ListenBrainz Documentation

Release 0.1.0

MetaBrainz Foundation

Aug 14, 2020

CONTENTS

This documentation is for:

- Developers *using our API* to submit and fetch listens
- System administrators managing a ListenBrainz installation
- Contributors to the ListenBrainz project

Review the *JSON* documentation if you plan to work with the ListenBrainz API. Most of the complexity comes from reading or constructing ListenBrainz JSON documents.

CHAPTER

ONE

CONTENTS

1.1 ListenBrainz API

The ListenBrainz server supports the following end-points for submitting and fetching listens. All endpoints have this root URL for our current production site[1].

- **API Root URL**: `https://api.listenbrainz.org`
- **Web Root URL**: `https://listenbrainz.org`

Note: All ListenBrainz services are only available on **HTTPS**!

1.1.1 Reference

Core API Endpoints

`POST /1/submit-listens`

Submit listens to the server. A user token (found on https://listenbrainz.org/profile/) must be provided in the Authorization header! Each request should also contain at least one listen in the payload.

Listens should be submitted for tracks when the user has listened to half the track or 4 minutes of the track, whichever is lower. If the user hasn't listened to 4 minutes or half the track, it doesn't fully count as a listen and should not be submitted.

For complete details on the format of the JSON to be POSTed to this endpoint, see *JSON Documentation*.

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – listen(s) accepted.
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Response Headers

- Content-Type – *application/json*

`GET /1/validate-token`

Check whether a User Token is a valid entry in the database.

[1] Provided for compatibility with other APIs, but we still recommend using `X-RateLimit-Reset-In` wherever possible

In order to query this endpoint, send a GET request with the token to check as the *token* argument (example: /validate-token?token=token-to-check)

A JSON response, with the following format, will be returned.

- If the given token is valid:

```
{
    "code": 200,
    "message": "Token valid.",
    "valid": True,
    "user": "MusicBrainz ID of the user with the passed token"
}
```

- If the given token is invalid:

```
{
    "code": 200,
    "message": "Token invalid.",
    "valid": False,
}
```

Status Codes

- 200 OK – The user token is valid/invalid.
- 400 Bad Request – No token was sent to the endpoint.

GET /1/users/(*user_list*)**/recent-listens**

Fetch the most recent listens for a comma separated list of users. Take care to properly HTTP escape user names that contain commas!

Status Codes

- 200 OK – Fetched listens successfully.
- 400 Bad Request – Your user list was incomplete or otherwise invalid.

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/listen-count**

Get the number of listens for a user `user_name`.

The returned listen count has an element 'payload' with only key: 'count' which unsurprisingly contains the listen count for the user.

Status Codes

- 200 OK – Yay, you have listen counts!

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/playing-now**

Get the listen being played right now for user `user_name`.

This endpoint returns a JSON document with a single listen in the same format as the `/user/<user_name>/listens` endpoint, with one key difference, there will only be one listen returned at maximum and the listen will not contain a `listened_at` element.

The format for the JSON returned is defined in our *JSON Documentation*.

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

GET /1/user/(*user_name*)**/listens**

Get listens for user `user_name`. The format for the JSON returned is defined in our *JSON Documentation*.

If none of the optional arguments are given, this endpoint will return the `DEFAULT_ITEMS_PER_GET` most recent listens. The optional `max_ts` and `min_ts` UNIX epoch timestamps control at which point in time to start returning listens. You may specify max_ts or min_ts, but not both in one call. Listens are always returned in descending timestamp order.

Parameters

- **max_ts** – If you specify a `max_ts` timestamp, listens with listened_at less than (but not including) this value will be returned.
- **min_ts** – If you specify a `min_ts` timestamp, listens with listened_at greater than (but not including) this value will be returned.
- **count** – Optional, number of listens to return. Default: `DEFAULT_ITEMS_PER_GET` . Max: `MAX_ITEMS_PER_GET`

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

GET /1/latest-import

Get the timestamp of the newest listen submitted by a user in previous imports to ListenBrainz.

In order to get the timestamp for a user, make a GET request to this endpoint. The data returned will be JSON of the following format:

```
{
    'musicbrainz_id': the MusicBrainz ID of the user,
    'latest_import': the timestamp of the newest listen submitted in previous␣
↪imports.
                     Defaults to 0
}
```

Parameters

- **user_name** – the MusicBrainz ID of the user whose data is needed

Status Codes

- 200 OK – Yay, you have data!

Response Headers

- Content-Type – *application/json*

POST /1/latest-import

Update the timestamp of the newest listen submitted by a user in an import to ListenBrainz.

In order to update the timestamp of a user, you'll have to provide a user token in the Authorization Header. User tokens can be found on https://listenbrainz.org/profile/.

The JSON that needs to be posted must contain a field named *ts* in the root with a valid unix timestamp. Example:

```
{
  'ts': 0
}
```

Request Headers

- Authorization – Token <user token>

Status Codes

- 200 OK – latest import timestamp updated
- 400 Bad Request – invalid JSON sent, see error message for details.
- 401 Unauthorized – invalid authorization. See error message for details.

Statistics API Endpoints

ListenBrainz now has a statistics infrastructure that collects and computes statistics from the listen data that has been stored in the database. The endpoints in this section offer a way to get this data programmatically. Right now, we calculate statistics for the top artists, releases and recordings that a user has listened to. However, we plan to add more statistics in the near future.

GET /1/stats/user/(*user_name*)**/listening-activity**

Get the listening activity for user `user_name`. The listening activity shows the number of listens the user has submitted over a period of time.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "listening_activity": [
            {
                "from_ts": 1587945600,
                "listen_count": 26,
                "time_range": "Monday 27 April 2020",
                "to_ts": 1588031999
            },
            {
                "from_ts": 1588032000,
                "listen_count": 57,
                "time_range": "Tuesday 28 April 2020",
                "to_ts": 1588118399
            },
            {
                "from_ts": 1588118400,
                "listen_count": 33,
                "time_range": "Wednesday 29 April 2020",
                "to_ts": 1588204799
            },
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Note:

- This endpoint is currently in beta
- The example above shows the data for three days only, however we calculate the statistics for the current time range and the previous time range. For example for weekly statistics the data is calculated for the current as well as the past week.
- For `all_time` listening activity statistics we only return the years which have more than zero listens.

Parameters

- **range** (`str`) – Optional, time interval for which statistics should be returned, possible values are `week`, `month`, `year`, `all_time`, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/daily-activity**

Get the daily activity for user `user_name`. The daily activity shows the number of listens submitted by the user for each hour of the day over a period of time. We assume that all listens are in UTC.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "daily_activity": {
            "Monday": [
                {
                    "hour": 0
                    "listen_count": 26,
                },
                {
                    "hour": 1
                    "listen_count": 30,
                },
                {
                    "hour": 2
                    "listen_count": 4,
                }...
            ],
```

(continues on next page)

(continued from previous page)


```
            "Tuesday": [...],
            ...
        },
        "stats_range": "all_time",
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
    }
}
```


Note:

- This endpoint is currently in beta

Parameters

- **range** (str) – Optional, time interval for which statistics should be returned, possible values are week, month, year, all_time, defaults to all_time

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check response['error'] for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/recordings**

Get top recordings for user user_name.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "recordings": [
            {
                "artist_mbids": [],
                "artist_msid": "7addbcac-ae39-4b4c-a956-53da336d68e8",
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "recording_mbid": "0fe11cd3-0be4-467b-84fa-0bd524d45d74",
                "recording_msid": "c6b65a7e-7284-433e-ac5d-e3ff0aa4738a",
                "release_mbid": "",
                "release_msid": "de97ca87-36c4-4995-a5c9-540e35944352",
                "release_name": "Delirium (Deluxe)",
                "track_name": "Love Me Like You Do - From "Fifty Shades of Grey""
            },
            {
                "artist_mbids": [],
                "artist_msid": "3b155259-b29e-4515-aa62-cb0b917f4cfd",
                "artist_name": "The Fray",
```


(continues on next page)

(continued from previous page)


```
                "listen_count": 23,
                "recording_mbid": "0008ab49-a6ad-40b5-aa90-9d2779265c22",
                "recording_msid": "4b5bf07c-782f-4324-9242-bf56e4ba1e57",
                "release_mbid": "",
                "release_msid": "2b2a93c3-a0bd-4f46-8507-baf5ad291966",
                "release_name": "How to Save a Life",
                "track_name": "How to Save a Life"
            },
        ],
        "count": 2,
        "total_recording_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- We only calculate the top 1000 all_time recordings
- `artist_mbids`, `artist_msid`, `release_name`, `release_mbid`, `release_msid`, `recording_mbid` and `recording_msid` are optional fields and may not be present in all the responses

Parameters

- **`count`** (`int`) – Optional, number of recordings to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **`offset`** (`int`) – Optional, number of recordings to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 recordings will be skipped, defaults to 0
- **`range`** (`str`) – Optional, time interval for which statistics should be collected, possible values are `week`, `month`, `year`, `all_time`, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`GET /1/stats/user/`(*user_name*)**`/artist-map`**

Get the artist map for user `user_name`. The artist map shows the number of artists the user has listened to from different countries of the world.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "from_ts": 1587945600,
        "last_updated": 1592807084,
        "artist_map": [
            {
                "country": "USA",
                "artist_count": 34
            },
            {
                "country": "GBR",
                "artist_count": 69
            },
            {
                "country": "IND",
                "artist_count": 32
            }
        ],
        "stats_range": "all_time"
        "to_ts": 1589155200,
        "user_id": "ishaanshah"
    }
}
```


Note:

- This endpoint is currently in beta
- We cache the results for this query for a week to improve page load times, if you want to request fresh data you can use the `force_recalculate` flag.

Parameters

- **`range`** (`bool`) – Optional, time interval for which statistics should be returned, possible values are `week`, `month`, `year`, `all_time`, defaults to `all_time`
- **`force_recalculate`** – Optional, recalculate the data instead of returning the cached result.

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

`GET /1/stats/user/`(*user_name*)**`/releases`**

Get top releases for user `user_name`.

A sample response from the endpoint may look like:

```
{
    "payload": {
        "releases": [
            {
                "artist_mbids": [],
                "artist_msid": "6599e41e-390c-4855-a2ac-68ee798538b4",
                "artist_name": "Coldplay",
                "listen_count": 26,
                "release_mbid": "",
                "release_msid": "d59730cf-f0e3-441e-a7a7-8e0f589632a5",
                "release_name": "Live in Buenos Aires"
            },
            {
                "artist_mbids": [],
                "artist_msid": "7addbcac-ae39-4b4c-a956-53da336d68e8",
                "artist_name": "Ellie Goulding",
                "listen_count": 25,
                "release_mbid": "",
                "release_msid": "de97ca87-36c4-4995-a5c9-540e35944352",
                "release_name": "Delirium (Deluxe)"
            },
            {
                "artist_mbids": [],
                "artist_msid": "3b155259-b29e-4515-aa62-cb0b917f4cfd",
                "artist_name": "The Fray",
                "listen_count": 25,
                "release_mbid": "",
                "release_msid": "2b2a93c3-a0bd-4f46-8507-baf5ad291966",
                "release_name": "How to Save a Life"
            },
        ],
        "count": 3,
        "total_release_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- `artist_mbids`, `artist_msid`, `release_mbid` and `release_msid` are optional fields and may not be present in all the responses

Parameters

- **`count`** (`int`) – Optional, number of releases to return, Default: `DEFAULT_ITEMS_PER_GET` Max: `MAX_ITEMS_PER_GET`
- **`offset`** (`int`) – Optional, number of releases to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 releases will be skipped, defaults to 0
- **`range`** (`str`) – Optional, time interval for which statistics should be collected, possible

values are `week`, `month`, `year`, `all_time`, defaults to `all_time`

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check `response['error']` for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

GET /1/stats/user/(*user_name*)**/artists**

Get top artists for user `user_name`.

A sample response from the endpoint may look like:


```
{
    "payload": {
        "artists": [
            {
               "artist_mbids": ["93e6118e-7fa8-49f6-9e02-699a1ebce105"],
               "artist_msid": "d340853d-7408-4a0d-89c2-6ff13e568815",
               "artist_name": "The Local train",
               "listen_count": 385
            },
            {
               "artist_mbids": ["ae9ed5e2-4caf-4b3d-9cb3-2ad626b91714"],
               "artist_msid": "ba64b195-01dd-4613-9534-bb87dc44cffb",
               "artist_name": "Lenka",
               "listen_count": 333
            },
            {
               "artist_mbids": ["cc197bad-dc9c-440d-a5b5-d52ba2e14234"],
               "artist_msid": "6599e41e-390c-4855-a2ac-68ee798538b4",
               "artist_name": "Coldplay",
               "listen_count": 321
            }
        ],
        "count": 3,
        "total_artist_count": 175,
        "range": "all_time",
        "last_updated": 1588494361,
        "user_id": "John Doe",
        "from_ts": 1009823400,
        "to_ts": 1590029157
    }
}
```


Note:

- This endpoint is currently in beta
- `artist_mbids` and `artist_msid` are optional fields and may not be present in all the responses

Parameters

- **count** (int) – Optional, number of artists to return, Default: DEFAULT_ITEMS_PER_GET Max: MAX_ITEMS_PER_GET
- **offset** (int) – Optional, number of artists to skip from the beginning, for pagination. Ex. An offset of 5 means the top 5 artists will be skipped, defaults to 0
- **range** (str) – Optional, time interval for which statistics should be collected, possible values are week, month, year, all_time, defaults to all_time

Status Codes

- 200 OK – Successful query, you have data!
- 204 No Content – Statistics for the user haven't been calculated, empty response will be returned
- 400 Bad Request – Bad request, check response['error'] for more details
- 404 Not Found – User not found

Response Headers

- Content-Type – *application/json*

Status API Endpoints

GET /1/status/get-dump-info

Get information about ListenBrainz data dumps. You need to pass the *id* parameter in a GET request to get data about that particular dump.

Example response:

```
{
    "id": 1,
    "timestamp": "20190625-165900"
}
```

Query Parameters

- **id** – Integer specifying the ID of the dump, if not provided, the endpoint returns information about the latest data dump.

Status Codes

- 200 OK – You have data.
- 400 Bad Request – You did not provide a valid dump ID. See error message for details.
- 404 Not Found – Dump with given ID does not exist.

Response Headers

- Content-Type – *application/json*

Rate limiting

The ListenBrainz API is rate limited via the use of rate limiting headers that are sent as part of the HTTP response headers. Each call will include the following headers:

- **X-RateLimit-Limit**: Number of requests allowed in given time window
- **X-RateLimit-Remaining**: Number of requests remaining in current time window
- **X-RateLimit-Reset-In**: Number of seconds when current time window expires (*recommended*: this header is resilient against clients with incorrect clocks)
- **X-RateLimit-Reset**: UNIX epoch number of seconds (without timezone) when current time window expires [#]_

Rate limiting is automatic and the client must use these headers to determine the rate to make API calls. If the client exceeds the number of requests allowed, the server will respond with error code `429: Too Many Requests`. Requests that provide the *Authorization* header with a valid user token may receive higher rate limits than those without valid user tokens.

Timestamps

All timestamps used in ListenBrainz are UNIX epoch timestamps in UTC. When submitting timestamps to us, please ensure that you have no timezone adjustments on your timestamps.

Constants

Constants that are relevant to using the API:

`listenbrainz.webserver.views.api_tools.`**`MAX_LISTEN_SIZE = 10240`**
: Maximum overall listen size in bytes, to prevent egregious spamming.

`listenbrainz.webserver.views.api_tools.`**`MAX_ITEMS_PER_GET = 100`**
: The maximum number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**`DEFAULT_ITEMS_PER_GET = 25`**
: The default number of listens returned in a single GET request.

`listenbrainz.webserver.views.api_tools.`**`MAX_TAGS_PER_LISTEN = 50`**
: The maximum number of tags per listen.

`listenbrainz.webserver.views.api_tools.`**`MAX_TAG_SIZE = 64`**
: The maximum length of a tag

1.2 JSON Documentation

Note: Do not submit copyrighted information in these fields!

1.2.1 Submission JSON

To submit a listen via our API (see: *ListenBrainz API*), `POST` a JSON document to the `submit-listens` endpoint. Submit one of three types JSON documents:

- `single`: Submit single listen
  - Indicates user just finished listening to track
  - `payload` should contain information about *exactly one* track
- `playing_now`: Submit `playing_now` notification
  - Indicates that user just began listening to track
  - `payload` should contain information about *exactly one* track
  - Submitting `playing_now` documents is optional
  - Timestamp must be omitted from a `playing_now` submission.
- `import`: Submit previously saved listens
  - `payload` should contain information about *at least one* track
  - submitting multiple listens in one go is allowed, but the complete JSON document may not exceed `MAX_LISTEN_SIZE` bytes in size

The `listen_type` element defines different types of submissions. The element is placed at the top-most level of the JSON document. The only other required element is the `payload` element. This provides an array of listens – the payload may be one or more listens (as designated by `listen_type`):

```
{
  "listen_type": "single",
  "payload": [
      --- listen data here ---
  ]
}
```

A sample listen payload may look like:

```
{
  "listened_at": 1443521965,
  "track_metadata": {
    "additional_info": {
      "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
      "artist_mbids": [
        "db92a151-1ac2-438b-bc43-b82e149ddd50"
      ],
      "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
      "tags": [ "you", "just", "got", "rick rolled!"]
    },
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
    "release_name": "Whenever you need somebody"
  }
}
```

A complete submit listen JSON document may look like:

```
{
  "listen_type": "single",
  "payload": [
    {
      "listened_at": 1443521965,
      "track_metadata": {
        "additional_info": {
          "listening_from": "spotify",
          "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
          "artist_mbids": [
            "db92a151-1ac2-438b-bc43-b82e149ddd50"
          ],
          "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
          "tags": [ "you", "just", "got", "rick rolled!"]
        },
        "artist_name": "Rick Astley",
        "track_name": "Never Gonna Give You Up",
        "release_name": "Whenever you need somebody"
      }
    }
  ]
}
```

1.2.2 Fetching listen JSON

The JSON documents returned from our API look like the following:

```
{
  "payload": {
    "count": 25,
    "user_id": "-- the MusicBrainz ID of the user --",
    "listens": [
      "-- listen data here ---"
    ]
  }
}
```

The number of listens in the document are returned by the top-level `count` element. The `user_id` element contains the MusicBrainz ID of the user whose listens are being returned. The other element is the `listens` element. This is a list which contains the listen JSON elements (described above).

The JSON document returned by the API endpoint for getting tracks being played right now is the same as above, except that it also contains the `payload/playing_now` element as a boolean set to True.

1.2.3 Payload JSON details

A minimal payload must include `track_metadata/artist_name` and `track_metadata/track_name` elements:

```
{
  "track_metadata": {
    "artist_name": "Rick Astley",
    "track_name": "Never Gonna Give You Up",
  }
}
```

`artist_name` and `track_name` elements must be simple strings.

The payload will also include the `listened_at` element which must be an integer representing the Unix time when the track was listened to. The only exception to this rule is when the listen is being played right now and has been retrieved from the endpoint to get listens being played right now. The `listened_at` element will be absent for such listens.

Add additional metadata you may have for a track to the `additional_info` element. Any additional information allows us to better correlate your listen data to existing MusicBrainz-based data. If you have MusicBrainz IDs available, submit them!

The following optional elements may also be included in the `track_metadata` element:

element	description
`release_name`	the name of the release this recording was played from.

The following optional elements may also be included in the `additional_info` element. If you do not have the data for any of the following fields, omit the key entirely:

element	description
`artist_mbids`	A list of MusicBrainz Artist IDs, one or more Artist IDs may be included here. If you have a complete MusicBrainz artist credit that contains multiple Artist IDs, include them all in this list.
`release_group_mbid`	A MusicBrainz Release Group ID of the release group this recording was played from.
`release_mbid`	A MusicBrainz Release ID of the release this recording was played from.
`recording_mbid`	A MusicBrainz Recording ID of the recording that was played.
`track_mbid`	A MusicBrainz Track ID associated with the recording that was played.
`work_mbids`	A list of MusicBrainz Work IDs that may be associated with this recording.
`tracknumber`	The tracknumber of the recording. This first recording on a release is tracknumber 1.
`isrc`	The ISRC code associated with the recording.
`spotify_id`	The Spotify track URL associated with this recording. e.g.: http://open.spotify.com/track/1rrgWMXGCGHru5bIRxGFV0
`tags`	A list of user defined tags to be associated with this recording. These tags are similar to last.fm tags. For example, you have apply tags such as `punk`, `see-live`, `smelly`. You may submit up to `MAX_TAGS_PER_LISTEN` tags and each tag may be up to `MAX_TAG_SIZE` characters large.
`listening_from`	The source of the listen, i.e the name of the client or service which submits the listen.

At this point, we are not scrubbing any superflous elements that may be submitted via the `additional_info` element. We're open to see how people will make use of these unspecified fields and may decide to formally specify or scrub elements in the future.

1.3 Client Libraries

Client Libraries have already been written by the community for some languages.

1.3.1 Haskell

- listenbrainz-client

1.3.2 Go

- go-listenbrainz

1.3.3 Rust

- listenbrainz-rust

1.3.4 .NET

- MetaBrainz.ListenBrainz

1.3.5 Python

- pylistenbrainz

1.4 API Usage Examples

Note: These examples are written in Python version **3.6.3** and use requests version **2.18.4**.

1.4.1 Prerequisites

All the examples assume you have a development version of the ListenBrainz server set up on `localhost`. Remember to set `DEBUG` to `True` in the config. When in production, you can replace `localhost` with `api.listenbrainz.org` to use the real API. In order to use either one, you'll need a token. You can find it under `ROOT/profile/` when signed in, with `ROOT` being either `localhost` for the dev version or `listenbrainz.org` for the real API.

Caution: You should use the token from the API you're using. In production, change the token to one from `listenbrainz.org`.

1.4.2 Examples

Submitting Listens

See *JSON Documentation* for details on the format of the Track dictionaries.

If everything goes well, the json response should be `{"status": "ok"}`, and you should see a recent listen of "Never Gonna Give You Up" when you visit `ROOT/user/{your-user-name}`.

```
from time import time
import requests

ROOT = '127.0.0.1'

def submit_listen(listen_type, payload, token):
    """Submits listens for the track(s) in payload.

    Args:
        listen_type (str): either of 'single', 'import' or 'playing_now'
        payload: A list of Track dictionaries.
        token: the auth token of the user you're submitting listens for

    Returns:
         The json response if there's an OK status.

    Raises:
         An HTTPError if there's a failure.
         A ValueError is the JSON in the response is invalid.
    """

    response = requests.post(
        url="http://{0}/1/submit-listens".format(ROOT),
        json={
            "listen_type": listen_type,
            "payload": payload,
        },
        headers={
            "Authorization": "Token {0}".format(token)
        }
    )

    response.raise_for_status()

    return response.json()


if __name__ == "__main__":
    EXAMPLE_PAYLOAD = [
        {
            # An example track.
            "listened_at": int(time()),
            "track_metadata": {
                "additional_info": {
                    "release_mbid": "bf9e91ea-8029-4a04-a26a-224e00a83266",
                    "artist_mbids": [
                        "db92a151-1ac2-438b-bc43-b82e149ddd50"
                    ],
                    "recording_mbid": "98255a8c-017a-4bc7-8dd6-1fa36124572b",
```

(continues on next page)

(continued from previous page)

```
                    "tags": ["you", "just", "got", "semi", "rick", "rolled"]
                },
                "artist_name": "Rick Astley",
                "track_name": "Never Gonna Give You Up",
                "release_name": "Whenever you need somebody"
            }
        }
    ]

    # Input token from the user and call submit listen
    token = input('Please enter your auth token: ')
    json_response = submit_listen(listen_type='single', payload=EXAMPLE_PAYLOAD,␣
↪token=token)

    print("Response was: {0}".format(json_response))
    print("Check your listens - there should be a Never Gonna Give You Up track,␣
↪played recently.")
```

Getting Listen History

See *JSON Documentation* for details on the format of the Track dictionaries.

If there's nothing in the listen history of your user, you can run `submit_listens` before this.

If there is some listen history, you should see a list of tracks like this:

```
import requests

ROOT = '127.0.0.1'
# The following token must be valid, but it doesn't have to be the token of the user␣
↪you're
# trying to get the listen history of.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}


def get_listens(username, min_ts=None, max_ts=None, count=None):
    """Gets the listen history of a given user.

    Args:
        username: User to get listen history of.
        min_ts: History before this timestamp will not be returned.
                DO NOT USE WITH max_ts.
        max_ts: History after this timestamp will not be returned.
                DO NOT USE WITH min_ts.
        count: How many listens to return. If not specified,
               uses a default from the server.

    Returns:
        A list of listen info dictionaries if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
```

(continues on next page)

(continued from previous page)

```
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/user/{1}/listens".format(ROOT, username),
        params={
            "min_ts": min_ts,
            "max_ts": max_ts,
            "count": count,
        },
        # Note that an authorization header isn't compulsary for requests to get␣
↪listens
        # BUT requests with authorization headers are given relaxed rate limits by␣
↪ListenBrainz
        headers=AUTH_HEADER,
    )

    response.raise_for_status()

    return response.json()['payload']['listens']

if __name__ == "__main__":
    username = input('Please input the MusicBrainz ID of the user: ')
    listens = get_listens(username)

    for track in listens:
        print("Track: {0}, listened at {1}".format(track["track_metadata"]["track_name
↪"],
                                                   track["listened_at"]))
```

```
Track: Never Gonna Give You Up, listened at 1512040365
Track: Never Gonna Give You Up, listened at 1511977429
Track: Never Gonna Give You Up, listened at 1511968583
Track: Never Gonna Give You Up, listened at 1443521965
Track: Never Gonna Give You Up, listened at 42042042
```

Latest Import

Set and get the timestamp of the latest import into ListenBrainz.

Setting

```
from time import time
import requests

ROOT = '127.0.0.1'

def set_latest_import(timestamp, token):
    """Sets the time of the latest import.

    Args:
        timestamp: Unix epoch to set latest import to.
        token: the auth token of the user you're setting latest_import of
```

(continues on next page)

(continued from previous page)

```
    Returns:
        The JSON response if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON response is invalid.
    """
    response = requests.post(
        url="http://{0}/1/latest-import".format(ROOT),
        json={
            "ts": timestamp
        },
        headers={
            "Authorization": "Token {0}".format(token),
        }
    )

    response.raise_for_status()

    return response.json()

if __name__ == "__main__":
    ts = int(time())
    token = input('Please enter your auth token: ')
    json_response = set_latest_import(ts, token)

    print("Response was: {0}".format(json_response))
    print("Set latest import time to {0}.".format(ts))
```

Getting

If your user has never imported before and the latest import has never been set by a script, then the server will return `0` by default. Run `set_latest_import` before this if you don't want to actually import any data.

```
import requests

ROOT = '127.0.0.1'
# The token can be any valid token.
TOKEN = 'YOUR_TOKEN_HERE'
AUTH_HEADER = {
    "Authorization": "Token {0}".format(TOKEN)
}

def get_latest_import(username):
    """Gets the latest import timestamp of a given user.

    Args:
        username: User to get latest import time of.

    Returns:
        A Unix timestamp if there's an OK status.

    Raises:
        An HTTPError if there's a failure.
        A ValueError if the JSON in the response is invalid.
```

(continues on next page)

(continued from previous page)

```
        An IndexError if the JSON is not structured as expected.
    """
    response = requests.get(
        url="http://{0}/1/latest-import".format(ROOT),
        params={
            "user_name": username,
        },
        headers=AUTH_HEADER,
    )

    response.raise_for_status()
    return response.json()["latest_import"]


if __name__ == "__main__":
    username = input('Please input the MusicBrainz ID of the user: ')
    timestamp = get_latest_import(username)

    print("User {0} last imported on {1}".format(username, timestamp))
```

You should see output like this:

```
User naiveaiguy last imported on 30 11 2017 at 12:23
```

1.5 ListenBrainz Data Dumps

ListenBrainz provides data dumps that you can import into your own server or use for other purposes. The full data dumps are created twice a month and the incremental data dumps twice a week. Each dump contains a number of different files. Depending on your use cases, you may or may not require all of them.

1.5.1 File Descriptions

A ListenBrainz data dump consists of three archives:

1. `listenbrainz-public-dump.tar.xz`
2. `listenbrainz-listens-dump.tar.xz`
3. `listenbrainz-listens-dump-spark.tar.xz`

listenbrainz-public-dump.tar.xz

This file contains information about ListenBrainz users and statistics derived from listens submitted to ListenBrainz calculated from users, artists, recordings etc.

listenbrainz-listens-dump.tar.xz

This is the core ListenBrainz data dump. This file contains all the listens submitted to ListenBrainz by its users.

listenbrainz-listens-dump-spark.tar.xz

This is also a dump of the core ListenBrainz listen data. These dumps are made for consumption by the ListenBrainz Apache Spark cluster, formatting all listens into monthly JSON files that can easily be loaded into dataframes.

1.5.2 Structure of the listens dump

The ListenBrainz listen dump consists of listens broken down by year and month. At the top level there are directories for each of the year for which we have data. Inside each year there are listens files with month number as its name:

1. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/1.listens`
2. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/2.listens`
3. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/3.listens`
4. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/4.listens`
5. `listenbrainz-listens-dump-183-20200727-001004-full/listens/2005/5.listens`

Each of the .listens files contains one JSON document per line – each of the JSON documents is one listen, formatted in the standard listens format.

1.5.3 Incremental dumps (BETA)

> **Warning:** The incremental dumps are in beta. We know of some data consistency issues where incremental dumps have fewer listens than they should. Make sure you use the full dumps if data accuracy is important.

ListenBrainz provides incremental data dumps that you can use to keep up to date with the ListenBrainz dataset without needing to download the full dumps everytime. These dumps have the same structure as the corresponding full dumps, but only contain data that has been submitted since the creation of the previous dump. We create incremental data dumps twice a week.

The basic idea here is that dumps create a linear timeline of the dataset based on the time of submission of data. In order to use the incremental dumps, you must start with the latest full dump and then, applying all incremental dumps since will give you the latest data. The series is consistent, if you take a full dump and apply all incremental dumps since that full dump until the next full dump, you will have the same data as the next full dump.

1.6 Set up ListenBrainz Server development environment

To contribute to the ListenBrainz project, you need a development environment. With your development environment, you can test your changes before submitting a patch to the project. This guide helps you set up a development environment and run ListenBrainz locally on your workstation. By the end of this guide, you will have...

- Installed system dependencies
- Registered a MusicBrainz application
- Initialized development databases

- Running ListenBrainz Server

1.6.1 Clone listenbrainz-server

ListenBrainz is hosted on GitHub at https://github.com/metabrainz/listenbrainz-server/. You can use `git` to clone it to your computer

```
git clone https://github.com/metabrainz/listenbrainz-server.git
```

1.6.2 Install docker

ListenBrainz uses Docker for development. This helps you to easily create your development environment. Therefore, to work on the project, you first need to install Docker. If you haven't already, follow the docker installation instructions for your platform.

1.6.3 Register a MusicBrainz application

Next, you need to register your application and get an OAuth token from MusicBrainz. This allows you to sign into your development environment with your MusicBrainz account.

To register, visit the MusicBrainz applications page. There, look for the option to register your application. Fill out the form with the following data:

- **Name**: (any name that you want and will recognize, e.g. `listenbrainz-server-devel`)
- **Type**: `Web Application`
- **Callback URL**: `http://localhost/login/musicbrainz/post`

After entering this information, you'll have an OAuth client ID and OAuth client secret. You'll use these for configuring ListenBrainz.

Update config.py

With your new client ID and secret, update the ListenBrainz configuration file. If this is your first time configuring ListenBrainz, copy the sample to a live configuration.

```
cp listenbrainz/config.py.sample listenbrainz/config.py
```

Next, open the file with your favorite text editor and look for this section.

```
# MusicBrainz OAuth
MUSICBRAINZ_CLIENT_ID = "CLIENT_ID"
MUSICBRAINZ_CLIENT_SECRET = "CLIENT_SECRET"
```

Update the strings with your client ID and secret. After doing this, your ListenBrainz development environment is able to authenticate and log in from your MusicBrainz login.

To use the Last.fm importer you need an API account at Last.fm. You can register for one at the Last.fm API page. Look for the following section in `config.py`.

```
# Lastfm API
LASTFM_API_URL = "https://ws.audioscrobbler.com/2.0/"
LASTFM_API_KEY = "USE_LASTFM_API_KEY"
```

Update the `LASTFM_API_KEY` field with your Last.fm API key.

You also need to update the `API_URL` field value to `http://localhost`.

To use the Spotify importer you need to register an application on the Spotify Developer Dashboard. Use `http://localhost/profile/connect-spotify/callback` as the callback URL.

After that, fill out the Spotify client ID and client secret in the following section of the file.

```
# SPOTIFY
SPOTIFY_CLIENT_ID = ''
SPOTIFY_CLIENT_SECRET = ''
```

Note: The hostname on the callback URL must be the same as the host you use to access your development server. If you use something other than `localhost`, you should update the `SPOTIFY_CALLBACK_URL` field accordingly.

1.6.4 Initialize ListenBrainz containers

Next, run

```
./develop.sh build
```

in the root of the repository. Using `docker-compose`, this will build multiple Docker images for the different services that make up the ListenBrainz server.

The first time you run this script it might take some time while it downloads all of the required dependencies and builds the services.

1.6.5 Initialize ListenBrainz databases

Your development environment needs some specific databases to work. Before proceeding, run these commands to initialize the databases.

```
./develop.sh manage init_db --create-db
./develop.sh manage init_msb_db --create-db
./develop.sh manage init_ts_db --create-db
```

Your development environment is now ready. Now, let's actually see ListenBrainz load locally!

1.6.6 Run the magic script

Now that the databases are initialized, you can start your development environment by running `develop.sh up`.

```
./develop.sh up
```

You will see the output of `docker-compose`. You can shut down listenbrainz by pressing CTRL^C. Once everything is running, visit your new site in a browser!

```
http://localhost
```

Now, you are all set to begin making changes and seeing them in real-time inside of your development environment. If you make changes to python code, the server will be automatically restarted. If you make changes to javascript code it will be automatically compiled.

Look at the develop.sh documentation for more details.

1.6.7 Listenbrainz containers

A listenbrainz development environment contains a number of different containers running different services. We provide a small description of each container here:

- `db`: A PostgreSQL server that contains data about users
- `redis`: A redis server to store temporary server data
- `timescale`: A PostgreSQL server with the TimescaleDB extension that stores users listens
- `rabbitmq`: Used for passing listens between different services
- `web`: This is the main ListenBrainz server
- `api_compat`: A Last.fm-compatible API server
- `follow_server`: A helper server used for the user-following component of ListenBrainz
- `static_builder`: A helper service to build Javascript/Typescript and CSS assets if they are changed
- `type_checker`: A helper service to type-check Typescript assets

Note: If you add new python dependencies to ListenBrainz by adding them to `requirements.txt` you will have rebuild the web server. Use

```
./develop.sh build static_builder
```

to do this.

If you add new Javascript dependencies you will have to rebuild the `static_builder`:

```
./develop.sh build static_builder
```

1.6.8 Test your changes with unit tests

Unit tests are an important part of ListenBrainz. It helps make it easier for developers to test changes and help prevent easily avoidable mistakes later on. Before committing new code or making a pull request, run the unit tests on your code.

```
./test.sh
```

This builds and runs the containers needed for the tests. This script configures test-specific data volumes so that test data is isolated from your development data.

To run tests faster, you can use some options to start up the test infrastructure once so that subsequent running of the tests is faster:

```
./test.sh -u # build unit test containers, start up and initialise the database
./test.sh    # run tests, do this as often as you need to
./test.sh -s # stop test containers, but don't remove them
./test.sh -d # stop and remove all test containers
```

If you made any changes to the frontend, you can run the tests for frontend using

```
./test.sh fe
```

You can also make use of the following frontend testing options for efficient testing.

```
./test.sh fe             run frontend tests
./test.sh fe -u          run frontend tests, update snapshots
./test.sh fe -b          build frontend test containers
./test.sh fe -t          run type-checker
```

Also, run the **integration tests** for ListenBrainz.

```
./test.sh int
```

When the tests complete, you will see if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

1.6.9 Lint your code

ListenBrainz uses ESLint to lint the frontend codebase, please make sure you lint all new frontend code using

```
./lint.sh
```

This command should list all issues with the code you've modified. Make sure to fix all errors.

We have a *FAQ page* for questions that come up often. Please take a look if you have any issues.

1.7 Set up ListenBrainz Spark development environment

There are two distinct part of the ListenBrainz development environment:

1. listenbrainz – the actual webserver components of ListenBrainz
2. listenbrainz_spark – the spark environment used for features that involve data processing (stats, recommendations etc.)

If you're just working on adding a feature to the ListenBrainz webserver, you **do not** need to set up the Spark development environment. However, if you're looking to add a new stat or improve our fledgling recommender system, you'll need both the webserver and the spark development environment.

This guide should explain how to develop and test new features for ListenBrainz that use Spark.

1.7.1 Set up the webserver

The spark environment is dependent on the webserver. Follow the steps in the *guide to set up the webserver environment*.

Create listenbrainz_spark/config.py

The spark environment needs a config.py in the listenbrainz_spark/ dir. Create it by copying from the sample config file.

```
cp listenbrainz_spark/config.py.sample listenbrainz_spark/config.py
```

1.7.2 Initialize ListenBrainz Spark containers

Run the following command to build the spark containers.

```
./develop.sh spark build
```

The first time you build the containers, you also need to format the `namenode` container.

```
./develop.sh spark format
```

Your development environment is now ready. Now, let's actually see ListenBrainz Spark in action!

1.7.3 Bring containers up

Start the ListenBrainz Spark containers by executing `develop.sh spark up`.

```
./develop.sh spark up
```

1.7.4 Import data into the spark environment

We provide small data dumps that are helpful for working with real ListenBrainz data. Download and import a data dump into your spark environment using the following commands.

```
./develop.sh spark run request_consumer python spark_manage.py upload_listens -i
```

Now, you are all set to begin making changes and seeing them in real-time inside of your development environment!

Once you are done with your work, shut down the containers using the following command.

```
./develop.sh spark down
```

Note: You'll need to run `./develop.sh spark down` every time you restart your environment, otherwise hadoop errors out.

1.7.5 Working with request_consumer

The ListenBrainz webserver and spark cluster interact with each other via the request consumer. For a more detailed guide on working with the request consumer, read this *document*.

1.7.6 Test your changes with unit tests

Unit tests are an important part of ListenBrainz Spark. It helps make it easier for developers to test changes and help prevent easily avoidable mistakes later on. Before committing new code or making a pull request, run the unit tests on your code.

```
./test.sh spark
```

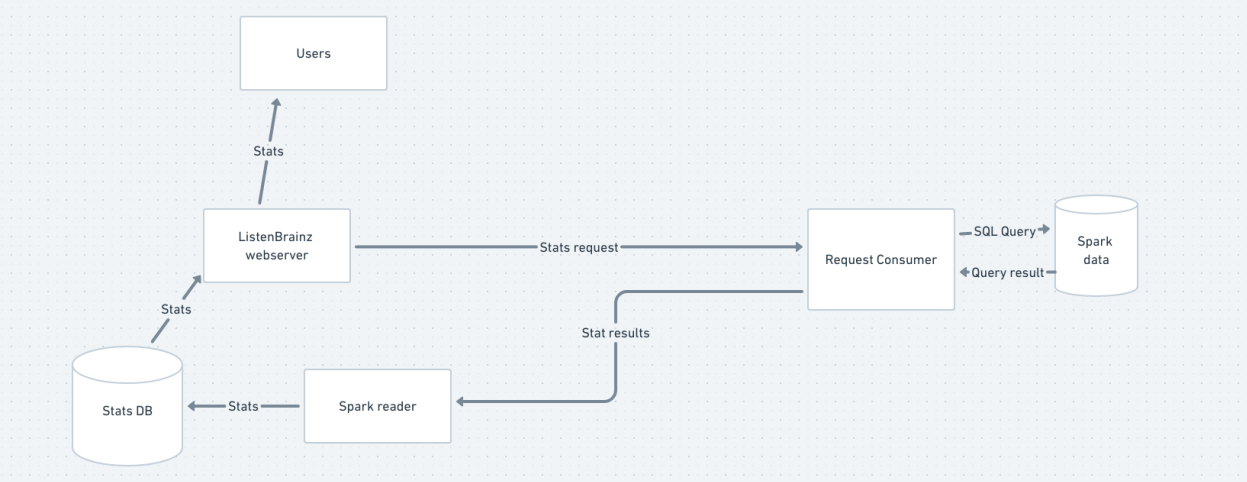
This builds and runs the containers needed for the tests. This script configures test-specific data volumes so that test data is isolated from your development data.

When the tests complete, you will see if your changes are valid or not. These tests are a helpful way to validate new changes without a lot of work.

Refer the *FAQs* to resolve the common errors that may arise when setting up the development environment.

1.8 ListenBrainz Spark Architecture

In order to actually build features that use Spark, it is important to understand how the ListenBrainz webserver and the Spark environment interact.



The ListenBrainz webserver and Spark cluster are completely seperate entities, only connected by RabbitMQ. This document explains how they interact with each other, taking the example of a stat.

The ListenBrainz environment sends a request to the *request_consumer* script via RabbitMQ. The request consumer, which is connected to Spark, takes the request and uses Spark to compute an appropriate response (or many responses). The request consumer then sends these responses via RabbitMQ to the *spark_reader* script, which runs alongside the webserver. The spark reader then takes the responses, and in the case of a stat, writes them to the ListenBrainz PostgreSQL database. Now that the stat has been updated in the database, users can view them on listenbrainz.org or via the API.

1.8.1 Developing request_consumer

Start the webserver

```
./develop.sh up
```

Start the spark containers

Follow the *instructions* to set up a Spark environment and import a small incremental dump so that you have some data.

Start the spark reader

We do not start the spark reader by default with develop.sh right now, because it isn't much use to non-spark developers. However, you can start it yourself.

```
./develop.sh run web python -m listenbrainz.spark.spark_reader
```

Now, you have everything needed to work with Spark. You can trigger a request like this

```
./develop.sh manage spark request_user_stats
```

1.9 Last.FM Compatible API for ListenBrainz

There are two versions of the Last.FM API used by clients to submit data to Last.FM.

1. The latest Last.FM API
2. The AudioScrobbler API v1.2

ListenBrainz can understand requests sent to both these APIs and use their data to import listens submitted by clients like VLC and Spotify. Existing Last.FM clients can be pointed to the ListenBrainz proxy URL and they should submit listens to ListenBrainz instead of Last.FM.

Note: This information is also present on the ListenBrainz website.

1.9.1 AudioScrobbler API v1.2

Clients supporting the old version of the AudioScrobbler API (such as VLC and Spotify) can be configured to work with ListenBrainz by making the client point to `http://proxy.listenbrainz.org` and using your MusicBrainz ID as username and the LB Authorization Token as password.

If the software you are using doesn't support changing where the client submits info (like Spotify), you can edit your `/etc/hosts` file as follows:

```
138.201.169.196    post.audioscrobbler.com
138.201.169.196    post2.audioscrobbler.com
```

1.9.2 Last.FM API

These instructions are for setting up usage of the Last.FM API for Audacious client on Ubuntu. These steps can be modified for other clients as well.

For development

1. Install dependencies from here, then clone the repo and install audacious.
2. Before installing audacious-plugins, edit the file *audacious-plugins/src/scrobbler2/scrobbler.h* to update the following setting on line L28. This is required only because the local server does not have https support.:

```
`SCROBBLER_URL` to "http://ws.audioscrobbler.com/2.0/".
```

3. Compile and install the plugins from the instructions given here.
4. Edit the `/etc/hosts` file and add the following entry:

```
127.0.0.1 ws.audioscrobbler.com
```

5. Flush dns and restart network manager using:

```
$ sudo /etc/init.d/dns-clean start
$ sudo /etc/init.d/networking restart
```

6. Register an application on MusicBrainz with the following Callback URL `http://<HOSTURL>/login/musicbrainz/post` and update the received MusicBrainz Client ID and Client Secret in config.py of ListenBrainz. `HOSTURL` should be as per the settings of the server. Example: `localhost`
7. In Audacious, go to File > Settings > Plugins > Scrobbler2.0 and enable it. Now open its settings and then authenticate.
8. **When you get a URL from your application which look like this `http://last.fm/api/auth/?api_key=as3..234`**

    - If you are running a local server, then `HOSTURL` should be similar to "localhost:8080".
    - If you are not running the server, then `HOSTURL` should be "api.listenbrainz.org".

For users

1. Repeat all the above steps, except for steps 2 and 6.
2. For Step 8, choose the 2nd option for `HOSTURL`.

1.10 FAQs

What to do if getting an error while running ``./develop.sh build`` command, 'ERROR: Couldn't connect to Docker daemon at http+docker://localhost - is it running?'?

- You need to add the user to the docker group by entering the following command in the terminal:

```
sudo usermod -aG docker $USER
```

- After this command, restart the computer and then again run `./develop.sh build`.

How to resolve 'datanode is running as process 1. Stop it first' or 'namenode is running as process 1. Stop it first'?

- You need to shut down the previous containers before bringing them up again. Run the following command to shut down the containers:

```
./develop.sh spark down
```

- When the containers shut down, run `./develop.sh spark up` again.

How to resolve 'sqlalchemy.exc.OperationalError: (psycopg2.OperationalError) FATAL: role "listenbrainz" does not exist' on running './test.sh'?

- You need to shut down the previous test containers before bringing them up again. Run the following command to shut down the test containers:

```
./test.sh -d
```

- When the containers shut down, run `./test.sh` to run the tests again.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

/1

GET /1/latest-import, 5
GET /1/stats/user/(user_name)/artist-map, 9
GET /1/stats/user/(user_name)/artists, 12
GET /1/stats/user/(user_name)/daily-activity, 7
GET /1/stats/user/(user_name)/listening-activity, 6
GET /1/stats/user/(user_name)/recordings, 8
GET /1/stats/user/(user_name)/releases, 10
GET /1/status/get-dump-info, 13
GET /1/user/(user_name)/listen-count, 4
GET /1/user/(user_name)/listens, 5
GET /1/user/(user_name)/playing-now, 4
GET /1/users/(user_list)/recent-listens, 4
GET /1/validate-token, 3
POST /1/latest-import, 5
POST /1/submit-listens, 3

INDEX

D

M