

MusicBrainz Database



The majority of the data in the **MusicBrainz Database** is released into the **Public Domain** and can be downloaded and used **for free**.

Data overview

Core data

Artists

Name, sort name, IPI, aliases, type, begin and end dates, disambiguation comment, MBID

Release Groups

Title, artist credit, type, disambiguation comment, MBID

Releases

Title, artist credit, type, status, language, date, country, label, catalog number, barcode, medium(s), disc ID(s), ASIN, disambiguation comment, MBID

Mediums

Format, list of tracks (title, artist credit, duration)

Recordings

Title, artist credit, duration, ISRC, PUIDs, relationships, disambiguation comment, MBID

Works

Title, ISWC, relationships, disambiguation comment, MBID

Labels

Name, sort name, aliases, country, type, code, begin and end dates, disambiguation comment, MBID

Relationships & URLs

Relationships are a way to link the above entities together and allow MusicBrainz to capture most of the data contained in the liner notes of a CD.

Si scarica il dump dei dati da
MusicBrainz tramite wget,
vengono poi processati
togliendo \N (i valori nulli) e
salvati come tsv utilizzando
multithreading per velocizzare
i processi

\$f = (tutti i files da trasferire)
hadoop fs -copyFromLocal \$f.tsv /mbdump/

A questo punto tutti i file si trovano su HDFS, pronti per essere elaborati con uno dei programmi di HADOOP

- I dati inizialmente vengono elaborati con Hive, usando sempre il motore Tez
- In seguito si è preferito usare Pig perché più performante per processare flow di dati
- Si poteva proseguire usando HCatalog per connettere Hive e Pig ma si è preferito a quel punto di utilizzare solo Pig

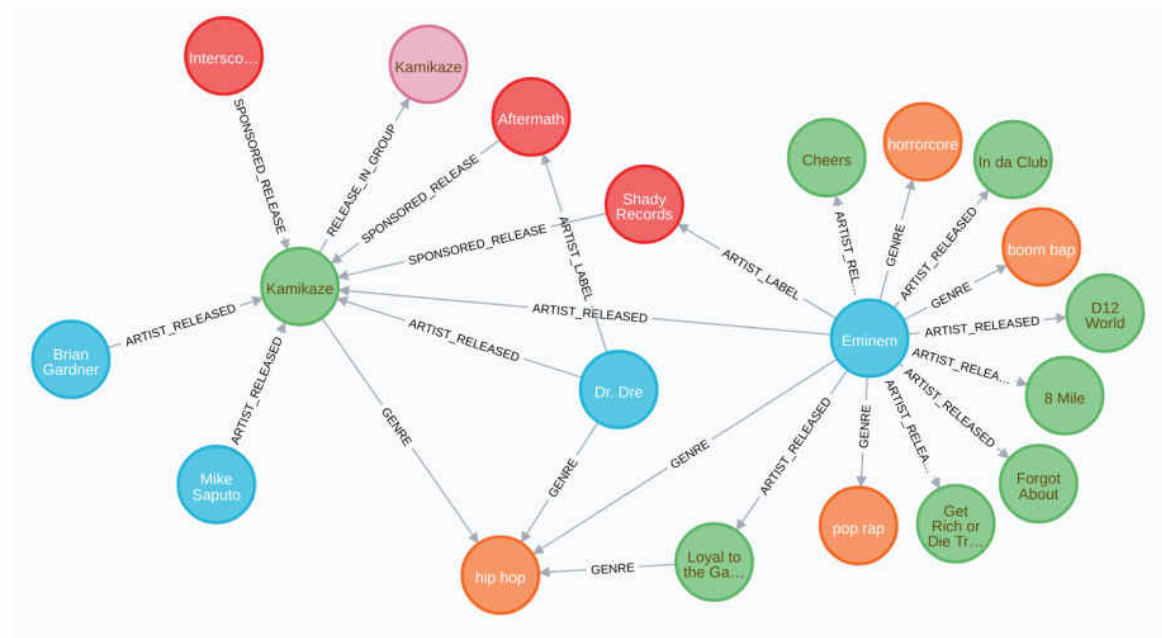
Pig viene utilizzato al posto di Hive per la sua velocità (tempo di esecuzione: 15-20 minuti)

I dati elaborati vengono trasferiti sul FS

Con neo4j_import in bash si caricano i dati in neo4j (tempo di esecuzione: circa 6 minuti)

Sui dati i comandi più utilizzati sono:

- Filter (Foreach ... Generate ...)
- Join
- Distinct

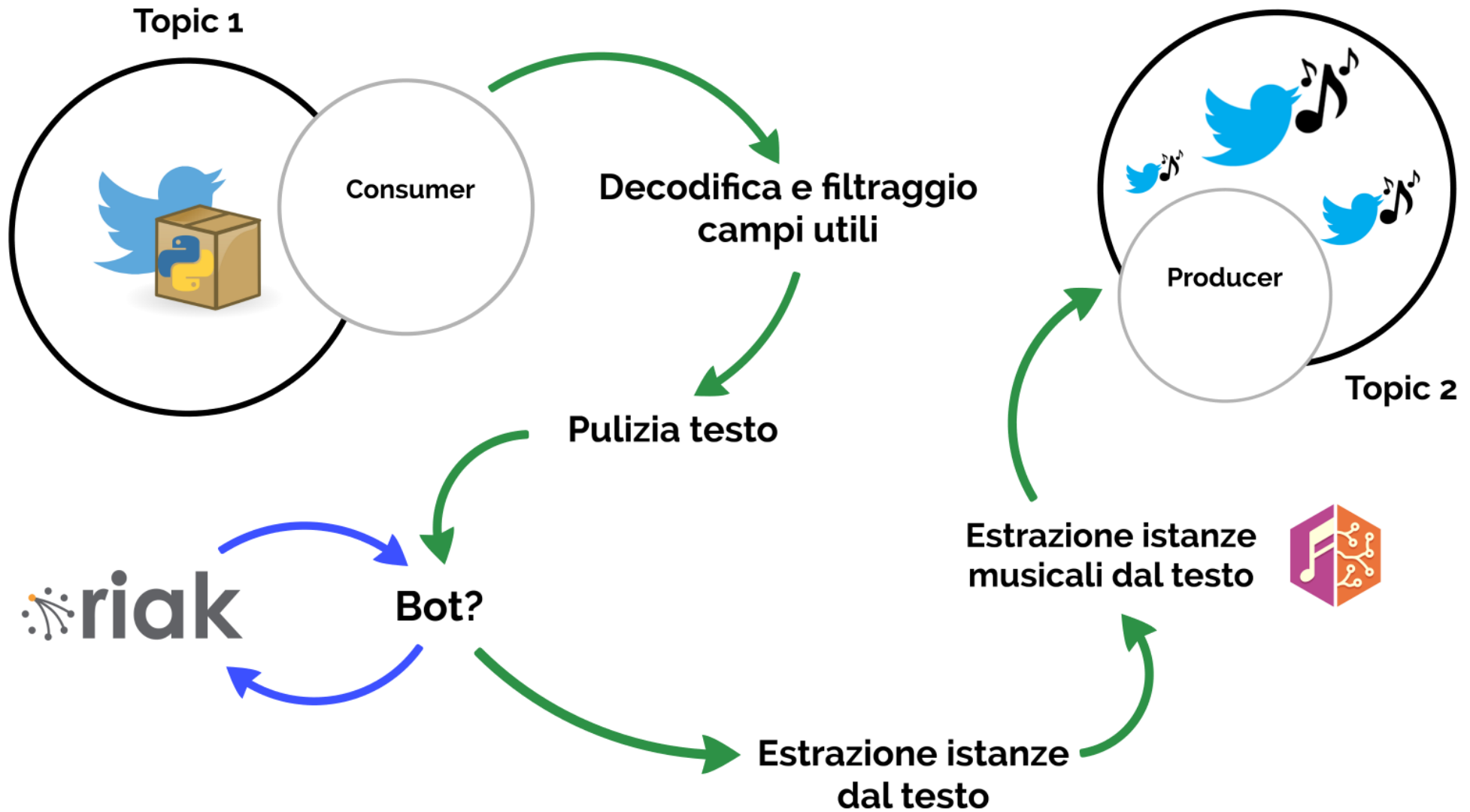


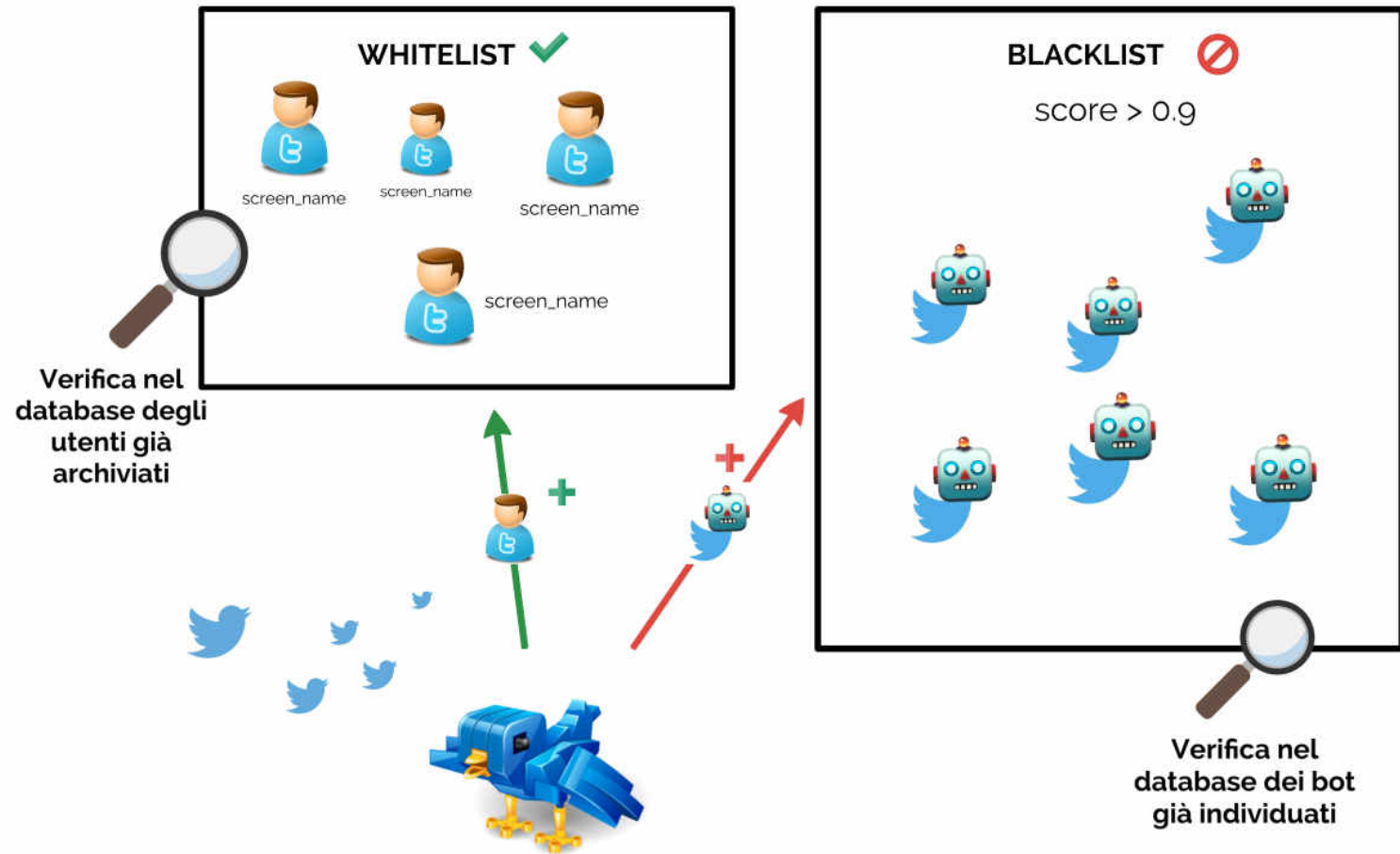
Attraverso l'API **Tweepy**,
viene estratto in tempo
reale un flusso di Tweets
scritti in lingua italiana e
che presentano hashtag
relativi a generi musicali



Ogni risultato di questa estrazione all'interno
dell'ambiente Python viene immediatamente
inviato come messaggio ad una Topic di Kafka,
attraverso un processo Producer









<@ScreenName; p(is_a_BOT?)>

Riak registra lo score dato da Botometer a ciascun utente, ovvero la probabilità di avere a che fare con un BOT.

Lo score viene verificato con una query



Se viene restituito un valore, si verifica che non superi la soglia imposta.



Altrimenti, viene effettuata una richiesta all'API di Botometer, registrato il valore e verificato che non superi la soglia.

Questo sistema adatta automaticamente il contenuto delle liste al variare della soglia.

**Instance
Recognition**



**Instance
Matching**

Performance

Il testo è ripulito da caratteri speciali, link, hashtag e abbreviazioni

TreeTagger individua tutti i sostantivi con la lettera maiuscola: son nomi propri?

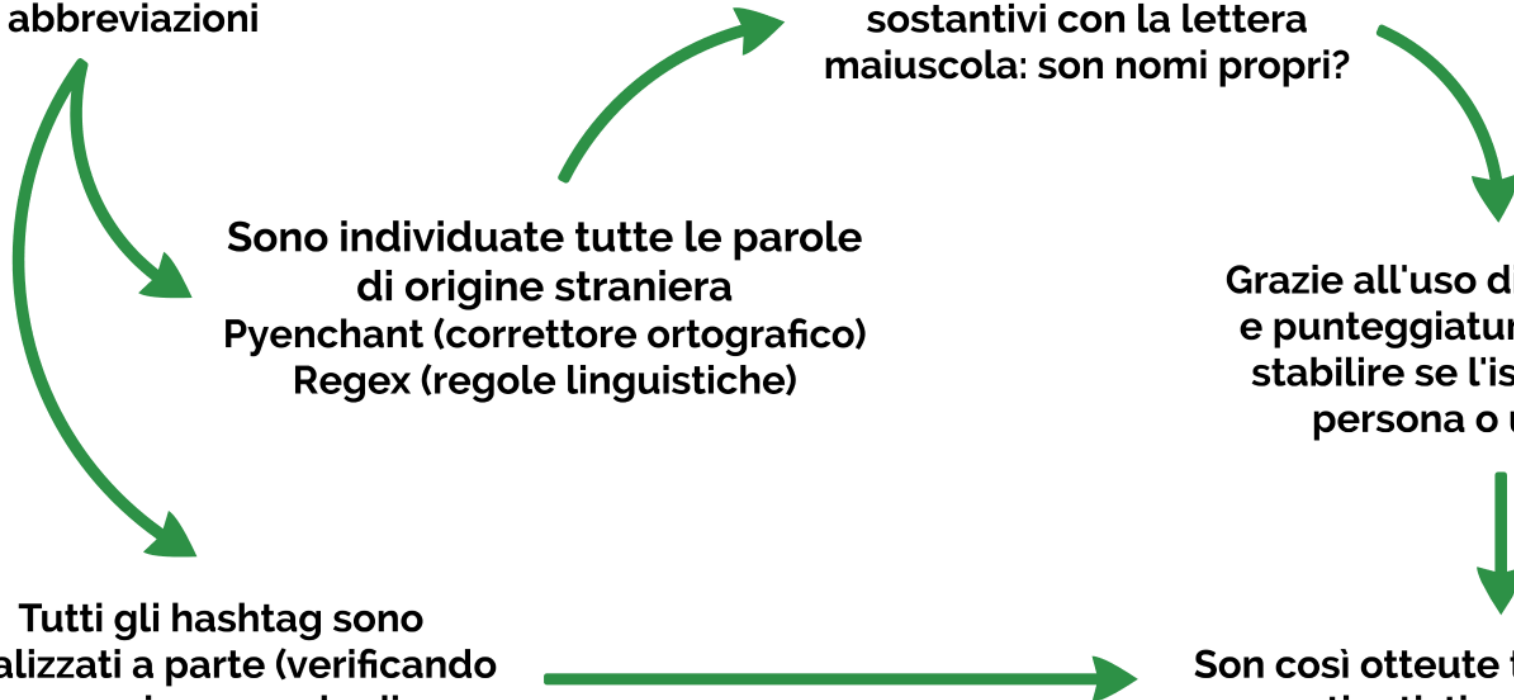
Sono individuate tutte le parole di origine straniera
Pyenchant (correttore ortografico)
Regex (regole linguistiche)

Grazie all'uso di preposizioni e punteggiatura, si tenta di stabilire se l'istanza è una persona o un'opera

Tutti gli hashtag sono analizzati a parte (verificando che non siano parole di uso comune); se di interesse sono aggiunti

Son così ottenute tre liste:

- presunti artisti
- presunte opere
- incerto



Instance Matching

Python + API Musicbrainz

Le entità riconosciute sono analizzate attraverso la libreria Python musicbrainzngs



Lo scopo è restituire gli id di musicbrainz, che sono chiamati "gid"



Attraverso le funzioni search è possibile consultare il database sfruttando un server di ricerca, costruito utilizzando Elastic Search

Problematica principale: omonimia tra canzoni ed album relativi ad artisti differenti



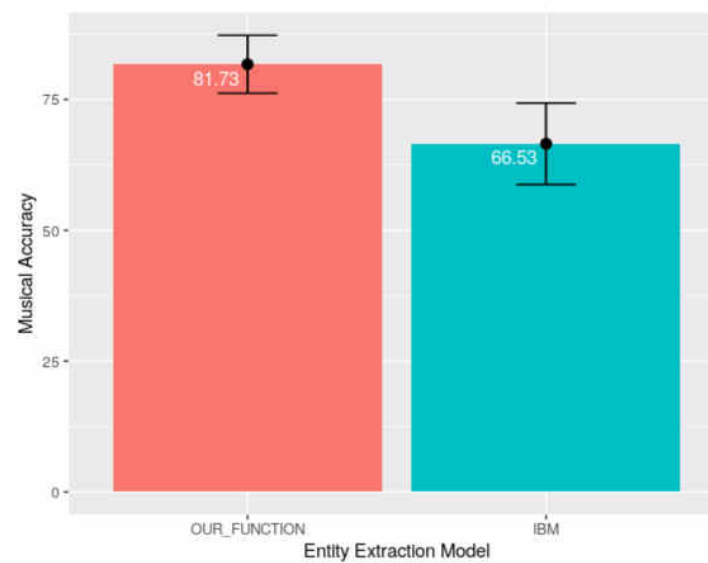
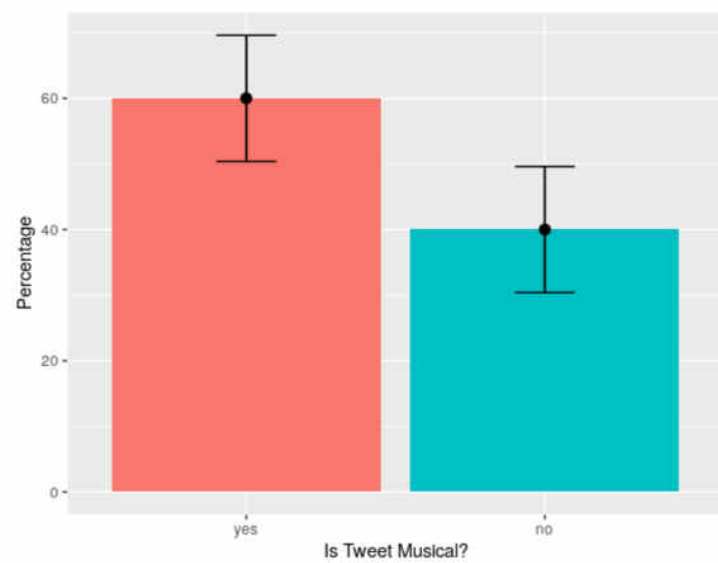
Impostato un valore di similarità pari a 0.95 nell'interrogazione al database



Controlli incrociati per trovare corrispondenze con le entità precedentemente individuate nel testo

Performance

eseguita su 100 Tweets



L'ingestione diretta e automatizzata dei tweet da Kafka in Neo4j avviene grazie al plug-in **Neo4j Streams Consumer**.

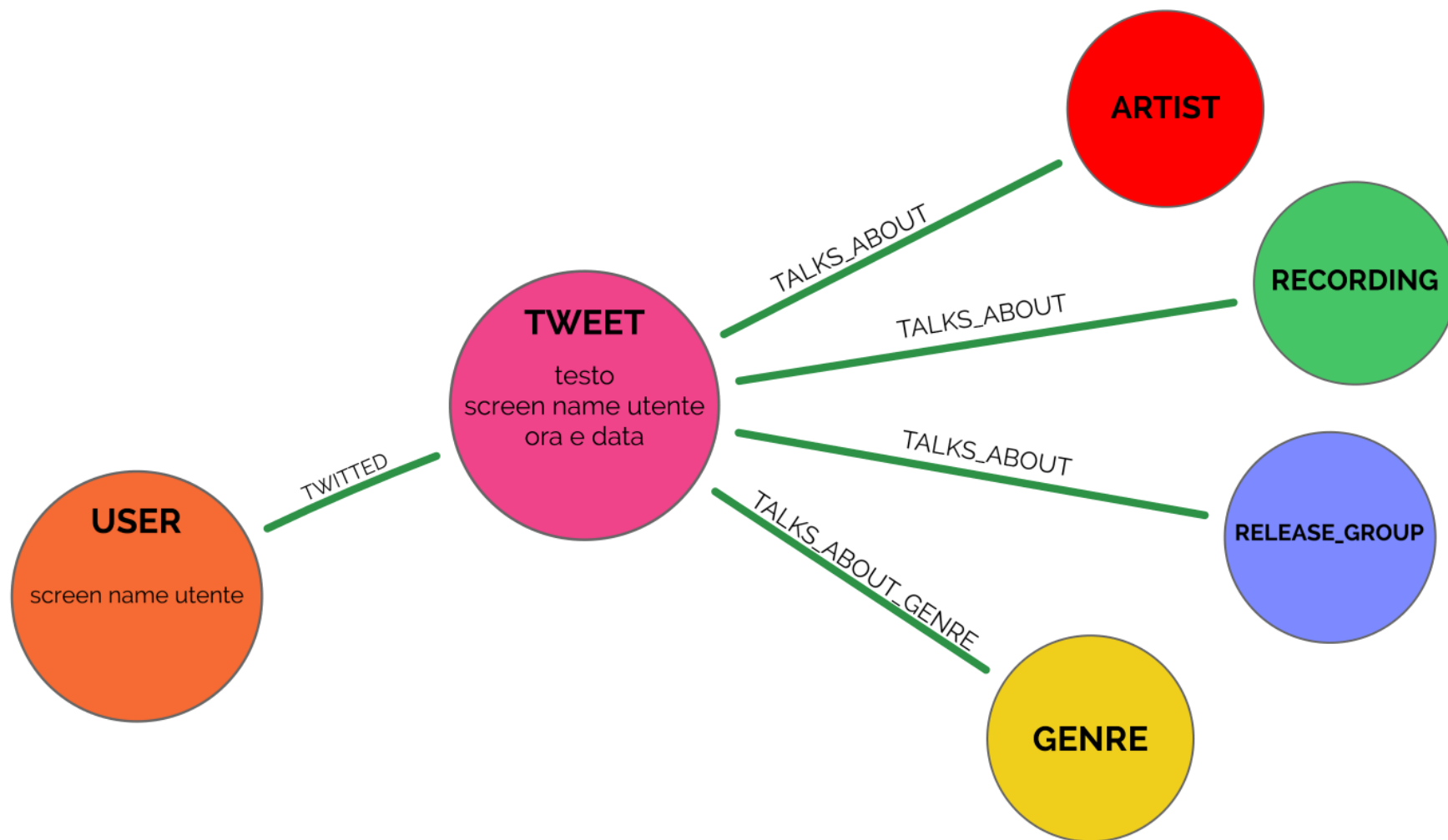
Grazie ad un Cypher template specificato nel file "docker-compose", esso permette all'utente di specificare arbitrariamente il tipo di relazioni, entità e proprietà in cui costituire i payloads di Kafka all'interno grafo.

Infine, la funzione streams.consume crea la progressiva struttura del grafo all'arrivo di ogni tweet, aggiungendo nuovi nodi ed effettuando una MERGE automatica con quelli già presenti.

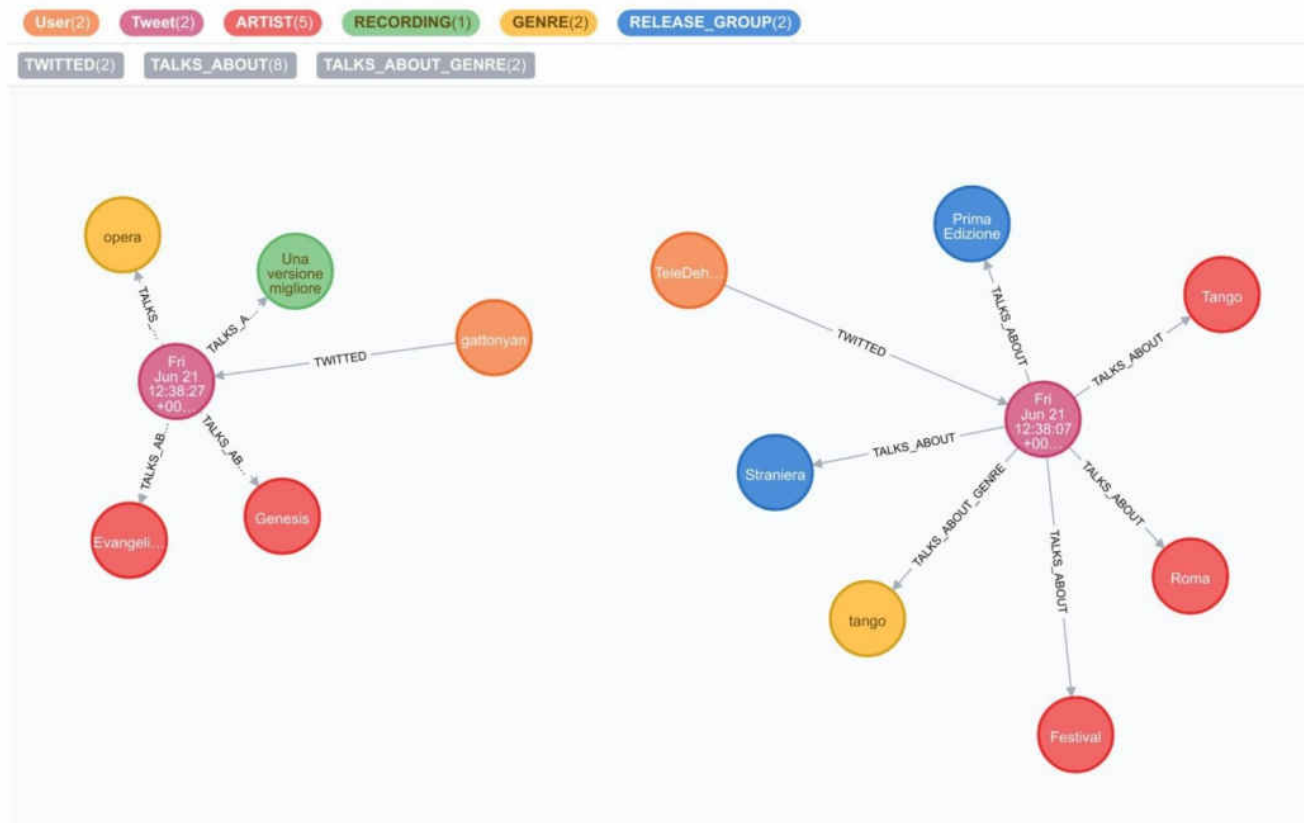
Query usato al consumo dei tweets da neo4j connector

```
MERGE (p:User{name:event.user})
CREATE (t:Tweet{text:event.text,
created_at:event.created_at})
MERGE (p)-[:TWITTED]->(t)
WITH t, event
OPTIONAL MATCH (g:GENRE) WHERE g.gid in event.genres
WITH t, event, collect(g) AS g
OPTIONAL MATCH (rc:RECORDING) WHERE rc.gid in
event.recordings
WITH t, event, g, collect(rc) AS rc
OPTIONAL MATCH (a:ARTIST) WHERE a.gid in event.artists
WITH t, event, g, rc, collect(a) AS a
OPTIONAL MATCH (rl:RELEASE) WHERE rl.gid in event.release
WITH t, event, g, rc, a, collect(rl) AS rl
OPTIONAL MATCH (rlg:RELEASE_GROUP) WHERE rlg.gid in
event.release
WITH t, event, g, rc, a, rl, collect(rlg) AS rlg
FOREACH(n in g      MERGE (t)-[:TALKS_ABOUT_GENRE]->(n))
FOREACH(n in rc     MERGE (t)-[:TALKS_ABOUT]->(n))
FOREACH(n in a      MERGE (t)-[:TALKS_ABOUT]->(n))
FOREACH(n in rl     MERGE (t)-[:TALKS_ABOUT]->(n))
FOREACH(n in rlg    MERGE (t)-[:TALKS_ABOUT]->(n))
```


Per ogni tweet verranno aggiunti i nodi e impostate le relazioni



La configurazione interna al grafo apparirà come segue:



Esempi di query possibili:

```
MATCH (t:Tweet)-[:TALKS_ABOUT_GENRE]->(g:GENRE)
RETURN t, COLLECT(g.name)
```



Lista dei tweet e dei generi presenti in essi

```
MATCH (t:Tweet)-[:TALKS_ABOUT]->(a:ARTIST)
WHERE a.name = "Queen"
RETURN t
```



Tweet che contengono l'artista Queen

```
MATCH (u:User)-[:TWITTED]->(t:Tweet)
WHERE t.created_at =~ '^Sat.*'
RETURN u.name, t
```



Utenti che hanno twittato il sabato, ed i loro relativi tweet

```
MATCH (u:User)-[*2..4]->(g:GENRE)
RETURN u.name, COLLECT(DISTINCT(g.name))
```



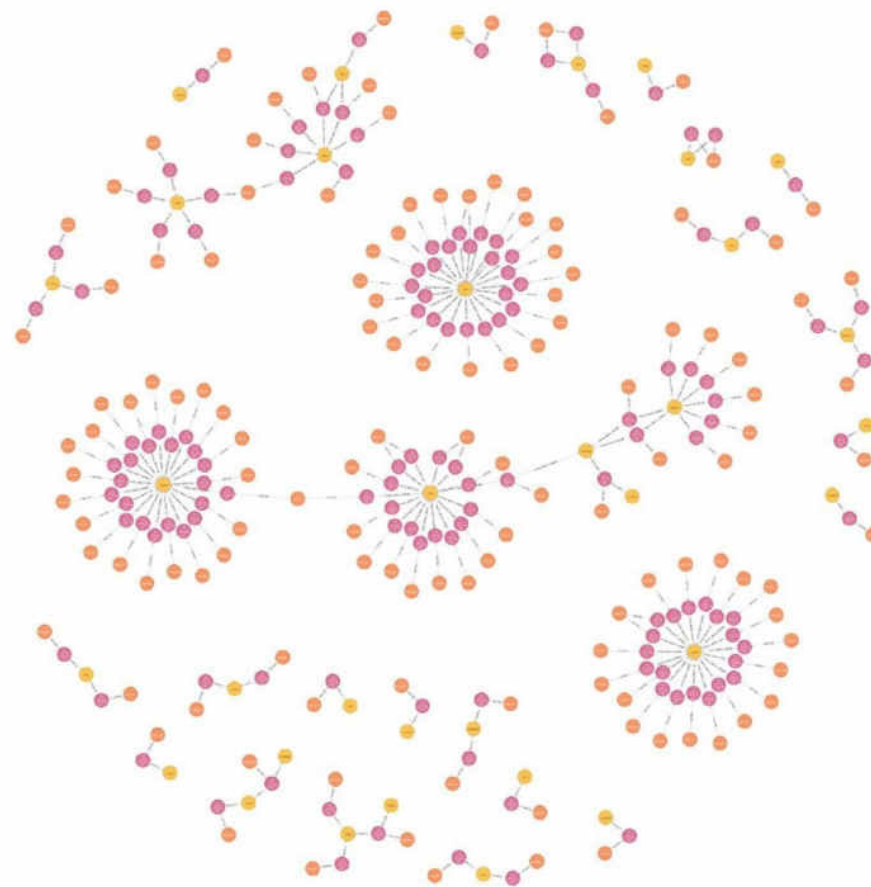
Generi collegati al singolo utente

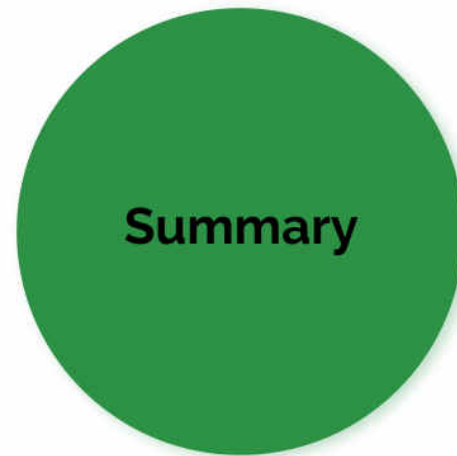
Il database scelto per lo storage è Neo4j, per le sue caratteristiche di:

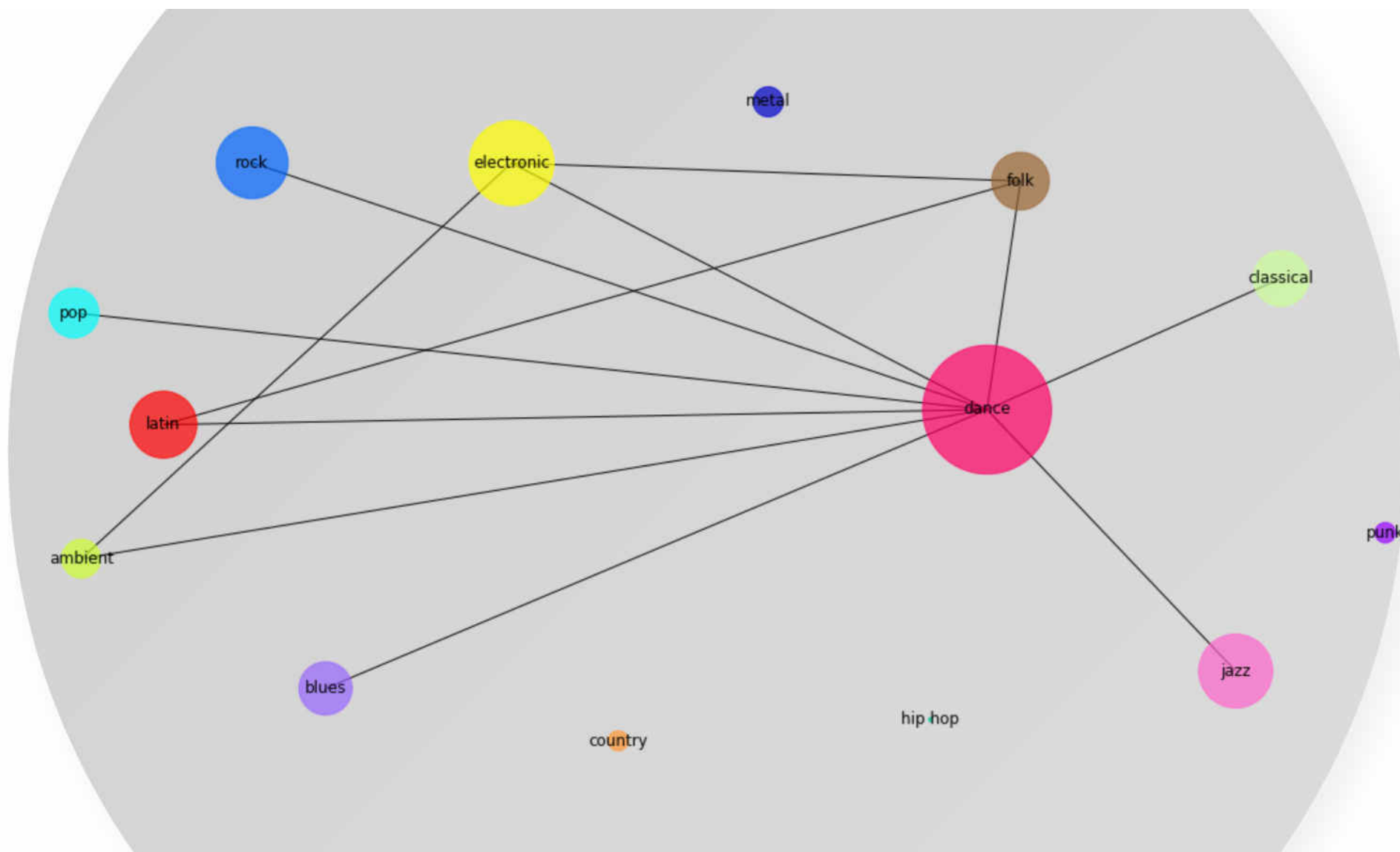
- estrema **scalabilità**
- **efficienza** nella gestione dei dati
- elevata **capacità di adattamento** al fenomeno di studio: le dinamiche sociali
- facile **interpretabilità**, anche a livello visivo.

Qui, Neo4j viene utilizzato come Consumer di Kafka, convertendo i payloads in nodi e relazioni.

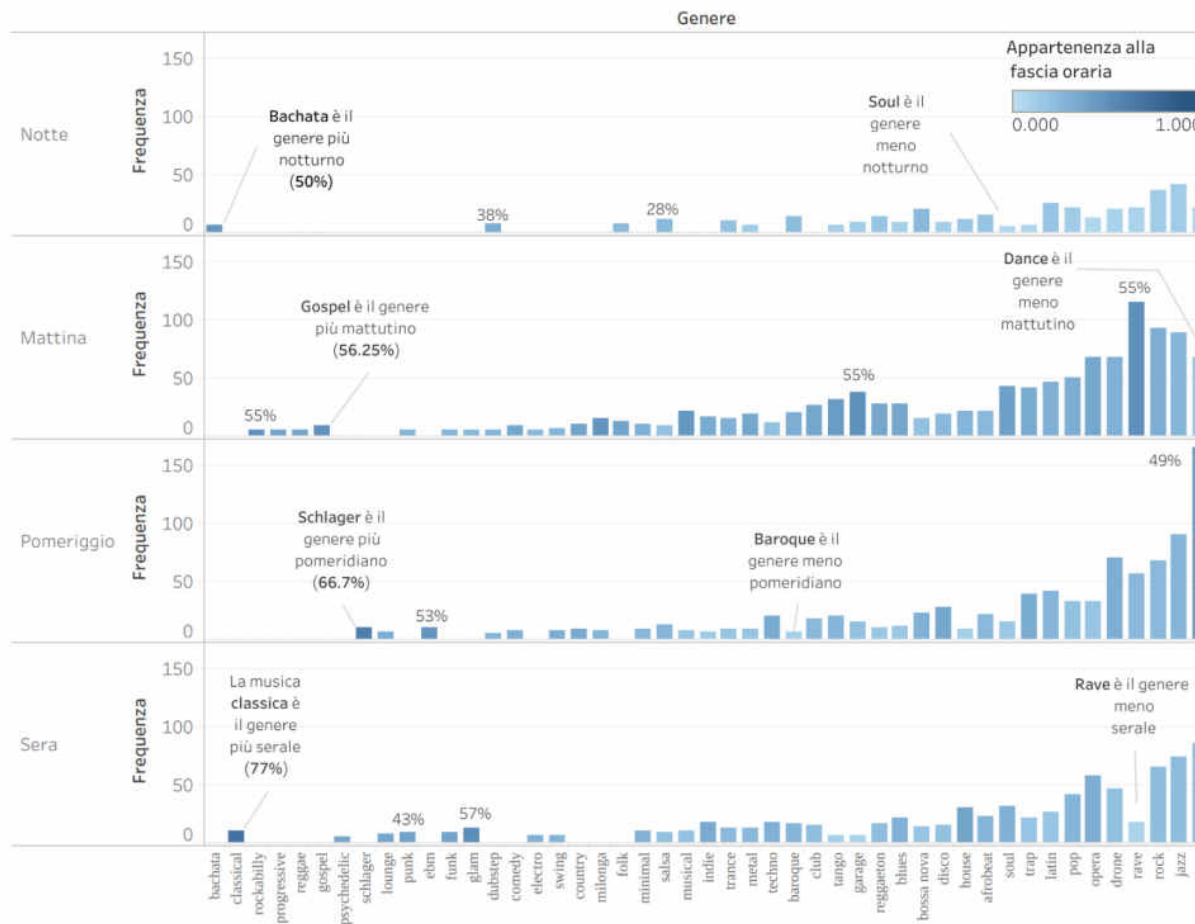
La clusterizzazione degli utenti in base al genere di cui parlano sarà simile a questa rappresentazione



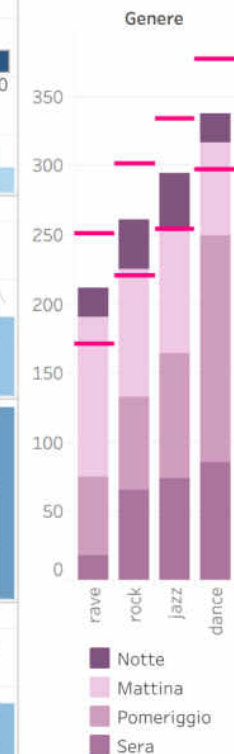




Distribuzione dei principali generi musicali per fascia oraria



Top 4 generi più citati



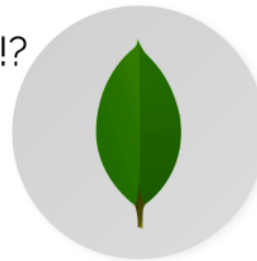
Frequenza assoluta globale dei quattro generi più citati, composta dalla somma delle grandezze assolute associate ad ogni fascia oraria. I segmenti orizzontali rossi rappresentano rispettivamente l'estremo inferiore e superiore dell'intervallo di confidenza calcolato al 95%.

Frequenza assoluta dei principali generi musicali analizzata separatamente nelle quattro fasce orarie. Il dettaglio di saturazione del colore rappresenta l'indice di appartenenza del genere alla fascia oraria, che viene inoltre specificato numericamente nei primi tre maggiori valori per ogni intervallo temporale. L'indice di appartenenza coincide con la tendenza percentuale di ogni genere a comparire in una particolare fascia oraria, rispetto al totale dei tweet che lo riguardano.



Riassumendo abbiamo usato

- Hadoop con Pig per preprocessare i dati sfruttando la velocità di processamento e fault tolerance di Hadoop, garantendo allo stesso momento una grande scalabilità
- Apache Kafka per gestire i tweets in RT con bassa latenza e alto parallelismo (concurrency)
- Riak KV per salvare i punteggi bot degli utenti in modo da diminuire la latenza dovuta alle API di botometer
- Neo4j per il DB di MusicBrainz, costruendo relazioni tra le entità musicali.
- E la scalabilità orizzontale!?



Integrazione di Neo4j con MongoDB

- Invio dei tweets elaborati a MongoDB
- Forma dei tweets:

```
{
  "user": "user_name",
  "Tweets": [{ "text": "testo primo",
                "created_at": "TimeStamp",
                "genres": ["pop", "rock"]
              },
              { "text": "testo secondo",
                "created_at": "TimeStamp",
                "genres": ["electronic", "ambient"]
              }
            ]
}
```

- Ora si sfrutta la potenza dello sharding di MongoDB con la integrazione di neo4j come una Knowledge base(graph) sulla musica.

