Kenzan Million Song Library Project

Last Updated: June 9, 2016

Kenzan Million Song Library Project

One does not simply build a rich, dynamic library with a million songs. It takes an array of microservices crafted with open technologies, a database that won't hold back performance, and an architecture that's ready to venture out to the cloud when you are.

With this demonstration project, you can deploy a Million Song Library and see what it's like to launch a microservices-based application of your own. If you have questions about the demo, feel free to drop us a line at support_msl@kenzan.com.

1. Million Song Library Overview

At Kenzan, we build Platform as a Service (PaaS) solutions that enable our customers to deliver secure, database-driven applications anywhere they want, anytime they need. We're a big believer in the open source software (OSS) tools created by Netflix, along with other cool things like microservices, NoSQL databases, and scalable cloud infrastructure. These technologies, and more, help power us through all phases of delivering a customized PaaS solution, from architecture and application development to testing and DevOps.

So why did we start the Million Song Library (MSL) project? There are a couple of reasons. First, we wanted to build a model to demonstrate the capability and flexibility offered by some of the Netflix OSS components when paired with a Cassandra database. Second, we thought it would be pretty neat to have a ready-to-go foundation on top of which database-driven applications can be rapidly developed, tested, and deployed to the cloud.

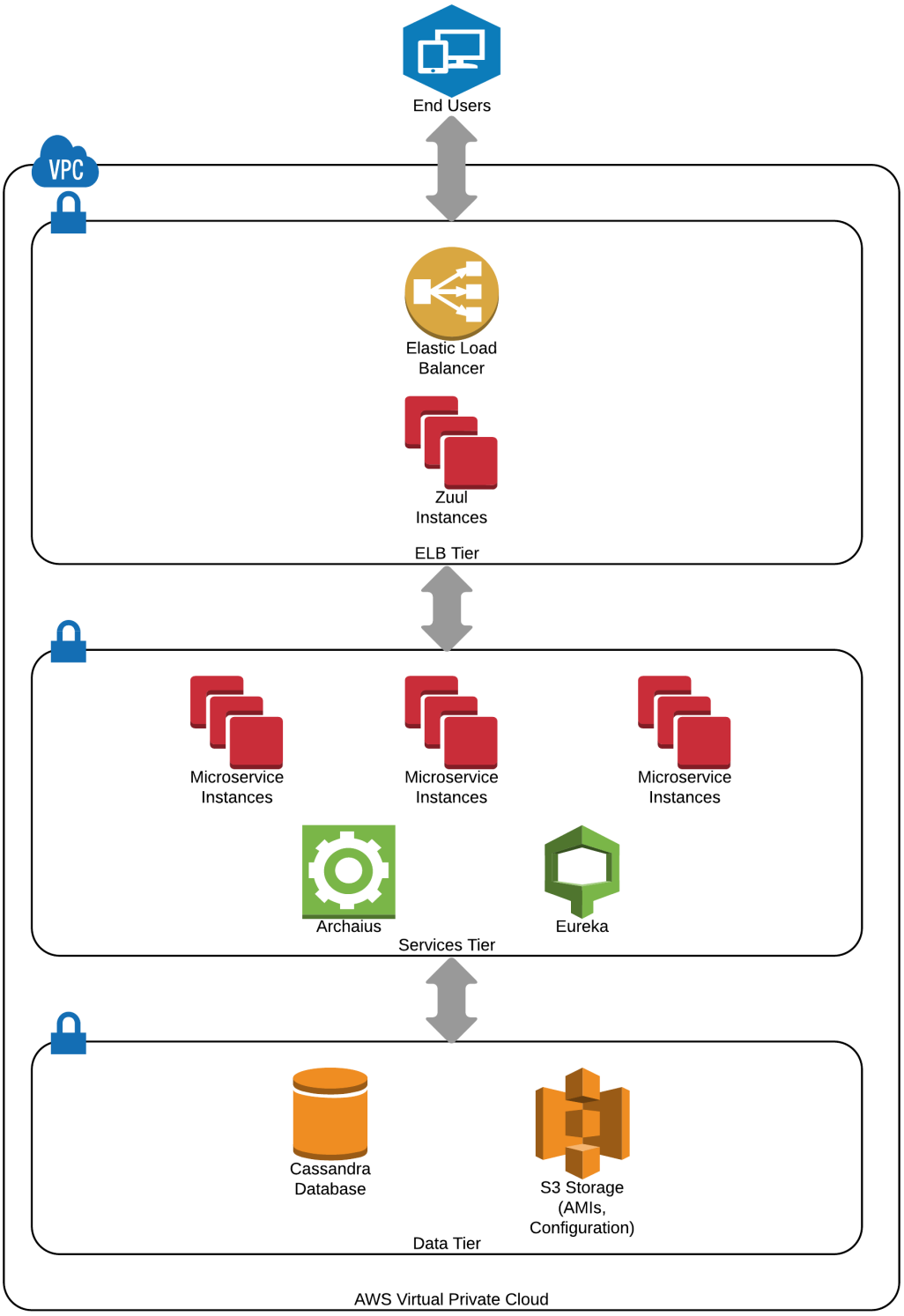
And here's the best part — we're taking the whole thing open source. Because at Kenzan, we want everyone to have the power to build a dynamic library with a million songs. Or a million funny videos. Or a million must-have gadgets. Or a million loyal customers. Or whatever it is that you're a big beliver in.

Need a companion on your journey to the cloud? We're always up for an adventure — just knock on our door at Kenzan.com.

1.1. Key Technologies

The diagram below shows some of the key techologies we use in creating a database-driven, microservices-based Web application like the Million Song Library project. Read on for information about each of the technologies.

Million Song Library Key Technologies



1.2. Microservices Architecture

In a microservices architecture, large applications are made up of multiple smaller services that can be independently deployed using automated methods. Each microservice is self-contained and provides a targeted piece of functionality. Microservices are discoverable, so clients and servers can automatically find one another. And they communicate using standard, lightweight protocols such as REST APIs.

At Kenzan, we have a lot of experience working with microservices. We've found that adopting a

microservices architecture results in Web applications that are more modular and easier to manage compared to traditional, monolithic applications. Because microservices can be scaled independently, distributing them across servers, you have the flexibility to quickly meet changing demands. Usage of one microservice is going way up? Just add more instances of it. Need to add another cabability to your application? It's as simple as updating a microservice or creating a new one. You don't have to rebuild and deploy the entire application just to push out a change.

While microservices bring a lot of advantages, building a microservices-based solution from the ground up could present a challenge. That's why the Million Song Library leverages a number of open source software (OSS) libraries from Netflix. The Netflix OSS components provide support for building applications using a suite of microservices, greatly easing the transition to a microservices-based architecture. What's more, they've been tried and tested in one of the most demanding cloud environments imaginable.

Here are some of the Netflix OSS tools that make the Million Song Library possible.

Karyon

Cloud infrastructure blueprint. Constructing a complex architecture requires a solid foundation. Karyon provides a scaffold for building cloud-based services, and it offers hooks for other Netflix OSS tools. When an application is based on the Karyon blueprint, it gains the ability to leverage a common core of functionality. A Karyon application can initialize itself and its dependencies, make itself available for discovery, and even let the rest of the platform know whether it's healthy or not.

Zuul

Intelligent request routing. You might remember Zuul as the gatekeeper from the movie Ghostbusters. In a microservices architecture, Zuul watches the front gate of your application and ensures that only the correct actors can access the correct services. Zuul uses pre-defined filters to allow or reject calls made to the application, and it dynamically routes requests to different application instances in response to changing demand. Zuul can even track statistics and perform stress testing to help make sure the system is prepared for times when usage spikes.

Eureka

Private load balancing. Using cloud-based load balancers in a microservices architecture can quickly become cumbersome. This is due to the vast number of microservices and possible routes between them — each of which could potentially need its own load balancer. What's more, while cloud-based load balancers are great for distributing loads for edge services that offer public access, they can't help with middle-tier services that reside within a virtual private cloud (VPC) — services that you don't want the outside world to acccess.

That's where Eureka comes in. Eureka is a service registry that manages all active services and ensures that loads are properly distributed. Each time an instance of a microservice starts up, it lets Eureka know that it's available. And if it ever stops sending out a heartbeat, Eureka quickly drops it from the list of available instances. Working in conjunction with Zuul, Eureka helps clients find active servers and gives them the information they need to communicate with one another, all within the VPC. And because server information is cached on the client, the system keeps working even if Eureka becomes unavailable.

Archaius

Contextual properties management. Managing configuration properties for a large number of microservices could be problematic. Archaius solves this by storing properties in a central library where they can be easily maintained. At runtime, Archaius checks the software version and environmental values for the application — that is, the context it's running in. Archaius determines the properties needed for that particular context, and then delivers the appropriate properties to the application over the network.

1.3. High-Performance Database

Today's modern Web is all about data — storing it, finding it, retrieving it. And all while users are impatiently tapping their screens. As data sets grow larger, having a reliable and blazingly-fast database is critical for delivering a sucessful database-driven application. That's why, for the Million Song Library, we went with a Cassandra database, along with a DataStax Java driver that provides an interface between the database and the application code.

Cassandra is a NoSQL database that offers much lower latency and higher scalablility than relational databases. Compared to a traditional SQL database, Cassandra can accept data at higher rates and volumes. Cassandra's node-based architecture makes it more resilient because there is no single point of failure. And it's easy to scale out a Cassandra database by adding more nodes (with no need to take the system down first).

Achieving such a high level of performance does require a different way of thinking. In a Cassandra database, transactions need to be simple and can't be nested. And data may be written in multiple locations rather than normalized to a single table. But predictable application performance and constant uptime are well worth the change.

1.4. Cloud-Based Infrastructure

At Kenzan, we build a lot of PaaS solutions on Amazon Web Services (AWS). AWS provides the infrastructure and core services needed to deploy a complex, database-driven application in the cloud:

- *Elastic Compute Cloud (EC2)* lets you create or decommision virtual server instances at will, so you always have just the amount of computing power you need.
- *Auto-Scaling Groups (ASGs)* can fire up instances automatically in response to increased load or other conditions.
- *Amazon Machine Images (AMIs)* enable a templated pipeline for successful and repeatable deployment of instances.
- *Elastic Load Balancing (ELB)* distributes traffic from the public web among the available instances so that one doesn't get overwhelmed while others sit idle.
- *Simple Storage Service (S3)* provides persistant storage for AMIs, configurations, or image assets.

AWS is hosted on multiple, independent regions and availability zones that provide resiliency — even if an AWS region goes down, the application is still available. And because the middle-tier and data store reside within the Virtual Private Cloud, the entire platform is secure and

protected.

The Million Song Library architecture isn't limited to running on AWS. It can be deployed locally. Or feel free to deploy your application to another cloud infrastructure, such as Microsoft Azure or Google Cloud Platform — whatever makes the most sense for you.

1.5. About the MSL Demo

To take a deeper dive into the architechure of the Million Song Library project and discover more about the technologies used to build it, just keep reading — more awaits below.

When you're ready to try out the Million Song Library demo on a Mac, Linux, or Windows computer, just follow the setup instructions, and you'll have your own microservices-based Web application up and running in about an hour:

- Mac OS X Automated Setup
- Linux Automated Setup
- Windows Automated Setup
- Manual Setup

If you have an AWS account, you can can also deploy the Million Song Library to an EC2 instance. See Deploying to AWS for step-by-step instructions.

2. Code and Documentation

The Million Song Library project is open source and hosted on GitHub. The code is written using common coding standards. And it's fully documented with the help of automated documentation tools.

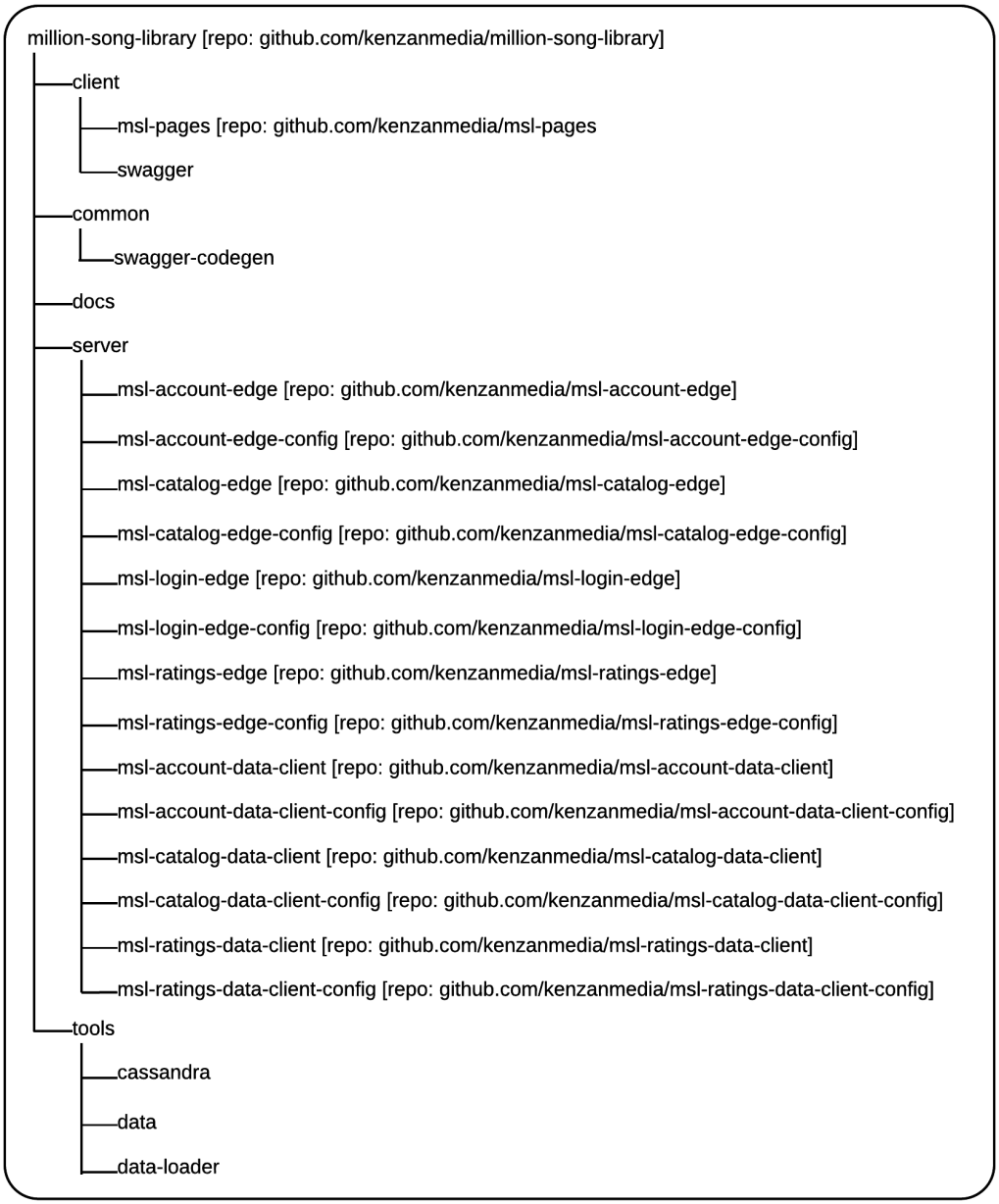
2.1. GitHub Repositories

You can find all of the source code for the Million Song Library project in the master repository on GitHub.

http://github.com/kenzanmedia/million-song-library

The master repository includes multiple sub-repositories that have been added as Git submodules using the `git submodule add` command. There is one submodule for each service, configuration, and UI service. Dividing the project into submodules like this makes it easier to configure continuous integration (CI) jobs. It also provides a cleaner separation of responsibilities within the code. At the same time, having one master repository means that it's simple to retrieve the entire project all at once.

Million Song Library Repository Structure



2.2. Coding Standards

We think the Google coding standards and style guides are concise and useful, so we based our own style guide on them. We wrote all of the code in the Million Song Library project to conform to these standards. See the Kenzan Style Guide to learn more.

2.3. Documentation Tools

The Million Song Library project leverages a number of tools to generate API, code, and other documentation.

2.3.1. Swagger API Documentation and Code Generation

We use the open-source Swagger framework to create the API documentation right alongside the client and server code. Swagger provides more than just a way to document as you develop because the documentation actually drives code generation. Using the Swagger codegen tools, you can generate client libraries and server stubs from the same definition file used to generate the documentation. By unifying the generation of the API documentation with the application code, Swagger assures that they always stay in sync. And it enables publishing of the API documentation to both static and interactive formats.

Recently, Swagger was donated to the Open API Initiative and was renamed the OpenAPI Specification.

To use Swagger, we first specify the API methods and models in multiple YAML files and combine them as necessary:

- **definitions.yaml** - (One file) Declares all of the objects that will be passed into and out of the APIs.
- **paths.yaml** - (One file for each microservice) Declares the API methods for that microservice.
- **index.yaml** - (One file for each microservice) Uses `$ref` commands to combine certain other YAML files to completely define the API for that microservice.
- **index.yaml** - (One file) Combines all of the API methods and models into a single file that is used to generate documentation for all API methods in all microservices.

Next, to generate documentation and server-side code stubs, we use JavaScript (**resolve.js**) to resolve the `$ref` commands and output the final YAML files. The final YAML files are sent through Swagger (**swagger-client-cli**) to build the server-side code stubs and the HTML API documentation.

For an example of static documentation generated with Swagger, see this static HTML representation. For instructions on how to run the Swagger mock server and view the interactive documentation, see the “Swagger” section in the README for the msl-pages sub-repository.

2.3.2. Javadoc and ESDoc

To generate documentation for the server Java classes, we use the open-source Javadoc tool. Information about each class, including its parameters and return object or value, is added to the code by means of standard comments. HTML or PDF documentation can then be generated from the code comments using command line tools. A number of popular IDEs also offer tools for generating JavaDocs.

Similarly, we use ESDoc to create code documentation for the JavaScript functions and methods that are part of the UI service.

2.3.3. AsciiDoc Documentation

We use AsciiDoc to create supplemental documentation for the Million Song Library project

(including the document you're reading now).

AsciiDoc is a text-based markup language for writing documentation such as articles, books, and more. Like other lightweight markup languages, AsciiDoc has a simple syntax that is human readable. Using open source tools, text files written in AsciiDoc can be easily processed to create output in a variety of formats, including HTML and PDF.

3. Client/UI Architecture

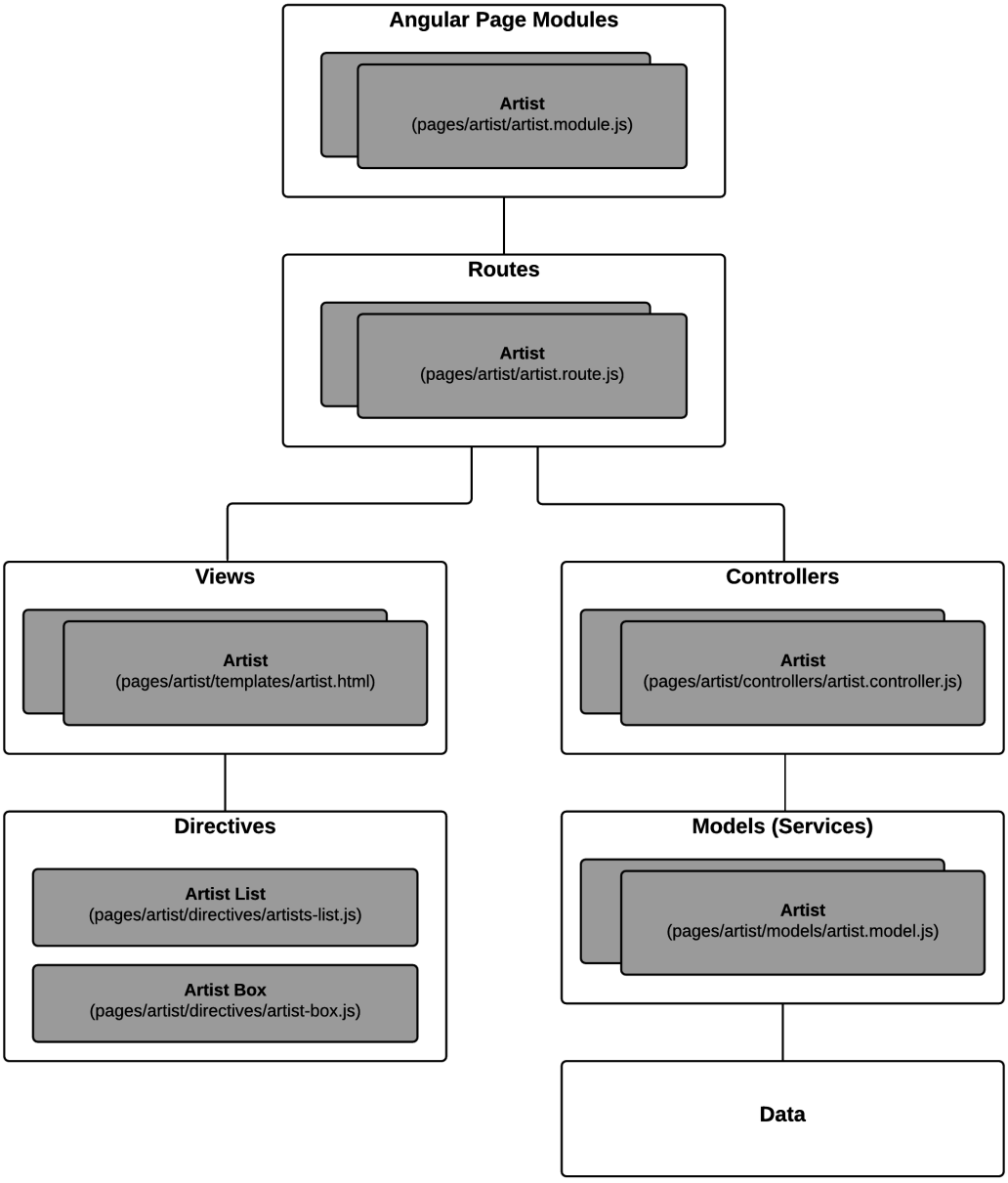
The Million Song Library client architecture is modular by design and takes advantage of modern Web application languages and frameworks.

3.1. AngularJS

The Million Song Library UI presents itself to the user as a single-page app built using AngularJS. AngularJS is an open-source JavaScript framework that extends HTML and enables the creation of dynamic Web applications. It provides a flexible toolset that works well with other libraries.

UI High-level Architecture

General Angular Architecture (Ex: Artist Page)



3.2. Sass

We use Sass (along with SCSS syntax) to help manage the complex CSS stylesheets that make the Million Song Library UI possible. Sass is a CSS pre-processor that extends the CSS language. It adds features that allow the use of variables, mixins, functions, and other techniques that make CSS much more maintainable, themable, and extendable.

3.3. Material Design

Underlying the CSS and Angular architecture of the Million Song Library UI is Google's Material Design, which provides an accessible look and feel that scales across browsers and devices. We use the Angular Material version of Material Design to leverage its built-in AngularJS components. Adopting Material Design allows for extensibility of the Million Song Library UI and makes it easy

for others to add their own features.

3.4. KSS Styleguide Generation

KSS provides a specification and styleguide format that helps you produce CSS documentation as well as example HTML that shows your styles in action. The KSS documentation syntax is readable by humans, yet is structured enough to be automatically parsed by a machine. It's designed with CSS preprocessors (such as Sass or Less) in mind, and is flexible enough to accommodate a multitude of CSS frameworks (such as YUI, Blueprint, or 960).

3.5. ES2015

The Million Song Library UI uses new features (like `await` and `async`) available in the latest version of the JavaScript programming language, ES2015. We use Babel to transpile ES2015 script into standard JavaScript syntax that today's browsers can understand.

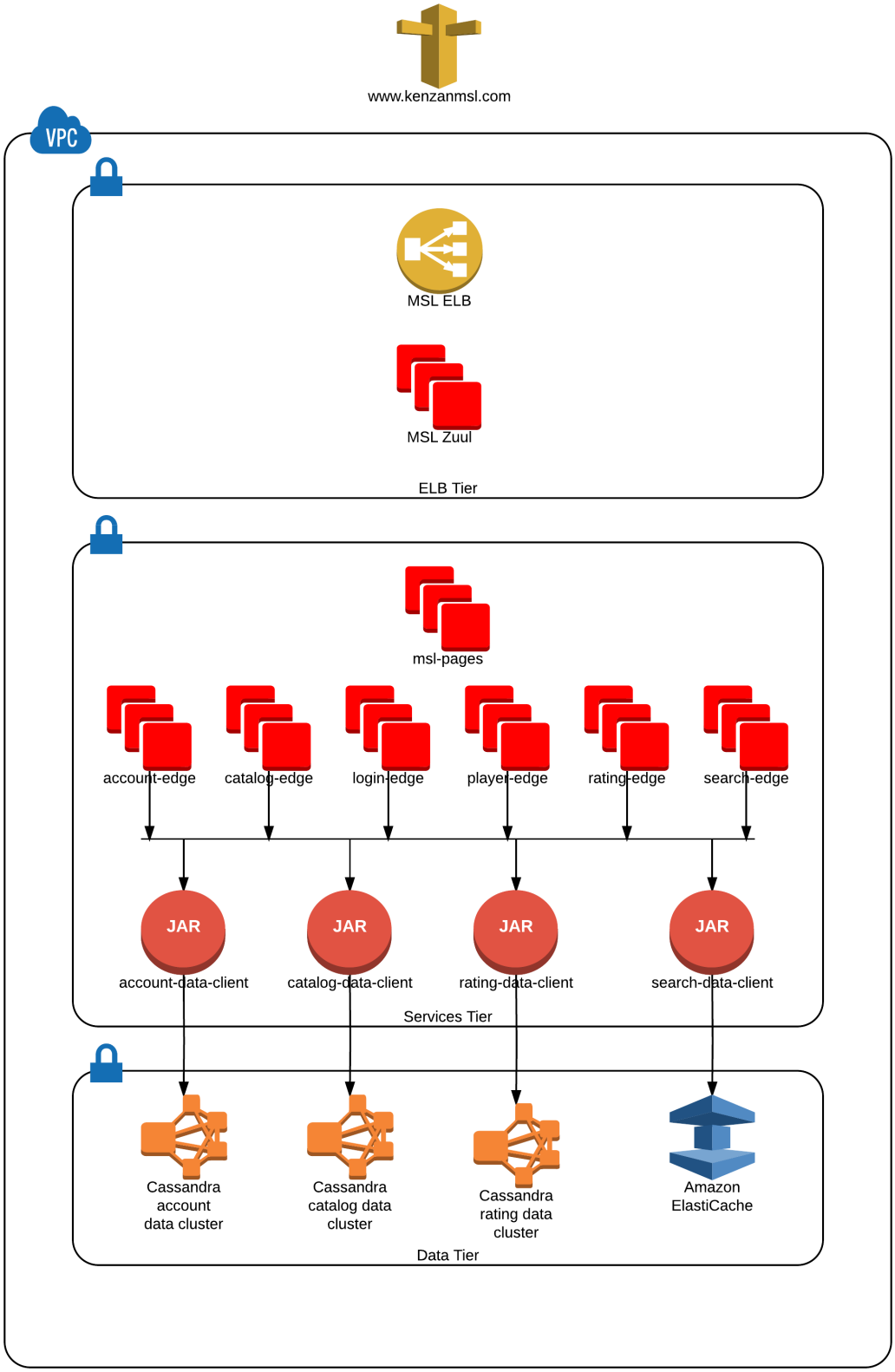
3.6. Webpack

Webpack is a module bundler. We use it to produce static assets from modules that have dependencies. The main `index.html` file is built using the HTML Webpack Plugin.

4. Server Architecture

Instead of a traditional edge/middle architecture, the Million Song Library project uses a simplified edge/data client architecture.

High-level Server Architecture



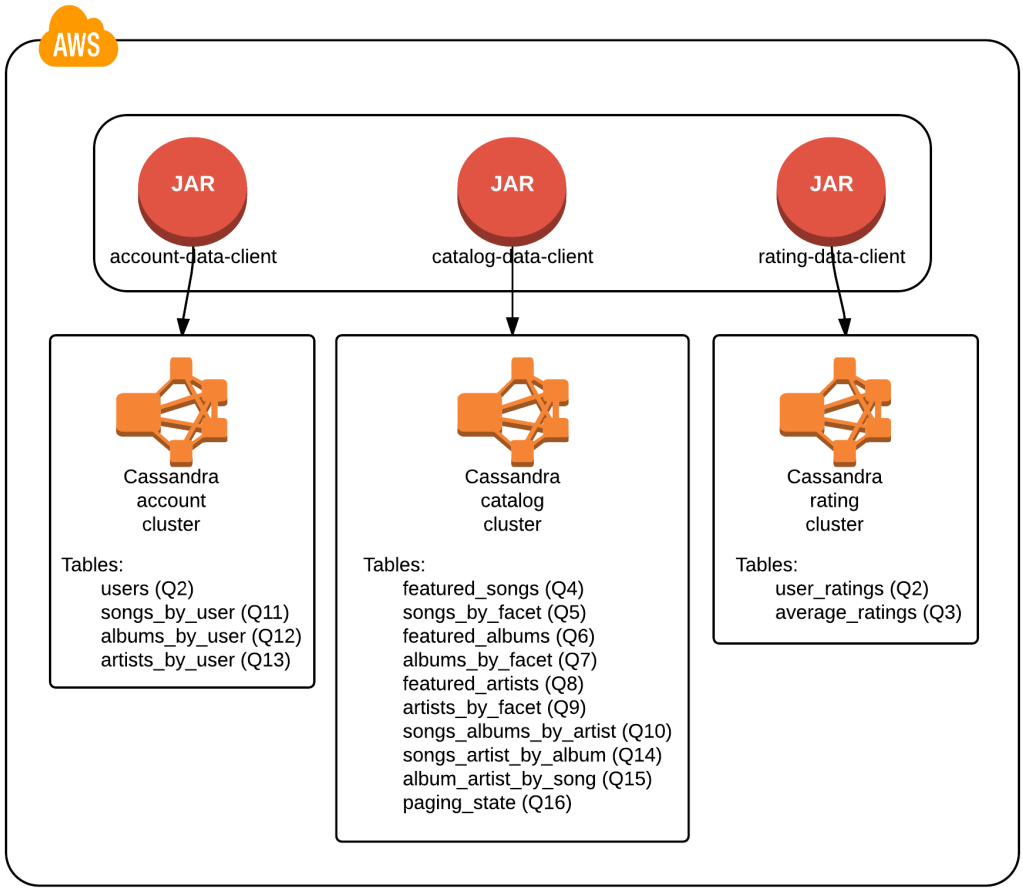
4.1. Data Client Layer

The data clients are JARs, each one containing the methods and data transfer objects (DTOs) needed to access all of the tables in a Cassandra cluster.

To enhance scalability and configuration flexibility, the Cassandra tables are split into three independent clusters: **account**, **catalog**, and **rating**. Each of these clusters has a data client JAR

dedicated to accessing it: **account-data-client**, **catalog-data-client**, and **rating-data-client**, respectively. This means that a microservice that needs to access Cassandra data will include one or more of the data client JARs.

Data Clients and Cassandra Clusters



4.1.1. Data Client DTOs and Methods

The tables below list the DTOs and methods available for each database table accessed by **account-data-client**, **catalog-data-client**, and **rating-data-client**.

For details about all of the available DTOs and methods, see the Javadoc documentation for each data client.

Table 1. Account Data Client DTOs

Table	DTO
users	UserDto
songs_by_user	SongsByUserDto
albums_by_user	AlbumsByUserDto
artists_by_user	ArtistsByUserDto

Table 2. Account Data Client Methods

Table	Method
users	Observable<Void> addOrUpdateUser(UserDto)
	Observable<UserDto> getUser(UUID userUuid)
	Observable<Void> deleteUser(UUID userUuid)
songs_by_user	Observable<Void> addOrUpdateSongsByUser(SongsByUserDto)
	Observable<SongsByUserDto> getSongsByUser(UUID userId, Timestamp favoritesTimestamp, UUID songUuid)
	Observable<ResultSet> getSongsByUser(UUID userId, Timestamp favoritesTimestamp, Optional<Integer> limit)
	Observable<ResultSet> getSongsByUser(UUID userId, Optional<Integer> limit)
	Observable<Result<SongsByUserDto> map(Observable<ResultSet>)
	Observable<Void> deleteSongsByUser(UUID userId, Timestamp favoritesTimestamp, UUID songUuid)
albums_by_user	Observable<Void> addOrUpdateAlbumsByUser(AlbumsByUserDto)
	Observable<AlbumsByUserDto> getAlbumsByUser(UUID userId, Timestamp favoritesTimestamp, UUID albumUuid)
	Observable<ResultSet> getAlbumsByUser(UUID userId, Timestamp favoritesTimestamp, Optional<Integer> limit)
	Observable<ResultSet> getAlbumsByUser(UUID userId, Optional<Integer> limit)
	Observable<Result<AlbumsByUserDto> map(Observable<ResultSet>)
	Observable<Void> deleteAlbumsByUser(UUID userId, Timestamp favoritesTimestamp, UUID albumUuid)
artists_by_user	Observable<Void> addOrUpdateArtistsByUser(ArtistsByUserDto)
	Observable<ArtistsByUserDto> getArtistsByUser(UUID userId, Timestamp favoritesTimestamp, UUID artistUuid)
	Observable<ResultSet> getArtistsByUser(UUID userId, Timestamp favoritesTimestamp, Optional<Integer> limit)
	Observable<ResultSet> getArtistsByUser(UUID userId, Optional<Integer> limit)
	Observable<Result<ArtistsByUserDto> map(Observable<ResultSet>)
	Observable<Void> deleteArtistsByUser(UUID userId, Timestamp favoritesTimestamp, UUID artistUuid)

Table 3. Catalog Data Client DTOs

Table	DTO
featured_songs	FeaturedSongsDto
songs_by_facet	SongsByFacetDto
featured_albums	FeaturedAlbumsDto
albums_by_facet	AlbumsByFacetDto
featured_artists	FeaturedArtistsDto
artists_by_facet	ArtistsByFacetDto
songs_albums_by_artist	SongsAlbumsByArtistDto
songs_artist_by_album	SongsArtistByAlbumDto
album_artist_by_song	AlbumArtistBySongDto
paging_state	PagingStateDto

Table 4. Catalog Data Client Methods

Table	Method
featured_songs	Observable<ResultSet> getFeaturedSongs(Optional<Integer> limit)
	Observable<Result<FeaturedSongsDto> map(Observable<ResultSet>)
songs_by_facet	Observable<ResultSet> getSongsByFacet(String facetName, Optional<Integer> limit)
	Observable<Result<SongsByFacetDto> map(Observable<ResultSet>)
featured_albums	Observable<ResultSet> getFeaturedAlbums(Optional<Integer> limit)
	Observable<Result<FeaturedAlbumsDto> map(Observable<ResultSet>)
albums_by_facet	Observable<ResultSet> getAlbumsByFacet(String facetName, Optional<Integer> limit)
	Observable<Result<AlbumsByFacetDto> map(Observable<ResultSet>)
featured_artists	Observable<ResultSet> getFeaturedArtists(Optional<Integer> limit)
	Observable<Result<FeaturedArtistsDto> map(Observable<ResultSet>)
artists_by_facet	Observable<ResultSet> getArtistsByFacet(String facetName, Optional<Integer> limit)
	Observable<Result<ArtistsByFacetDto> map(Observable<ResultSet>)
songs_albums_by_artist	Observable<ResultSet> getSongsAlbumsByArtist(UUID artistUuid, Optional<Integer> limit)
	Observable<Result<SongsAlbumsByArtistDto> map(Observable<ResultSet>)

Table	Method
songs_artist_by_album	Observable<ResultSet> getSongsArtistByAlbum(UUID albumUuid, Optional<Integer> limit)
	Observable<Result<SongsArtistByAlbumDto> map(Observable<ResultSet>)
album_artist_by_song	Observable<ResultSet> getAlbumArtistBySong(UUID songUuid, Optional<Integer> limit)
	Observable<Result<AlbumArtistBySongDto> map(Observable<ResultSet>)
paging_state	Observable<Void> addOrUpdatePagingState(PagingStateDto)
	Observable<PagingStateDto> getPagingState(UUID pagingStateUuid)
	Observable<Void> deletePagingState(UUID pagingStateUuid)

Table 5. Rating Data Client DTOs

Table	DTO
average_ratings	AverageRatingDto
user_ratings	UserRatingsDto

Table 6. Rating Data Client Details

Table	Method
average_ratings	Observable<Void> addOrUpdateAverageRating(AverageRatingDto)
	Observable<AverageRatingDto> getAverageRating(UUID contentId, String contentType)
	Observable<Void> deleteAverageRating(UUID contentId, String contentType)
user_ratings	Observable<Void> addOrUpdateUserRatings(UserRatingsDto)
	Observable<UserRatingsDto> getUserRatings(UUID userUuid, String contentType, UUID contentUuid)
	Observable<ResultSet> getUserRatings(UUID userUuid, String contentType, Optional<Integer> limit)
	Observable<ResultSet> getUserRatings(UUID userUuid, Optional<Integer> limit)
	Observable<Result<UserRatingsDto> map(Observable<ResultSet>)
	Observable<Void> deleteUserRatings(UUID userUuid, String contentType, UUID contentUuid)

4.2. APIs

To learn more about the Million Song Library server APIs, see the API documentation we generated

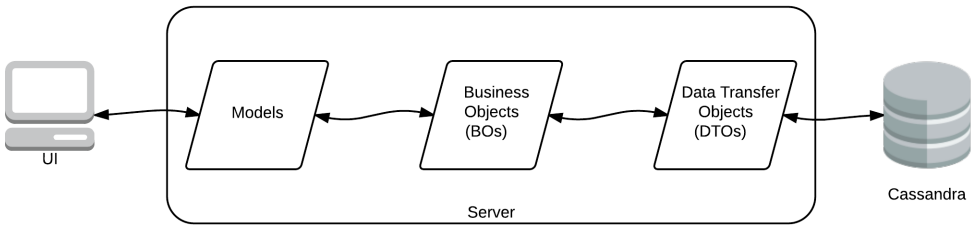
using Swagger. You can find links to all of the documentation in the README file on the Million Song Library GitHub page.

4.3. Data POJOs

The server has three different types of data POJOs (plain old Java objects):

- **Models** - Classes that represent the data payloads that are passed between client and server.
- **Data Transfer Objects (DTOs)** - Classes that represent the data persisted in Cassandra.
- **Business Objects (BOs)** - Classes that help to convert data between models and DTOs.

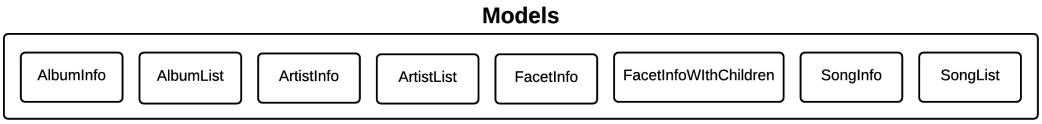
Data POJOs Interaction



4.3.1. Models

The model classes are generated by Swagger and represent the data payloads that are passed back and forth between client and server.

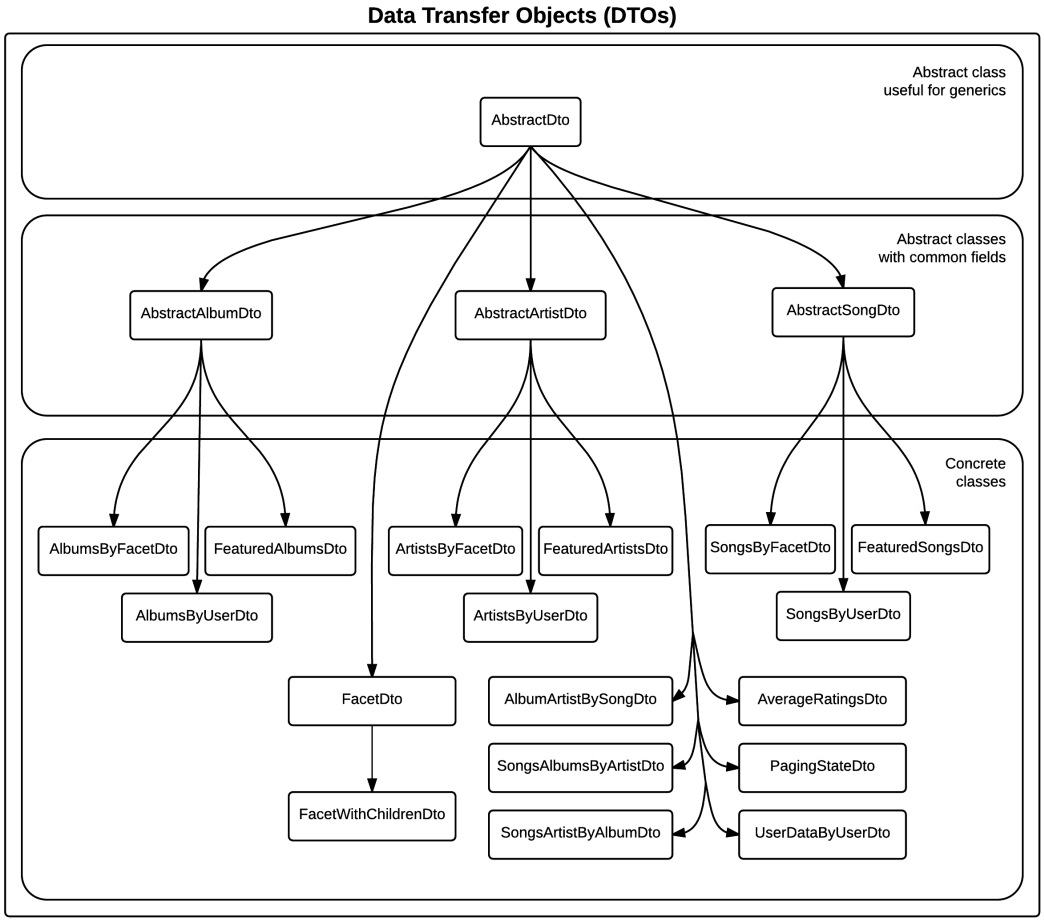
Model Classes



4.3.2. Data Transfer Objects (DTOs)

Data Transfer Objects represent the data as it is persisted in Cassadra. Simply, there is a one-to-one relationship between Cassandra tables and concrete DTO classes. In many cases this data is optimized for Cassandra, which means it needs to undergo some manipulation to convert it to a model.

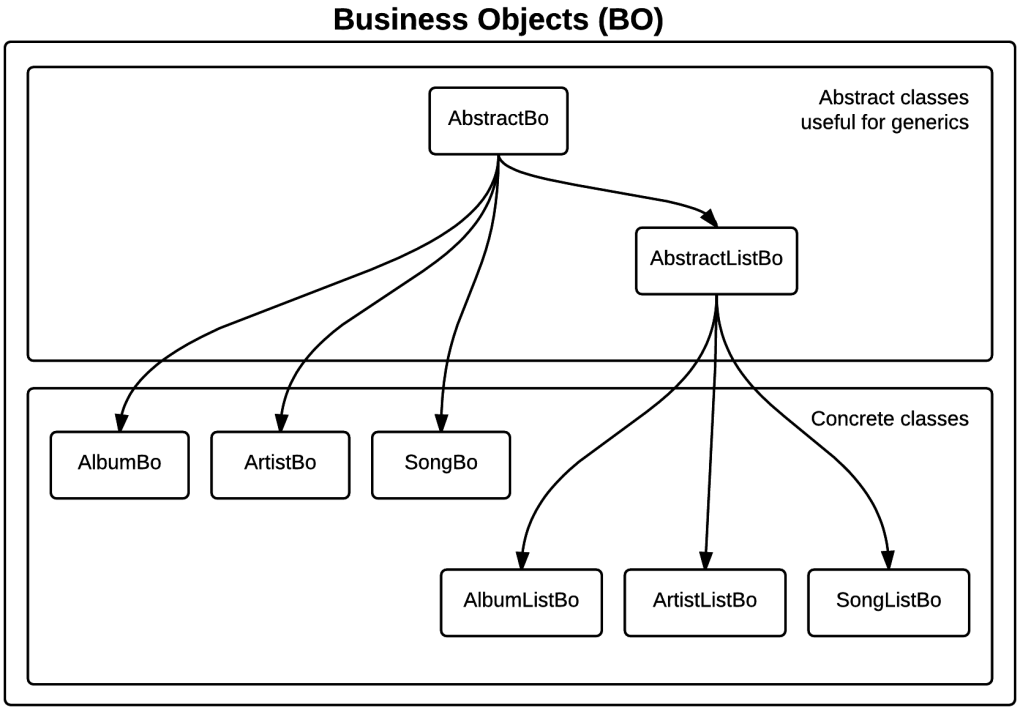
Data Transfer Object Classes



4.3.3. Business Objects (BOs)

Insider the server, business object classes represent the glue that holds together model classes and DTO classes. In many cases there is not a simple one-to-one connection between reading a DTO from Cassandra and sending that data as a model to the client. For example, when building an **AlbumInfo** model, the album's metadata, community, and user ratings need to be merged. The BO contains the merged representation of the data.

Business Object Classes



4.4. Languages and Frameworks

The back-end server is written in Java — Java Enterprise Edition 1.8, to be exact. Communication between client and server is performed using RESTful Web services over HTTP. The REST code stubs are generated by Swagger and use JAX-RS annotations. Java's Jersey library is used as the implementation of the JAX-RS API. Finally, JUnit is used as our unit testing framework.

As described in the Microservices Architecture section above, portions of the Netflix OSS stack are used to provide cloud services. At its foundation, the server is based on Karyon, which enables the use of other Netflix OSS tools. For example, Archaius is used as the configuration manager, allowing a service to dynamically react to changes in configuration parameters.

All data is persisted using Apache's Cassandra. Cassandra is a scalable, highly-available NoSQL database. The DataStax Java driver and row→POJO mapper provide an interface between the Java code and the Cassandra database.

The table below summarizes the languages and frameworks used in the Million Song Library server.

Table 7. Server Languages and Frameworks

Use	Name	Vendor	Link
Language	Java EE 1.8	Oracle	link
API Documention/Code Generator	Swagger	open-souce	link
ReST API	JAX-RS	Oracle	link
ReST Implementation	Jersey	Oracle	link

Use	Name	Vendor	Link
Unit Testing	JUnit	JUnit	link
Cloud Infrastructure	Karyon	Netflix OSS	link
Configuration Management	Archaius	Netflix OSS	link
Database	Cassandra	Apache	link
Database Driver	DataStax	DataStax	link
Database Row→POJO Mapper	Datastax	DataStax	link
Reactive Code Library	RxJava	Netflix	link

4.5. Album Cover Artwork

Album cover artwork is retrieved from MusicBrainz, "an open music encyclopedia that collects music metadata and makes it available to the public", and Cover Art Archive, "whose goal is to make cover art images available to everyone on the Internet in an organised and convenient way". The Million Song Library data set contains an artist's MusicBrainz ID (**artist_mbid**). As part of the data import process, this identifier is used to retrieve the links to artwork using the MusicBrainz and Cover Art Archive APIs.

All interactions with MusicBrainz and the Cover Art Archive are performed during the data import process — no queries to either of these sites occur when the Million Song Library server is running.

4.5.1. MusicBrainz Access

We use the **musicbrainzws2-java** Java library to access the MusicBrainz API.

4.5.2. Parsing and Using MusicBrainz Data

Multiple releases (that is, albums) can be returned for the requested artist. The data import code reviews the available releases/albums for the artist and selects album art using the following algorithm:

1. Consider only those releases/albums whose title matches (not case-sensitive) the name of the album from the Million Song Library data.
2. Consider only those release/albums for which `cover-art-archive/front = true`.
3. Give preference to multiple possible candidates based on packaging in this order:
    a. Jewel Case (indicates a CD)
    b. Cardboard/Paper Sleeve (idicates an LP)
    c. Cassette Case (indicates cassette tape)
    d. Other packaging

If a particular piece of art is *not* available, the image link field in the database is left blank/null. If artwork is available, then the data import process retrieves the image URL through the Cover Art Archive REST API (using **CoverArtArchiveClient**) and writes it to the database.

4.5.3. Cover Art Archive Access

The **CoverArtArchiveClient** class from the Cover Art Archive API allows us to retrieve the image URL using the release MBID (MusicBrainz Identifier).

4.5.4. Parsing and Using CoverArtArchive Data

As you can see in the code example above, multiple images can be returned for the requested release/album. The data import code will review the available images for the release/album and select the image URL using the following algorithm:

1. Consider only those images for which `isFront() == true`.
2. Give preference to multiple possible URLs in this order:
    a. small thumbnail
    b. large thumbnail
    c. image (this is the high resolution image)

If, based on this algorithm, a particular piece of art is *not* available, the image link field in the database is left blank/null.

4.5.5. "Image Unavailable" Images

When preparing to send an artwork link in response to a request, if the image link URL from the database is blank/null, the server instead inserts a URL that points to an "artwork unavailable" image on the pages server. This URL is defined as a configuration parameter. Three "artwork unavailable" images (one each for album, artist, and song) are available for HTTP retrieval from the pages server.

5. Database Architecture

Due to the large amount of data to be persisted, and the need for quick reads and linear scalable performance, the Million Song Library project uses Cassandra as the back-end datastore, along with the Datastax Java driver as the interface.

5.1. Design

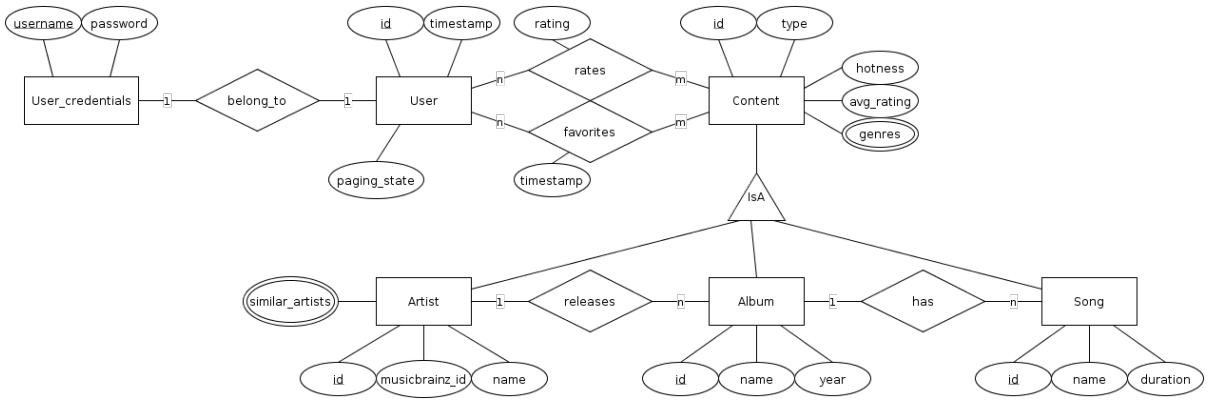
We used the Kashlev Data Modeler (KDM) to model the Cassandra schema.

5.1.1. Entity-Relationship Model

Initially, we performed a traditional data analysis — looking at what entities are required based on the use cases, and what the relationships are between those entities. The result is illustrated in the

entity-relationship model below which was created with KDM and uses Chen's notation.

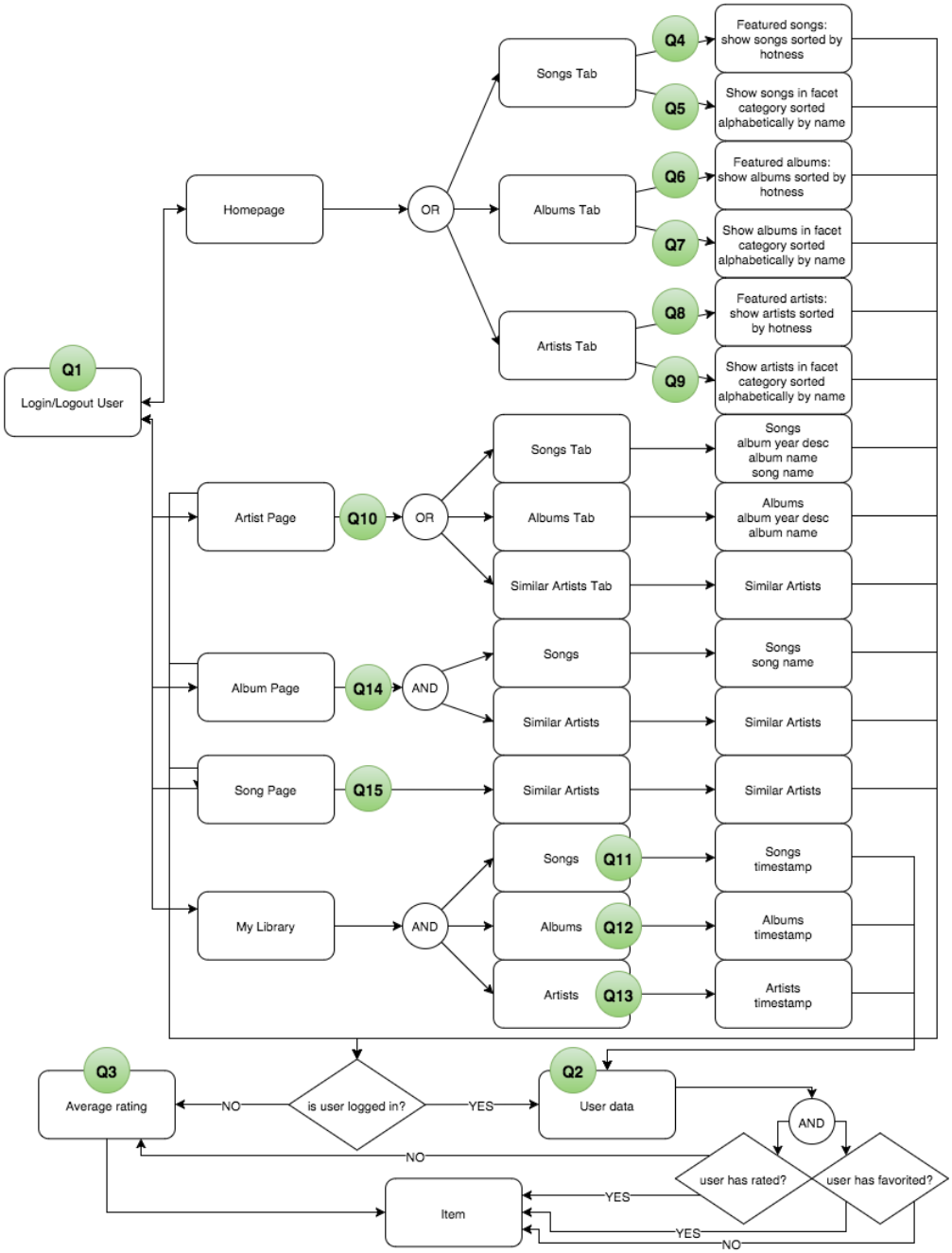
Entity Relationship Model



5.1.2. Access Patterns

Next, we used those same use cases to determine how the data in the entities would be accessed. Given the use patterns of a Cassandra datastore, it's important to think in terms of queries when designing the schema. This can be a challenging transition for those accustomed to building SQL schemas. Maybe even more challenging is the need to denormalize the schema — a thought that could keep an SQL database designer up at night. Luckily, there are many good tutorials and documents online dicussing this process.

Access Patterns Diagram

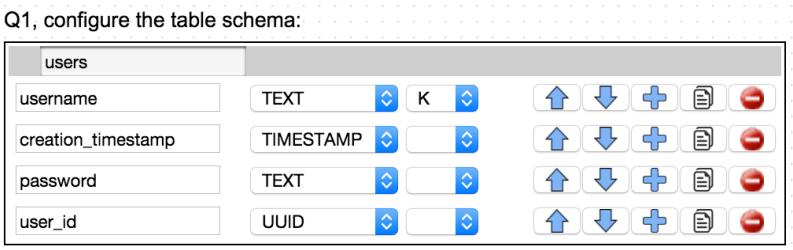
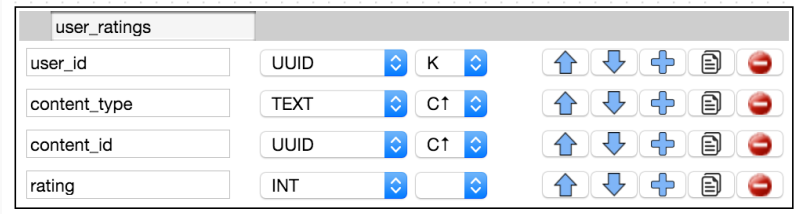
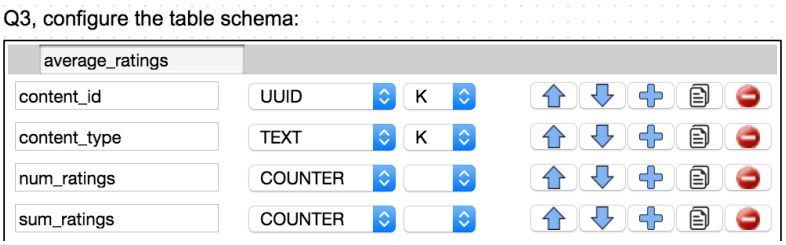
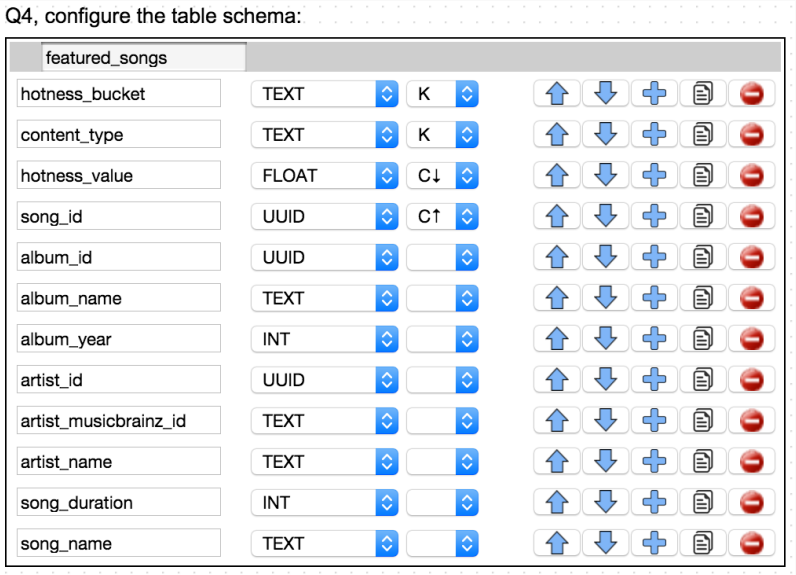


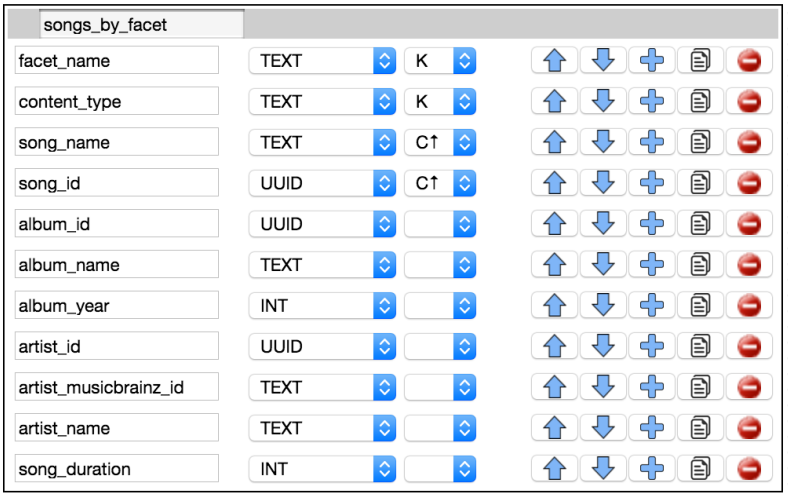
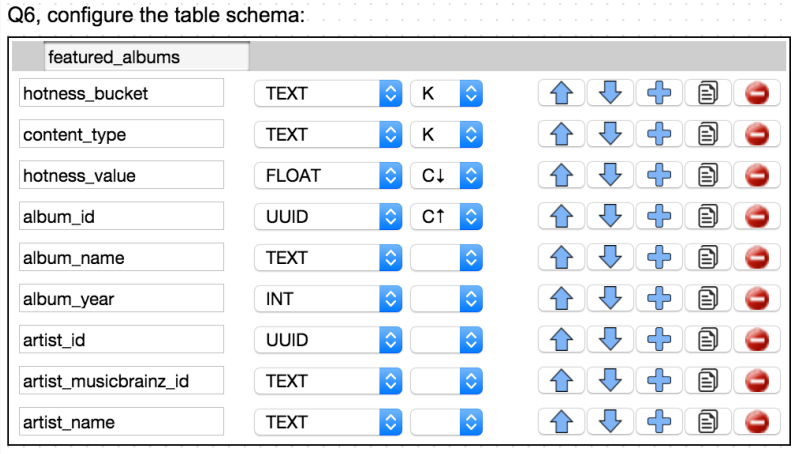
5.2. Implementation

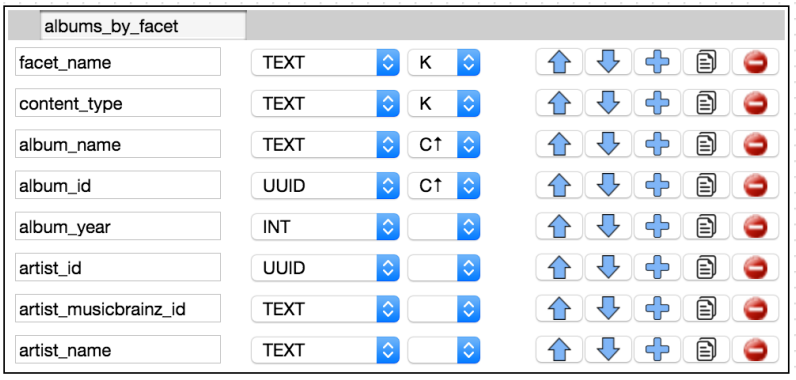
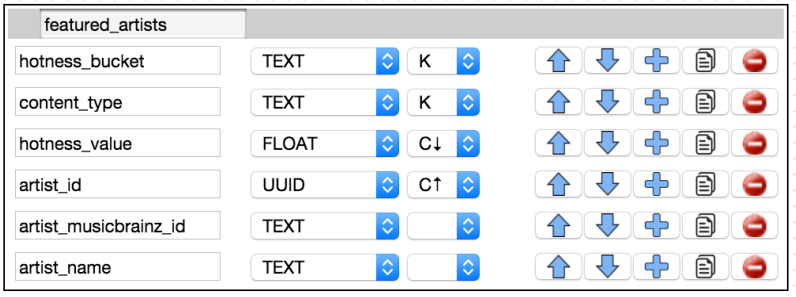
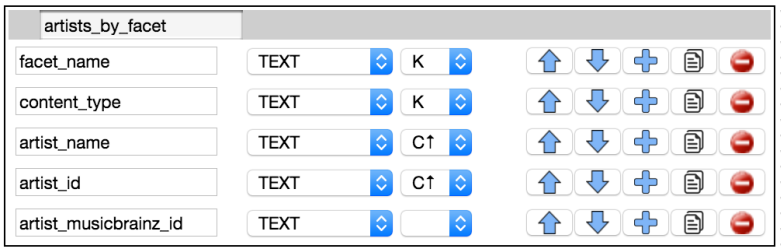
The following table shows the details for each query included in the access patterns diagram above.

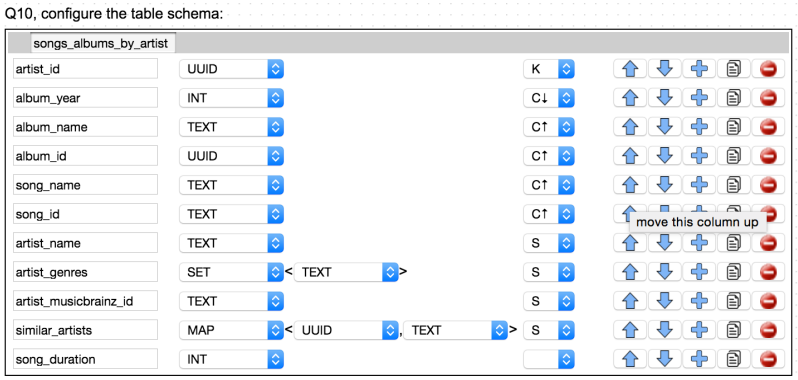
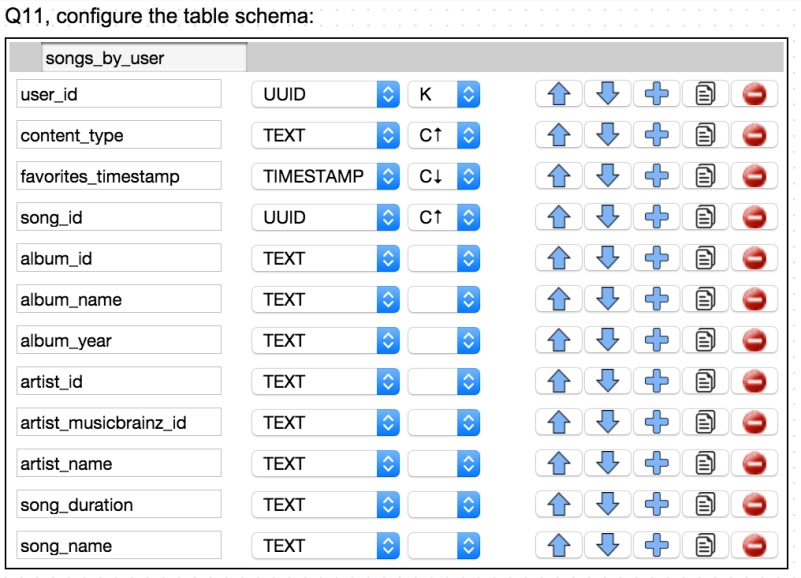
Fields marked with **K** are partition keys. Fields marked with **C** are cluster keys.

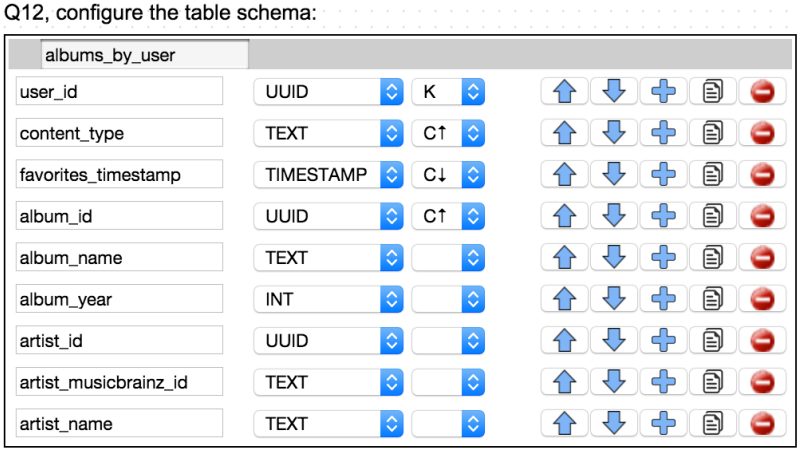
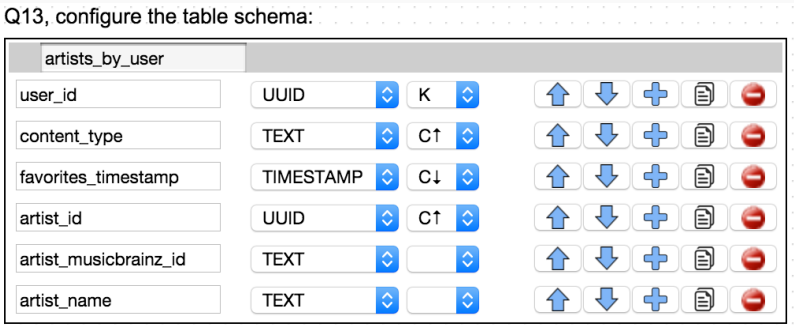
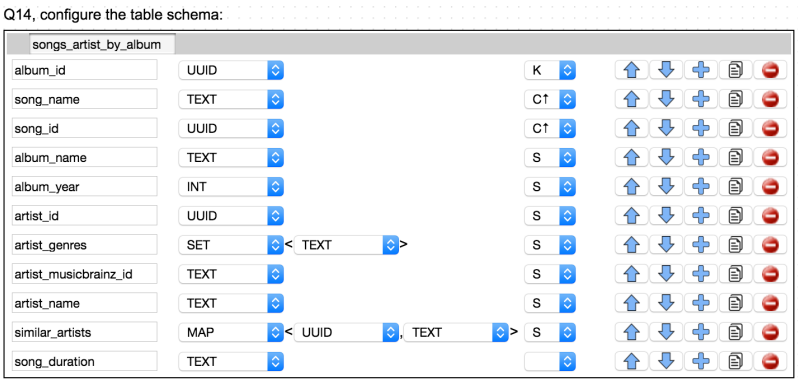
Table 8. Queries

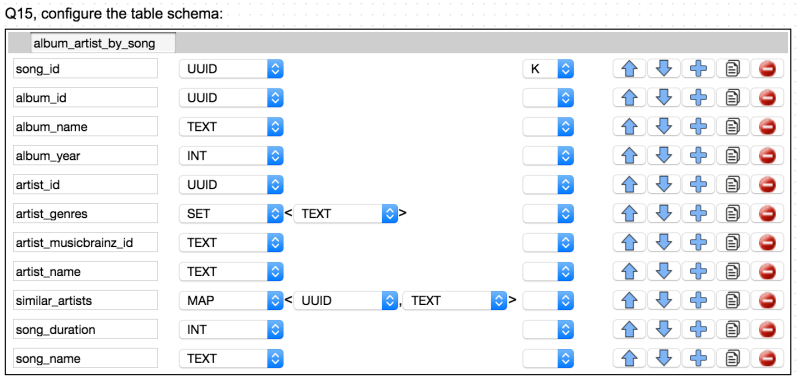
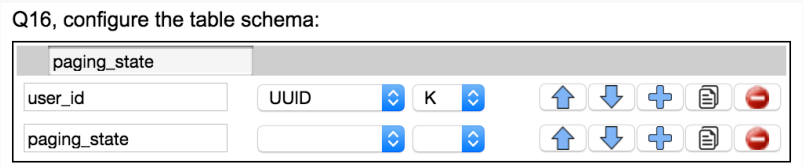
Query Number	Description	Table Schema
Q1	Users	
Q2	User ratings	
Q3	Average Ratings	
Q4	Featured Songs	

Query Number	Description	Table Schema
Q5	Songs by Facet	Q5, configure the table schema: 
Q6	Featured Songs	Q6, configure the table schema: 

Query Number	Description	Table Schema
Q7	Songs by Facet	Q7, configure the table schema: 
Q8	Featured Artists	Q8, configure the table schema: 
Q9	Artists by Facet	Q9, configure the table schema: 

Query Number	Description	Table Schema
Q10	Songs/Albums by Artist	
Q11	Songs by User	

Query Number	Description	Table Schema
Q12	Albums by User	
Q13	Artists by User	
Q14	Songs/Artist by Album	

Query Number	Description	Table Schema
Q15	Album/Artist by Song	
Q16	Paging State	

5.3. Data Import

The Million Song Library project uses data from the Million Song Dataset. (A 10,000 song subset of the full data set is also available.) Downloading the data set yields a gzipped file that contains two directories: **AdditionalFiles** and **data**. The **data** directory has a number of subdirectories that contain an **HDF** file (with the extension **h5**) for each song in the data set. The **AdditionalFiles** directory contains several files that summarize the data set — these files are not used for the import.

HDF or Hierarchical Data Format files contain all the data needed for albums, artists, and songs. We use the Python library written by Thierry Bertin-Mahieux to read these files. (We initially tried using Java and JHI5 but ran into issues with the native library aspect of that implementation.)

At a high level, the data import process uses Python to traverse a given directory tree, consuming all of the **.h5** files and creating a single CSV file with one line for each song. This CSV file is then consumed by a Java program to create the CSVs that are inserted into the Cassandra cluster.

The tags within an **.h5** file (**/metadata/artist_terms**) essentially indicate the genre of the song. There are so many genres used in the data set that it makes them not particularly useful as a searching mechanism. For this reason, a list of genres is defined in **Genre.java** to declare which of the genres we are interested in persisting. Any genres not found in the list are ignored.

6. Mac OS X Automated Setup

Follow the steps in this section to set up the Million Song Library demo on a Mac.

6.1. Get Your System Ready

Before setting up the demo, make sure you have everything you need.

Mac OS X 10.11

We tested this procedure on Mac OS X 10.11 (El Capitan). Check your OS X version to see if you're already running El Capitan. If you're running an older version of OS X, you should upgrade your Mac to El Capitan.

Apple Xcode

On Mac OS X, some necessary command line tools aren't installed by default, so to get them you'll need to install the latest version of Xcode from the Mac App Store.

After installation, make sure to open Xcode, accept the license agreement, and (if prompted) install any additional tools.

Java 1.8 JDK

Install the Java 1.8 JDK. Make sure to select the **Mac OS X x64** download option. To install, just double-click the DMG file to open it, and then double-click the installer.

Git

Install Git if you don't have it already. One easy method is to download the Mac OS X installer. To install, just double-click the DMG file to open it, and then double-click the installer.

Terminal

You'll need a terminal application so you can enter commands at the command prompt. The Terminal app included with Mac OS X will work just fine. You can find it in **Applications→Utilities**.

Where is My Home Directory?

For this process, it's important to know where your home directory is. Your home directory is the directory that has the same name as your user name.

For example, if the user name you use to log on to your Mac is jdoe, your home directory is named **jdoe**. In Terminal, your home directory is represented by the tilde (~) sign. For example, the path **~/stuff** points to a directory named **stuff** located inside your home directory.

6.2. Update Existing Tools (If Needed)

The Million Song Library setup script will automatically install the required tools and frameworks, so in most cases there's nothing more you need to install before getting started. There's one

exception—if you already have some of the required tools installed on your system, then you need to make sure they are at the required version. Check the list below.

Maven 3.3.9 or later

To see what version of Maven is installed, enter `mvn -version` at the command prompt. If you get the message `command not found`, it means Maven is not installed, and there's no need to do anything.

If the command reports that an older version is installed, upgrade your Maven installation to version 3.3.9 or later.

Node 0.12.x or later

To see what version of Node is installed, enter `node --version` at the command prompt. If you get the message `command not found`, it means Node is not installed, and there's no need to do anything.

If the command reports that an older version is installed, download the Node.js Mac OS X installer. Just double-click the installer to run it. This will also install npm.

npm 2.7.x or later

To see what version of npm is installed, enter `npm -version` at the command prompt. If you get the message `command not found`, it means npm is not installed, and there's no need to do anything.

If the command reports that an older version is installed, download the Node.js Mac OS X installer, which includes npm. Just double-click the installer to run it.

6.3. Enable the Root User

On Mac OS X, the root user is not enabled by default, so you'll need to enable it so the setup script runs properly. To enable the root user, just follow these instructions.

6.4. Clone the MSL Repository

To run the setup script, you first need to clone the million-song-library repository to a local directory.

1. Open a new Terminal window and make sure you are in your home directory.
2. Enter the following command:

```
git clone https://github.com/kenzanmedia/million-song-library
```

   Git creates a directory called **million-song-library** inside your home directory, and checks out a working copy of the repository.

3. Enter the following command to change directories to the cloned repository:

```
cd million-song-library
```

4. Enter the folowing command to initialize the repository and submodules:

```
git submodule init && git submodule update
```

6.5. Install and Start Cassandra

Now let's get Cassandra up and running.

1. Download Cassandra 2.1.11 (Community version) from the Datastax download site.

Make sure to download the correct file: **dsc-cassandra-2.1.11-bin.tar.gz**

2. Create a new directory named **cassandra** inside your home (~/) directory.
3. Move the **dsc-cassandra-2.1.11-bin.tar.gz** file into the **~/cassandra** directory, and then double-click the file to extract it.
4. To start Cassandra, open a new Terminal window and enter the following command:

```
sh ~/cassandra/dsc-cassandra-2.1.11/bin/cassandra
```

6.6. Run the Setup Script

Now that Cassandra is running, you're ready to run the setup script.

1. In Terminal, press **Command+T** to open a new tab.
2. Enter the following command to change to the script directory:

```
cd ~/million-song-library/common
```

3. Enter the following command to make the setup script executable:

```
chmod +x setup.sh
```

4. Enter the following command to run the setup script:

```
./setup.sh -c ~/cassandra/dsc-cassandra-2.1.11 -n -s -g
```

Watch the script output — you will be prompted to press <**Enter**> and also to enter your administrator password.

5. Wait for the script to complete (about 30 minutes).

It's normal to see some warnings while the script runs. If the script encounters an error that prevents it from finishing, make sure you installed all the required tools. See Get Your System Ready and Update Existing Tools (If Needed) above. Then try running the script again.

6.7. Start the MSL Demo

Setup is complete — now it's time to start the Million Song Library demo.

1. In Terminal, press **Command**+**T** to open another new tab.
2. Enter the following command to change to the application directory:

```
cd ~/million-song-library/msl-pages
```

3. Enter the following command to start the application front end:

```
npm run full-dev
```

   Wait for the front end to start up — this will take just a couple of minutes.

4. In Terminal, press **Command**+**T** to open another new tab. You should still be in the **msl-pages** directory.
5. Enter the following command to start the server instances:

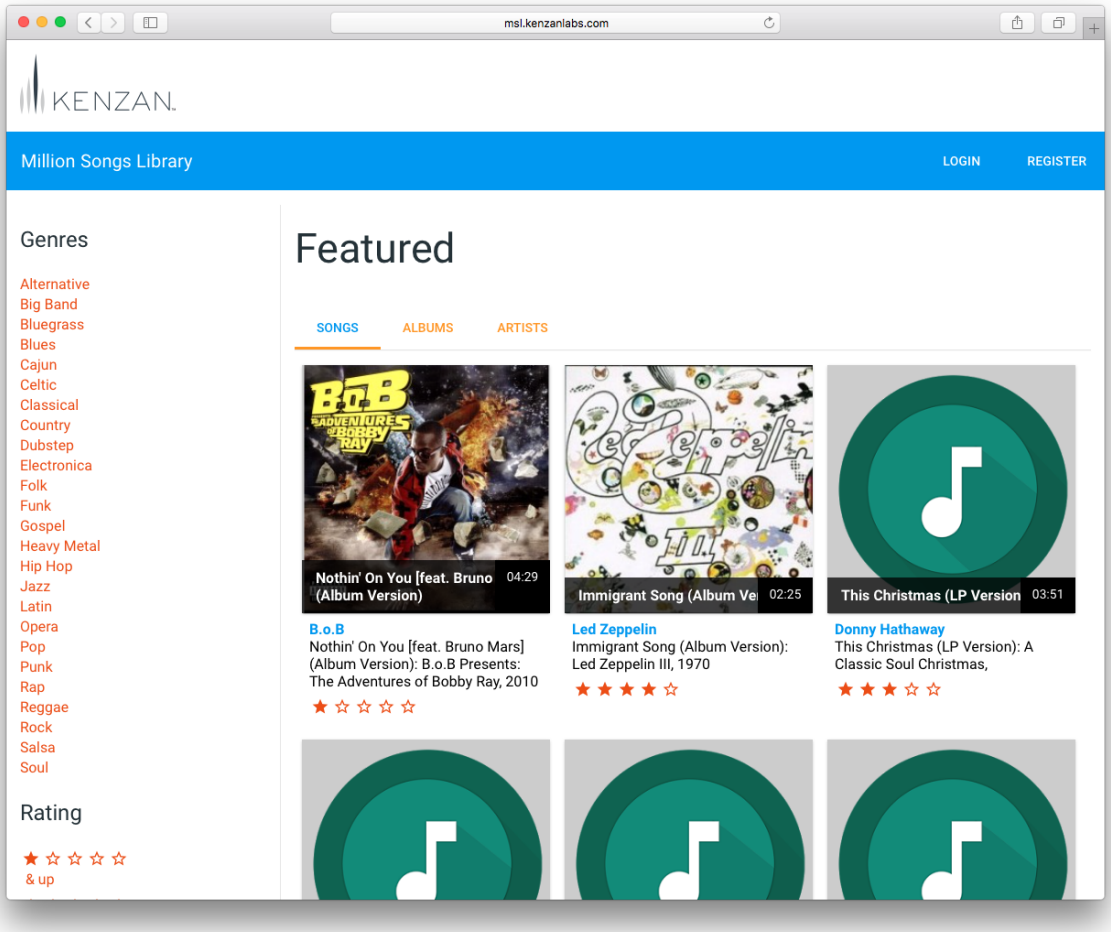
```
npm run serve-all
```

   Wait for the server instances to start up — again, this will take just a couple of minutes.

6. Open a Web browser and point it to: **msl.kenzanlabs.com:3000**

   The Million Song Library home page displays. (If you don't see data right away, wait a couple of minutes and then refresh the page.)

Million Song Library Home Page



Now that the Million Song Library demo is working, here are some fun things to try:

- Click **Register** to register for an account.
- Click the labels to switch the view between **Songs**, **Albums**, and **Artists**.
- Click a genre or star rating on the left to filter songs. (Click the **x** to clear a filter.)
- Mouse over a song and click **Add to library** to add it to your music library (you must be logged in).
- Click the stars below a song to rate it (you must be logged in).

6.8. Stop the MSL Demo

Done having fun for now? Follow these steps to stop the Million Song Library demo.

1. In Terminal, switch to the tab where you started the server instances and press **Control+C** to stop the server.
2. In Terminal, switch to the tab where you started the application and press **Control+C** to stop the application.
3. In Terminal, switch to the tab where you started Cassandra and enter the following command:

```
ps auwx | grep cassandra
```

Look at the output from the command and note the first 3–5 digit number that appears in the output. This is the process ID for Cassandra.

4. Enter the following command where **pid** is the process ID you found (you'll be prompted for your administrator password):

```
sudo kill pid
```

You don't need to run the setup script if you want to start the Million Song Library demo again. Just start Cassandra (see Install and Start Cassandra) and then start the application and server instances (see Start the MSL Demo).

7. Linux Automated Setup

Follow the steps in this section to set up the Million Song Library demo on a system running Linux.

7.1. Get Your System Ready

Before setting up the demo, make sure you have everything you need.

Ubuntu 14.04

We tested this procedure on Ubuntu 14.04 (Trusty Tahr). Check your Ubuntu version to see if you're already running Trusty Tahr. If you're running an older version of Ubuntu, you should upgrade your system to Trusty Tahr.

Java 1.8 JDK

Install the Java 1.8 JDK:

1. In a Terminal window, enter the following command to add the WebUpd8 Java PPA repository (you'll be prompted for your administrator password):

```
sudo add-apt-repository ppa:webupd8team/java
```

2. Enter the following command to update the package index:

```
sudo apt-get update
```

3. Enter the following command to install the Java 1.8 JDK:

```
sudo apt-get install oracle-java8-installer
```

Git

Install Git if you don't have it already. In a Terminal window, enter the following command (you'll be prompted for your administrator password):

```
sudo apt-get install git
```

Terminal

You'll need a terminal application so you can enter commands at the command prompt. The Terminal app included with Ubuntu will work just fine. You can find it in **Applications**.

Where is My Home Directory?

For this process, it's important to know where your home directory is. Your home directory is the directory that has the same name as your user name.

For example, if the user name you use to log on to your system is jdoe, your home directory is named **jdoe**. In Terminal, your home directory is represented by the tilde (~) sign. For example, the path **~/stuff** points to a directory named **stuff** located inside your home directory.

7.2. Update Existing Tools (If Needed)

The Million Song Library setup script will automatically install the required tools and frameworks, so in most cases there's nothing more you need to install before getting started. There's one exception—if you already have some of the required tools installed on your system, then you need to make sure they are at the required version. Check the list below.

Maven 3.3.9 or later

To see what version of Maven is installed, enter `mvn -version` at the command prompt. If the system does not return a version number, it means Maven is not installed, and there's no need to do anything.

If the command reports that an older version is installed, upgrade your Maven installation to version 3.3.9 or later.

Node 0.12.x or later

To see what version of Node is installed, enter `node --version` at the command prompt. If the system does not return a version number, it means Node is not installed, and there's no need to do anything.

If the command reports that an older version is installed, update your installation of Node.js.

npm 2.7.x or later

To see what version of npm is installed, enter `npm -version` at the command prompt. If the system does not return a version number, it means npm is not installed, and there's no need to do anything.

If the command reports that an older version is installed, update your installation of Node.js,

which includes npm.

7.3. Clone the MSL Repository

To run the setup script, you first need to clone the million-song-library repository to a local directory.

1. Open a new Terminal window and make sure you are in your home directory.
2. Enter the following command:

```
git clone https://github.com/kenzanmedia/million-song-library
```

   Git creates a directory called **million-song-library** inside your home directory, and checks out a working copy of the repository.

3. Enter the following command to change directories to the cloned repository:

```
cd million-song-library
```

4. Enter the folowing command to initialize the repository and submodules:

```
git submodule init && git submodule update
```

7.4. Install and Start Cassandra

Now let's get Cassandra up and running.

1. Download Cassandra 2.1.11 (Community version) from the Datastax download site.

Make sure to download the correct file: **dsc-cassandra-2.1.11-bin.tar.gz**

2. Create a new directory named **cassandra** inside your home (~/) directory.
3. Move the **dsc-cassandra-2.1.11-bin.tar.gz** file into the ~/**cassandra** directory, and then extract the contents into the ~/**cassandra** directory.
4. To start Cassandra, enter the following command in Terminal:

```
sh ~/cassandra/dsc-cassandra-2.1.11/bin/cassandra
```

7.5. Run the Setup Script

Now that Cassandra is running, you're ready to run the setup script.

1. In Terminal, press **Shift+Control+T** to open a new tab.
2. Enter the following command to change to the script directory:

```
cd ~/million-song-library/common
```

3. Enter the following command to make the setup script executable:

```
chmod +x setup.sh
```

4. Enter the following command to run the setup script (you'll be prompted for your administrator password):

```
sudo ./setup.sh -c ~/cassandra/dsc-cassandra-2.1.11 -n -s -g
```

5. Wait for the script to complete (about 30 minutes).

It's normal to see some warnings while the script runs. If the script encounters an error that prevents it from finishing, make sure you installed all the required tools. See Get Your System Ready and Update Existing Tools (If Needed) above. Then try running the script again.

7.6. Start the MSL Demo

Setup is complete — now it's time to start the Million Song Library demo.

1. In Terminal, press **Shift+Control+T** to open another new tab.
2. Enter the following command to change to the application directory:

```
cd ~/million-song-library/msl-pages
```

3. Enter the following command to start the application front end:

```
npm run full-dev
```

   Wait for the front end to start up — this will take just a couple of minutes.

4. In Terminal, press **Shift+Control+T** to open another new tab. You should still be in the **msl-pages** directory.
5. Enter the following command to start the server instances (you'll be prompted for your administrator password):

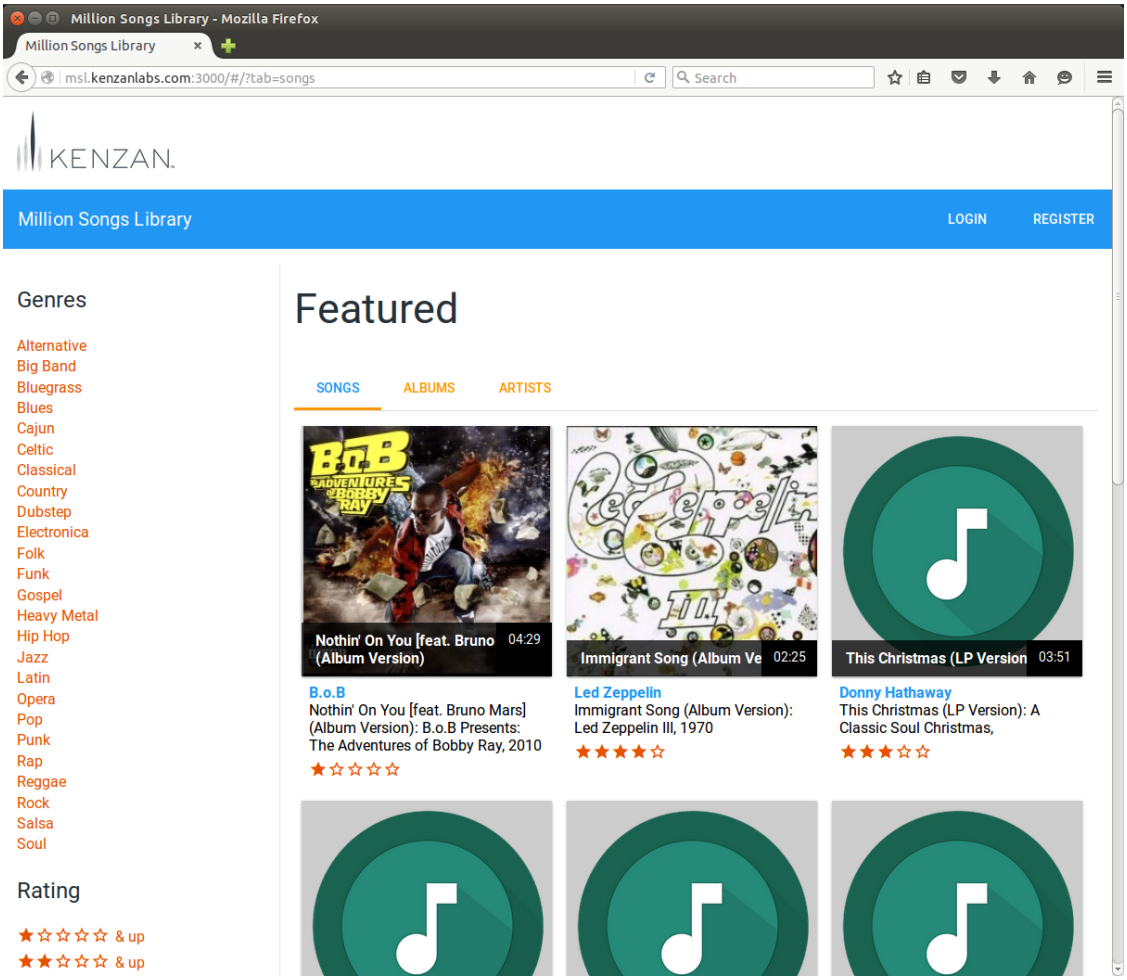
```
sudo npm run serve-all
```

Wait for the server instances to start up — again, this will take just a couple of minutes.

6. Open a Web browser and point it to: **msl.kenzanlabs.com:3000**

   The Million Song Library home page displays. (If you don't see data right away, wait a couple of minutes and then refresh the page.)

Million Song Library Home Page



Now that the Million Song Library demo is working, here are some fun things to try:

- Click **Register** to register for an account.
- Click the labels to switch the view between **Songs**, **Albums**, and **Artists**.
- Click a genre or star rating on the left to filter songs. (Click the **x** to clear a filter.)
- Mouse over a song and click **Add to library** to add it to your music library (you must be logged in).
- Click the stars below a song to rate it (you must be logged in).

7.7. Stop the MSL Demo

Done having fun for now? Follow these steps to stop the Million Song Library demo.

1. In Terminal, switch to the tab where you started the server instances and press **Control+C** to stop the server.
2. In Terminal, switch to the tab where you started the application and press **Control+C** to stop the application.
3. In Terminal, switch to the tab where you started Cassandra and enter the following command:

   ```
   ps auwx | grep cassandra
   ```

   Look at the output from the command and note the first 3–5 digit number that appears in the output. This is the process ID for Cassandra.

4. Enter the following command where **pid** is the process ID you found (you'll be prompted for your administrator password):

   ```
   sudo kill pid
   ```

You don't need to run the setup script if you want to start the Million Song Library demo again. Just start Cassandra (see Install and Start Cassandra) and then start the application and server instances (see Start the MSL Demo).

8. Windows Automated Setup

Follow the steps in this section to set up the Million Song Library demo on a system running Windows. You will actually be installing the demo inside a Linux virtual machine using Vagrant — a process that's both easy and free.

8.1. Get Your System Ready

Before setting up the demo, make sure you have everything you need.

Windows 10

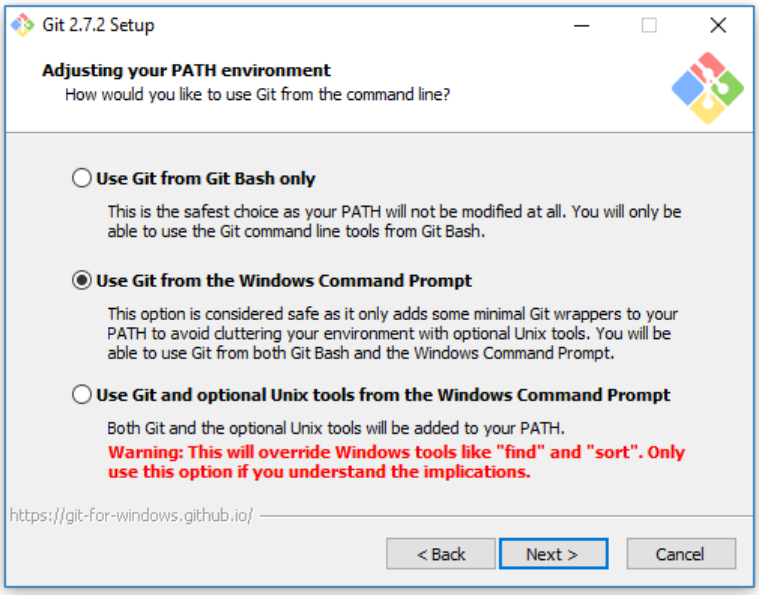
We tested this procedure on Windows 10 Pro. Check your Windows version to see if you're already running Windows 10 Pro, and to find out if you're running a 32-bit or 64-bit version of Windows. If you're running an older version of Windows, you should upgrade your system to Windows 10.

Git

Install Git if you don't have it already. One easy method is to download the Windows installer. To install, just double-click the EXE file.

During Git installation, make sure to select the **Use Git from the Windows Command Prompt** option. Otherwise you won't be able use Git commands in PowerShell.

Installing Git for Windows



Vagrant

The easiest way to set up the Million Song Library on Windows is with Vagrant, a tool for automating the creation of development environments. If you don't already have it, install Vagrant.

PowerShell

You'll need to use Windows PowerShell to enter commands at the command prompt. You can find it in **Start → All Apps → Windows PowerShell**.

For some commands, you'll need to use Git Bash instead (we'll let you know when). You can find it in **Start → All Apps → Git → Git Bash**.

Where is My Home Directory?

For this process, it's important to know where your home directory is. Your home directory is the directory that has the same name as your user name.

For example, if the user name you use to log on to Windows is jdoe, then the path **C:\Users\jdoe** points to your home directory.

In Git Bash, your home directory is represented by the tilde (~) sign. For example, the path **~/stuff** points to a directory named **stuff** located inside your home directory.

8.2. Clone the MSL Repository

To use Vagrant to set up a virtual machine for the demo, you first need to clone the million-song-library repository to a local directory.

1. Open a new PowerShell window and make sure you are in your home directory.
2. Enter the following command:

   ```
   git clone https://github.com/kenzanmedia/million-song-library
   ```

   Git creates a directory called **million-song-library** inside your home directory, and checks out a working copy of the repository.

3. Enter the following command to change directories to the cloned repository:

   ```
   cd million-song-library
   ```

4. Enter the folowing command to initialize the repository:

   ```
   git submodule init
   ```

5. Enter the folowing command to update all submodules:

   ```
   git submodule update
   ```

8.3. Set up the Virtual Machine

The Vagrant file contains all of the information needed to set up and configure the Linux virtual machine to run the Million Song Library Demo. Let's get it running.

1. Open a new PowerShell window and make sure you are in your home directory.
2. Enter the following command to change to the million-song-library repository:

   ```
   cd million-song-library
   ```

3. Enter the following command to set up the virtual machine:

   ```
   vagrant up ubuntu
   ```

4. Wait for the setup process to complete (about 30 minutes).

If you get an error stating `The box 'ubuntu/trusty64' could not be found`, it means you don't have the correct version of the Microsoft Visual C++ Redistributable Package on your system. First install the Microsoft Visual C++ 2010 SP1 Redistributable Package, and then run the `vagrant up ubuntu` command again.

8.4. Edit the Hosts File

While you're waiting for the setup process to complete, take a few minutes to update the **hosts** file on your local system. This will allow you to access the Million Song Library application using a Web browser.

1. In a text editor, open the **hosts** file located in the **C:\Windows\System32\Drivers\etc** directory.
2. Add the following line to the **hosts** file.

```
127.0.0.1     msl.kenzanlabs.com
```

3. Save and close the **hosts** file.

8.5. Start the MSL Demo

This is where we need to start using Git Bash, which already has an ssh client (unlike PowerShell).

1. Open a new Git Bash window and make sure you are in your home directory.
2. Enter the following command to change to the million-song-library repository:

```
cd million-song-library
```

3. Enter the following command to connect to the virtual machine and forward the necessary ports (it's really long, so use copy and paste, and make sure to select the entire command):

```
vagrant ssh ubuntu -- -L 3000:localhost:3000 -L 3002:localhost:3002 -L
3003:localhost:3003 -L 3004:localhost:3004 -L 9002:localhost:9002 -L
9003:localhost:9003
```

4. After you are connected to the virtual machine, enter the following command to change to the application directory:

```
cd kenzan/million-song-library/msl-pages
```

5. Enter the following command to start the application front end:

```
npm run full-dev
```

Wait for the front end to start up — this will take a few minutes.

6. Open a new Git Bash window and make sure you are in your home directory.
7. Enter the following command to change to the million-song-library repository:

```
cd million-song-library
```

8. Enter the following command to connect to the virtual machine (we don't need such a long command this time as we already forwarded the necessary ports):

```
vagrant ssh ubuntu
```

9. After you are connected to the virtual machine, enter the following command to change to the application directory:

```
cd kenzan/million-song-library/msl-pages
```

10. Enter the following command to start the server instances:

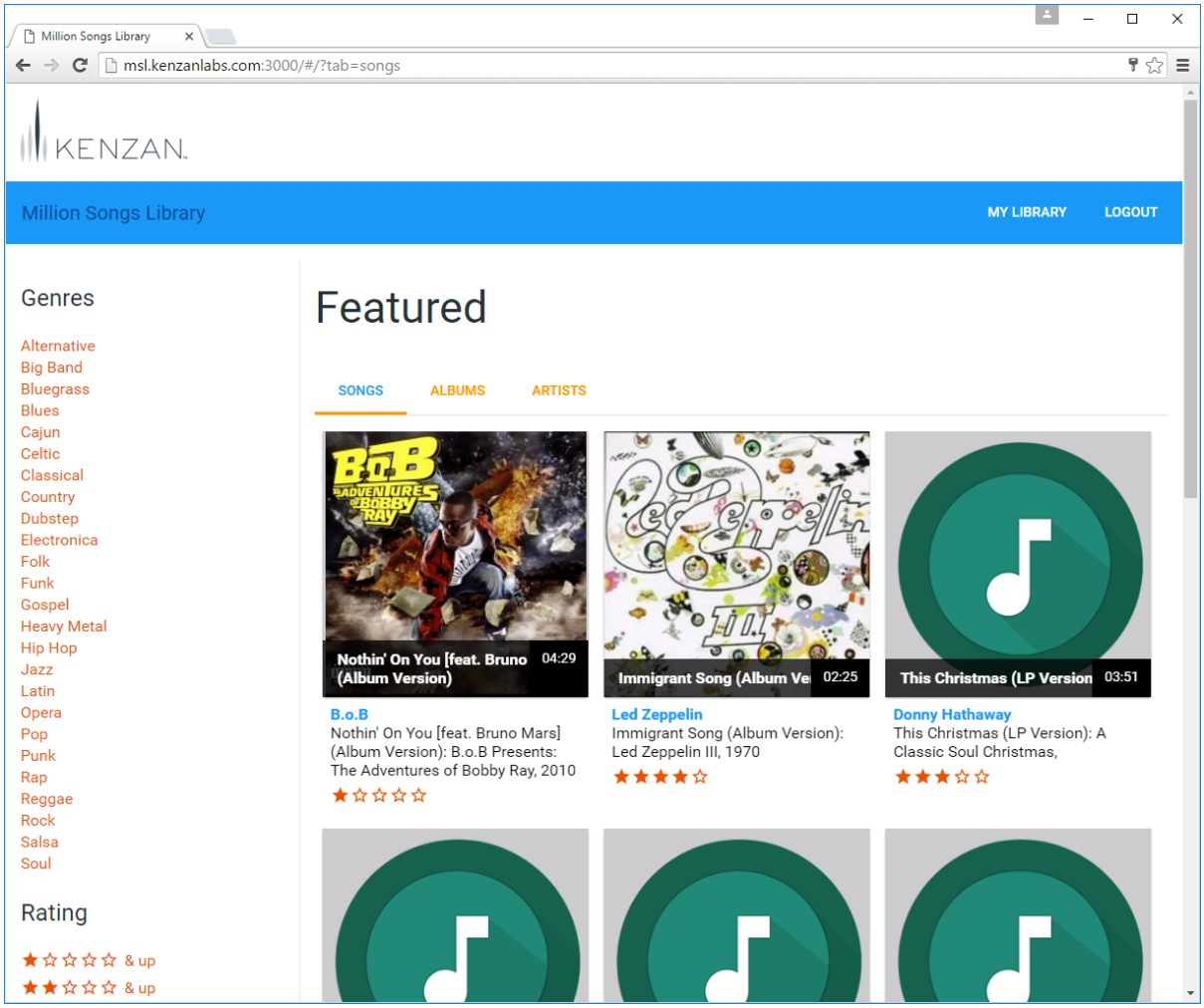
```
sudo npm run serve-all
```

Wait for the server instances to start up — again, this will take several minutes.

11. Open a Web browser and point it to: **msl.kenzanlabs.com:3000**

The Million Song Library home page displays. (If you don't see data right away, wait a couple of minutes and then refresh the page.)

Million Song Library Home Page



Now that the Million Song Library demo is working, here are some fun things to try:

- Click **Register** to register for an account.
- Click the labels to switch the view between **Songs**, **Albums**, and **Artists**.
- Click a genre or star rating on the left to filter songs. (Click the **x** to clear a filter.)
- Mouse over a song and click **Add to library** to add it to your music library (you must be logged in).
- Click the stars below a song to rate it (you must be logged in).

8.6. Stop the MSL Demo

Done having fun for now? Here's how to terminate the virtual machine:

1. Open a new PowerShell window and make sure you are in your home directory.
2. Enter the following command to change to the million-song-library repository:

```
cd million-song-library
```

3. Enter the following command to destroy the virtual machine:

```
vagrant destroy ubuntu -f
```

The virtual machine is deleted. If you want to run the demo again, just follow this procedure to create a new Million Song Library virtual machine any time.

9. Manual Setup

Are you the type who likes to see all the magic that happens behind the scenes? Or maybe you prefer that things are installed on your system *just so*. In either case, you can can manually install the Million Song Library on a Mac or Linux system without using the automated setup script. Just follow the steps in this section.

If you want to get up and running as quickly as possible, using the automated setup script is the easiest way to install the Million Song Library demo on a Mac or Linux system. Or use Vagrant to automate setup on a Windows system.

9.1. Get Your System Ready

Before setting up the demo, make sure all of the following tools are installed on your system and are at the required version or higher.

This procedure assumes that you're running the Million Song Library demo on Mac OS X 10.11 (El Capitan) or Ubuntu 14.04 (Trusty Tahr). Things might work a little differently on other operating systems.

Entering Commands

We tested these steps using the default Terminal applications on Mac OS X and Ubuntu.

Where is My Home Directory?

For this process, it's important to know where your home directory is. Your home directory is the directory that has the same name as your user name.

For example, if the user name you use to log on to your system is jdoe, your home directory is named **jdoe**. In Terminal, your home directory is represented by the tilde (~) sign. For example, the path **~/stuff** points to a directory named **stuff** located inside your home directory.

Java 1.8 JDK

To check the installed version, use the `java -version` command. If necessary, install the Java 1.8 JDK.

Git

To check the installed version, use the `git --version` command. If necessary, install Git.

Maven 3.3.9 or later

To check the installed version, use the `mvn -version` command. If necessary, set up Maven.

Node 0.12.x or later

To check the installed version, use the `node --version` command. If necessary, install Node.

npm 2.7.x or later

To check the installed version, use the `npm -version` command. If necessary, install Node, which includes npm.

Bower

To check the installed version, use the `bower --version` command. If necessary, install Bower.

If you get a permissions error, use `sudo bower --version` instead.

RubyGem

To check the installed version, use the `gem --version` command. If necessary, install RubyGem for Mac or Linux.

AsciiDoctor

To check the installed version, use the `asciidoctor --version` command. If necessary, install AsciiDoctor.

Cassandra

1. Download Cassandra 2.1.11 (Community version) from the Datastax download site.

Make sure to download the correct file: **dsc-cassandra-2.1.11-bin.tar.gz**

2. Create a new directory named **cassandra** inside your home directory.
3. Move the **dsc-cassandra-2.1.11-bin.tar.gz** file into the new **cassandra** directory, and then extract the contents into the **cassandra** directory.

Apple Xcode (Mac OS X Only)

On Mac OS X, some necessary command line tools aren't installed by default, so to get them you'll need to install the latest version of Xcode from the Mac App Store.

After installation, make sure to open Xcode, accept the license agreement, and (if prompted) install any additional tools.

Enable Root User (Mac OS X Only)

On Mac OS X, the root user is not enabled by default, so you'll need to enable it so the setup script runs properly. To enable the root user, just follow these instructions.

9.2. Clone the MSL Repository

1. Make sure you are in your home directory.
2. Clone the million-song-library repository:

```
git clone https://github.com/kenzanmedia/million-song-library
```

3. Change directories to the cloned repository:

```
cd million-song-library
```

4. Initialize the repository and submodules:

```
git submodule init && git submodule update
```

9.3. Set up the Client

1. Change to the **msl-pages** directory inside the **million-song-library** directory

```
cd msl-pages
```

2. Download and install all dependencies:

```
npm install && bower install
```

3. Install Protractor and Selenium WebDriver globally:

```
npm install -g -y protractor && npm install -g -y selenium-webdriver
```

== Set up the Server

4. Change to the **server** directory inside the **million-song-library** directory:

```
cd ../server
```

5. Run the Maven file to set up the server:

```
mvn clean compile
```

9.4. Set up Cassandra

1. Open a new Terminal window, or a PowerShell window with administrator privileges.
2. Start Cassandra:

   ```
   sh ~/cassandra/dsc-cassandra-2.1.11/bin/cassandra
   ```

3. After Cassandra finishes starting, press <**Enter**> to display the command prompt again.
4. Enter the Cassandra console:

   ```
   sh ~/cassandra/dsc-cassandra-2.1.11/bin/cqlsh
   ```

5. Import data by entering the following commands at the **cqlsh**> prompt, pressing <**Enter**> after each command:

   ```
   SOURCE 'msl_ddl_latest.cql';
   SOURCE 'msl_dat_latest.cql';
   ```

   Wait for the database to import the data — this will take several minutes.

6. Exit the console:

   ```
   exit
   ```

9.5. Start the MSL Demo

1. Open a new Terminal window, or a PowerShell window with administrator privileges.
2. Change to the application directory:

   ```
   cd ~/million-song-library/msl-pages
   ```

3. Start the application front end:

   ```
   npm run full-dev
   ```

   Wait for the front end to start up — this will take just a couple of minutes.

4. Open a new Terminal window, or a PowerShell window with administrator privileges.
5. Change to the application directory:

```
cd ~/million-song-library/msl-pages
```

6. Start the server instances:

   Linux:

   ```
   sudo npm run serve-all
   ```

   Mac:

   ```
   npm run serve-all
   ```

   Wait for the server instances to start up — again, this will take just a couple of minutes.

7. Open a Web browser and point it to: **msl.kenzanlabs.com:3000**

   The Million Song Library home page displays. (If you don't see data right away, wait a couple of minutes and then refresh the page.)

9.6. Stop the MSL Demo

1. Switch to the Terminal or PowerShell window where you started the server instances and press **Control+C** to stop the server.
2. Switch to the Terminal or PowerShell window where you started the application and press **Control+C** to stop the application.
3. Switch to the Terminal or PowerShell window where you started Cassandra and enter the following command:

   ```
   ps auwx | grep cassandra
   ```

   Look at the output from the command and note the first 3–5 digit number that appears in the output. This is the process ID for Cassandra.

4. Enter the following command where **pid** is the process ID you found (you'll be prompted for your administrator password):

   ```
   sudo kill pid
   ```

10. Deploying to AWS

Deploying the Million Song Library demo locally is a great way to see it in action, but a

microservices-based application like MSL is really designed for the cloud. Follow the steps in this section to deploy the Million Song Library demo to an AWS EC2 instance.

10.1. Get Your System Ready

Before setting up the demo, make sure you have everything you need.

AWS Account

You will need an AWS account to complete this procedure. If you don't already have one, you can set up an AWS account for free.

You may incur costs on your AWS account based on your EC2 instance usage while running the Million Song Library demo. Be sure to stop your MSL instance when you're not using it.

Git

You will need Git to make a local copy of the million-song-library repository. If you don't already have it, install Git.

(Windows only) During Git installation, make sure to select the **Use Git from the Windows Command Prompt** option. Otherwise you won't be able use Git commands in PowerShell.

Vagrant

The easiest way to set up the Million Song Library in an EC2 instance is with Vagrant, a tool for automating the creation of development environments. If you don't already have it, install Vagrant.

Entering Commands

We tested these steps using the default Terminal applications on Mac OS X and Ubuntu. To enter commands on Windows 10, we recommend running Windows PowerShell as an Administrator.

Where is My Home Directory?

For this process, it's important to know where your home directory is. Your home directory is the directory that has the same name as your user name.

- On Mac and Linux, your home directory is represented by the tilde (~) sign in Terminal commands.
- On Windows, your home directory is: **C:\Users\<your_username>**

10.2. Clone the MSL Repository

To edit the Vagrant file, you first need to clone the million-song-library repository to a local directory.

1. Open a new Terminal or PowerShell window and make sure you are in your home directory.
2. Enter the following command to clone the million-song-library repository:

```
git clone https://github.com/kenzanmedia/million-song-library
```

3. Enter the following command to change directories to the cloned repository:

```
cd million-song-library
```

4. Enter the following command to initialize the repository and submodules:

```
git submodule init && git submodule update
```

For Windows, you will need to run the commands separately: first `git submodule init` and then `git submodule update`.

10.3. Set up Vagrant for AWS

The AWS plug-in enables Vagrant to automtically create EC2 instances using a dummy virtual machine.

1. Enter the following command to install the AWS plug-in for Vagrant:

```
vagrant plugin install vagrant-aws
```

2. Enter the following command to download the dummy virtual machine:

```
vagrant box add dummy https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box
```

10.4. Generate an SSH Key Pair

Generate a new SSH key pair for use on the EC2 instance.

On Windows, use the Git Bash terminal app for this step. You can find it uder **Start → All apps → Git → Git Bash**.

1. Generate a new key pair:
   a. Enter the following command where *myemailaddress@gmail.com* is your email address:

   ```
   ssh-keygen -t rsa -C myemailaddress@gmail.com
   ```

   b. Enter the file path and name where you want to save the key pair. For example:

   ```
   /Users/jdoe/.ssh/aws_msl
   ```

   c. Press <**Enter**> twice to skip entering a passphrase.

2. Import the new SSH key pair into your AWS account:
   a. Sign in to your AWS EC2 Management Console and switch to your preferred region.
   b. Click **Key Pairs** (under **Network & Security**).
   c. Click **Import Key Pair**.
   d. Click **Choose File** and select the new public key file (for example, **aws_msl.pub**).

      Alternately, you can open the public key in a text editor, copy the contents, and paste the contents in the **Public key contents** box.

   e. In the **Key pair name** box, type the name of the key pair (for example, **aws_msl**).
   f. Click **Import**.

10.5. Generate an AWS Key Pair

Generate a new AWS key pair to allow Vagrant to log on to the EC2 instance.

1. In your EC2 Management Console, click **Key Pairs** (under **Network & Security**).
2. Click **Create Key Pair**.
3. In the **Key pair name** box, type the name of the key pair (for example, **kenzan_msl**).
4. Click **Create** to create and download the private key file.
5. Place the private key file in the same location as the SSH key you created. For example: **~/.ssh/kenzan_msl.pem**

10.6. Generate an AWS Access Key

Generate a new AWS access key to allow Vagrant to authenticate with AWS.

1. In your EC2 Management Console, click your user name, and then click **Security Credentials**.
2. Under **Access Keys**, click **Create New Access Key**.
3. Click **Download Key File** to download the access key file.

4. Place the access key file in the same location as the other keys you created. For example: **~/.ssh/rootkey.csv**

The access key file is a CSV file that contains values for two keys: **AWSAccessKeyId** and **AWSSecretKey**. You will need these two key values in a later step.

10.7. Create a Security Group

Create a new AWS Security Group to allow inbound access to the EC2 instance.

1. In your EC2 Management Console, click **Security Groups** (under **Network & Security**).
2. Click **Create Security Group**.
3. In the **Security group name** box, type the name of the security group (for example, **kenzanmsl**).
4. In the **Description** box, type a description for the security group (for example, **Million Song Library Demo**).
5. On the **Inbound** tab, click **Add Rule**.
6. For **Type**, select **All traffic**.
7. For **Source**, select **Anywhere**.
8. Click **Create**.

10.8. Edit the Vagrant File

The Vagrant file contains all of the information needed to set up and configure the EC2 instance. You need to edit the Vagrant file to add information about your AWS account — this lets Vagrant know where to set up the instance.

1. In a text editor, open the file named **Vagrantfile** located in the **million-song-library** directory (in your home directory).
2. Replace the following variables with the correct values:

Variable	Value
<<PATH_TO_SSH_PRIVATE_KEY>>	File path and name for your private SSH key. For example: **~/.ssh/aws_msl**
<<PATH_TO_SSH_PUBLIC_KEY>>	File path and name for your public SSH key. For example: **~/.ssh/aws_msl.pub**
<<KEY_PAIR_NAME>>	Name of your AWS key pair. For example: **kenzan_msl**

Variable	Value
<<PATH_TO_YOUR_AWS_KEY.pem>>	File path and name for your private AWS key. For example: **~/.ssh/kenzan_msl.pem**
<<AWS_ACCESS_KEY_ID>>	**AWSAccessKeyId** value from the **rootkey.csv** access key file you downloaded.
<<AWS_SECRET_ACCESS_KEY>>	**AWSSecretKey** value from the **rootkey.csv** access key file you downloaded.
<<SECURITY_GROUP_WITH_SSH_AND_HTTP_PORTS_ENABLED>>	Name of the AWS security group you created for use with the MSL demo. For example: **kenzanmsl**
aws.region = "us-west-2"	If necessary, replace the region code with your preferred AWS region.
aws.instance_type="t2.small"	Replace the instance type with the desired type. We recommend **t2.large** or better to avoid running out of resources.

3. Save and close the **Vagrant** file.

10.9. Provision an EC2 Instance

After you finish editing the Vagrant file, you're ready to use Vagrant to create and provision the EC2 instance with the Million Song Library Demo.

1. Open a new Terminal or PowerShell window and make sure you are in your home directory.
2. Enter the following command to change to the million-song-library repository:

```
cd million-song-library
```

3. Enter the following command to start the provisioning process:

```
vagrant up prod
```

4. Wait for the provisioning process to complete (about 30 minutes).

If you get an error stating that the **.gitconfig** file does not exist, enter the command `touch ~/.gitconfig` to create one, and then run `vagrant up prod` again.

10.10. Edit the Hosts File

While you're waiting for the provisioning process to complete, take a few minutes to update the **hosts** file on your local system. This will allow you to access the Million Song Library application using a Web browser.

1. In your EC2 Management Console, click **Instances** to view all running instances. You should see the **Production-aws** instance created by Vagrant.
2. Copy the **Public IP** of the instance.
3. In a text editor, open the **hosts** file on your computer. The location of the **hosts** file depends on your operating system:

Operating Syystem	**Location of hosts file**
Mac	/private/etc/hosts
Linux	/etc/hosts
Windows	C:\Windows\System32\Drivers\etc

4. Add the following line to the **hosts** file where *xxx.xxx.xxx.xxx* is the public IP address of the EC2 instance.

```
xxx.xxx.xxx.xxx     msl.kenzanlabs.com
```

5. Save and close the **hosts** file.

10.11. Start the MSL Demo

Provisioning is complete — now it's time to start the Million Song Library demo.

1. Open a new Terminal or PowerShell window and make sure you are in your home directory.
2. Enter the following command to connect to the EC2 instance:

```
vagrant ssh prod
```

3. After you are connected to the instance, enter the following command to change to the application directory:

```
cd million-song-library/msl-pages
```

4. Enter the following command to start the application front end:

```
npm run deploy
```

Wait for the front end to start up — this will take just a couple of minutes.

5. Open a new Terminal or PowerShell window and make sure you are in your home directory.
6. Enter the following command to change to the application directory:

```
cd million-song-library/msl-pages
```

7. Enter the following command to start the server instances:

```
npm run serve-all
```

If your EC2 instance is low on free memory, you can start up just the core server instances by running the `npm run catalog-edge-server` command instead.

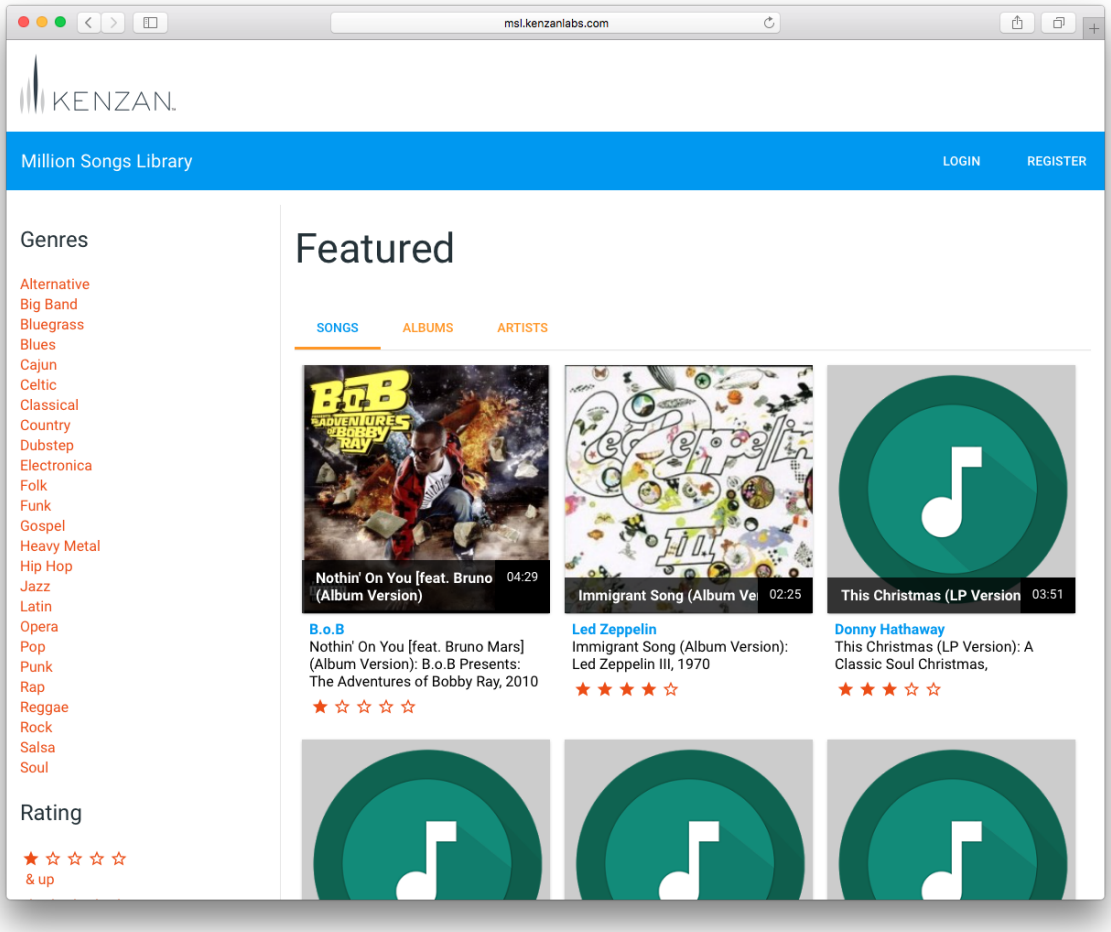
Wait for the server instances to start up — again, this will take just a couple of minutes.

8. Open a Web browser and point it to: **msl.kenzanlabs.com:3000**

The Million Song Library home page displays. (If you don't see data right away, wait a couple of minutes and then refresh the page.)

If the Million Song Library home page does not display, make sure that you allowed all inbound traffic in your AWS security group and that you correctly edited the **hosts** file.

Million Song Library Home Page



Now that the Million Song Library demo is working, here are some fun things to try:

- Click **Register** to register for an account.
- Click the labels to switch the view between **Songs**, **Albums**, and **Artists**.
- Click a genre or star rating on the left to filter songs. (Click the **x** to clear a filter.)
- Mouse over a song and click **Add to library** to add it to your music library (you must be logged in).
- Click the stars below a song to rate it (you must be logged in).

10.12. Stop the MSL Demo

Done having fun for now? Here's how to terminate the EC2 instance:

1. Open a new Terminal or PowerShell window and make sure you are in your home directory.
2. Enter the following command to change to the million-song-library repository:

```
cd million-song-library
```

3. Enter the following command to destroy the EC2 instance:

```
vagrant destroy prod -f
```

The EC2 instance is terminated. If you want to run the demo again, just follow this procedure to create a new Million Song Library instance any time.