The Evolution of Spotify Home Architecture
Our mission is to unlock the potential of human creativity — by giving a million creative artists the opportunity to live off their art and billions of fans the opportunity to enjoy and be inspired by it.
<table>
<thead>
<tr>
<th>FAST FACTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of subscribers (as at December 31 2018)</td>
<td>96 Million</td>
</tr>
<tr>
<td>Number of monthly active users (as at December 31 2018)</td>
<td>207 Million</td>
</tr>
<tr>
<td>Revenue paid to rightsholders (as at August 31 2018)</td>
<td>€10 Billion</td>
</tr>
<tr>
<td>Number of songs</td>
<td>40 Million +</td>
</tr>
<tr>
<td>Number of playlists</td>
<td>3 Billion +</td>
</tr>
<tr>
<td>Spotify is available in</td>
<td>79 Markets</td>
</tr>
</tbody>
</table>
Overview

- Started with a Batch architecture
- Used services to hide complexity and be more reactive
- Leveraged GCP and added streaming pipelines to build a product based on user activity
Batch

- Songs Played Logs
- Word2Vec
**word2vec**

A natural language processing model to learn **vector representations** of words ("embeddings") from text.

https://www.tensorflow.org/tutorials/word2vec
Input: Playlists

Output: Vector representation of tracks
**Input:**
Playlists

**Output:**
Vector representation of tracks

**Examples:**
- 2Pac
- Bach
- Mozart
Batch

Songs Played Logs

Word2Vec
Batch

- Songs Played Logs
- Hadoop Jobs
- Word2Vec
Batch

Songs Played Logs

Hadoop Jobs

Cassandra

Word2Vec
Batch

Songs Played Logs

Hadoop Jobs

Cassandra

Word2Vec
Batch

Songs Played Logs

Hadoop Jobs

Cassandra

Word2Vec

CMS
Batch

Songs Played Logs

Hadoop Jobs → Cassandra

Fetch Shelf for Home → CMS

Word2Vec
Pros & Cons

+ Low latency to load Home

+ Fallback to old data if it fails to generate recommendations

- Recommendations updated once every 24 hours

- Calculate recommendations for every user, even if they aren’t active

- Experimentation can be difficult

- Operational overhead to maintain Cassandra and Hadoop
Batch

- Songs Played Logs
- Word2Vec
- Hadoop Jobs
- Cassandra
- Fetch Shelf for Home
- CMS
Batch

Songs Played Logs

Hadoop Jobs

Word2Vec

CMS

Fetch Shelf for Home
Services

2017
Services

- Songs Played Service
- Word2Vec Service
Services

- Songs Played Service
- Word2Vec Service

CMS
Services

Songs Played Service

Word2Vec Service

Create Shelf for Home

CMS
Services

- Songs Played Service
- Word2Vec Service
- Create Shelf for Home
- CMS
Pros & Cons

+ Updates recommendations at request time
+ Calculate recommendations for Home users only
+ Simplified stack
+ Easier to Experiment
+ Google managed infrastructure

- High latency to load Home
- No fallback if request fails
Streaming ++ Services

2018 - Present
Streaming Pipelines

- Google Dataflow pipelines using Spotify Scio - scala wrapper on Apache Beam
- Real time data - Unbounded stream of user events
  - All user events are available as Google Pubsub topics
- Perform aggregation operations using time based windows
  - `groupBy`, `countBy`, `join`...
- Store the results
  - Pubsub, BigQuery, GCS, Bigtable
Real time Signals
Real time Signals

follow

pubsub

pubsub

pubsub
Real time Signals

follow

pubsub

pubsub

pubsub

Streaming Pipeline
Streaming Pipeline

Real time Signals

follow

pubsub

pubsub

pubsub

Streaming Pipeline

pubsub
Real time Signals

follow

Streaming Pipeline

pubsub

Create Shelves
Real time Signals

follow

Streaming Pipeline

pubsub

Create Shelves
Real time Signals

Streaming Pipeline

pubsub

Create Shelves

Write Shelf

BT

Fetch Shelf

Songs Played Service

Word2Vec Service

Create Shelves

Follow

Real time Signals

Follow
Pros & Cons

+ Updates recommendations based on user events
+ Computing recommendations out of request path
+ Fresher content, driven by user sessions
+ Fallback to previously generated recommendations
+ Easy to experiment

- More complex stack
- More tuning in the system
  - Event spikes
  + Guardrails
- Debugging is more complicated
# Lessons Learned

<table>
<thead>
<tr>
<th>Batch</th>
<th>Services</th>
<th>Streaming ++ Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Fallback to old recommendations</td>
<td>+ Updates are fast</td>
<td>+ Updates are frequent/fast</td>
</tr>
<tr>
<td>+ Low latency to load Home</td>
<td>- High Latency to load Home</td>
<td>+ Low latency to load Home</td>
</tr>
<tr>
<td>- Updates are slow</td>
<td>- No fallback if request fails</td>
<td>+ Fallback to old recommendations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Balance computation frequency and downstream system load</td>
</tr>
</tbody>
</table>
# Lessons Learned

<table>
<thead>
<tr>
<th>Batch</th>
<th>Services</th>
<th>Streaming ++ Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Fallback to old recommendations</td>
<td>+ Updates are fast</td>
<td>+ Updates are frequent/fast</td>
</tr>
<tr>
<td>+ Low latency to load Home</td>
<td>- High Latency to load Home</td>
<td>+ Low latency to load Home</td>
</tr>
<tr>
<td>- Updates are slow</td>
<td>- No fallback if request fails</td>
<td>+ Fallback to old recommendations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Balance computation frequency and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>downstream system load</td>
</tr>
</tbody>
</table>
Takeaways

- Less overhead with managed infrastructure. Focus more on product.
- If you care about timeliness, then adopt streaming pipelines.
  - Beware of event spikes.
- Optimize for developer productivity and ease of experimentation.
  - Creating a new shelf is as simple as writing a new function.
Hi! I’m Luna,
Any questions?