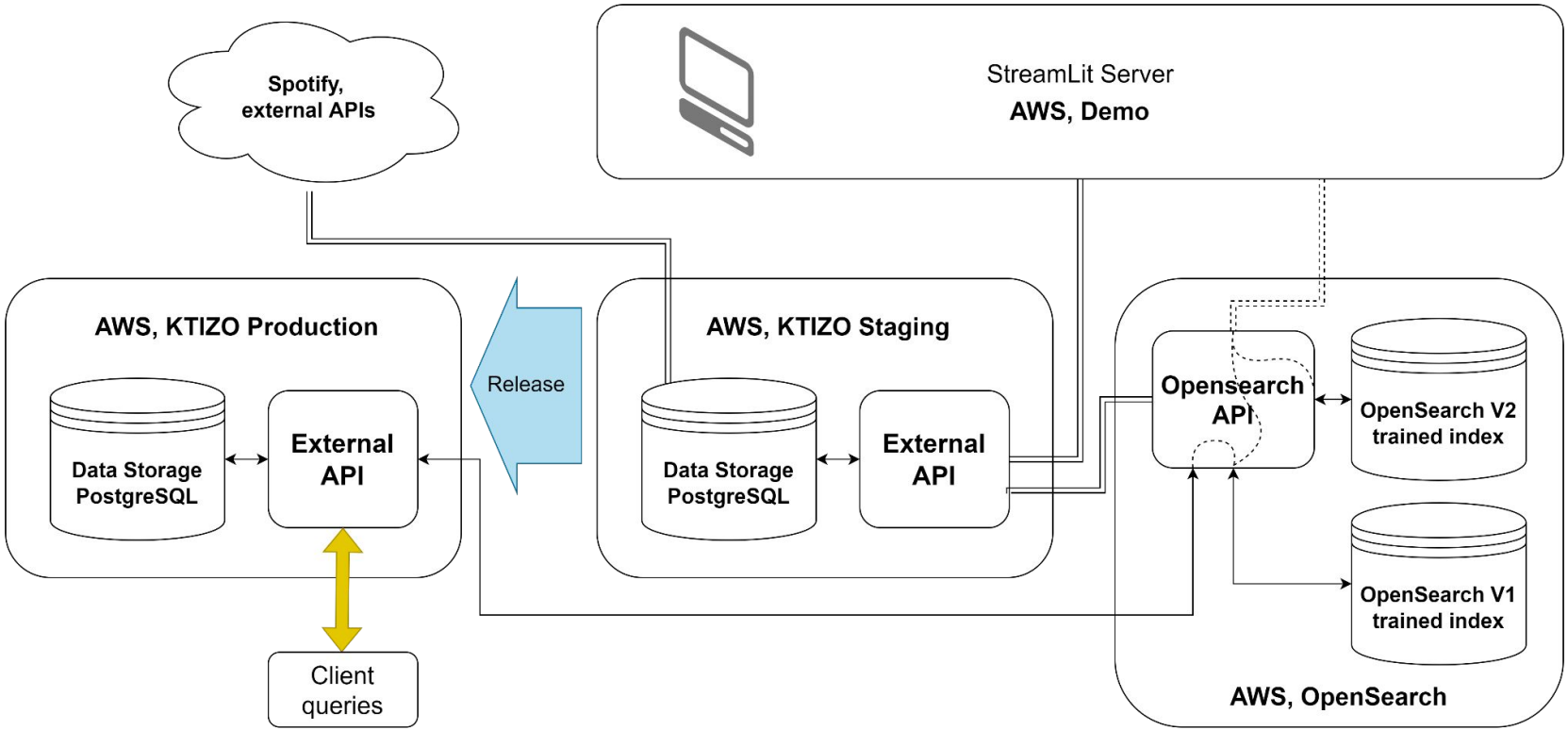
How KTIZO works?

Explanation and comparison of versions and algorithms

Architecture



Use Cases V1 & V2

Work Case: - **stay the same in V2**

1. Pick popular song for reference.
    a. Ask API KTIZO staging for the popular artists. Same with tracks of chosen Artist
    b. Choose a track
    c. Press button match (=Ask API KTIZO staging to give us specified number of closest matches)
2. Receiving matches:
    a. From DEMO server we ask KTIZO staging for a specific song.
    b. Making correct reference vector of a reference song
    c. Sending API OpenSearch reference vector for a match
    d. Receiving number of matches
    e. Sending it to DEMO

Add new unknown Artists - **stay the same in V2**

1. Going to admin page
2. Loading CSV with Artist's names
3. Asking Spotify API to find Spotify's IDs of the artists. And so on… (as on previous slide)
4. After receiving all necessary tracks into our DB we have to manually upload it to the OpenSearch and train the OpenSearch index.

Add new popular Artists - **Changed to automatic update in V2**

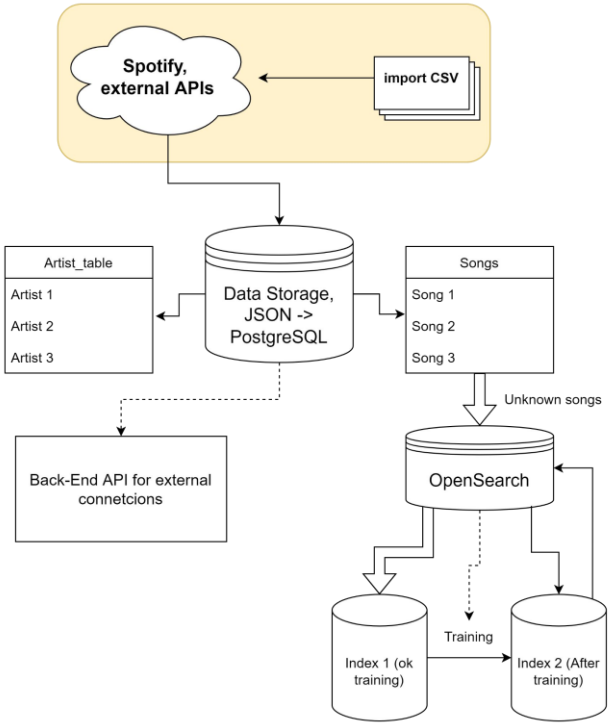
1. Same procedure like in 1-3

How Back-End works for song extraction V1

We always talk about Artists - “import TOP-500”, “50 each genre” and so on.. Why? It’s a standard way to import songs!

Once we get an Artist to import (no matter it comes from - websites, .CSV-files) - we have a task to import his(her) songs in the DB. So for the moment Back-End’s main function is to scrape songs from Spotify, put it in the OpenSearch engine, and get back recommendations from OpenSearch when User asks.

- ➔ Find Artist by it’s name on Spotify
- ➔ Detecting genre of the Artist (**this has been modified for V2**)
- ➔ Check for having uploaded Artist in our DB? If yes - skip (**and that’s the issue, but medium priority**), if no..
- ➔ ..Save Artist in separate table; start finding Albums of that Artist (we are uploading only albums with not “compilation” parameter to prevent numerous duplicates); we are not storing the albums.
- ➔ Find Songs from those albums; put these songs into separate table + Update those songs with detected genre of the Artist
- ➔ If Songs are from popular Artists, they stay still in DB, if they are from unknown Artists - they could be manually uploaded in OpenSearch.



V2 Back End

On weekly basis we will check TOP-500 popular songs and TOP-50 Artists of each genre from special genre list of Spotify

Also we have to import 100 legendary Artists in out BE database. It will be manual procedure for obvious reasons. :)

First import of TOP-500 and TOP-50 of each genre should be done manually:

1. Need to go admin panel, "Import" tab. Need to choose what to import: "top-500" or "top by genres". User will be automatically redirected to Spotify login page.
2. Log in in spotify manually
3. After authentication User will be redirected back into import tab. Our back-end will have received access token and refresh token. These tokens will be used for automatic import

Automatic procedure of scraping TOP-500 artists:

1. Using access token send AJAX-query to Spotify. Receiving TOP-200.
2. Sending query to https://chartmaster.org. Get protective token.
3. Sending query to API https://chartmaster.org with protective token. Get TOP-1000.
4. Adding to Spotify's TOP-200 300 unique artists from TOP-1000.
5. Starting standard import procedure: Name of the Artist, Presence check, Genre detection, Albums (real), Tracks from each Album. Adding genre to each track and save tracks in the DB.

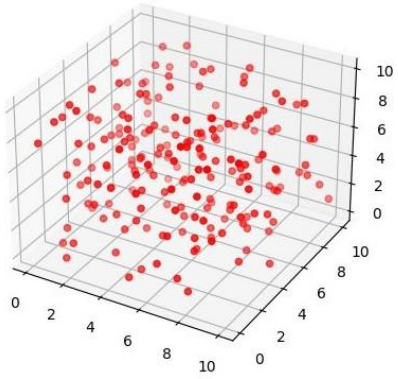
Automatic procedure of scraping TOP-50 by genre artists:

1. Using access token send AJAX-query to Spotify. Receiving 17 genres in the list.
2. Using access token send AJAX-query to Spotify. Receiving top songs of each genre.
3. Take top-50 artists for each genre. Making a common list.
4. Starting standard import procedure: Name of the Artist, Presence check, Genre detection, Albums (real), Tracks from each Album. Adding genre to each track and save tracks in the DB.

How the Matching algorithm works right now V1

- ➔ Download a song from Spotify. Each song has following parameters (from Spotify):
  'Danceability', 'energy', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'Time_signature' (**11 parameters**)
- ➔ Normalizing this vector from 11 parameters (standard math procedure, needed for correct comparison)
- ➔ Apply manually found factors to each of the parameters above
- ➔ adding "genre" from following:
  'jazz', 'rock', 'pop', 'dance', 'latin', 'country', 'classical', 'hip hop', 'r&b', 'folk', 'easy listening', 'new age', 'lo fi' (**13 genres**)
  Problem here that Spotify doesn't point a genre for each song. Genre corresponds only for Artists. So we assume that all songs of the Artist are in one Genre.
- ➔ Apply manually found factors to each of the parameters above

- ➔ At the end we have a vector with 11+13=24 parameters
- ➔ Upload a song to OpenSearch DB
- ➔ Produce teaching of OpenSearch DB. It builds up a "tree" in N-dimensional space (in this case - 24 dimensions)
- ➔ After this it's ready - you can ask to produce number of closest songs from the reference one and it will return you this number in terms of euclidean distance between points. With sorting from the closest to distant.



V2 - **genre** + KNN vector

We reworked genre grouping. Before we reworked around 500 sub-genres into 13 genres, and for the V2 we did grouping for 3600 sub-genres into 21. We believe that it will help us improve matching because genre is very valuable parameter in our vector.

We did a research for genres classification and made improvements based on our experience with previous genre choice.

- Separated asian pop and indian pop from pop

- Separated metal from rock

- Separated blues from jazz

- Separated reggae from r&b

So at the end we extend old genre set with 8 new genres: "blues", "christian music", "reggae", "asian pop", "comedy", "children's music", "indian pop", "metal"
In total we use 21 main genre.

And exactly this we presented you few days ago. As an intermediate result. Vector contained a 11+21=32 parameters.

V2 - genre (21) + **KNN vector (216) + old(11)**

Main difference between V1 and V2 is that we start to use parameters for audio analysis of the track /audio-analysis. It provides the opportunity to divide music track in parts (sections and segments) and use parameters of each part for matching. We take first 8 sections of song (first half of the song) and perform analysis of this duration.

1. *Sections_num* - number of Sections for analyse - **1 parameter**
2. *Sections_tempo_diff_{1-7}* - We take a tempo of each section and put differences in the array (S2_tempo - S1_tempo;..;S8_Tempo-S7_Tempo} - **7 parameters**;
3. *Sections_loudness_diff_{1-7}* - Same idea as in tempo differences, but with loudness - **7 parameters**;
4. *Sections_mode_1-8;* Values of modality (major or minor) of a section, the type of scale from which its melodic content is derived - **8 parameters**;
5. *Segments_duration_mean_1-8:* mean time of the first 8 segments in the each section - **8 parameters**;
6. *Segments_pitches_1-8_1-12:* **96 parameters** of pitch of each segment. Pitch content is given by a “chroma” vector, corresponding to the 12 pitch classes C, C#, D to B, with values ranging from 0 to 1 that describe the relative dominance of every pitch in the chromatic scale.
7. *Segments_timbre_1-8_2-12:* same idea like with pitch but with timbre of the segments. Timbre is the quality of a musical note or sound that distinguishes different types of musical instruments, or voices. It is a complex notion also referred to as sound color, texture, or tone quality, and is derived from the shape of a segment’s spectro-temporal surface, independently of pitch and loudness - **88 parameters**.
8. *Tatum_duration:* mean duration of tatums for all length of the track. A tatum represents the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments). - **1 parameter.**