), tj. objava na *Twitter* društvenoj mreži, a svaku sekundu se dogodi 99000 pretraživanja na *Googlu*. Svi ovi brojevi zapravo dočaravaju koliko podataka neprestano teče i koliko njih se koristi, a da ljudi toga nisu ni svjesni.

Ovo poglavlje govori općenito o načinima i alatima za prikupljanje velikih skupova podataka. Objašnjava se način rada *Apache Kafke*, alata koji se koristi za dohvaćanje podataka sa *Spotify API*-ja. Dohvaćeni podaci se skladište u *Azure Blob Storageu* pa se u ovom poglavlju objašnjava stvaranje poveznice između *Apache Kafke* i *Azure Blob Storagea* (Slika 3.1).



Slika 3.1 Drugi korak primjera: Dohvaćanje podataka iz *Spotify API*-ja pomoću *Apache Kafke* te stvaranje poveznice s *Azure Blob Storageom*

3.1. Prikupljanje i tipovi podataka

Podaci koji se prikupljaju nisu uzaludni i zapravo su danas potrebni za rad mnogih tvrtki, ali i čovjeku za obavljanje svakodnevnih aktivnosti. Kako bi se podaci uopće mogli iskoristiti, prvo ih je potrebno prikupiti iz već spomenutih izvora.

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 [19]. Strukturirani podaci su čuvani u unaprijed definiranom formatu, dok nestrukturirani podaci postoje u onom obliku u kojem su stvoreni. Primjer nestrukturiranih podataka su objave na društvenim mrežama. Polu-strukturirani podaci kombinacija su strukturiranih i nestrukturiranih podataka [20].

Jedan primjer polu-strukturiranih podataka su *json* podaci jer ne zahtijevaju shemu. JSON (engl. *JavaScript Object Notation*) je format podataka koji se temelji na tekstu i mogu ga čitati ljudi i strojevi [21]. Široko je korišten te nudi dodatnu fleksibilnost za pohranjivanje podataka i postavljanje upita koji se pridržavaju uvijek fiksnih shema i vrsta podataka [22]. Primjer *json-a* prikazan je u prethodnom poglavlju kao odgovor na GET zahtjev upućen na *Spotify API*.

3.2. Alati za dohvaćanje podataka

Dohvaćanje podataka s nekog izvora 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 jednog sustava na drugu lokaciju, kao što je baza podataka. Cilj im je da automatiziraju proces dohvaćanja podataka iz različitih izvora prije slanja tih podataka na lokaciju. Alati su nužni za brži prijenos velikih količina podataka, bez potrebe ručne intervencije [23]. Kada bi se prijenos podataka morao raditi ručno, troškovi u obliku vremena i novca bili bi znatno veći. Osim manjih troškova, neke dodatne prednosti korištenja alata za dohvaćanje (engl. *ingestion*) podataka su:

- Korištenje **različitih izvora**: 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 zadatka, bez potrebe za ručnim pokretanjima i obavljanjima zadataka.
- **Rješavanje i oporavak od pogreške:** 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 tijekom dohvaćanja podataka [23].

3.2.1. Tipovi i stvarni primjeri alata za dohvaćanje podataka

Neki od tipova alata za dohvaćanje podataka su:

- 1) **Web pretraživači** (engl. *web crawlers*): Alati koji sakupljaju podatke iz web stranica skeniranjem i povlačenjem dostupnih informacija.
- 2) **Aplikacije za strujanje podataka:** Alati koji omogućuju prijenos podataka u stvarnom vremenu pomoću toka podataka.
- 3) **skripte ili ručno izrađen kod:** Moguće je pisati vlastite skripti za sakupljanje podataka.
- 4) **ETL alati:** Dolazi od engleskih riječi *extract*, *transform* i *load* jer oni povlače (engl. *extract*) podatke iz različitih izvora, transformiraju (engl. *transform*) ih u korisne formate i na kraju učitavaju (engl. *load*) u središnju bazu podataka ili repozitorij.
- 5) **Alati za repliciranje baza podataka:** Repliciraju strukture i sadržaje jedne baze podataka u drugi primjerak te baze podataka.
- 6) **Alati za rad s oblacima:** Na siguran način i čuvajući integritet podataka, unose podatke u okruženja u oblaku [23].

Neki od poznatijih alata koji se koriste 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*). Ima jednostavno i intuitivno sučelje, a uglavnom se koristi u tvrtkama za analizu prodaje i oglašavanja

prikupljanjem svih podataka na jedno mjesto [25]. *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 je *NiFi* procesor sam odgovoran za interpretiranje sadržaja podataka koje primi [26]. *Apache Flume* distribuirana je i otporna usluga za učinkovito prikupljanje, agregiranje i premještanje velikih količina podataka. Uglavnom je riječ o zapisima podataka, tj. o logovima. *Apache Kafka* je alat koji je poznat po svojim visokim performancama, koje mu omogućuju rad s tisućama poruka u sekundi [26].

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

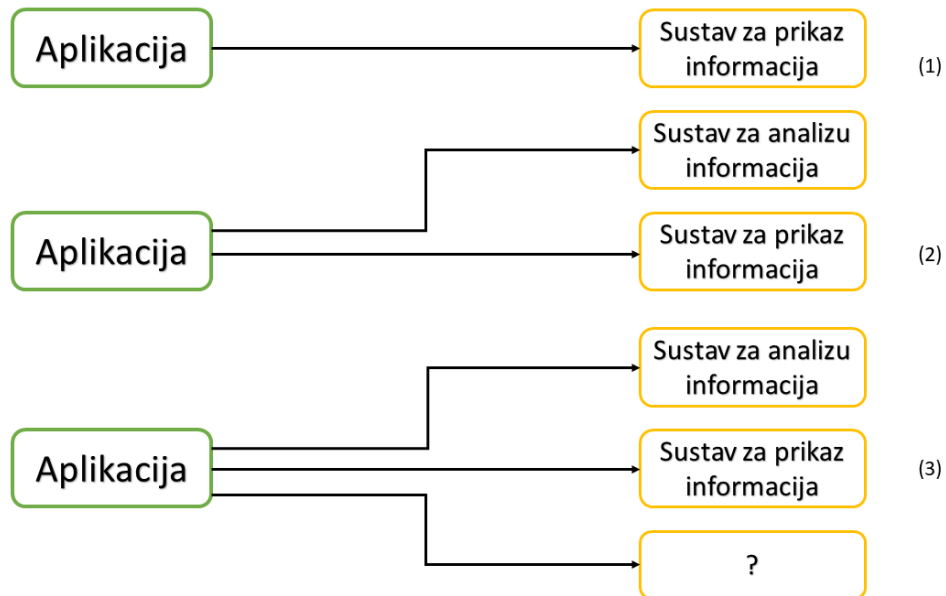
3.3. Apache Kafka

3.3.1. Model objavi-pretplati

Prije nego što se objasni što je uopće *Apache Kafka*, kako radi i kako se koristi u ovom radu, važno je razumjeti koncept slanja poruka korištenjem modela *objavi-pretplati* (engl. *publish/subscribe*). Model *objavi-pretplati*, ili prema engleskom izrazu skraćeno *pub/sub*, model je slanja poruka u kojem pošiljalac (engl. *publisher*) poruke, tj. podatka, ne šalje poruku određenom primatelju, nego pošiljalac klasificira podatak koji objavljuje, a primatelj se pretplaćuje na primanje određenih klasa podataka. Takvi sustavi često imaju posrednika (engl. *broker*), tj. središnju točku na koju se poruke objavljuju i s koje se prikupljaju [27].

Mnogi slučajevi *pub/sub* sustava započnu s jednostavnim redovima poruka (engl. *message queue*). Na primjer, nakon stvaranja aplikacije potrebno je s nje slati nadzorne informacije na nadzorni sustav ili aplikaciju koja će te informacije prikazati (Slika 3.2 pod (1)). Nakon nekog vremena, osim prikaza nadzornih informacija, potrebno je te informacije analizirati, ali prethodni sustav nema tu mogućnost pa će se iste informacije slati na novi sustav. Sada aplikacija šalje podatke u druge dvije (Slika 3.2 pod (2)). U budućnosti će možda biti potrebno dodatno proširiti cijeli sustav (Slika 3.2 pod (3)), a tada već postaje kompleksniji. Kako bi se taj problem riješio, može se uvesti *pub/sub* sustav [27].

Pretpostavimo da je sada potrebno, umjesto nadzornih, slati neku drugu vrstu informacija ili drugog formata, ali informacije se ne smiju slati svima ili možda ne svima u istom obliku. Rješenje s pub/sub sustavom je u redu, ali naglo će doći do pojave dupliciranja veza između aplikacije i drugih sustava koje se vide na slici 3.2. Kao jedno od mogućih rješenja ovog problema dolazi *Apache Kafka*.



Slika 3.2 Primjer rasta kompleksnosti sustava

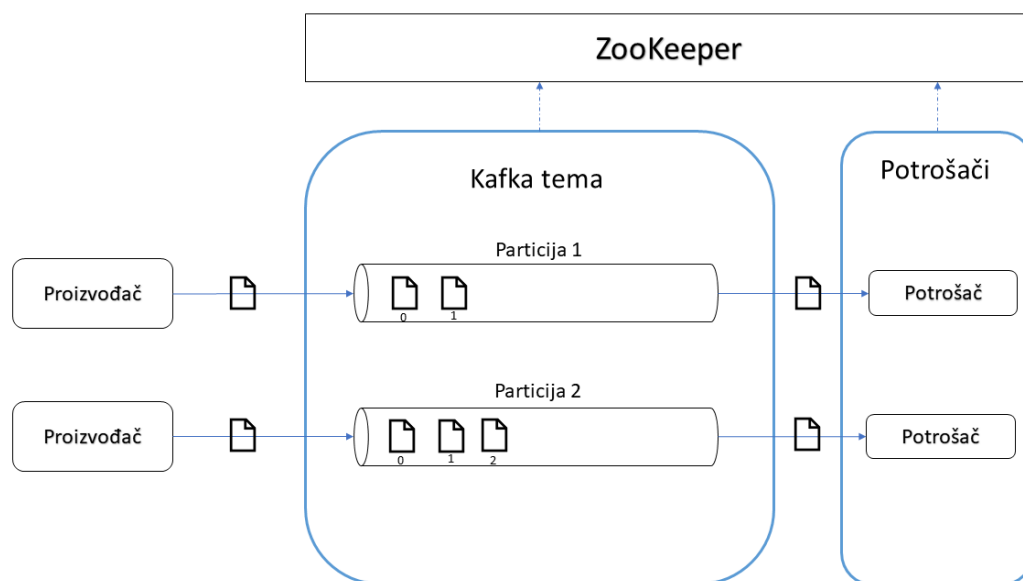
3.3.2. Uvod u rad Apache Kafke

Apache Kafka originalno je stvoren u *LinkedInu* kako bi obrađivao zapise u stvarnom vremenu s minimalnim kašnjenjem od ne više od nekoliko sekundi. Dizajniran je da bude distribuiran, skalabilan i izdržljiv sustav bez pogrešaka s visokom propusnošću. *LinkedIn* se oslanja na skalabilnost i pouzdanost *Kafke* za slučajeve poput izgradnje nadzornih sustava ili u praćenju rada stranice. Trenutno, zbog svoje jake izdržljivosti i visoke propusnosti, *Apache Kafka* je široko primijenjen u internetskim kompanijama [28].

Kafka se ponaša kao posrednik između dvaju sustava: proizvođača (engl. *producer*) i potrošača (engl. *consumer*) [29]. Jedinica podatka u *Kafki* zove se poruka. Poruka se može zamisliti kao jedan zapis u bazi podataka, a *Kafki* jedna poruka predstavlja polje bajtova koje sadrži podatke koji ne moraju biti nekog određenog formata ili značenja [27]. Konkretno, u primjeru kojeg prati ovaj rad jedna poruka bi bila odgovor na GET zahtjev sa

Spotify API-ja. *Kafka* podatke čuva u kategorijama koje se zovu teme (engl. *topic*). Tema je dio *Kafke* koji sprema i objavljuje poruke. Sve *Kafkine* poruke se organiziraju unutar tema. Proizvođači su procesi koji objavljuju poruke na *Kafkine* teme, a potrošači procesi koji se pretplaćuju na *Kafkine* teme te dohvaćaju poruke [30]. U primjeru je proizvođač *Spotify API*, a potrošač *Azure Blob Storage* koji dohvaća i skladišti dobivene podatke. *Kafkina* tema može se dijeliti na jednu ili na više particija, a svaka particija je uređeni, nezamjenjiv slijed poruka.

Kafka se izvršava na klasteru (engl. *cluster*) koji se sastoji od jednog ili više poslužitelja od kojih se svaki zove posrednik (engl. *broker*) [29]. Oni su osmišljeni da rade kao dio klastera. Unutar klastera, jedan posrednik je upravitelj klastera te je odgovoran za zadatke kao što su dodjeljivanje particija posrednicima i nadgledanje pogrešaka drugih posrednika.



Slika 3.3 Prikaz *Apache Kafka* sustava

Na prethodnoj slici (Slike 3.3) prikazan je *Apache Kafka* sustav koji se sastoji od jedne teme unutar koje su dvije particije. Na svaku od particija po jedan proizvođač šalje poruke te su na svaku pretplaćeni po jedan potrošač. Na slici se mogu vidjeti dvije strelice koje vode u *ZooKeepera*. *ZooKeeper* je servis za koordinaciju procesa distribuiranih aplikacija koji se koristi za upravljanje i koordiniranje *Kafka* posrednika. Njegove usluge se uglavnom koriste da obavijeste proizvođača i potrošača o prisutnosti novog posrednika

ili o pojavi pogreške unutar posrednika [30]. Kafka osigurava pretpostavljeni i jednostavni konfiguracijski dokument *ZooKeepera* koji se koristi za pokretanje njegove instance [29].

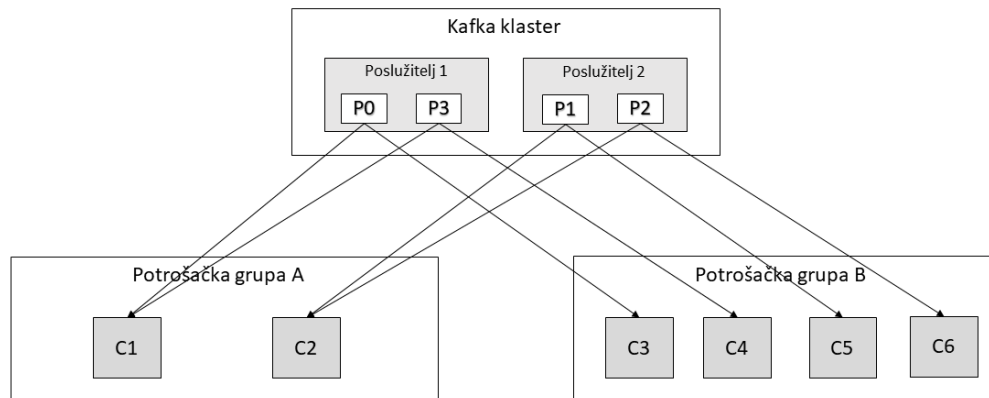
Kako bi se teret balansirao (engl. *load balancing*), teme se dijele na nekoliko particija i svaki posrednik čuva jednu ili više od tih particija. Također, više proizvođača i potrošača mogu objavljivati, odnosno povlačiti poruke u isto vrijeme. Svaka poruka u particiji ima svoj sekvencijalni broj koji se zove pomak (engl. *offset*) [29]. Pomak je jedinstveni identifikator koji predstavlja položaj poruke unutar particije. Omogućuje praćenje napretka i pouzdan nastavak potrošnje za potrošače. Na slici 3.3 vidi se numeriranje podataka unutar particija od nule. Potrošač se pretplaćuje na jednu ili više tema te preuzima poruke onim redoslijedom kojim su poruke stigle u particiju. U isto vrijeme, prati koje je poruke već primio na način da pamti pomak poruke. Pomak se konstantno povećava dolaskom sljedeće poruke i *Kafka* pomak prenosi kao metapodatak uz poruku. Zbog toga što potrošač pamti pomak, može se zaustaviti i nakon nekog vremena nastaviti čitati poruke bez da izgubi mjesto na kojem je stao.

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 [29]. 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 u porukama koristi ključ koji se odnosi na točno specifičnu particiju [29].

Poruke se općenito mogu slati na dva načina: u redovima poruka (engl. *queue*) ili u modelu pub/sub. Kada se koriste 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 pripisuju potrošačkoj grupi i svaka poruka koja se objavi na temu se dostavlja jednom potrošaču unutar pretplaćene potrošačke grupe. Ako su svi potrošači raspoređeni u zasebne potrošačke grupe, onda *Kafka* radi kao da se radi o redovima poruka te se teret jednoliko raspoređuje na potrošače. 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 se poruke odašilju svim potrošačima [29].

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 potrebna nikakva koordinacija među potrošačkim grupama. Rezultat toga je da broj particija zapravo ograničava 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 [28].



Slika 3.4 Primjer potrošačkih grupa

Slika 3.4 prikazuje primjer u kojem na jednom klasteru postoje dva poslužitelja. Svaki poslužitelj sastoji se od dvije particije. Na slici se može uočiti da maksimalni broj potrošača u potrošačkoj grupi ne prelazi ukupan broj particija. Također, potrošači iste potrošačke grupe ne primaju poruke s iste particije, dok ista potrošačka grupa može primati poruke s različitih tema.

3.4. Dohvaćanje podataka sa Spotify API-ja pomoću Kafke

Apache Kafka može se koristiti samostalno ili s dodatnom tehnologijom iz *Confluenta* [31]. I *Apache Kafka* i *Confluent Kafka* temelje se na istoj *Apache Kafka* tehnologiji, ali se razlikuju u nekim značajkama [32]. *Confluent* su razvili isti inženjeri kao i *Apache Kafku* tako da pruža upravljane usluge izgrađene na *Kafki* koje je timovima lakše implementirati, održavati i nadzirati [33]. *Confluent Kafka* pruža komercijalnu verziju *Apache Kafke* s dodatnim značajkama kao što su replikacija, sigurnost i nadzor [32]. U nastavku ovog rada koristit će se platforma *Confluent Platform*. Platforma pruža dodatne značajke i alate na razini poduzeća za poboljšanje *Kafkinog* rada [34].

3.4.1. Pokretanje Confluent Platforme

Za potrebe ovog rada za rad s *Apache Kafkom* korištena je platforma *Confluent*. Kako bi se platforma koristila lokalno, potrebno ju je instalirati i ispravno podesiti. Jedan način postavljanja je preuzimanje ZIP ili TAR arhive s njihove stranice. Nakon preuzimanja arhive, potrebno ju je raspakirati. Trebali bi se stvoriti direktoriji pojašnjeni u sljedećoj tablici.

Direktorij	Pojašnjenje
/bin/	Sadrži upravljačke skripte za pokretanje i zaustavljanje usluga.
/etc/	Sadrži konfiguracijske dokumente.
/lib/	Sadrži <i>systemd</i> usluge. <i>Systemd</i> usluge su konfiguracijske datoteke koje definiraju kako bi <i>systemd</i> trebao upravljati određenim procesom ili aplikacijom. Ove usluge obično imaju ekstenziju datoteke <i>.service</i> . Npr. <i>confluent-zookeeper.service</i> je datoteka koje je odgovorna za pokretanje i zaustavljanje <i>ZooKeeper</i> komponente.
/libexec/	Sadrži CLI (engl. <i>command-line interface</i>) binarne datoteke za više platformi.
/share/	Sadrži JAR datoteke i dozvole koje koriste komponente platforme. JAR datoteke omogućuju uvođenje dodatnih funkcionalnosti, proširenja ili ovisnosti koje zahtijeva <i>Kafka</i> ekosustav.
/src/	Sadrži izvorne datoteke koje zahtijevaju izgradnju nezavisnu o platformi.

Tablica 3.1 Pojašnjenja direktorija stvorenih instalacijom *Confluent Platform*

Kako bi platforma ispravno radila potrebno je postaviti varijablu okruženja za početni direktorij *Confluent Platforme*:

```
export CONFLUENT_HOME=<Direktorij u kojem je Confluent
```

```
Platforma instalirana>
```

Kod 3.1 – Postavljanje CONFLUENT_HOME varijable okruženja

Zatim treba dodati direktorij *bin* u vlastiti PATH:

```
export PATH=$PATH:$CONFLUENT_HOME/bin
```

Kod 3.2 – Dodavanje direktorija bin u PATH

Pozivanjem *confluent* naredbe moguće je provjeriti ispravnost postavljanja CONFLUENT_HOME varijable:

```
confluent -help
```

Kod 3.3 – Pozivanje *confluent* naredbe

Za potrebe korištenja *Confluent* lokalnih naredbi, nužno je imati instaliranu *Javu 11* ili *8*. Za rad s platformom bez korištenja lokalnih naredbi moguće je imati novije verzije *Jave*. Trenutna preporučena verzija je *Java 17*. Konačno sve prethodno navedene usluge pokreću se naredbom:

```
confluent local services start
```

Kod 3.4 – Naredba za pokretanje usluga platforme *Confluent*

Rezultat pokretanja naredbe treba izgledati kao na slici 3.5. Na slici vidimo da su se osim *Kafke* pokrenule i neke druge komponente sadržane u platformi *Confluent Platform*. Slijedi kratko pojašnjenje svake od komponenti.

Spomenimo prvo *ZooKeepera* koji je predstavljen u prethodnom dijelu rada. Platforma se brine za njegovo postavljanje i pokretanje. Pored *ZooKeepera*, platforma pokreće i *ksqlDB* komponentu. To je komponenta koja nudi jednostavno i interaktivno SQL sučelje za obradu toka podataka na *Kafki*, bez potrebe pisanja koda u programskom jeziku kao što su *Java* ili *Python*.

Kako bi se podaci mogli povlačiti ili spremati u skladišta podataka, poput baza podataka, koriste se *Kafka* konektori. Platforma nudi korištenje *Kafka Connect API*-ja za povezivanje *Kafke* s drugim sustavima [34]. U ovom radu koristit će se *Kafka* poveznica za povezivanje s *Azure Blob Storageom*.

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.

Pored svega navedenog, *Confluent* nudi *REST Proxy* koji pruža *RESTful* sučelje za *Apache Kafka* klaster. Na taj način olakšava proizvodnju i konzumiranje poruka, pregled stanja klastera i izvođenje administrativnih radnji bez korištenja izvornog *Kafkinog* protokola [36].

Za kraj spomenimo *Control Center*. To je sustav koji se temelji na GUI-ju 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].

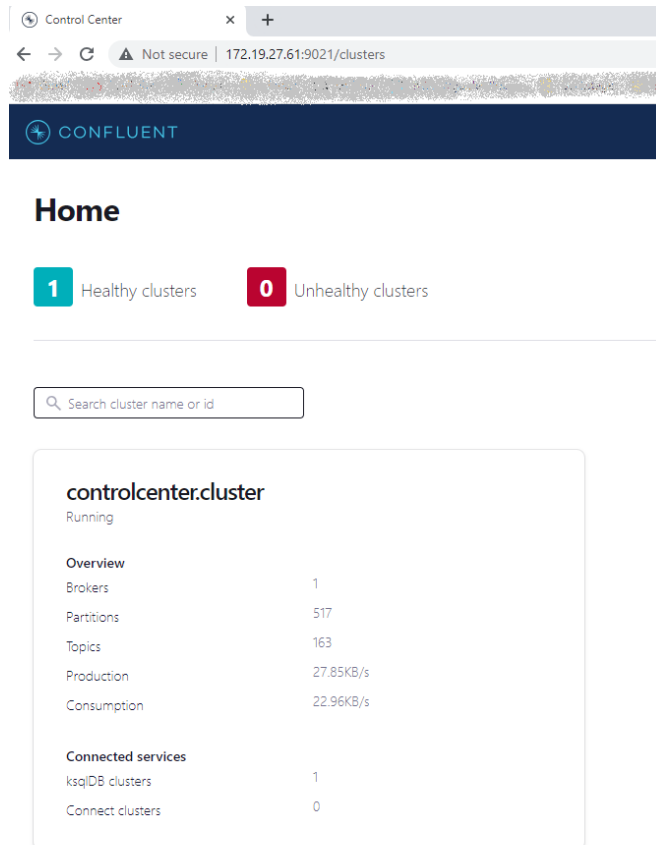
Kako bi pristupili pokrenutom sučelju *Control Center* trebamo otkriti na kojoj se IP adresi nalazi. To je moguće pozivanjem `ifconfig` naredbe. Na slici 3.6 zaokružena je IP adresa na izlazu *eth0*. U ovom primjeru to je 172.19.27.61. U pregledniku je sada moguće na toj adresi i portu 9021 posjetiti UI *Control Center* komponente. Nakon odlaska na adresu vidjet će se sučelje kao na slici 3.7.

```
The local commands are intended for a single-node development environment only,  
NOT for production usage. https://docs.confluent.io/current/cli/index.html  
  
Using CONFLUENT_CURRENT: /tmp/confluent.119443  
ZooKeeper is [UP]  
Kafka is [UP]  
Schema Registry is [UP]  
Kafka REST is [UP]  
Connect is [UP]  
ksqlDB Server is [UP]  
Control Center is [UP]
```

Slika 3.5 Rezultat pokretanja *Confluent Platforme*

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 172.19.27.61 netmask 255.255.240.0 broadcast 172.19.31.255  
    ether 02:00:0c:00:00:00  
    RX errors 0 dropped 2 overruns 0 frame 0  
    TX packets 263 bytes 30141 (30.1 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 54079 bytes 54642620 (54.6 MB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 54079 bytes 54642620 (54.6 MB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

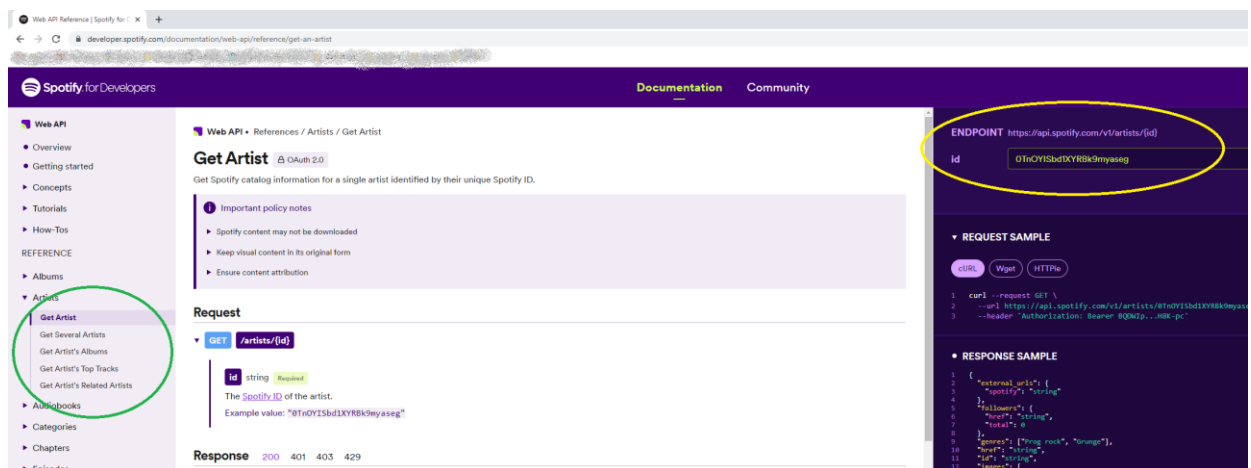
Slika 3.6 Rezultati `ifconfig` naredbe



Slika 3.7 Control Center sučelje

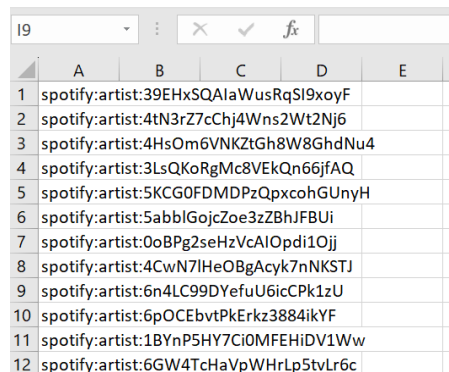
3.4.2. Pokretanje Spotify API Kafka proizvođača

U drugom poglavlju opisane su mogućnosti *Spotify API-ja*. Kao primjeri podataka koji će se povlačiti sa *Spotify API-ja* odabrani su podaci o izvođačima (engl. *artists*), njihovim albumima (engl. *albums*) i glazbenim zapisima (engl. *track*). Moguće je isprobati zahtjeve za dohvaćanje izvođača na njihovoj web stranici (Slika 3.8).



Slika 3.8 Get Artists metoda na Spotify API sučelju

Na prethodnoj slici žuto je zaokružen primjer slanja zahtjeva za izvođačem s ID-jem 0TnOYISbd1XYRBk9myaseg. Za dohvaćanje bilo kojeg izvođača potreban je njegov ID. Zbog toga je na kaggle.com pronađena csv datoteka koja sadrži ID-jeve izvođača (Slika 3.9). Također, pored naslova *Get Artist* vidi se oznaka OAuth 2.0 koja nam govori da je za obavljanje ovakvih zahtjeva potrebna autorizacija. Kao autorizaciju koristit ćemo token čije se generiranje objasnilo u drugom poglavlju.



	A	B	C	D	E
1	spotify:artist:39EHxSQAiaWusRqSI9xoyF				
2	spotify:artist:4tN3rZ7cChj4Wns2Wt2Nj6				
3	spotify:artist:4HsOm6VNKZtGh8W8GhdNu4				
4	spotify:artist:3LsQKoRgMc8VEkQn66jfAQ				
5	spotify:artist:5KCG0FMDPzQpxcohGUnyH				
6	spotify:artist:5abbIgojcZoe3zZBhJFBui				
7	spotify:artist:0oBPg2seHzVcAlOpdi1Ojj				
8	spotify:artist:4CwN7lHeOBgAcyk7nNKSTJ				
9	spotify:artist:6n4LC99DYefuU6icCPk1zU				
10	spotify:artist:6pOCEbvtPkErkz3884ikYF				
11	spotify:artist:1BYnP5HY7Ci0MFEHiDV1Ww				
12	spotify:artist:6GW4TcHaVpWHrLp5tvLr6c				

Slika 3.9 .csv datoteka s ID-jevima izvođača

U nastavku će se objasniti dohvaćanje izvođača. Dohvaćanje njihovih zapisa i albuma je slično uz razlike u izgledu URL zahtjeva. U nastavku slijedi primjer koda dohvaćanja podataka o izvođačima uz pomoć generiranja tokena korištenjem već opisanih metoda te pozivanjem *Kafka* proizvođača (engl. *producer*).

```
from requests import get
import common_functions as cf
import json
import csv
import time

topic = "artists"
header = cf.get_auth_header(cf.get_token())

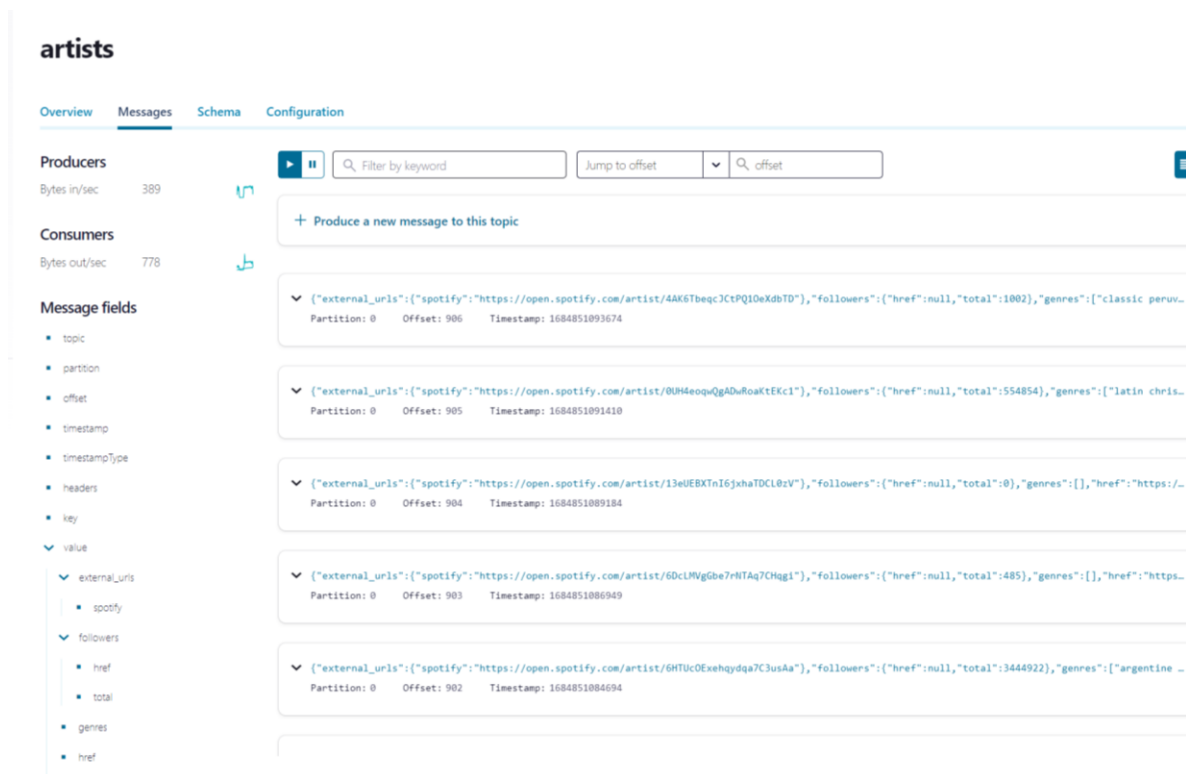
producer = cf.producer
producer.flush()
with open('artists_spotify.csv', mode='r') as file:
    csvFile = csv.reader(file)
    for lines in csvFile:
        id = lines[0].split(":")[2]
        r = get("https://api.spotify.com/v1/artists/" + id, headers =
header).json()
        producer.produce(topic, value = json.dumps(r,
```

```
indent=2).encode('utf-8'))
    producer.flush()
    time.sleep(1)
producer.close()
```

Kod 3.5 Dohvaćanje izvođača sa *Spotify API*-ja i slanje na *Kafkinu* temu

U kodu uočavamo petlju koja iterira po spomenutoj *csv* datoteci. U njoj se u svakom koraku petlje dohvaća jedan ID i GET zahtjevu predaje token u zaglavlju (engl. *headers*). Dobiveni odgovor (*r*) se pretvara u *json* format kako bi ga proizvođač *producer* mogao zaprimiti. Proizvođaču se osim vrijednosti odgovora na zahtjev (u kodu *value*) predaje i tema na koju šalje poruke. U ovom slučaju tema se zove *artists*. Na kraju se poziva naredba *producer.flush()*. Pozivanjem *flush()* metode moguće je osigurati da se proizvedeni zapisi odmah zapišu u *Kafkine* posrednike. To smanjuje rizik gubitka podataka u slučaju kvarova proizvođača ili pada sustava. Metoda *close()* poziva se kako bi se osiguralo zaustavljanje proizvođača nakon što se poruka pošalje *Kafki* [37].

Konačno, kada je programski kod napisan i platforma *Confluent* pokrenuta, potrebno je pokrenuti kod. Nakon pokretanja koda moguće je na sučelju *Control Center* pratiti učitavanje poruka (Slika 3.10).



Slika 3.10 Prikaz učitavanja podataka u temu *artists*

Na slici se jasno vidi da se stvorila nova tema imena *artists* kako je definirano u kodu. Pretpostavljeno je unutar platforme da se unutar teme stvara nulta particija u koju se šalju podaci. U drugom poglavlju objašnjeno je da autorizacijski token za *Spotify API* traje 60 minuta te je nakon 60 minuta potrebno zatražiti novi token i nastaviti slanje poruka. Slično kako je napisan programski kod za dohvaćanje izvođača, napisan je kod i za dohvaćanje albuma i glazbenih zapisa. Svaki od njih je spremljen u svoju temu, tj. za izvođače je to tema *artists*, za albume *albums*, a za zapise *artists_tracks*.

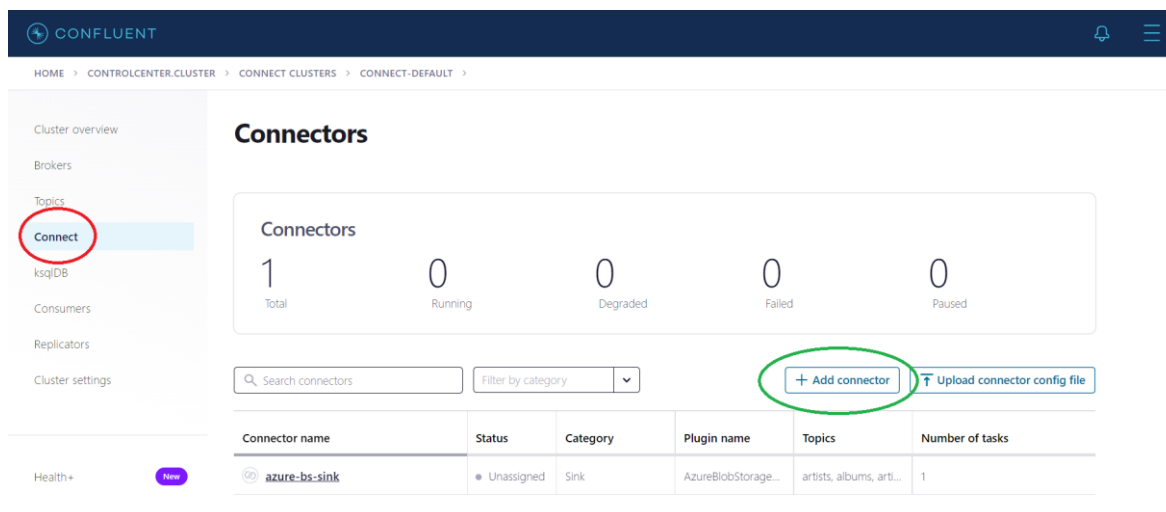
Za potrebe nastavka rada, u svaku od tema učitano je približno 20000 podataka. Podatke je nakon učitavanja u temu potrebno poslati potrošaču. Za potrošača je odabran *Azure Blob Storage*, o kojem će biti više riječi kasnije. Ono što nam je potrebno dodati je već spomenuti *Kafka Connect*.

3.4.3. Postavljanje Azure Blob Storage Sink Connectora

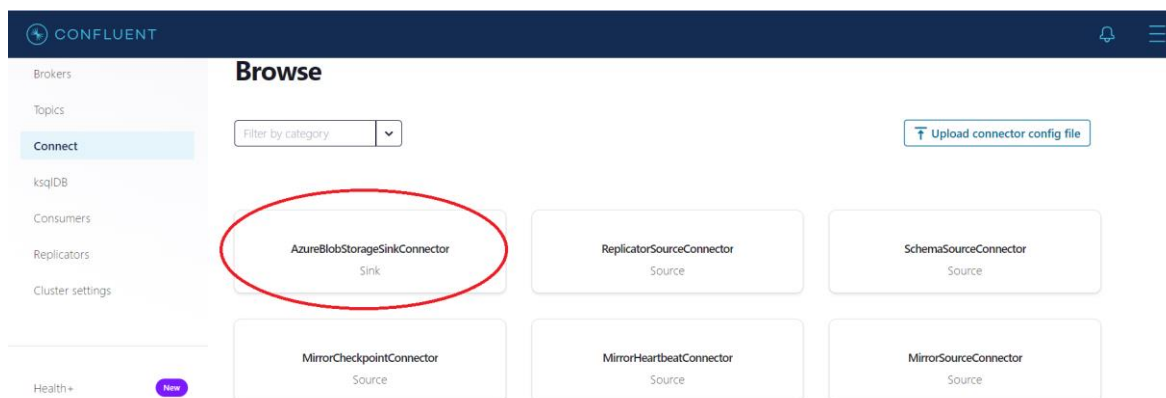
Kafka Connect uključuje dvije vrste konektora:

- Izvorni konektor (engl. *source connector*): Izvorni konektori povezuju čitave baze podataka i usmjeravaju podatke na *Kafkine* teme.
- Ponorni konektori (engl. *sink connector*): Ponorni konektori dostavljaju podatke iz *Kafkinih* tema u druge sustave po želji [38].

S obzirom na to da je ovdje potrebno poslati podatke iz *Kafkine* teme, potreban je ponorni konektor. Podatke želimo poslati na *Azure Blob Storage* i zbog toga tražimo konektor prikladan za navedene potrebe. Riječ je o *Azure Blob Storage Sink Connectoru*. To je ponorni konektor za izvoz podataka iz *Apache Kafka* teme na *Azure Blob Storage* objekte u *Avro*, *JSON*, bajtovnom ili *parquet* formatu. On povremeno provjerava podatke iz *Kafke* i zatim učitava u *Azure Blob Storage*. Konektor se može preuzeti na stranici *Confluent Platforme*. Zatim je potrebno dodati konektor u popis konektora na *Control Center* sučelju. Na sljedećoj slici zeleno je zaokružen gumb kojim se dodaje konektor. Nakon što se konektor preuzeo, platforma će ga automatski prepoznati i dodati u popis svojih konektora (Slika 3.12).

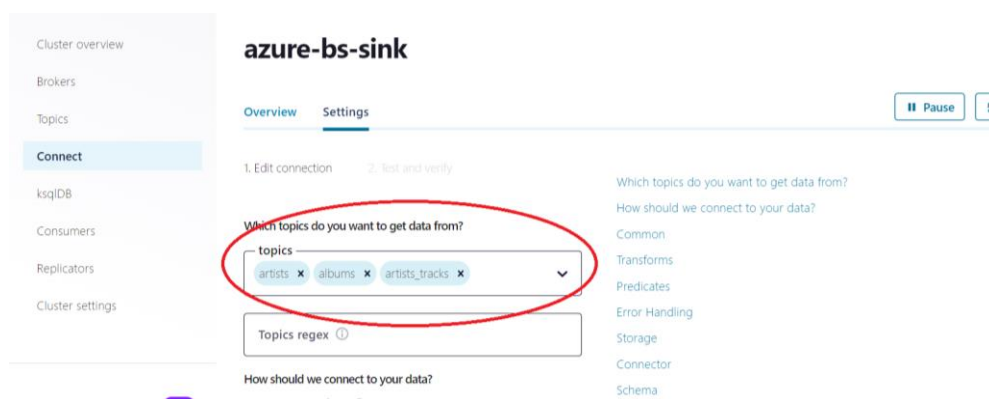


Slika 3.11 Dodavanje novog konektora



Slika 3.12 Odabir Azure Blob Storage Sink Connectora

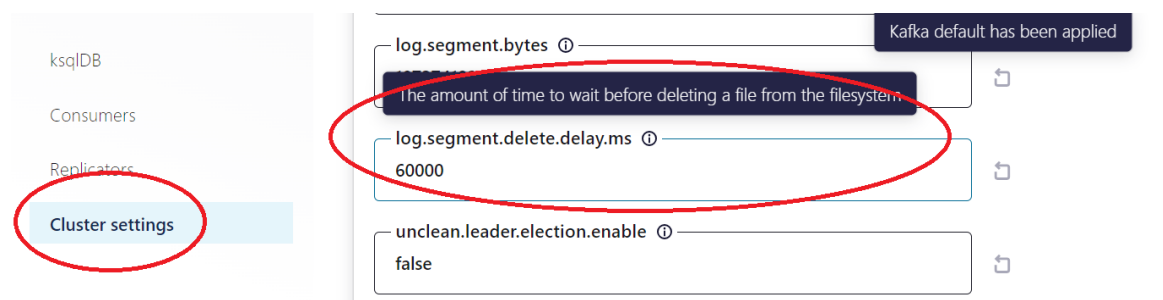
Prilikom postavljanja konektora potrebno je pripaziti da se dodaju teme s kojih se žele primati podaci. U ovom slučaju to su *artists*, *albums* i *artists_tracks* (Slika 3.13). Ako je konektor ispravno postavljen i pokrenut podaci će se preuzimati s teme i slati u *Azure Blob Storage*.



Slika 3.13 Dodavanje tema konektoru

3.5. Prednosti i mane korištenja Apache Kafka tehnologije

Glavna značajka *Kafke* je čuvanje podataka unutar vlastitog sustava. *Kafkini* posrednici imaju postavljene vrijednosti čuvanja podataka unutar teme na određeno vrijeme, na primjer četiri dana, ili dok se ne dosegne određena veličina u bajtovima. Jednom kada se to ograničenje dosegne, poruke se smatraju isteklima i one se brišu [27]. Slika 3.14 pokazuje da je moguće vrijeme čuvanja podataka postaviti u postavkama klastera unutar *Control Center* sučelja.



Slika 3.14 Postavljanje vremena zadržavanja podataka u klasteru unutar *Control Center* sučelja

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

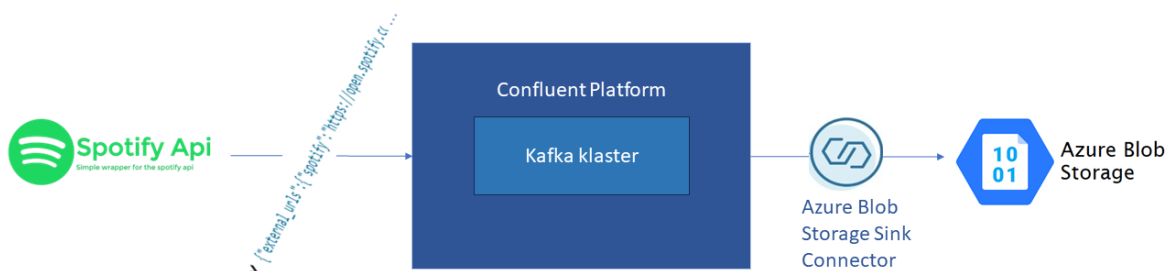
Kafka je skalabilna. U radu s *Kafkom*, može se započeti s jednim posrednikom i razviti svoj sustav sve do velikog klastera koji se sastoji od više stotina posrednika koji s vremenom rastu skalirajući se prema količini podataka [27].

Još jedna važna značajka je podrška za različite formate i tipove podataka. Tipovi podataka koji su podržani variraju između različitih baza podataka i drugih skladišnih sustava [27]. Također, *Kafka* može raditi s velikim skupovima podataka i poznat je po svom niskom kašnjenju [30].

Jedan od problema rada s *Kafkom* je njezina kompleksnost postavljanja i održavanja. Kako bi ju se svladalo potrebno je vremena, čak i osobama koje imaju tehnološko predznanje. Također, *Kafka* nema svoje sučelje za nadzor rada te je zato potrebno puno vremena u analizi i provjeri ispravnog slanja poruka. Ipak, postoje rješenja

koja se mogu koristiti zajedno s *Kafkom* kako bi se izbjeglo ručno pregledavanje *Kafkinog* rada. Ovisnost o *ZooKeeperu* može biti mana samim time što nadodaje složenosti *Kafke*. Također, ne dozvoljava *Kafka* da ima više od 200000 particija i usporava rad posrednika kada se priključuje ili kada napušta klaster. Trenutno se radi na novom rješenju bez *ZooKeepera* [31].

U ovom poglavlju općenito su predstavljeni alati za dohvaćanje podataka. Detaljnije je pojašnjen rad *Apache Kafke*. Korištenjem *Kafka* usluga unutar platforme *Confluent* dohvaćeno je oko 20000 podataka o izvođačima, isto toliko i o albumima te o glazbenim zapisima. Stvorena je poveznica između *Apache Kafke* i *Azure Blob Storagea*. Trenutni tok podataka prikazuje slika 3.15. Sljedeće poglavlje opisuje načine skladištenja velikih skupova podataka i pojašnjava skladištenje u *Azure Blob Storageu*.



Slika 3.15 Detaljniji prikaz dohvaćanja podataka iz *Spotify API*-ja pomoću *Apache Kafke* te stvaranja poveznice s *Azure Blob Storageom*

4. Skladištenje velikih skupova podataka

Zbog svojih veličina, velikim skupovima podataka rukuje se koristeći drugačija spremišta od tradicionalnih. Tvrtke poput *Googlea* i *Amazona* imaju velike podatkovne centre koji mogu spremati i obrađivati podatke s minimalnim kašnjenjem [39]. U nastavku poglavlja ukratko se navode različite vrste sustava za pohranu te opisuju razlozi skladištenja u oblacima. Na kraju se daje uvod u *Azure Blob Storage* i objašnjava kako su spremljeni podaci čije se dohvaćanje pojasnilo u prethodnom poglavlju.

4.1. Različite vrste sustava za pohranu

- 1) **Distribuirani datotečni sustavi** (engl. *Distributed File Systems*): Oni nude mogućnost pohranjivanja velikih količina nestrukturiranih podataka na pouzdan način. Jedan primjer datotečnog sustava je *Hadoop File System* (HDFS). *Hadoop* je dizajniran za velike podatkovne datoteke i vrlo je prikladan za skupnu obradu i brzi unos podataka. Naziva se distribuiranim jer je distribuiran na više datotečnih poslužitelja ili više lokacija. Programeri mogu dohvaćati datoteke s bilo koje mreže ili računala. Glavna uloga im je omogućiti korisnicima fizički distribuiranih sustava da dijele podatke i resurse koristeći zajednički datotečni sustav [40].
- 2) **NoSQL baze podataka**: Vjerojatno najvažnija vrsta za pohranu velikih skupova podataka su NoSQL sustavi za upravljanje bazama podataka. Koriste modele podataka koji se ne pridržavaju nužno transakcijskih svojstava atomičnosti, dosljednosti, izolacije i trajnosti (engl. *atomicity, consistency, isolation and durability – ACID*) [40].
- 3) **NewSQL baze podataka**: Moderni oblik relacijskih baza podataka koji čini most između SQL i NoQL baza podataka. Cilj im je održati skalabilnost kao i kod NoSQL baza podataka, ali i održati konzistentnost obavljanja transakcija tradicionalnih SQL baza podataka [40].
- 4) **Platforme za slanje upita nad velikim skupovima podataka** (engl. *Big Data Querying Platforms*): Tehnologije koje pružaju upite nad spremištima velikih

skupova podataka. Glavno svojstvo je pružanje kvalitetnog sučelja, koje omogućuje npr. slanje upita i nisku razinu kašnjenja na upite [40].

4.2. Skladištenje u oblacima

Računalni oblak odnosi se na poslužitelje koji su dostupni putem interneta, kao i na softver i baze podataka koje se vrte na tim poslužiteljima. Danas postoje mnogi pružatelji usluga u oblaku, poput *Google Cloud Platform*, *Amazon Web Services*, *IBM Clouda* i drugih. U ovom radu koristit će se *Microsoft Azure* pružatelj usluga u oblaku.

Mnoge tvrtke koriste tehnologije oblaka kako bi riješili goleme potrebe za pohranom podataka. Pohrana u oblaku je skalabilna i pristupačna. Omogućuje tvrtkama sigurnu pohranu podataka takvu da im ovlašteni korisnici mogu pristupiti kad god i s bilo kojeg mjesta. Prednost oblaka je što ima veliki kapacitet koji se neprestano razvija i širi [41].

Oblak mogu koristiti i poduzeća i krajnji korisnici. Za krajnje korisnike pohranjivanje podataka u oblak omogućuje pristup s bilo koje lokacije i sa svakog uređaja na pouzdan način. Pohrana u oblaku pruža fleksibilan pristup s više lokacija i brz i jednostavan kapacitet skaliranja te bolju podršku za poduzeća čija se potreba za pohranom mijenja tijekom vremena na više ili na manje.

Rješenja za pohranu u oblaku mogu se razlikovati između objektna i blokova pohrane. Skladištenje objekata je generički pojam koji opisuje pristup adresiranju i manipuliranju diskretnim jedinicama pohrane koje se nazivaju objekti. Nasuprot tome, podaci se u blokovnoj pohrani pohranjuju u volumene koji se nazivaju blokovima [40].

4.3. Azure Blob Storage

Azure Blob Storage Microsoftovo je rješenje za objektnu pohranu u oblaku. *Blob Storage* optimiziran je za pohranjivanje velikih količina nestrukturiranih podataka te je dizajniran za:

- posluživanje slika ili dokumenata
- pohranjivanje datoteka za distribuiran pristup
- strujanje videa i slika

- pohranjivanje podataka za sigurnosno kopiranje, oporavak od katastrofe i arhiviranje
- pohranjivanje podataka za analizu

Objekti su dostupni putem *Azure Storage REST API*-ja, *Azure PowerShell*a, *Azure CLI* ili klijentske biblioteke za *Azure Storage*.

Blob Storage nudi tri tipa resursa na *Azureu*. Resursi na *Azureu* su entiteti kojima *Azure* upravlja. To su npr. virtualne mašine (računalni resurs), virtualne mreže (mrežni resurs) ili skladišni računi (skladišni resurs). Resursi povezani s *Blob Storageom* su:

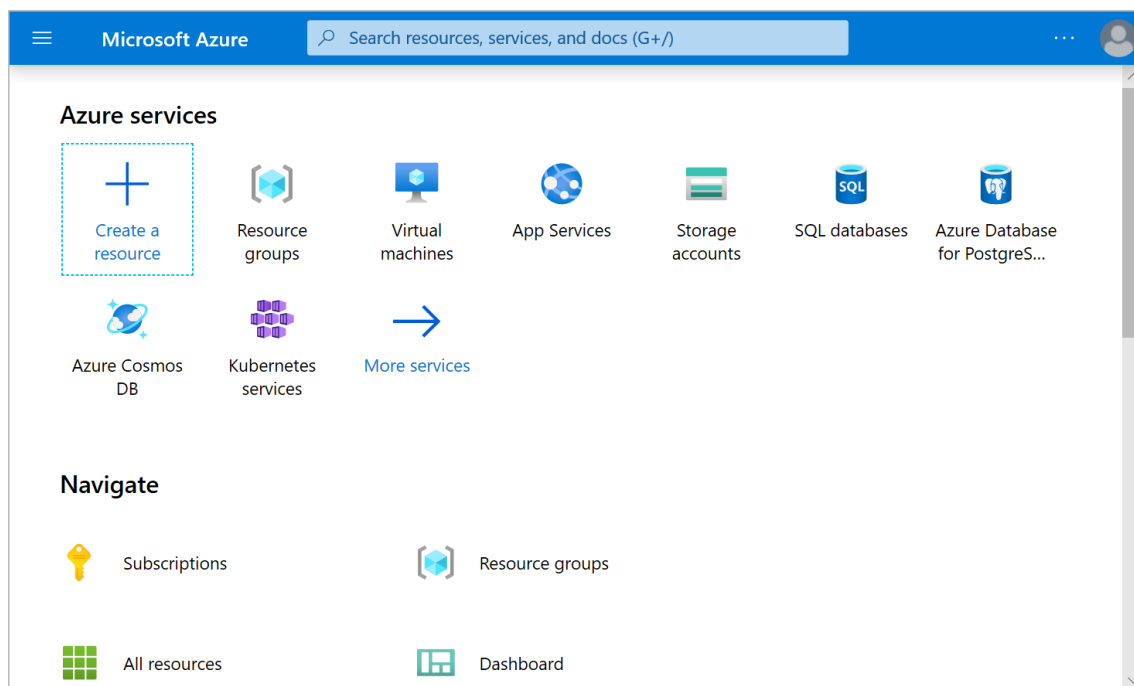
- 1) skladišni račun (engl. *storage account*),
- 2) kontejner (engl. *container*) u skladišnom računu i
- 3) blob u kontejneru.

Skladišni račun pruža jedinstveni prostor u *Azureu* za vlastite podatke. Svaki objekt kojeg se pohrani u *Azure Storageu* ima adresu koja uključuje jedinstveni naziv skladišnog računa. Kombinacija naziva računa i krajnje točke *Blob Storagea* čini osnovnu adresu za mjesto pohrane objekata. Na primjer, ako je račun za pohranu nazvan *mystorageaccount*, tada je zadana krajnja točka za *Blob Storage*: <https://mystorageaccount.blob.core.windows.net>.

Blobovi se organiziraju u direktorijima u datotečnom sustavu. Račun za pohranu može uključivati neograničen broj kontejnera, a kontejner može pohraniti neograničen broj blobova. Krajnja točka za kontejner *mycontainer* glasila bi: <https://mystorageaccount.blob.core.windows.net/mycontainer>, a krajnja točka za blob *myblob*: <https://myaccount.blob.core.windows.net/mycontainer/myblob>.

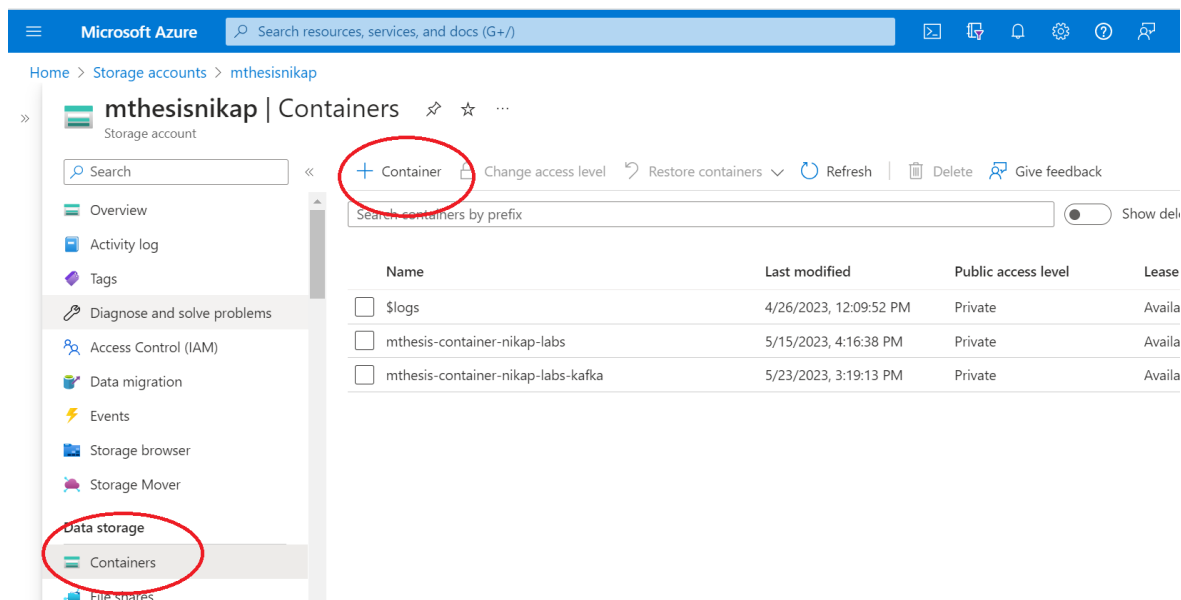
4.4. Stvaranje potrebnih resursa u Azure Cloudu

Sljedeća slika prikazuje *Azure* portal do kojeg se dolazi odabirom izbornika gore lijevo i zatim gumba Home (Slika 4.1). Pomoću njega može se upravljati *Azure* pretplatom.



Slika 4.1 Prikaz Azure portala

Na gornjoj slici vidljiv je resurs *Storage accounts* kojeg je potrebno kreirati. Ako ga nema ponuđenog, moguće ga je pronaći klikom na gumb *Create a resource*. Unutar kreiranog računa mogu se kreirati kontejneri klikom na gumb *Create* (Slika 4.2).



Slika 4.2 Stvaranje kontejnera

Prethodno su već spomenute krajnje točke do skladišnog računa, kontejnera i bloba. Te su informacije važne jer su potrebne za povezivanje drugih sustava s njima. Upravo se na taj način *Azure Blob Storage Sink Connector* povezao s točno određenim računom na *Azureu*. Slika 4.3 prikazuje koje su informacije unesene za povezivanje konektora s

Azureom. Tajni ključ računa se može pronaći unutar *Azure* računa pod *Access keys* odlomkom.

Azure

Account Name* ⓘ
mthesisnikap

Account Key* ⓘ
.....

Container Name ⓘ
mthesis-container-nikap-labs-kafka

Azure Block Size ⓘ

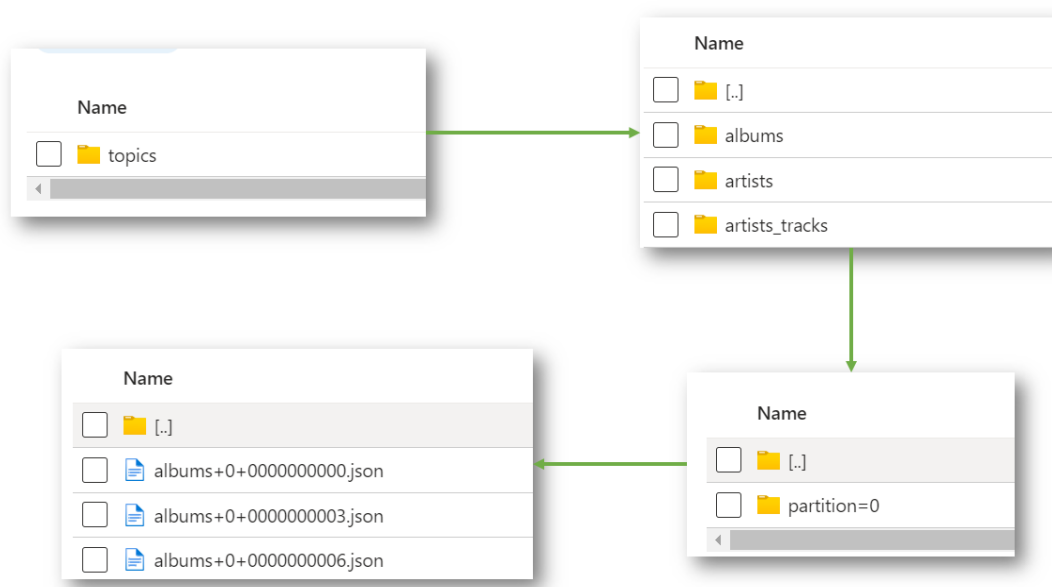
Retry Policy Type ⓘ ▼

Slika 4.3 Postavljanje *Azure* postavki za *Azure Blob Storage Sink Connector*

Slika 4.4 prikazuje odnos između stvorenog *Azure* računa *mthesisnikap*, kontejnera *mthesis-container-nikap-labs-kafka* koji je povezan s *Kafkom* te stvorenih blobova, odnosno podataka učitanih u kontejner. Svi podaci učitani u *Kafku* se šalju u isti kontejner. Unutar *Azure* kontejnera se automatski stvaraju poddirektoriji koji odgovaraju nazivima *Kafkinih* tema (Slika 4.5). Stvoreni direktoriji nisu stvarni direktoriji, već virtualni način organiziranja blobova koji daje dojam hijerarhijskog datotečnog sustava. Takav hijerarhijski odnos će kasnije omogućiti da *Apache Spark* specificira do kojih podataka želi doći, tj. ako želi dohvatiti podatke o izvođačima onda u putanji navodi direktorije `topics/artists/partition=0/*.json`.



Slika 4.4 Prikaz odnosa računa, kontejnera i blobova



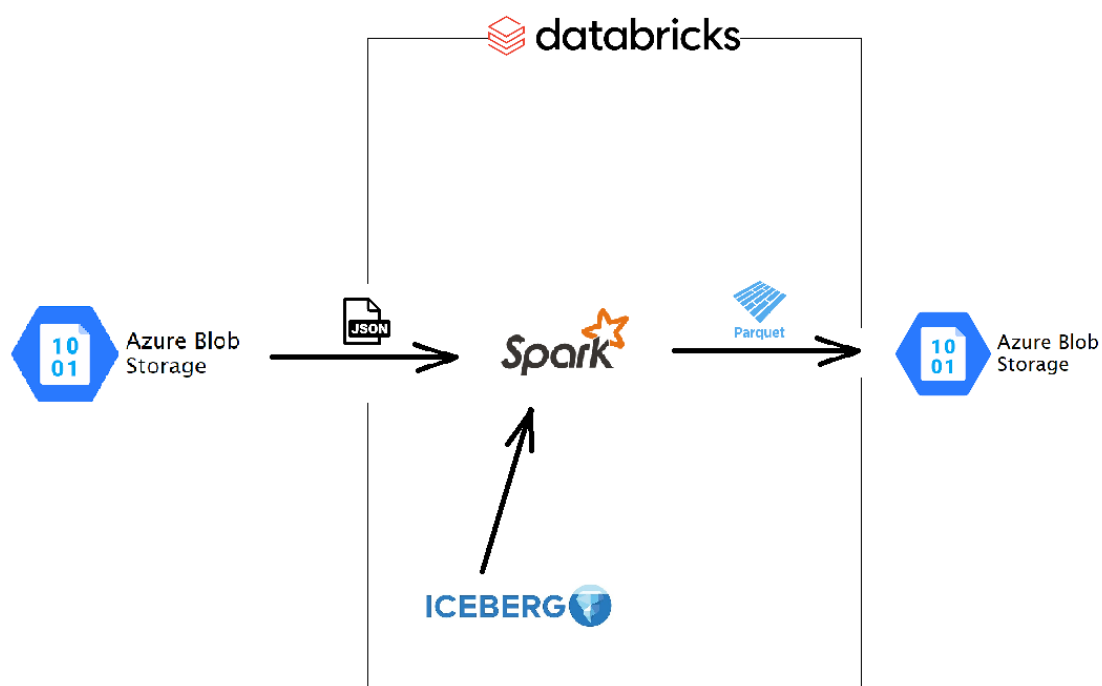
Slika 4.5 Izgled datoteka unutar kontejnera

U ovom poglavlju se dao pregled o skladištenju podataka u *Azure Blob Storage*, a u idućem poglavlju će se prikazati dohvaćanje tih podataka u platformu *Azure Databricks* i obavljanje transformacija nad dohvaćenim podacima pomoću *Apache Sparka* i *Apache Iceberga*.

5. Transformacija velikih skupova podataka

Transformacija podataka proces je pretvaranja podataka iz jednog formata u drugi, obično iz formata izvornog sustava u traženi format odredišnog sustava. Sastavni je zadatak kod upravljanja podacima, uz dohvaćanje i skladištenje podataka [42].

U ovom poglavlju pojašnjen je pojam transformacije velikih skupova podataka, a zatim je predstavljena platforma *Azure Databricks* u kojoj se podaci čitaju i transformiraju pomoću *Apache Sparka* i *Apache Iceberga*. Na kraju se transformirani podaci spremaju natrag u *Azure Blob Storage* u *parquet* formatu.



Slika 5.1 Treći korak: Transformacija podataka korištenjem *Apache Sparka* i *Apache Iceberga* unutar platforme *Azure Databricks*

5.1. Općenito o transformaciji velikih skupova podataka

Danas podaci dolaze s mnogo različitih izvora i u mnogo različitih oblika. Također, podaci izvučeni iz izvorne lokacije često su nestrukturirani i ne mogu se koristiti u izvornom obliku. Jedna od mogućih opasnosti u radu s podacima je kompatibilnost podataka. Transformacija podataka omogućuje pretvaranje podataka iz bilo kojeg izvora u

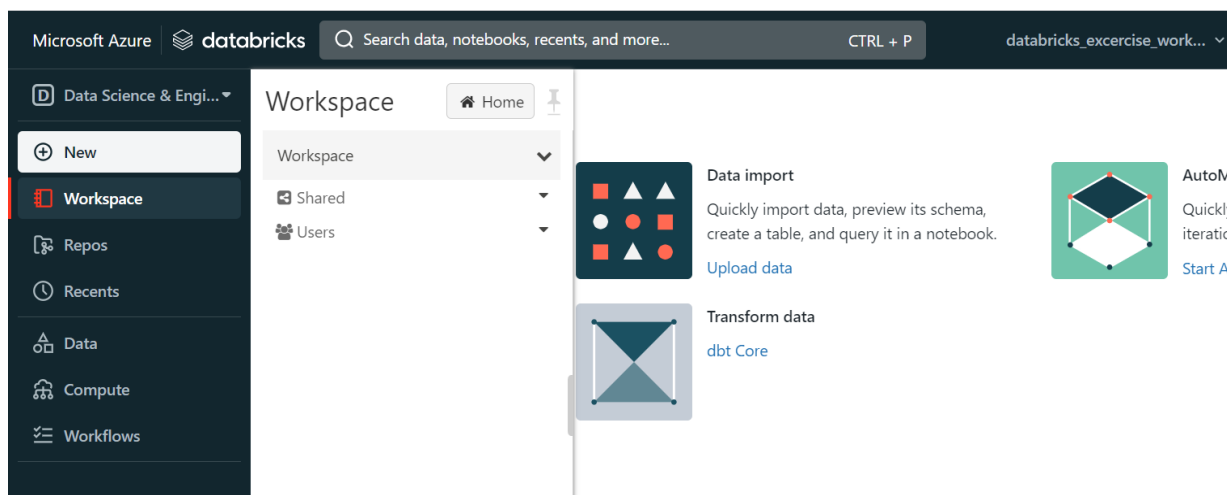
format koji se može integrirati, pohraniti i na kraju analizirati [42]. Transformacija podataka odvija se u sljedećih pet koraka:

- 1) **otkriće** (engl. *discovery*): Prvi korak je identificirati i razumjeti podatke u njihovom izvornom formatu uz pomoć alata za profiliranje podataka. Potrebno je pronaći sve izvore i vrste podataka koje je potrebno transformirati. Ovaj korak pomaže u razumijevanju kako podatke treba transformirati da bi se uklopili u željeni format.
- 2) **preslikavanje** (engl. *mapping*): Preslikavanje uključuje određivanje trenutne strukture kao i transformaciju koja je potrebna za dobivanje željene strukture.
- 3) **generiranje koda** (engl. *code generation*): U ovom koraku kreira se kod koji je potreban za pokretanje procesa transformacije pomoću platforme ili alata za transformaciju podataka.
- 4) **izvršenje** (engl. *execution*): Podaci se konačno, uz pomoć koda, pretvaraju u željeni format. Transformirani podaci se šalju odredišnom sustavu koji može biti skladište podataka.
- 5) **pregled** (engl. *review*): Procjenjuje se je li pretvorba podataka imala željene rezultate nakon transformacije [43].

5.2. Azure Databricks

Na početku rada spomenuto je da podaci koji su spremljeni u *Azure Blob Storage* odlaze u *Azure Databricks*. *Azure Databricks* je skup alata za izgradnju, implementaciju, dijeljenje i održavanje velikih podatkovnih rješenja. Koristi se kao platforma za izradu i implementaciju procesa rada s podacima. *Azure Databricks* pruža objedinjeno sučelje i alate za većinu podatkovnih zadataka, uključujući upravljanje radnim procesima obrade podataka, rad u SQL-u, unos podataka, generiranje vizualizacija i nadzornih ploča, upravljanje sigurnošću te modeliranje strojnog učenja [45].

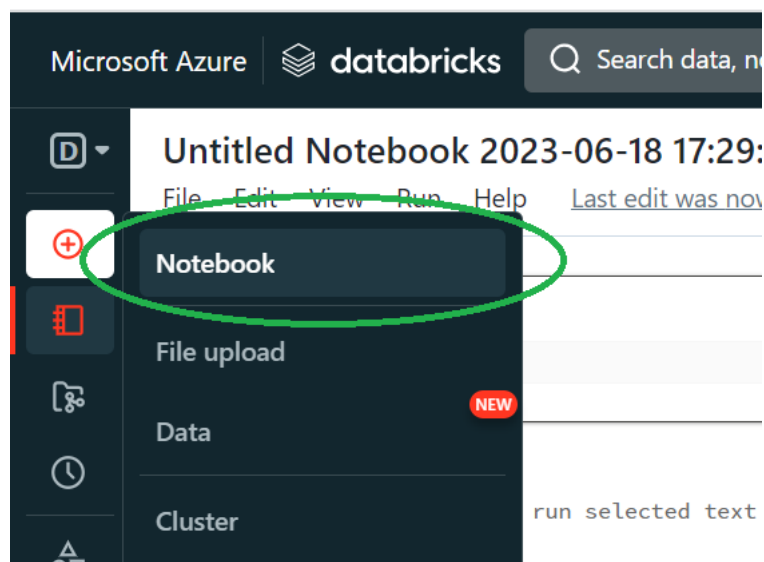
Azure Databricks je usko povezan s *Azureovim* skladištima podataka i računalnim resursima kao što je ranije spomenuti *Azure Blob Storage*. Obavljanje posla unutar klastera *Azure Databricks* svodi se na stvaranje bilježnice (engl. *notebook*) unutar koje se piše željeni kod [46]. Sljedeća slika prikazuje izgled sučelja platforme *Azure Databricks*.



Slika 5.2 Prikaz Azure Databricks sučelja

Prednost korištenja *Azure Databricksa* je što može obraditi velike količine podataka i kako je dio *Azurea*, podaci su izvorno u oblaku. Jednostavno je postaviti i konfigurirati klastere te podržava više jezika. Glavni jezik je *Scala*, ali dobro radi i s *Pythonom*, *SQL-om* i *R-om*. Ipak nedostatak je što nije integriran s Gitom ili bilo kojim drugim alatom za kontrolu verzija [47].

U nastavku ovog rada koristit će se *Apache Spark* te će se kod pisati u *PySparku*. *PySpark* je *Pythonov API* za *Apache Spark*. Za početak je potrebno unutar *Databricks* sučelja kreirati novu bilježnicu za pisanje programskog koda u *PySparku* (Slika 5.3).



Slika 5.3 Stvaranje bilježnice unutar Azure Databricksa

Prije pisanja samog koda i obavljanja transformacija nad podacima potrebno je pojasniti što je *Apache Spark*, koje su njegove prednosti i kako radi.

5.3. Uvod u Apache Spark

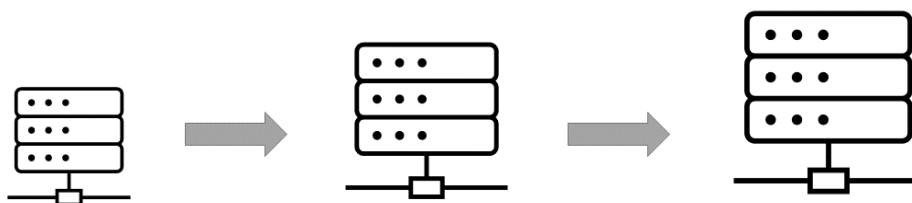
Spomenuti *Azure Databricks* usko je povezan s *Apache Sparkom*. *Spark* je 2009. godine začeo Matei Zaharia na sveučilištu *Berkeley*. Isti tvorac stvorio je i *Databricks*. *Spark* se pojavio kao klasterski računalni okvir za analizu velikih skupova podataka [48].

Spark je ubrzo postao vodeća platforma za rad u SQL-u nad velikim skupovima podataka, skupnu obradu, obradu toka i strojno učenje. Može raditi brzo nad velikim skupovima podataka, a zadatke obrade podataka može razdijeliti na više računala. *Spark* se može dočarati kao tvornica u kojoj su ulazi podatci, a izlazi razni uvidi, modeli ili vizualizacije. Razdiobom zadataka na više računala *Spark* zapravo čini horizontalno skaliranje (engl. *scaling out*) preko više manjih računala (Slika 5.4). Drugi način bi bio vertikalno skaliranje. Tada se dodaje više resursa, poput CPU-a, RAM-a i prostora na disku jednog računala (Slika 5.5).

Od jezika pruža potporu za rad s *Java*, *Scala*, *Python* i *R* jezicima. Koriste ga banke, telekomunikacijske tvrtke, tvrtke za izradu igara, vlade i svi tehnološki divovi kao što su *Apple*, *IBM*, *Meta* i *Microsoft* [49].



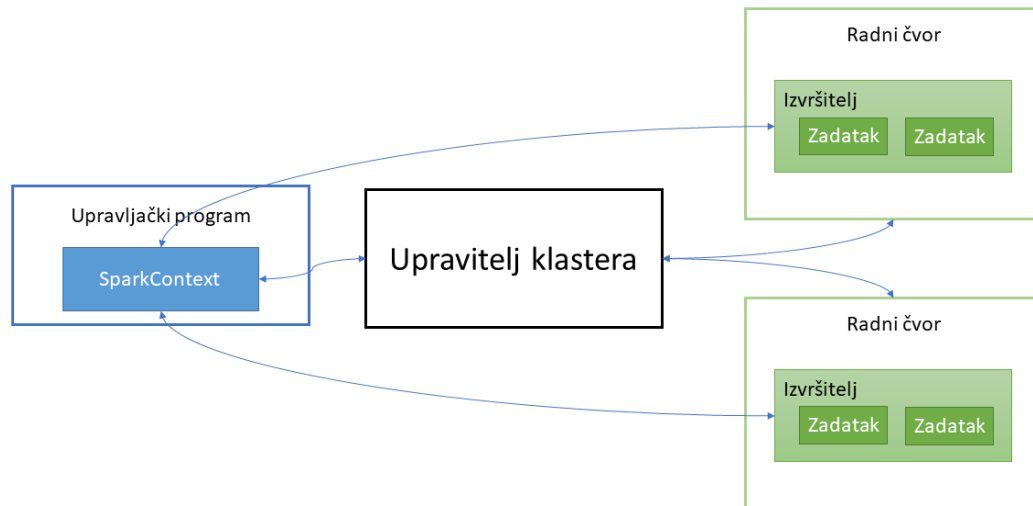
Slika 5.4 Horizontalno skaliranje



Slika 5.5 Vertikalno skaliranje

Spark aplikacije pokreću se kao neovisni procesi na klasteru. Njima upravlja *SparkContext* objekt unutar glavnog programa koji se još naziva upravljačkim programom (engl. *driver program*). *SparkContext* povezuje se s upraviteljem klastera koji raspodjeljuje resurse među aplikacijama. Nakon što se poveže, *Spark* dobije izvršitelje (engl. *executors*)

na čvorovima (engl. *node*) u klasteru. Izvršitelji su procesi koji izvođe zadatke koje im *SparkContext* pošalje na izvođenje. Na slici 5.6 mogu se uočiti veze koje prikazuju komunikaciju između *SparkContext* objekta, upravitelja klasterom i radnih čvorova, tj. izvršitelja na njima [50].



Slika 5.6 Komunikacija *Apache Sparka* i klastera

Upravitelj klastera je entitet ili program koji se brine za dodjelu resursa zadatku. On pregledava radne čvorove i osigurava zadatku onoliko prostora koliko mu je potrebno. Informacije o količini i lokaciji resursa predaje *SparkContextu* koji predstavlja vezu s klasterom. Upravljački program je odgovoran da uzme programski kod, npr. u *Pythonu*, prevede ga u *Spark* korake i zatim obradi preko radnih čvorova.

5.4. Primjena *Apache Sparka*

Do sada je pojašnjeno zašto se unutar toka podataka obavljaju transformacije, zatim je predstavljena platforma *Azure Databricks* i konačno *Apache Spark*. *Spark* je alat koji će se u nastavku koristiti za transformiranje podataka povučenih iz *Spotify API*-ja.

5.4.1. Povezivanje s *Azure Blob Storageom*

Nakon što je dan kratak uvod u *Apache Spark* i stvorena bilježnica unutar *Databricksa*, sljedeće što je potrebno napraviti je povezati stvorenu bilježnicu s *Azure Blob Storageom* u kojem su spremljeni podaci o izvođačima, albumima i glazbenim zapisima.

Za povezivanje su potrebni sljedeći podaci: naziv *Azure* računa, njegov ključ i naziv kontejnera. U bilježnicu se unosi sljedeći isječak *PySpark* koda:

```
storage_account_name = "mthesisnikap"
storage_account_key = "bXloQdM*****taunB/g=="
container = "mthesis-container-nikap-labs-kafka"

spark.conf.set(f"fs.azure.account.key.{storage_account_name}"
               ".blob.core.windows.net", storage_account_key)
```

Kod 5.1 Postavljanje varijabli za autentifikaciju s *Azure Blob Storage*om

Prva tri retka spremaju naziv računa, njegov ključ i naziv kontejnera u varijable. Posljednja naredba *spark.conf.set()* je metoda koja se koristi za postavljanje konfiguracijskih svojstava unutar *Apache Sparka*, odnosno omogućuje programsku izmjenu *Spark* konfiguracije. Rezultat izvođenja prethodnih redaka koda je postavljeni ključ za *Azure* račun te sada *Spark* može pristupiti podacima unutar *Azure Blob Storagea* i komunicirati s njima.

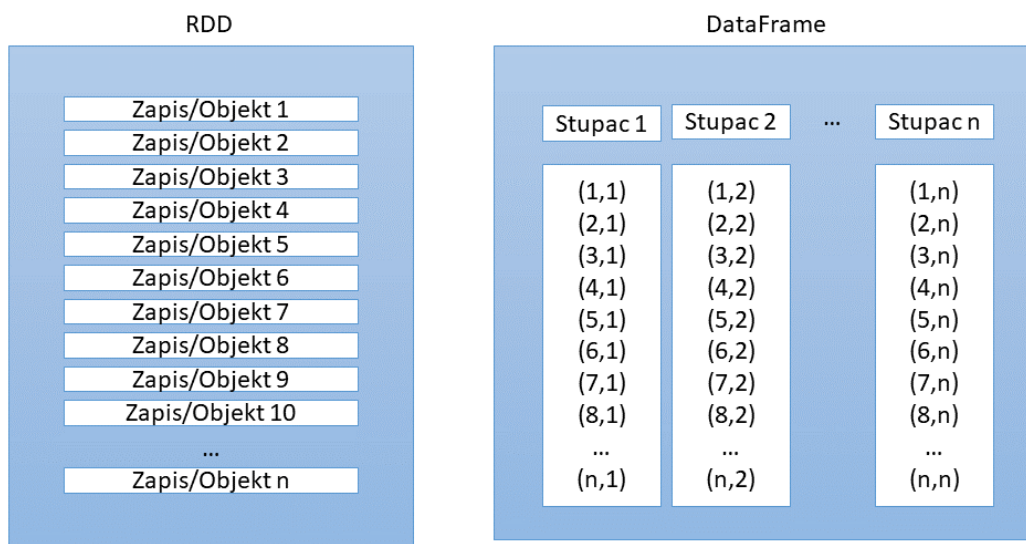
U nastavku će se koristiti *DataFrame* kao struktura u koju će se učitavati podaci. Osim *DataFramea* postoji i *RDD* (engl. *Resilient Distributed Datasets*) struktura podataka. Prije nego što se pojasni učitavanje podataka iz *Azure Blob Storagea* u *DataFrame*, slijedi kratak pregled razlika između *RDD* i *DataFrame* struktura podataka.

5.4.2. Strukture podataka u *Apache Sparku*

Dvije glavne strukture podataka u *Sparku* su *RDD-ovi* i *DataFrameovi* (DF). Na niskoj razini podaci su predstavljeni kao otporni distribuirani skupovi podataka, tj. *RDD-ovi* (engl. *Resilient Distributed Datasets*). Otpornost (engl. *resilient*) se odnosi na sposobnost obnove podataka u slučaju kvara, distribuiranost (engl. *distributed*) pojašnjava mogućnost distribuiranja podataka među različitim čvorovima u klasteru, a skup podataka (engl. *dataset*) označava kolekciju podataka. *RDD* je kolekcija zapisa namijenjena samo za čitanje (engl. *read-only*). *RDD-ovi* omogućuju eksplicitno pohranjivanje podataka na disk ili u memoriju tako što pružaju strukturu podataka koja je otporna na pogreške i koja kontrolira particioniranje podataka te njima manipulira korištenjem bogatog skupa operatora [48].

Iako je *RDD* dugo vremena bila jedina i dominantna struktura podataka *Sparka*, počeo se koristiti i *DataFrame*. Razlika između *RDD*-ja i *DataFramea* je što su podaci u *DataFrameu* raspoređeni u imenovane stupce. Pobljiže to opisuje slika 5.7.

Unutar *RDD*-a svaki zapis je neovisan objekt te se transformacije izvode na razini zapisa. Naglasak je na cijeloj kolekciji, a ne na strukturi. S druge strane, *DataFrame* organizira zapise u stupce. Iz tog razloga je *DataFrame* sličniji relacijskim bazama podataka. Transformacije se obavljaju nad stupcima ili nad cijelim *DataFrameom*. Obično se kod *DataFramea* zapisima ne pristupa horizontalno, tj. zapis po zapis, kao kod *RDD*-a. *DataFrame* može obraditi strukturirane i polu-strukturirane podatke zbog svoje sličnosti s relacijskim bazama podataka. Pored toga, može upravljati shemom. Za rad s *Apache Spark* tehnologijama, sve se više kao struktura podataka koristi *DataFrame*.



Slika 5.7 Primjer izgleda RDD-a i DF-a

5.4.3. Čitanje podataka iz Azure Blob Storagea u Spark DataFrame

Određeno je da će se kao struktura u koju će se učitavati podaci koristiti *DataFrame*, a razlog tomu je što *DataFrame* nalikuje relacijskim bazama podataka i ima prikaz podataka po stupcima. Pregledom jednog *json* podatka unutar *Azure Blob Storagea* može se pretpostaviti koji će se stupci pokazati unutar *DataFramea*, a malo detaljnijom

analizom mogu se početi sagledavati transformacije koje će se obavljati nad podacima (Slika 5.8).

Slika 5.8 Isječak *json* podatka unutar *Azure* kontejnera

```
df_artists =
spark.read.option("inferSchema", "true").option("multiline", "true").json
(f"wasbs://{container}@{storage_account_name}.blob.core.windows.net/top
ics/artists/partition=0/*.json")

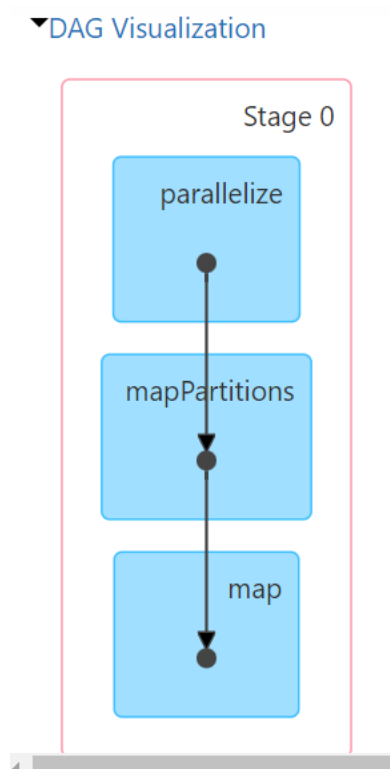
df_artists.show()
```

Prethodne naredbe dohvaćaju sve *json* podatke s lokacije [mthesis-container-nikap-labs-kafka@mthesisnikap.blob.core.windows.net/topics/artists/partition=0](https://mthesisnikap.blob.core.windows.net/topics/artists/partition=0) i spremaju unutar *DataFramea* pod imenom `df_artists`. Za kraj se poziva naredba, tj. akcija, `show()` kako bi se vidjeli rezultati spremljeni u *DataFrame*. Pokretanjem koda 5.2 pokreću se poslovi na upravljačkom programu (engl. *driver*) što prikazuje slika 5.9.

Cancel •• Running command...

Slika 5.9 Pokretanje koda za dohvaćanje podataka i spremanje u *DataFrame*

Nakon što se podaci dohvate i zadatak izvrši, *Databricks* omogućava detaljniji pogled na svaki od zadataka pa ako se pobliže analiziraju zadaci može se vidjeti da je svaki od njih kreirao direktni aciklički graf, DAG (engl. *Direct Acyclic Graph*). Primjer za nulti zadatak prikazan na slici 5.10. Na slici se vidi da postoje određene faze (engl. *stages*) pod kojima se zadatak izvršavao. Osim toga,, mogu se vidjeti i drugi detalji izvršavanja zadatka kao što su podaci o upravljačkom programu (engl. *driver*) (Slika 5.11). Kako bi se lakše razumjelo zašto *Spark* poslovi stvaraju DAG-ove potrebno je objasniti pojam lijene evaluacije u *Apache Sparku*.



Slika 5.10 Primjer DAG-a

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	
driver	10.139.64.4:38645	Active	0	18.2 KiB / 3.3 GiB	0.0 B	4

Slika 5.11 Detalji o upravljačkom programu

5.4.4. Lijena evaluacija u Apache Sparku

Lijena evaluacija (engl. *lazy evaluation*) je koncept koji je uveden u *Apache Spark* i samo je jedna od mnogih strategija evaluacije. To je strategija koja odgađa evaluaciju izraza dok njegova vrijednost ne bude potrebna. Suprotno od lijene evaluacije je striktna evaluacija (engl. *eager evaluation*) kojom se koristi većina programskih jezika. Kod striktno evaluacije se evaluacija izraza odvija odmah [53].

U *Sparku*, lijena evaluacija označava da je moguće primijeniti onoliko transformacija koliko se želi, ali izvršavanje procesa nad njima neće započeti dok se ne pozove akcija. *Spark* transformacije su funkcije koje strukture uzimaju kao ulaz i stvaraju jednu ili više struktura kao izlaz. Ulaznu strukturu ne mijenjaju, ali nad izlazom izvršavaju transformacije kao što su mapiranje – *map()*, filtriranje – *filter()*, redukcija po ključu – *reduceByKey()* i slične [54].

Ključna stvar je da kada se transformacija primjeni nad strukturom, ona se neće izvršiti odmah. *Spark* će, koristeći primijenjene operacije i izvornu strukturu, stvoriti usmjereni aciklički graf – DAG (engl. *Directed Acyclic Graph*). Graf će se graditi sve dok se ne pozove akcija. Akcija je jedan od načina slanja podataka od izvršitelja do upravljačkog programa. Primjeri akcija su *show()*, *count()*, *collect()* i *save()*. Zato su transformacije u *Sparku* lijene [54].

Transformacije stvaraju RDD-ove/DF-ove jedan iz drugog, ali kada želimo raditi sa stvarnim skupom podataka, tada se poziva akcija. Rezultati akcija se spremaju unutar upravljačkog programa ili u vanjski sustav za pohranu [54].

5.4.5. Obavljanje transformacija nad podacima

Na slici 5.8 narančasto su istaknuti neki od mogućih imena stupaca unutar *DataFramea*. Ipak, *json* podatak je kompleksan pa će neki od podataka koji bi mogli biti samostalni stupci ostati sadržani unutar stupaca. Transformiranjem se želi razdvojiti sve važne podatke u samostalne stupce i ukloniti one koji nose nevažne informacije. Prvo će se to učiniti za podatke o izvođačima, zatim za albume i konačno za glazbene zapise. Za svaki od njih kreiran je po jedan *DataFrame* nad kojim će se izvršavati transformacije. Konačni oblici *DataFrameova* će se spremiti u jedan veliki *DataFrame* koji će u sebi sadržavati sve ključne informacije o glazbeniku, njegovim albumima i glazbenim zapisima.

Kada se podaci dohvate, kao rezultat naredbe `show()` podaci su prikazani u obliku tablice sa sljedećim stupcima: `external_urls`, `followers`, `genres`, `href`, `id`, `images`, `name`, `popularity`, `type` i `uri`. Opisi tih vrijednosti dani su u drugom poglavlju unutar tablice 2.1. Ipak, unutar tih stupaca nalaze se podaci koje bi htjeli izdvojiti u vlastite stupce. Kako bi se dobilo više informacija poziva se naredba `printSchema()` koja ispisuje izgled sheme (Slika 5.12). Shema jasnije prikazuje koje se sve informacije mogu dobiti iz jednog zapisa o izvođaču. Također, unutar sheme uočavamo dva tipa podataka koja se mogu raščlaniti na manje dijelove: `struct` i `array`, odnosno strukturu i polje.

Apache Spark nudi nekoliko funkcija koje će riješiti gore navedeni problem. *Spark withColumn()* funkcija koristi se za dodavanje novog stupca u *DataFrame*, promjenu vrijednosti postojećeg stupca, pretvaranje tipa podataka stupca ili izvođenje novog stupca iz postojećeg. Rezultat funkcije je novi *DataFrame* [55]. Funkcija `withColumn()` može se koristiti u kombinaciji s funkcijama `col()` i `explode()`. Metoda `col()` koristi se nad strukturama tako što iz strukture dohvaća jedan element i pretvara ga u novi stupac te predaje metodi `withColumn()` da nadoda stupac na *DataFrame*. S druge strane, metoda `explode()` koristi se nad poljima tako što stupce koji su tipa polja pretvori u tip nad kojim se može pozvati funkcija `col()` pomoću koje se stvaraju novi željeni stupci.

```
1 df_artists.printSchema()

root
 |-- external_urls: struct (nullable = true)
 |   |-- spotify: string (nullable = true)
 |-- followers: struct (nullable = true)
 |   |-- href: string (nullable = true)
 |   |-- total: long (nullable = true)
 |-- genres: array (nullable = true)
 |   |-- element: string (containsNull = true)
 |-- href: string (nullable = true)
 |-- id: string (nullable = true)
 |-- images: array (nullable = true)
 |   |-- element: struct (containsNull = true)
 |       |-- height: long (nullable = true)
 |       |-- url: string (nullable = true)
 |       |-- width: long (nullable = true)
 |-- name: string (nullable = true)
 |-- popularity: long (nullable = true)
 |-- type: string (nullable = true)
 |-- uri: string (nullable = true)
```

Slika 5.12 Prikaz sheme

Osim prethodno navedenih metoda, koristi se i metoda *drop()* za uklanjanje neželjenih stupaca. Recimo, nakon što su se iz određenog polja izvukli željeni stupci, stupac koji je sadržavao to polje se može ukloniti metodom *drop()*. Slijedi primjer koji pojašnjava korištenje prethodnih metoda nad *DataFrameom* *df_artists*.

```
from pyspark.sql.functions import explode, col

df_artists_final = df_artists.withColumn("images_explode",
explode("images"))\
.withColumn("images_url", col("images_explode.url"))\
.drop("images", "images_explode")
df_artists_final = df_artists_final.withColumn("genres_explode",
explode("genres")).drop("genres")
df_artists_final = df_artists_final.withColumn("spotify_url",
col("external_urls.spotify")).drop("external_urls")
df_artists_final = df_artists_final.withColumn("total_followers",
col("followers.total")).drop("followers")
df_artists_final = df_artists_final.drop("href", "spotify_url")
```

Kod 5.3 Obavljanje transformacija nad *DataFrameom*

U prvom retku se dohvaćaju metode *explode()* i *col()* iz *pyspark.sql.functions* biblioteke. Zatim se definira novi *DataFrame* *df_artists_final* u koji će se spremati rezultati naredbe *withColumn()*. Ako se promotri slika 5.12 može se uočiti da je jedno od polja unutar zapisa stupac *images*. Nad tim stupcem se poziva metoda *explode()* čiji su rezultati vidljivi u novom stupcu *images_explode*. Zatim se nad tim stupcem poziva metoda *col()* kako bi se kreirao stupac koji u sebi sadrži URL slike, tj. *images_url*. Nakon toga se uklanjaju stupci *images* i *images_explode* koji više nisu potrebni. Slični koraci se ponavljaju nad ostalim poljima i strukturama dok se ne dobije željeni izgled *DataFramea* (Slika 5.13).

20 display(df_tracks_final)

▶ (3) Spark Jobs

▶ df_tracks_final: pyspark.sql.dataframe.DataFrame = [album_track_id: string, is_album_playable: boolean ... 10 more fields]

Table

▼

+

	album_track_id	is_album_playable	artist_track_id	disc_number	duration_ms	explicit
1	6GHPteYITukOGAdLTPA3B4	true	4aP1p10BRYZO658B2NwkG	1	233066	false
2	6GHPteYITukOGAdLTPA3B4	true	5ZEAzHE2TzAwUcOj6jMlgf	1	233066	false
3	6GHPteYITukOGAdLTPA3B4	true	2PT695UML2m3qBDsamlcB3	1	233066	false
4	6GHPteYITukOGAdLTPA3B4	true	4aP1p10BRYZO658B2NwkG	1	227626	false
5	6GHPteYITukOGAdLTPA3B4	true	5ZEAzHE2TzAwUcOj6jMlgf	1	227626	false
6	6GHPteYITukOGAdLTPA3B4	true	2PT695UML2m3qBDsamlcB3	1	227626	false
7						

Slika 5.13 Isječak konačnog *DataFramea*

Koraci dohvaćanja i transformiranja *DataFramea* su pojašnjeni na primjeru izvođača, ali isto vrijedi i za albume i glazbene zapise. Nakon transformiranja tih triju grupa podataka, dobiju se tri *DataFramea* sa stupcima koji će se radi preglednosti navesti u tablici 5.1.

DataFrame	Nazivi stupaca
df_artists_final	id, name, popularity, type, uri, images_url, genres_explode, total_followers
df_albums_final	album_group, artists_id, image_url, album_name, release_date, release_date_precision, total_tracks,
df_tracks_final	album_track_id, is_album_playable, artist_track_id, disc_number, duration_ms, explicit, track_id, is_track_playable, track_name, track_popularity, track_number, track_type

Tablica 5.1 Krajnji nazivi stupaca nakon transformacija

Konačno, želimo spojiti sva tri *DataFramea* u jedan koji će se povezati preko izvođača, tj. želimo imati izvođača i informacije o njegovim albumima i glazbenim zapisima. Koristi se metoda *join()* koja ima istu ulogu kao i u SQL-u, tj. spaja dvije tablice u jednu.

```
df_final =
df_artists_final.join(df_albums_final).where(df_artists_final["id"] ==
df_albums_final["artists_id"])
df_final = df_final.join(df_tracks_final).where(df_final["id"] ==
```

```
df_tracks_final["artist_track_id"])
```

Kod 5.4 Spajanje triju *DataFrame*ova u jedan veliki *DataFrame*

Prvo se spaja *df_artists_final DataFrame* s *df_albums_final DataFrame*om preko *id*-ja izvođača. Zatim se rezultat tog spajanja spaja s *df_tracks_final* također preko *id*-ja izvođača. Rezultat je veliki *DataFrame* koji sadrži sljedeće stupce: *id*, *name*, *popularity*, *type*, *uri*, *images_url*, *genres_explode*, *total_followers*, *album_group*, *artists_id*, *image_url*, *album_name*, *release_date*, *release_date_precision*, *total_tracks*, *album_track_id*, *is_album_playable*, *artist_track_id*, *disc_number*, *duration_ms*, *explicit*, *track_id*, *is_track_playable*, *track_name*, *track_popularity*, *track_number* i *track_type*. Stupci sadržani u konačnom *DataFrameu* sadrže sve ključne informacije o glazbeniku, njegovim albumima i glazbenim zapisima koje su se mogle dohvatiti iz *Spotify API*-ja. Kao posljednji korak transformiranja je pohrana podataka u *parquet* obliku natrag u *Azure Blob Storage* korištenjem *Apache Iceberg* tablice što se detaljnije objašnjava u nastavku poglavlja.

5.5. Apache Iceberg

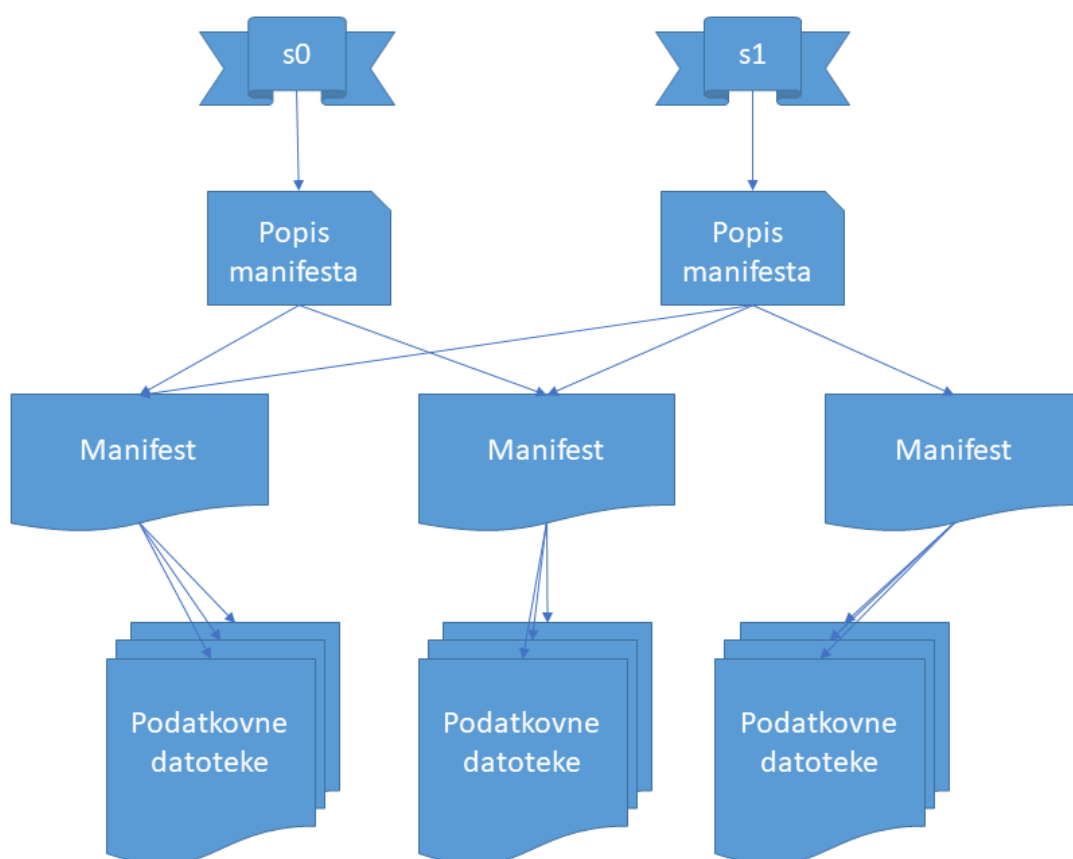
Apache Iceberg otvoreni je format tablice dizajniran za tablice velikih veličina koji pomaže pojednostaviti obradu podataka na velikim skupovima podataka. Brz je, učinkovit i pouzdan te vodi evidenciju o tome kako se skupovi podataka mijenjaju tijekom vremena. *Apache Iceberg* nudi jednostavnu integraciju s popularnim okvirima za obradu kao što su *Apache Flink*, *Apache Hive*, *Presto* i *Apache Spark* [56]. *Iceberg* je dizajniran unutar *Netflix*a kako bi riješio probleme rada s velikim količinama podataka u smislu poboljšanja dosljednosti i performansi [57].

5.5.1. Struktura Apache Iceberga

Svrha formata tablice je odrediti kako upravljati, organizirati i pratiti sve datoteke koje čine tablicu. *Apache Iceberg* može se zamisliti kao sloj apstrakcije između datoteka koje nose fizičke podatke i načina na koji su strukturirane da tvore tablicu [57]. *Iceberg* prati potpuni popis svih datoteka unutar tablice koristeći strukturu stabla (Slika 5.14). Prednost praćenja pojedinačnih datoteka je ta što više nisu potrebne skupe operacije što dovodi do poboljšanja performansi prilikom izvođenja operacija kao što su upiti nad

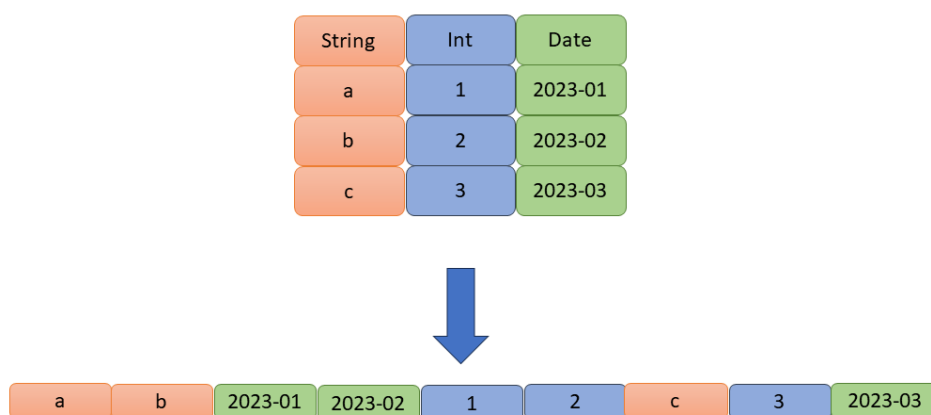
podacima u tablici. Kako *Iceberg* donosi poboljšanja može se lakše objasniti proučavanjem njegove strukture na slici 5.14.

Stanje tablice pohranjuje se u više različitih datoteka s metapodacima. Na slici su sa s0 i s1 označene datoteke s metapodacima snimke (engl. *snapshot*), Snimka, tj. *snapshot*, predstavlja verziju tablice s pripadajućom vremenskom oznakom. Snimka sadrži metapodatke o tablici kao što je shema tablice, specifikacija particije kao i put do popisa manifesta (engl. *manifest list*). Sljedeći u nizu je popis manifesta (engl. *manifest list*) koji sadrži po jedan zapis za svaku datoteku manifesta koja je povezana sa snimkom. Zapis uključuje put do manifesta i njegove metapodatke, poput broja podatkovnih datoteka. To znanje se može iskoristiti za izbjegavanje čitanja manifesta koji nisu potrebni za obavljanje operacije. Manifest sadrži popis putova do podatkovnih datoteka. Svaki unos za podatkovnu datoteku uključuje neke metapodatke o datoteci. I konačno, sama podatkovna datoteka je fizička datoteka napisana u formatima kao što su *parquet*, *avro* ili *ORC* [57].



Slika 5.14 Struktura *Apache Iceberga*

Spomenuto je da će konačna transformacija *Spotify* podataka biti pretvorba u *parquet* format. *Parquet* je format otvorenog koda koji se temelji na stupčanoj pohrani podataka. Suprotno od pohrane podataka u stupcima je pohrana u redovima koju koristi *csv* format. Razlika između ta dva tipa pohrane podataka uočljiva je na slici 5.15 na kojoj je prvo prikazana pohrana u redovima, a zatim u stupcima. U radu s velikim skupovima podataka primarno se koriste formati koji se temelje na stupčanoj pohrani podataka. Takvi formati omogućavaju lakšu kompresiju što olakšava skladištenje podataka. *Parquet* format osim podataka sadrži i metapodatke o tablici koji se mogu iskoristiti za lakše obavljanje upita.



Slika 5.15 Pohrana podataka u redovima i pohrana podataka u stupcima

5.5.2. Svojstva Apache Iceberga

1) Evolucija sheme

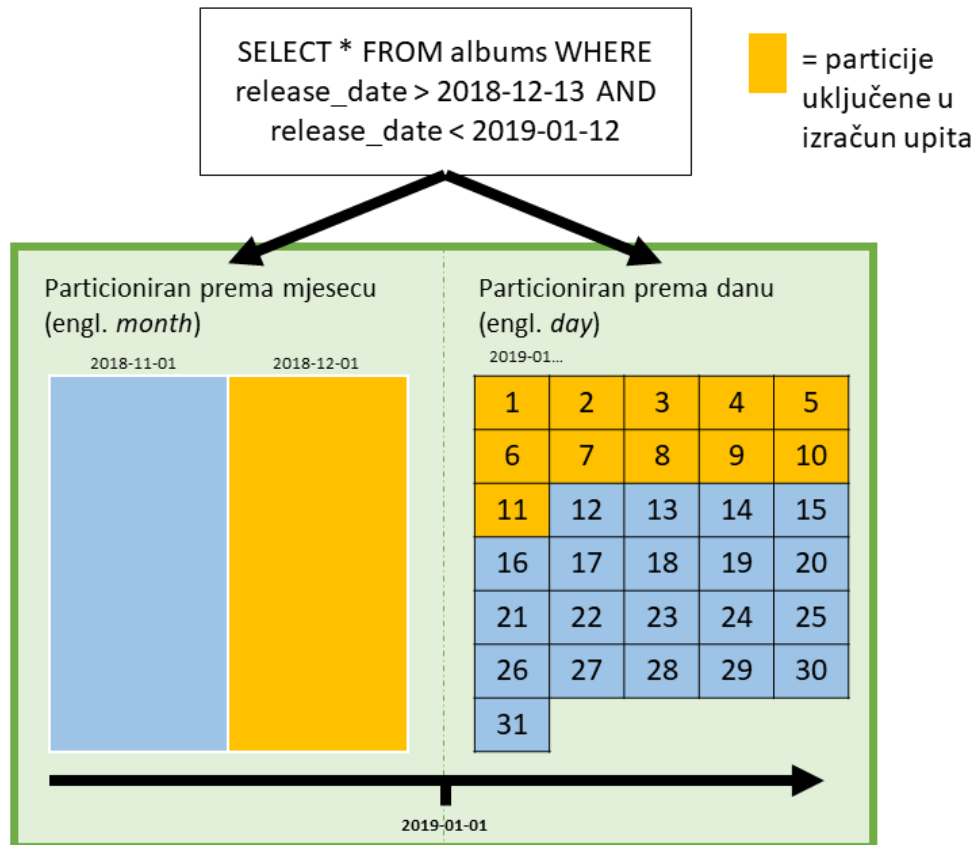
Apache Iceberg omogućuje jednostavno obavljanje promjena strukture podataka, tj. evolucije sheme. Što znači da korisnici mogu dodavati, preimenovati ili uklanjati stupce iz tablice bez ometanja podataka [56]. *Apache Iceberg* jamči da su promjene sheme neovisne i da nemaju nuspojava [57].

2) Evolucija particije

Koristeći *Iceberg* moguće je promijeniti stupac prema kojem se radi particioniranje bez razbijanja tablice. Evolucija particije objasniti će se kroz jedan jednostavan primjer (Slika 5.16).

Iceberg implementira podijeljeno planiranje. Plan upita za prvi skup podataka odvija se pomoću stare specifikacije, a zatim drugi plan upita za drugi skup s novom specifikacijom

i nakon toga kombinira sve datoteke. Na donjoj slici, *albums* tablica je do 1.1.2019. podijeljena prema mjesecima kada se specifikacija particije mijenja u dan. Stari podaci ostaju u starom formatu particije, a svi novi podaci koji dolaze zapisuju se u novom formatu [57].



Slika 5.16 Primjer particioniranja

3) Putovanje kroz vrijeme

Iceberg vodi evidenciju prethodnih snimaka tablice i to omogućuje izvođenje upita o vraćanju tablice na neku stariju vrijednost korištenjem snimaka, tj. *snapshota* [57].

```
spark.read.format("iceberg").option("snapshot-id", 1588341995546L).load("albums.name")
```

Kod 5.5 Učitavanje podataka o albumu koristeći se snimkom

Prethodni isječak koda koristeći se snimkom pod oznakom 1588341995546L pokušava učitati imena albuma. Što znači da će se učitati imena kakva su zabilježena u toj snimci bez promjena koje su se dogodile nakon te snimke.

4) Poznavanje SQL-a

SQL (engl. *Structured Query Language*) popularni je jezik za obavljanje upita nad podacima. *Apache Iceberg* omogućuje svakome tko je upoznat s SQL-om obavljanje većine operacije bez potrebe za učenjem novog jezika [56].

5.5.3. Primjena Apache Iceberga

Zbog svih prednosti i svojstava koje donosi, *Apache Iceberg* je uveden u projekt koji se provlači kroz ovaj rad. Podaci koji su prikupljeni u jedan veliki *DataFrame* će se pomoću *PySpark* naredbi spremati u obliku *Iceberg* tablice u formatu *parquet* natrag u *Azure Blob Storage*. Naziv tablice glasi `all_about_artists`, a metoda kojom se piše natrag u bazu je `writeTo()`.

```
df_final.writeTo("default.all_about_artists").option("encoding",  
"UTF-8").create()
```

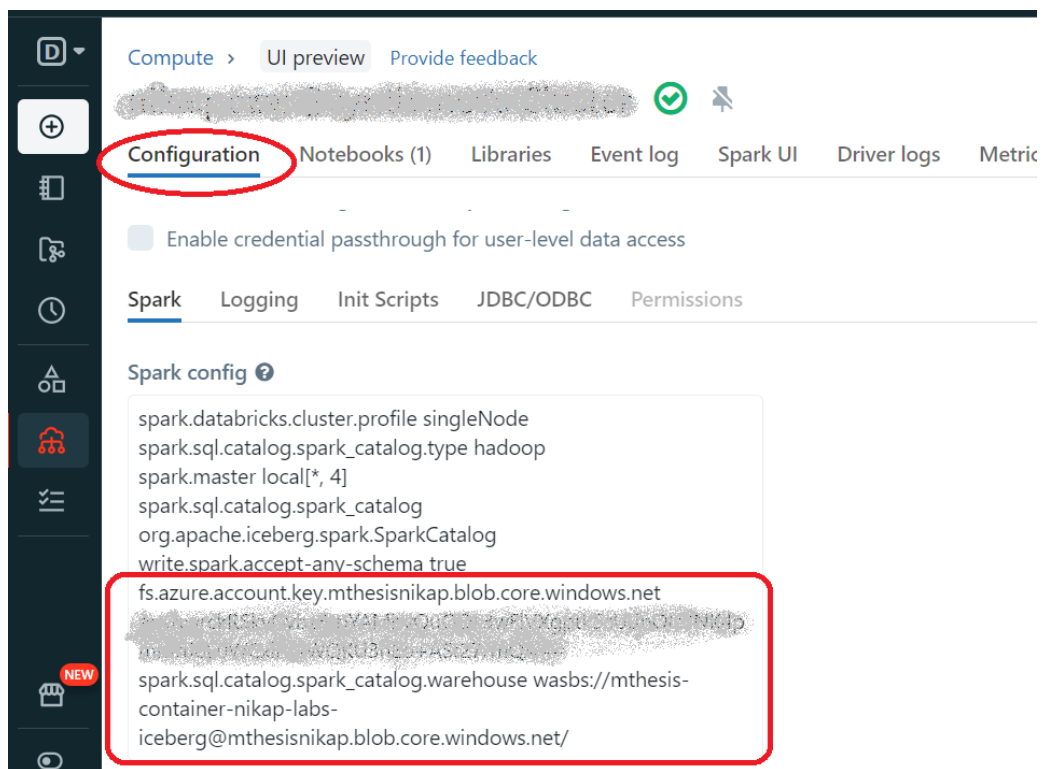
Kod 5.6 Zapisivanje podataka natrag u bazu

Prije pokretanja naredbe potrebno je povezati *Apache Iceberg* s *Azure Blob Storageom* preko *Azure Databricksa*. To se radi postavljanjem konfiguracijskih varijabli u postavkama *Azure Databricksa*. Unutar *Azure Databricks* sučelja potrebno je pronaći postavljanje konfiguracija: *Configuration* -> *Advanced Options* -> *Spark Config*. Zatim u njih treba dodati sljedeće linije:

```
spark.sql.catalog.spark_catalog org.apache.iceberg.spark.SparkCatalog  
spark.sql.catalog.spark_catalog.type hadoop  
spark.sql.catalog.spark_catalog.warehouse /<folder for iceberg data>/
```

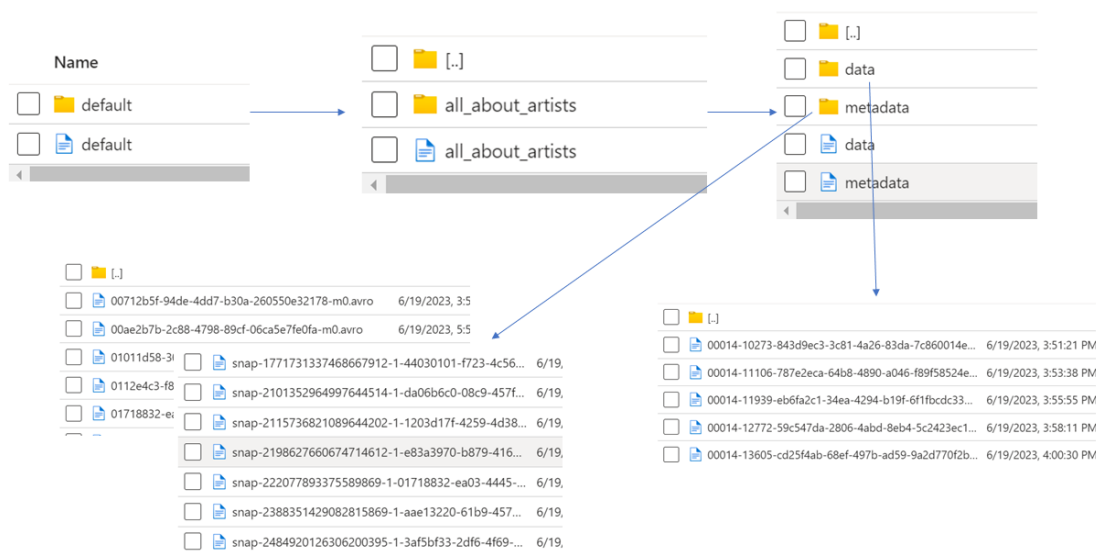
Kod 5.7 Konfiguracijske postavke za povezivanje *Apache Iceberga* s *Azure Blob Storageom*

Stvara se katalog koji je potreban za rad s *Iceberg* tablicama. `spark.sql.catalog.spark_catalog.warehouse` isječak konfiguracijskih postavki definira gdje će se spremati *Iceberg* tablice. Kako bi se mogao povezati s *Azure Blob Storageom* potrebno je predati ime *Azure* računa i pripadajući ključ kao i naziv kontejnera u koji će se spremati *parquet* podaci. Konačno, rezultat postavljanja konfiguracijskih varijabli izgleda kao na slici 5.17.



Slika 5.17 Postavljanje konfiguracijskih varijabli

Nakon što je konfiguracija ispravno postavljena i pisanje u *Azure Blob Storage* pokrenuto, podaci se spremaju u kontejner. Osim podataka, spremaju se i *metapodaci*. Izgled hijerarhije kontejnera prikazan je na slici 5.16.



Slika 5.18 Struktura kontejnera nastala spremanjem *Apache Iceberg* tablice

Iz strukture uočavamo direktorije u kojima se spremaju metapodaci i one u kojima se spremaju podaci. Također, vidljiva su i spremanja snimki (engl. *snapshot*) čiji nazivi započinju sa „*snap*“.

Iz stvorene tablice sada se mogu ponovno pročitati podaci. To je moguće učiniti korištenjem SQL-a recimo sljedećom naredbom:

```
spark.sql(" SELECT * FROM default.table_new3").show()
```

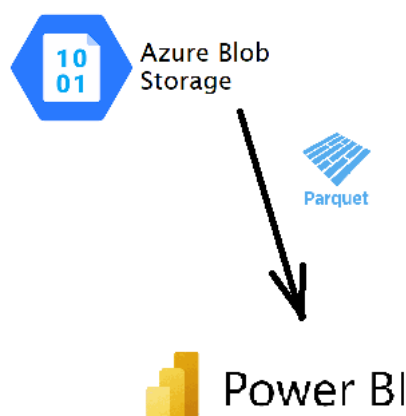
Kod 5.8 Primjer SQL naredbe nad *Apache Iceberg* tablicom

Podaci su konačno iz *json* formata transformirani u *parquet* oblik nastao iz stvaranja *Apache Iceberg* tablice. Primjer koji se proteže kroz ovaj rad bilo bi moguće obaviti i bez korištenja *Apache Iceberg* tablice tako što bi se korištenjem *PySpark* naredbi *json* podaci transformirali u *parquet* format i na sličan način spremili u *Azure Blob Storage*. Ipak, u primjer je odlučeno uvesti korištenje *Apache Iceberg* alata kako bi se skrenula pozornost na mogući budući rad s podacima kojeg *Iceberg* olakšava zbog svojih svojstava od kojih su neke mogućnost jednostavnih obavljanja SQL upita i mogućnost vraćanja u vremenu korištenjem snimki.

Ovim poglavljem završen je proces transformiranja podataka i ponovnog skladištenja u *Azure Blob Storage*. Sada ih je potrebno vizualizirati korištenjem *Microsoft Power BI* alata što će se detaljnije pojasniti u sljedećem poglavlju.

6. Vizualizacija velikih skupova podataka

Već je spomenuto da su neki od glavnih izazova u radu s velikim skupovima podataka njihovo dohvaćanje, skladištenje, transformiranje, a na kraju i vizualizacija. Vizualizacija podataka pruža korisnicima intuitivan način za interaktivno istraživanje i analizu podataka, omogućujući im da učinkovito identificiraju obrasce među podacima, donose zaključke o korelacijama i uzročno-posljedičnim vezama [58]. U nastavku poglavlja pojašnjava se pojam vizualizacije velikih skupova podataka, opisuju se neki od poznatih alata za vizualizaciju te se na kraju kroz primjer objašnjava vizualizacija korištenjem *Microsoft Power BI* (Slika 6.1).



Slika 6.1 Četvrti korak: Vizualizacija podataka dohvaćenih iz *Azure Blob Storagea* pomoću alata *Microsoft Power Bi*

6.1. Općenito o vizualizaciji velikih skupova podataka

Većina tradicionalnih sustava za vizualizaciju ne može podnijeti veličine današnjih skupova podataka, tj. ograničeni su na rad s malim veličina skupova podataka, s kojima se može lako rukovati. Zbog pojave velikih količina podataka, vizualizacija podataka postala je izazov koji uključuje nekoliko problema vezanih uz pohranu podataka, postavljanje upita, vizualnu reprezentaciju i interakciju [58].

Vizualizacija podataka igra važnu ulogu u donošenju odluka u različitim sektorima. Ljudski mozak lakše i učinkovitije pronalazi uzorke kada su podaci predstavljeni vizualno. Vizualizacija podataka postoji već stoljećima, a za čovjeka predstavlja jednostavan i brz način prenošenja poruka i predstavljanja složenih stvari. Danas se problem rada s velikim skupovima podataka pokušava riješiti stvaranjem odgovarajućih alata za vizualizaciju. Neki od važnih problema s kojima se ti alati susreću su pojava vizualnog šuma (engl. *visual noise*) i gubitak informacija (engl. *information loss*). Vizualni šum je pojava kada je većina objekata u skupu podataka relativna jedna prema drugoj pa ih postaje vrlo teško razdvojiti. S druge strane, gubitak informacija sam po sebi govori da dolazi do pojave nestajanja informacija. To se događa kada se zbog povećanja vremena odziva, smanjuje vidljivost skupa podataka [59].

6.2. Primjeri primjene vizualizacije i primjeri poznatih alata

Vizualizacija velikih skupova podataka je široko primjenjiva. Jedan primjer je primjena u fizici i astronomiji. Sateliti svakodnevno prikupljaju velike količine podataka. Vizualna analiza podataka može astronomima omogućiti da identificiraju neočekivane astronomske fenomene. Također, olakšavaju obavljanje nekih složenijih operacija koji nisu izvedivi koristeći tradicionalne pristupe analizi. Drugo područje primjene u znanosti je meteorologija. U toj domeni se velike količine podataka prikupljaju putem senzora i satelita. Osim toga, primjenjivo je i u bioinformatici. Na primjer, jako je izazovno analizirati velike količine bioloških podataka koje se proizvode iz DNK sekvenci. Vizualizacijom je lakše uočiti „zanimljiva“ područja gena. Tehnike vizualizacije se često koriste i u području poslovne inteligencije. Vizualna analitika omogućuje praćenje tržišta, prepoznavanje trendova i izvođenje predviđanja [58].

Kao rješenja za probleme vizualizacije velikih skupova podataka pojavili su se mnogi alati. U nastavku će se ukratko predstaviti neki od njih.

1) Tableau

Tableau je interaktivni alat za vizualizaciju podataka koji je usmjeren na poslovnu inteligenciju. Brz je i fleksibilan alat koji omogućuje stvaranje prilagođene vizualizacije. Iako je *Tableau* besplatno javno dostupan, svoju uslugu pruža *online* s 1 GB prostora za

pohranu. Za *offline*, tj. izvanmrežnu, verziju potrebno je kupiti licencu. Za obavljanje složenijih izračuna u *Tableau* alatu potrebno je poznavati kodiranje u jeziku R [59].

2) Plotly

Plotly, koji je još poznat kao *Plot.ly*, izvodi radnje analize i vizualizacije podataka. Izgrađen je pomoću *Pythona* i *Djanga*. Besplatan je za korisnike, ali s ograničenim značajkama. Za ostale značajke potrebno je učlaniti se. Može se koristiti izvan mreže koristeći *Python* bilježnice. Ipak, čak i uz učlanjenje, veličina datoteke za učitavanje seže samo do 5 MB [59].

3) Gephi

Gephi je alat koji se koristi za rukovanje velikim i složenim skupovima podataka, ali s naglaskom na analizu uz pomoć grafova. Za korištenje nije potrebno imati dobre vještine programiranja, ali je potrebno dobro poznavati grafikone. Njegova glavna prednost vizualizacije uz pomoć grafikona, njegova je i glavna mana, tj. ne može se koristiti za druge vrste vizualizacija [59].

4) Microsoft Excel

Microsoft Excel moćan je alat za rad s velikim skupovima podataka i obavljanje statističke analize, ali pored toga dobar je i za vizualizaciju. Ipak, *Excel* nije besplatan. Njegov API dostupan je samo uz pretplatu na *Office 365* [59].

5) Microsoft Power BI

Power BI (engl. *Business Intelligence - BI*) moćna je usluga za obavljanje poslovne analize koja se temelji na oblaku. Ima interaktivne i bogate alate za vizualizaciju. Značajka koja ju razlikuje od drugih alata je da se može prirodni jezik za obavljanje upita. Moguće je pokretati *R* skripte, ali poznavanje vještina programiranja nije potrebno. Desktop verzija softvera dostupna je besplatno, ali kako bi se pristupilo uslugama u oblaku potrebno je imati radni račun za prijavu [59].

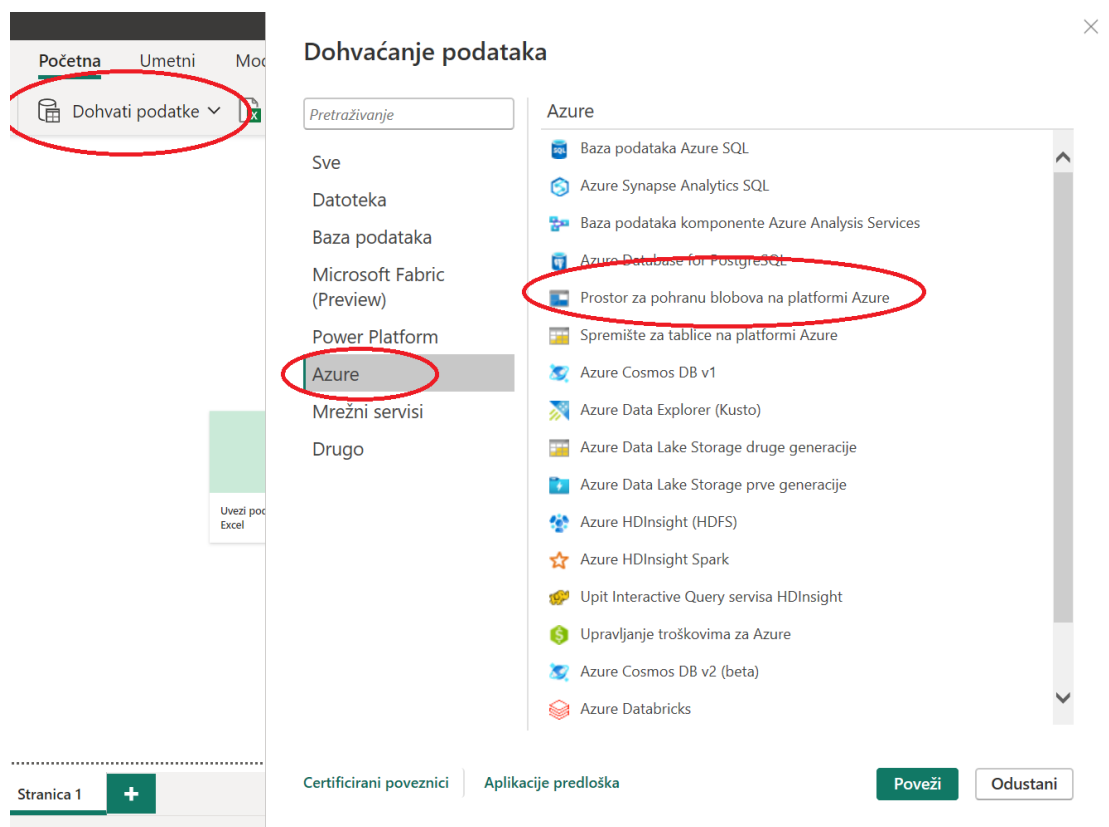
6.3. Microsoft Power BI

Microsoft Power BI alat je koji pretvara podatke iz različitih izvora podataka u interaktivne elemente za nadzor i kreiranje izvješća. *Power BI* nudi desktop verziju aplikacije koja je korištena u ovom radu. Osim toga nudi uslugu temeljenu na SaaS-u (engl. *Software as a Service*) i mobilnu aplikaciju [60].

6.3.1. Postavljanje i korištenje Microsoft Power BI-ja

Power BI verzija za desktop dostupna je na sljedećoj poveznici <https://powerbi.microsoft.com/en-us/downloads/>. Kada se *Power BI* instalira, pokrene se zaslon dobrodošlice koji se dalje koristi za pokretanje različitih opcija izrade vizualizacija na temelju podataka.

Power BI podržava veliki raspon izvora podataka. Slika 6.2 prikazuje kako se dolazi do odabira izvora podataka. Prvo je potrebo kliknuti na gumb *Dohvati podatke* (engl. *Get data*). Na slici je vidljiv široki raspon mogućih izvora podataka. Podaci koje želimo povući su spremljeni u *Azure Blob Storage* pa prikladno tome se odabire *Prostor za pohranu blobova na platformi Azure*. Nakon odabira te opcije, zatražit će se unos naziva računa na *Azureu* (Slika 6.3). Zatim je potrebno odabrati kontejner iz kojeg želimo pročitati podatke. To će biti kontejner *mthesis-container-nikap-labs-iceberg* u kojeg su spremljeni podaci u krajnjem *parquet* obliku. Na slici 6.4 crveno je zaokružen gumb *Transformacija podataka* kojeg treba odabrati kako bi se dalje prilagodili podaci.



Slika 6.2 Odabir izvora podataka u *Power BI*-ju

Prostor za pohranu blobova na platformi Azure

Naziv računa ili URL

mthesisnikap

U redu

Odustani

Slika 6.3 Unos Azure računa s kojim će se *Power BI* spojiti

Već se na slici 6.4 vidi u prvim nekoliko redaka podataka da su učitani neki podaci koji nisu u *parquet* formatu. Zbog toga je potrebna dodatna ručna transformacija. Kada se otvori pregled podataka, odabire se stupac *Extention* u kojem se mogu podesiti ekstenzije koje se žele prihvatiti kao unos (Slika 6.5). Nakon tog koraka je sve podešeno kako bi se podaci mogli unijeti u *Power BI*. Slika 6.6 prikazuje učitavanje svih podataka u *Power BI* te dočarava količinu redaka koja se učitava.

Navigator

Prikaz mogućnosti

mthesisnikap [3]

mthesis-container-nikap-labs

☒

mthesis-container-nikap-labs-iceberg

mthesis-container-nikap-labs-kafka

mthesis-container-nikap-labs-iceberg

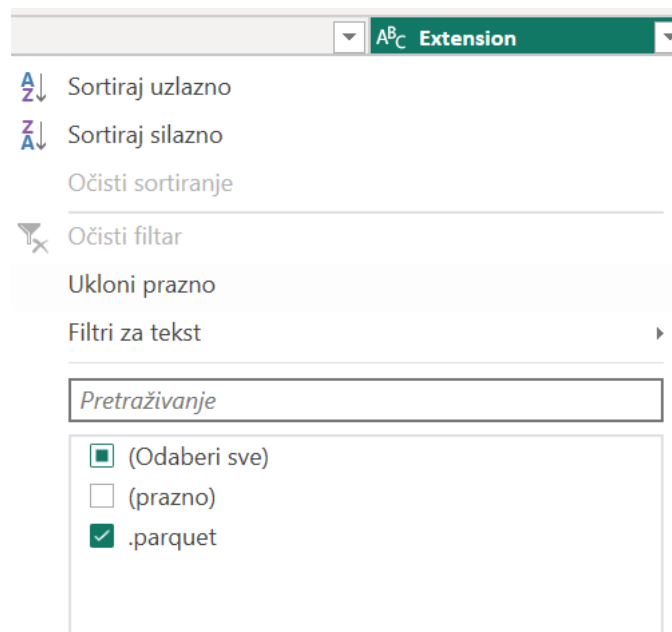
Content	Name	Extension
Binary	default	
Binary	default/all_about_artists	
Binary	default/all_about_artists/data	
Binary	default/all_about_artists/data/00014-10273-843d9ec3-3c81-4a26-83da-7	.parque
Binary	default/all_about_artists/data/00014-11106-787e2eca-64b8-4890-a046-f	.parque
Binary	default/all_about_artists/data/00014-11939-eb6fa2c1-34ea-4294-b19f-6f	.parque
Binary	default/all_about_artists/data/00014-12772-59c547da-2806-4abd-8eb4-5	.parque
Binary	default/all_about_artists/data/00014-13605-cd25f4ab-68ef-497b-ad59-9e	.parque
Binary	default/all_about_artists/data/00014-14438-25144ae3-edc2-445d-ad68-3	.parque
Binary	default/all_about_artists/data/00014-15108-ba9f4ca2-c476-4fd5-99d1-74	.parque
Binary	default/all_about_artists/data/00014-15941-05d5c21e-dfc8-4850-a460-2	.parque
Binary	default/all_about_artists/data/00014-16774-12011824-825f-4f1b-a635-d	.parque
Binary	default/all_about_artists/data/00014-17444-e2444154-3b4f-457b-b5f7-9	.parque
Binary	default/all_about_artists/data/00014-18114-6b36380a-2b54-4770-a5c4-e	.parque
Binary	default/all_about_artists/data/00014-18947-a774a248-e710-4b49-9c4c-4	.parque
Binary	default/all_about_artists/data/00014-19617-1b640263-9dd5-4f55-beb7-5	.parque
Binary	default/all_about_artists/data/00014-20287-12ac8567-7950-48e8-aedf-c	.parque
Binary	default/all_about_artists/data/00014-2055-0ab2c536-fbd7-4114-a695-fac	.parque
Binary	default/all_about_artists/data/00014-21120-3af3667f-97f6-4a6e-8f87-ea	.parque
Binary	default/all_about_artists/data/00014-21953-8f42f24a-19e6-4d1b-bed2-2	.parque
Binary	default/all_about_artists/data/00014-22786-c14c35d3-0353-4003-89c6-8	.parque
Binary	default/all_about_artists/data/00014-23456-16b11f7f-0a7a-45b6-a110-6	.parque

Učitaj

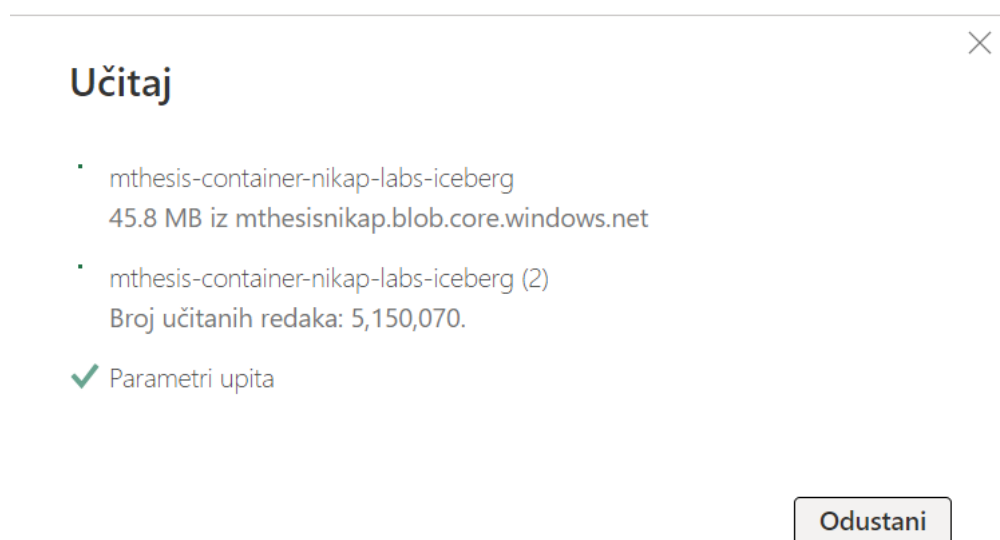
Transformacija podataka

Odustani

Slika 6.4 Odabir kontejnera unutar *Power BI*-ja



Slika 6.5 Odabir tipa podataka za učitati u *Power BI*

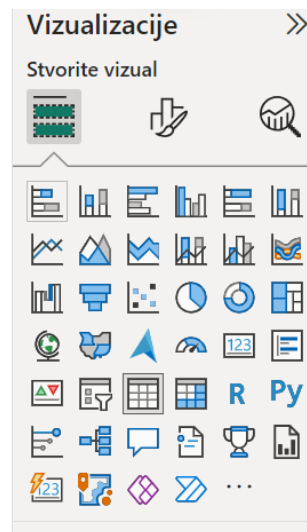


Slika 6.6 Učitavanje podataka u *Power BI*

Nakon što se doda izvor podataka i podaci se učitaju, oni se prikazuju na traci s desne strane. S lijeve strane ekrana nalaze se sljedeća tri gumba: *Prikaz izvješća* (engl. *Report*), *Prikaz podataka* (engl. *Data*) i *Prikaz modela* (engl. *Relationships*). *Prikaz izvješća* otvara ploču unutar koje se stvaraju grafikoni za vizualizaciju podataka, *Prikaz podataka* otvara sve podatke prema definiranom odnosu iz izvora podataka, a *Prikaz modela* otvara prikaz modela odnosa između izvora podataka.

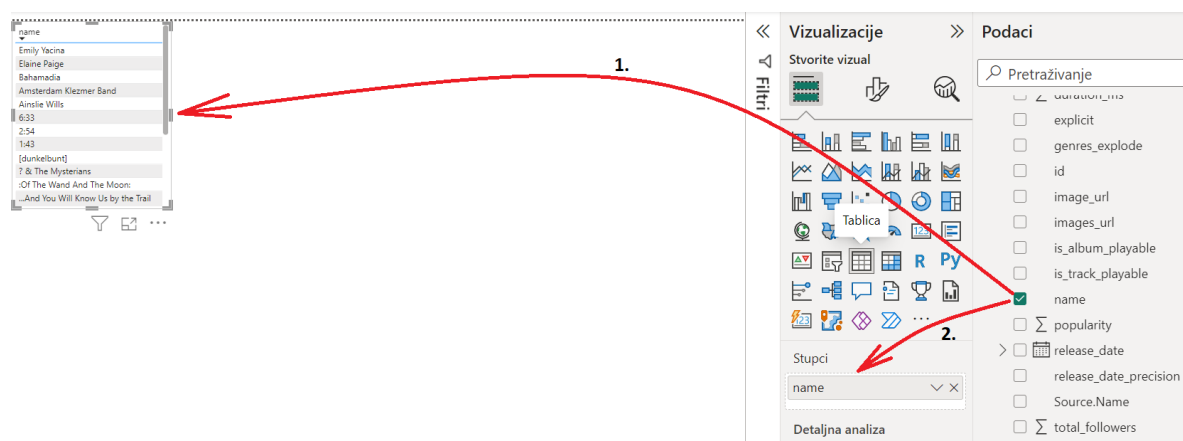
Unutar izvješća postoje razne vizualizacije, a ovisno o onome što se želi prikazati treba odabrati jednu od njih. Pretpostavljeno je da se uvijek otvori prikaz tablice, a pored

toga mogu se odabrati jednostavni stupčasti pa sve do kružnih grafikona, mapa, vodopada, mjeraca i sličnog. Slika 6.6 prikazuje ikone za odabir željene vizualizacije.



Slika 6.7 Popis vizualizacija u *Power BI*-ju

U *Power BI*-ju moguće je stvoriti vizualizaciju na dva načina. Prvi način je direktno dodavanje stupca s desne strane trake direktno na platno. Rečeno je da je pretpostavljen izgled vizualizacije tipa tablice. Drugi način je da se s desne trake stupac po želji povuče u polja unutar odabrane vizualizacije. Na slici 6.8 prvi način označen je brojem jedan, a drugi brojem dva. Unutar vizualizacija moguće je kombinirati nekoliko različitih elemenata, recimo može se napraviti graf s imenima izvođača i njihovom popularnošću, tj. sa stupcima *name* i *popularity*. Vizualizacije su potpuno interaktivne. Tako da odabirom imena izvođača se direktno pronalaze njegovi albumi i pjesme. Jednako vrijedi i obrnuto.

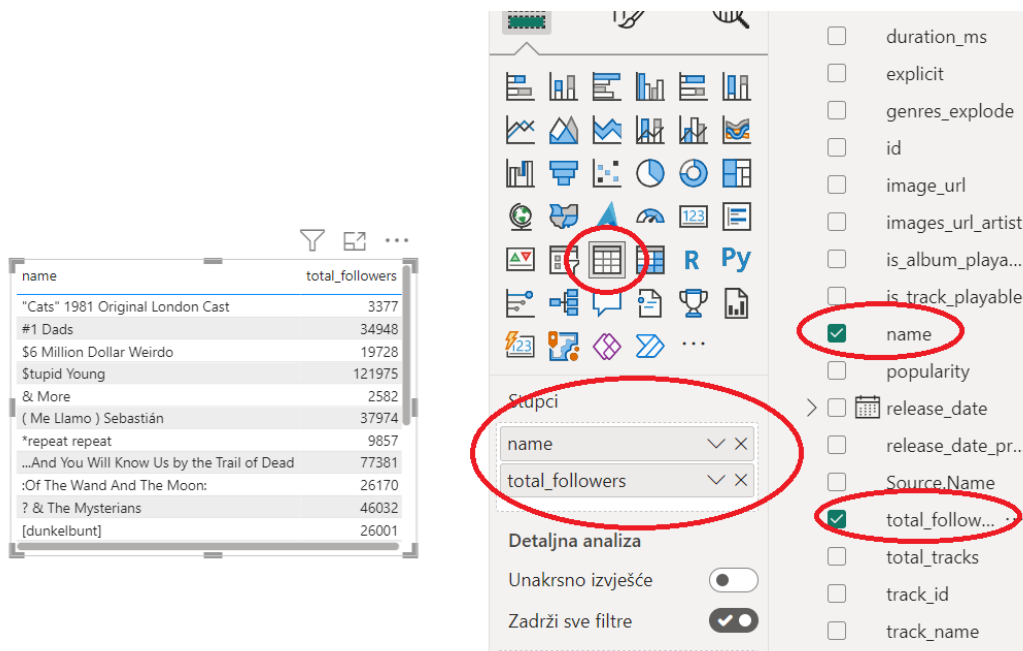


Slika 6.8 Primjeri stvaranja vizualizacija u *Power BI*-ju

Na slici 6.8 se vidi isječak nekih od stupaca koji su učitani u *Power BI* alat. Sada je potrebno promotriti koje podatke se želi vizualizirati i međusobno iskombinirati. Podaci će

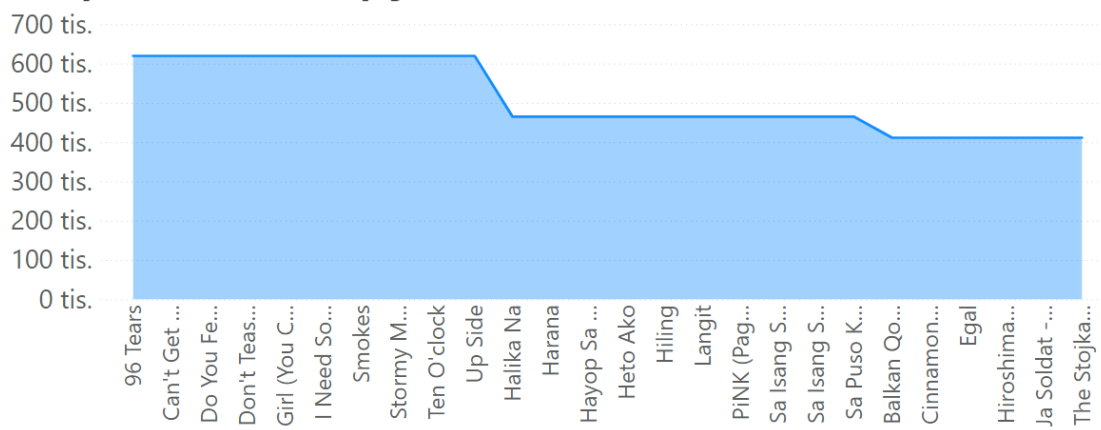
se vizualno podijeliti u tri kategorije: podaci o glazbenicima, podaci o pjesmama i podaci o albumima. Unatoč tome što će se ti podaci vizualno odvojiti, svi će biti smješteni na istom platnu te će biti moguće pratiti njihovu međusobnu komunikaciju.

Za početak sagledajmo izradu jednostavne tablice koja će sadržavati imena izvođača i njihov broj pratitelja na *Spotify* aplikaciji. U desnoj traci pronalazimo ta dva stupca: *name* i *total_followers* te među vizualizacijama odabiremo izgled tablice. Na slici 6.9 prikazan je način izrade tablice i prikazan je krajnji rezultat. Na sličan način moguće je izraditi i druge tablice. Na primjer, jedna zanimljiva tablica bi mogla sadržavati ime albuma, godinu i mjesec izdavanja te broj pjesama koji se nalaze na albumu. Osim tablica zanimljivi mogu biti i grafovi. Tako primjerice se mogu u odnos staviti imena pjesama i njihova popularnost pa bi izgled jednog takvog grafa mogao izgledati kao na slici 6.10.



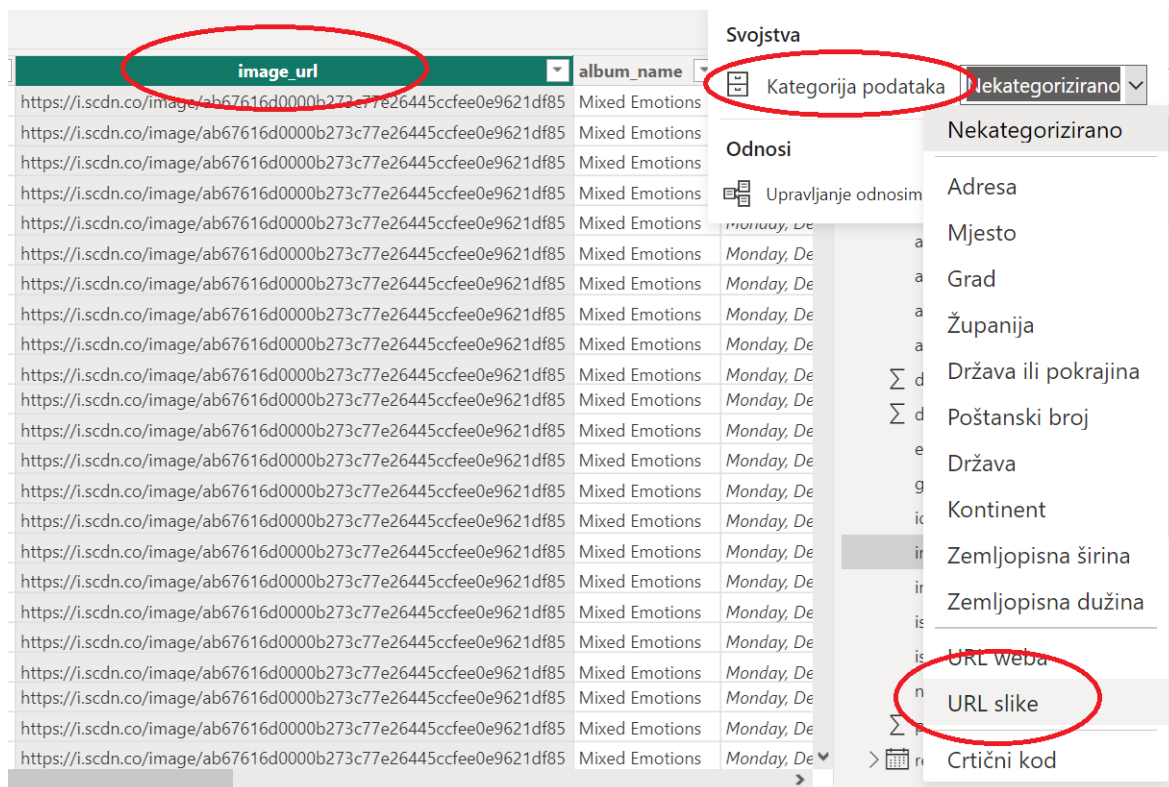
Slika 6.9 Izrada tablice s dva stupca u Power BI-ju

Popularnost pjesama



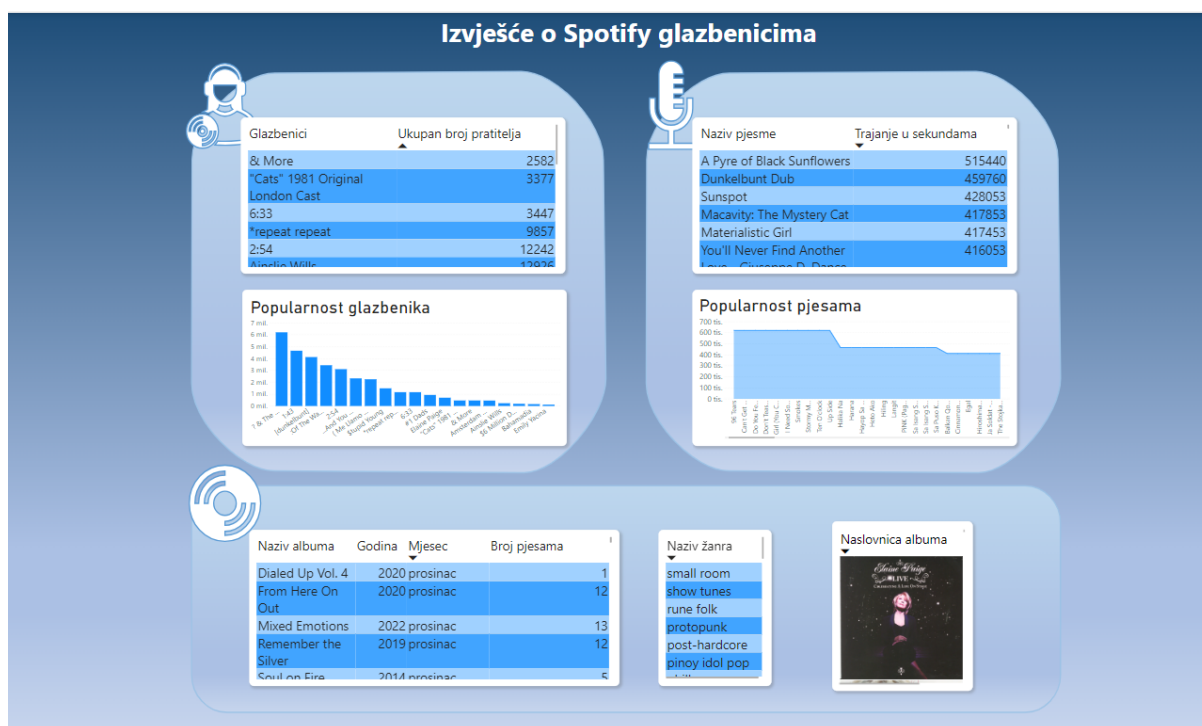
Slika 6.10 Izrada grafa popularnosti pjesama u *Power BI*-ju

Osim brojčanih i tekstualnih podataka, sa *Spotifyja* su se prenijele i slike u obliku URL-a. *Power BI* nudi opciju generiranja slike iz URL-a. U tablici imamo stupce koji sadrže slike albuma. Odabere se vizualizacija u obliku tablice. U nju se nadoda stupac koji sadrži URL-ove slika. Odabirom opcije *Prikaz podataka* (engl. *data*) pronalazi se navedeni stupac i pod izbornikom *Kategorija podataka* (engl. *Data category*) odabere se opcija *image url*. Postupak je prikazan na slici 6.11.



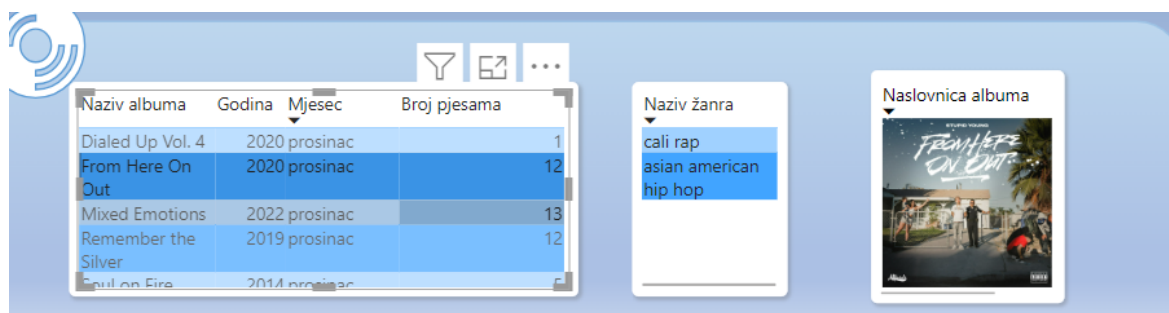
Slika 6.11 Pretvaranje URL-a u sliku u Power BI-ju

Nakon odabira svih željenih vizualizacija podataka. Krajnji rezultati prikazani su u nekoliko sljedećih slika. Slika 6.12 prikazuje cjelokupni izgled izvješća.



Slika 6.12 Krajnji izgled izvješća

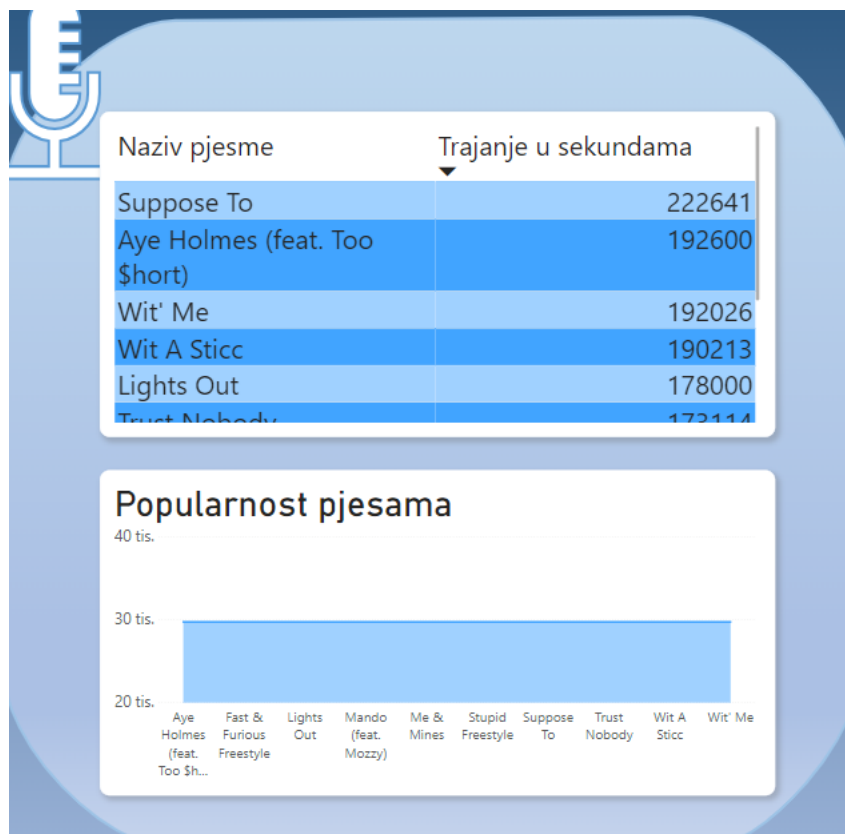
Recimo da se među albumima odabere album imena *From Here On Out*. Sve druge vizualizacije će reagirati na taj odabir i filtrirati podatke vezane isključivo uz taj album. Slika 6.13 prikazuje dio izvješća vezan uz albume kada se odabere album *From Here On Out*. Nakon toga, na slici 6.14 vidi se izgled odjeljka povezanog uz glazbenika, a na slici 6.15 popis pjesama koje se nalaze na odabranom albumu i informacije o njima.



Slika 6.13 Izgled izvješća nakon odabira albuma *From Here On Out* – odjeljak albuma



Slika 6.14 Izgled izvješća nakon odabira albuma *From Here On Out* – odjeljak glazbenika



Slika 6.15 Izgled izvješća nakon odabira albuma *From Here On Out* - odjeljak pjesama

Na isti način moguće je pronaći informacije o drugim glazbenicima, albumima ili pjesmama. U ovom primjeru pregledavaju se podaci koji su se povukli sa *Spotify API*-ja, ali slični načini vizualizacije i primjene međusobne interakcije jako su korisni u nadzoru rada poslovanja kao i provođenju raznih statističkih zaključaka iz prikupljanja velikih skupova podataka, a *Microsoft Power BI* samo je jedan od alata za vizualizaciju koji se može koristiti.

Zaključak

Iznimno brzim napretkom tehnologije kroz posljednje stoljeće, došlo je do pojave velikih količina podataka s kojima se trebalo naučiti raditi. Problemi rada s velikim skupovima podataka sežu od njihovog nastajanja, prikupljanja, transformiranja, skladištenja pa sve do vizualizacije i konačne primjene. Stručnjaci se još uvijek bave razvijanjem i prilagođavanjem tehnologija kako bi se olakšao tok velikih skupova podataka.

Rad upoznaje čitatelja s problematikom rada s velikim skupovima podataka. Pored toga, rad progovara o novim tehnologijama, a sve skupa potkrepljuje primjerom toka podataka. Podaci se prikupljaju iz *Spotify API*-ja kao odabranog izvora podataka pomoću *Apache Kafka* tehnologije. *Apache Kafka* svojom nadogradnjom na model *objavi-pretplati*, rješava problem pojave velikog broja različitih pretplatnika na podatke, kao i različitih tipova podataka koji se pojavljuju u toku podataka. Velike skupove podataka ne može se skladištiti na tradicionalnim lokacijama pa se za rad s njima uglavnom koriste oblaci. Jedan od dostupnih oblaka je *Microsoft Azure* na kojeg se skladište podaci koje je *Apache Kafka* povukao iz *Spotify API*-ja. Podaci nisu nužno uvijek strukturirani i ne pojavljuju se uvijek u formatima u kojima ih njihovi potrošači žele. Zato je potrebno obaviti transformaciju podataka. Danas postoje mnogi alati koji se time bave, a u ovom radu detaljnije je pojašnjen rad *Apache Sparka* u kombinaciji s *Apache Icebergom*. Podaci se konačno prilagode u *parquet* format i spremaju natrag na oblak kako bi ih pročitao *Microsoft Power BI* alat za vizualizaciju. Cijeli tok podataka koji prati ovaj rad prikazan je u kratkom videu na poveznici <https://youtu.be/e5Qhq3V11Yg>.

U radu su naglašeni problemi rada s velikim skupovima podataka, ali i istaknuta mnoga tehnološka rješenja današnjice koja taj rad olakšavaju. Alati koji se spominju kroz rad imaju svoje prednosti i mane, ali svakim njihovim poboljšanjem se tok podataka ubrzava i podaci postaju dostupniji. Zaključujemo da je za rad s velikim skupovima podataka potreban razvoj u tehnologiji, a upravo poboljšanje rada s velikim skupovima podataka dovodi do novih tehnoloških i znanstvenih razvitaka.

Literatura

- [1] The Conversation, Academic rigour, journalistic flair. Poveznica: <https://theconversation.com/the-worlds-data-explained-how-much-were-producing-and-where-its-all-stored-159964>; pristupljeno 26.5.2023.
- [2] Ward, J.S., Barker A., Undefined By Data: A Survey of Big Data Definitions, Poveznica: <https://arxiv.org/pdf/1309.5821.pdf>; pristupljeno 26.5.2023.
- [3] Bigdataldn, Big Data: The 3 Vs Explained. Poveznica: <https://bigdataldn.com/news/big-data-the-3-vs-explained/>; pristupljeno 26.5.2023.
- [4] National Research Council. 2013. *Frontiers in Massive Data Analysis*. Washington, DC: The National Academies Press. <https://doi.org/10.17226/18374>.
- [5] Top Big Data Technologies You Must Know. Poveznica: <https://www.interviewbit.com/blog/big-data-technologies/>; pristupljeno 26.5.2023.
- [6] Spotify for Developers, Documentation. Poveznica: <https://developer.spotify.com/documentation/web-api>; pristupljeno 31.5.2023.
- [7] APACHE KAFKA, Get Started. Poveznica: <https://kafka.apache.org/>; pristupljeno 31.5.2023.
- [8] CONFLUENT Developer, Confluent Platform Docs. Poveznica: <https://kafka.apache.org/>; pristupljeno 31.5.2023.
- [9] Azure, Azure Blob Storage. Poveznica: <https://kafka.apache.org/>; pristupljeno 31.5.2023.
- [10] Talend, A Qlik Company, What is a Data Source? Poveznica: <https://www.talend.com/resources/data-source/#:~:text=A%20data%20source%20is%20the,process%20accesses%20and%20Utilizes%20it.>; pristupljeno 2.6.2023.
- [11] ALPHA|SERVE, What is a Data Source: Definitions, Types, Examples. Poveznica: <https://www.alphaservesp.com/blog/what-is-a-data-source-definitions-types-examples>; pristupljeno 2.6.2023.
- [12] Tech Target Network, application programming interface (API). Poveznica: <https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API>; pristupljeno 2.6.2023.
- [13] Newline, Okay, but what is REST API? Poveznica: <https://www.newline.co/courses/build-a-spotify-connected-app/okay-but-what-is-a-rest-api>; pristupljeno 4.6.2023.
- [14] Spotify R&D, Understanding the Spotify Web API. Poveznica: <https://engineering.atspotify.com/2015/03/understanding-spotify-web-api/>; pristupljeno 4.6.2023.
- [15] Spotify for Developers, Authorization. Poveznica: <https://developer.spotify.com/documentation/web-api/concepts/authorization>; pristupljeno 4.6.2023.

- [16] Towards Data Science, Extracting Song Data From the Spotify API Using Python. Poveznica: <https://towardsdatascience.com/extracting-song-data-from-the-spotify-api-using-python-b1e79388d50>; pristupljeno 4.6.2023.
- [17] Postman, What is Postman? Poveznica: [https://www.postman.com/#:~:text=What%20is%20Postman%3F,can%20create%20better%20APIs%E2%80%94faster.](https://www.postman.com/#:~:text=What%20is%20Postman%3F,can%20create%20better%20APIs%E2%80%94faster.;); pristupljeno 4.6.2023.
- [18] Techjury, How Much Data Is Created Every Day in 2023? Poveznica: <https://techjury.net/blog/how-much-data-is-created-every-day/>; pristupljeno 7.6.2023.
- [19] Tech Target Network, How big data collection works: Process, challenges, techniques. Poveznica: [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](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;); pristupljeno 7.6.2023.
- [20] K21Academy, Structured, Semi Structured and Unstructured Data. Poveznica: <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>; pristupljeno 7.6.2023.
- [21] Snowflake, WHAT IS JSON? Poveznica: <https://www.snowflake.com/guides/what-is-json>; pristupljeno 7.6.2023.
- [22] Google Cloud, Announcing preview of BigQuery's native support for semi-structured data. Poveznica: <https://cloud.google.com/blog/products/data-analytics/bigquery-now-natively-supports-semi-structured-data>; pristupljeno 7.6.2023.
- [23] SOURCEFORGE, Best Data Ingestion Tools. Poveznica: <https://sourceforge.net/software/data-ingestion/>; pristupljeno 7.6.2023.
- [24] Hevo, Top Data Ingestion Tool sin 2023. Poveznica: <https://hevodata.com/learn/data-ingestion-tools/>; pristupljeno 8.6.2023.
- [25] DATAVID, 7 data ingestion tools you should consider [Updated list]. Poveznica: <https://datavid.com/blog/data-ingestion-tools>; pristupljeno 8.6.2023.
- [26] Integrate.io, Top 11 Data Ingestion Tools for 2023. Poveznica: <https://www.integrate.io/blog/top-data-ingestion-tools/>; pristupljeno 8.6.2023.
- [27] O'Reilly Media, Inc., Shapira G., Palino T., Sivaram R., Petty K., Kafka The Definitive Guide: Real-Time Data and Stream Processing at Scale, 2. izdanje, 1005 Gravenstein Highway North, Sebastopol, 2022.
- [28] H. Wu, Z. Shang and K. Wolter, *Performance Prediction for the Apache Kafka Messaging System*, 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, (2019), str. 154-161
- [29] K.M.M. Thein, *Apache Kafka: Next Generation Distributed Messaging System*, International Journal of Scientific Engineering and Technology Research, Volume.03, Issue No.46, (2014), str. 9478-9483

- [30] B. R. Hiran, C. Viresh M. and K. Abhijeet C., *A Study of Apache Kafka in Big Data Stream Processing*, 2018 International Conference on Information , Communication, Engineering and Technology (ICICET), Pune, India, (2018), str. 1-3
- [31] Dattell, Comparing Confluent Kafka and Apache Kafka. Poveznica: <https://dattell.com/data-architecture-blog/comparing-confluent-kafka-and-apache-kafka/>; pristupljeno 11.6.2023.
- [32] Alibaba Cloud, What Is The Difference Between Apache Kafka And Confluent Kafka. Poveznica: [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](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;); pristupljeno 11.6.2023.
- [33] Estuary, Confluent Kafka vs Apache Kafka vs Estuary: 2023 Comparison. Poveznica: <https://estuary.dev/confluent-kafka-vs-apache-kafka/#what-is-apache-kafka>; pristupljeno: 11.6.2023.
- [34] Confluent Developer, What is Confluent Platform? Poveznica: <https://docs.confluent.io/platform/current/platform.html>; pristupljeno 12.6.2023.
- [35] Confluent Developer, Confluent Schema Registry. Poveznica: <https://developer.confluent.io/learn-kafka/apache-kafka/schema-registry/>; pristupljeno: 12.6.2023.
- [36] Confluent Developer, REST Proxy. Poveznica: <https://docs.confluent.io/platform/current/kafka-rest/index.html>; pristupljeno 12.6.2023.
- [37] Conduktor, Complete Kafka Producer with Java. Poveznica: <https://www.conduktor.io/kafka/complete-kafka-producer-with-java/>; pristupljeno 13.6.2023.
- [38] Confluent Developer, Kafka Connect. Poveznica: <https://docs.confluent.io/platform/current/connect/index.html>; pristupljeno 13.6.2023.
- [39] Bocasay, How to Store Big Data. Poveznica: <https://www.bocasay.com/how-to-store-big-data/>; pristupljeno 15.6.2023.
- [40] M. Strohhach, J. Daubert, H. Ravkin, M. Lischka, *Big Data Storage*, New Horizons for Data-Driven Economy, Springer Open, Galway Ireland, (2016), str. 119-142
- [41] SmartData Collective, 5 Big Data Storage Solutions. Poveznica: <https://www.smartdatacollective.com/5-big-data-storage-solutions/>; pristupljeno 15.6.2023.
- [42] Talend, Data Transformation Defined. Poveznica: <https://www.talend.com/resources/data-transformation-defined/>; pristupljeno 16.6.2023.
- [43] TIBCO, What is Data Transformation? Poveznica: <https://www.tibco.com/reference-center/what-is-data-transformation>; pristupljeno 16.6.2023.
- [44] Techtarget, What is data transformation? Poveznica: <https://www.techtarget.com/searchdatamanagement/definition/data-transformation>; pristupljeno 16.6.2023.

- [45] Microsoft, What is Azure Databricks? Poveznica: <https://learn.microsoft.com/en-us/azure/databricks/introduction/>; pristupljeno 16.6.2023.
- [46] Element 61, Microsoft Azure Databricks. Poveznica: <https://www.element61.be/en/competence/microsoft-azure-databricks>; pristupljeno 16.6.2023.
- [47] IntelliPaat, What is Azure Databricks? Poveznica: <https://intellipaat.com/blog/what-is-azure-databricks/?US>; pristupljeno 16.6.2023.
- [48] Salloum, S., Dautov, R., Chen, X. et al. *Big data analytics on Apache Spark*. Int J Data Sci Anal 1, (2016), str. 145–164
- [49] InfoWorld, What is Apache Spark? The big data platform that crushed Hadoop. Poveznica: <https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html>; pristupljeno 18.6.2023.
- [50] Apache Spark, Cluster Mode Overview. Poveznica: <https://spark.apache.org/docs/latest/cluster-overview.html>; pristupljeno 18.6.2023.
- [51] Apache Spark, RDD Programming Guide. Poveznica: <https://spark.apache.org/docs/latest/rdd-programming-guide.html>; pristupljeno 18.6.2023.
- [52] Algoscale, Apache Spark – RDD vs Dataframe. Poveznica: <https://algoscale.com/blog/apache-spark-rdd-vs-dataframe/>; pristupljeno 18.6.2023.
- [53] Towards Data Science, 3 Reasons Why Spark's Lazy Evaluation is Useful. Poveznica: <https://towardsdatascience.com/3-reasons-why-sparks-lazy-evaluation-is-useful-ed06e27360c4>; pristupljeno 18.6.2023.
- [54] Knoldus, Deep Dive into Apache Spark Transformations and Actions. Poveznica: <https://blog.knoldus.com/deep-dive-into-apache-spark-transformations-and-action/>; pristupljeno 18.6.2023.
- [55] SparkByExamples, Spark DataFrame with Column. Poveznica: <https://sparkbyexamples.com/spark/spark-dataframe-with-column/>; pristupljeno 19.6.2023.
- [56] AWS, What Is Apache Iceberg? Poveznica: <https://aws.amazon.com/what-is/apache-iceberg/>; pristupljeno 19.6.2023.
- [57] Medium, A Short Introduction to Apache Iceberg. Poveznica: <https://medium.com/expedia-group-tech/a-short-introduction-to-apache-iceberg-d34f628b6799>; pristupljeno 19.6.2023.
- [58] Bikakis N., *Big Data Visualization Tools*, Encyclopedia of Big Data Technologies, Springer, (2018)
- [59] S. M. Ali, N. Gupta, G. K. Nayak and R. K. Lenka, *Big data visualization: Tools and challenges*, 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, (2016), str. 656-660
- [60] Tutorialspoint, Power BI – Quick Guide. Poveznica: https://www.tutorialspoint.com/power_bi/power_bi_quick_guide.htm; pristupljeno 20.6.2023.

Sažetak

Kao jedna od modernih problematika je rad s velikim skupovima podataka. Ovaj rad upoznaje čitatelja s tom problematikom te iznosi tehnološka rješenja koja su se razvila za rad s tokom velikih skupova podataka. Detaljnije se opisuje primjer toka podataka od *Spotify API*-ja čiji se podaci povlače pomoću *Apache Kafka* tehnologije, preko *Azure* oblaka i transformacija pomoću *Apache Sparka* i *Apache Iceberga* do *Microsoft Power BI* alata za vizualizaciju. Koristeći primjer, objašnjavaju se prednosti i mane modernih tehnologija i naglašava važnost rada s velikim skupovima podataka.

Ključne riječi: veliki skupovi podataka, dohvaćanje podataka, transformacija podataka, vizualizacija podataka, Spotify API, Apache Kafka, Azure Blob Storage, Apache Spark, Apache Iceberg, Microsoft Power BI

Summary

One of the modern issues is working with large data sets. This paper introduces the reader with this problem and presents technological solutions that have been developed for working with the flow of large data sets. It presents a detailed example of a data flow from the *Spotify API* whose data is pulled using *Apache Kafka* technology, through the *Azure* cloud and transformed using *Apache Spark* and *Apache Iceberg* to the *Microsoft Power BI* visualization tool. Using an example, the advantages and disadvantages of modern technologies are explained and the importance of working with large data sets is emphasized.

Keywords: big data sets, data retrieval, data transformation, data visualization, Spotify API, Apache Kafka, Azure Blob Storage, Apache Spark, Apache Iceberg, Microsoft Power BI