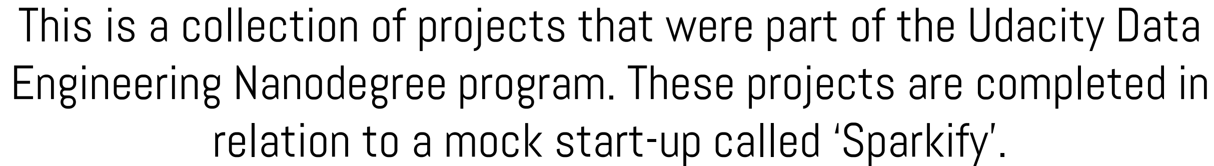
Data Engineering Nanodegree



Sparkify is a music streaming application that wants to start analyzing their songs, song plays and user data. So using modern data engineering tools I built 4 projects to assist Sparkify in accomplishing it's goals.

Projects

01. Data Modeling

Data Modeling with PostgreSQL & Cassandra

03. Data Lakes

Data Lakes with AWS & Apache Spark

02. Data Warehouse

Data Warehousing with Amazon Redshift

04. Data Pipelines

Data Pipelines with Apache Airflow

Datasets

Song Dataset

The first dataset is a subset of real data from the Million Song Dataset. Each file is in JSON format and contains metadata about a song and the artist of that song. The files are partitioned by the first three letters of each song's track ID. For example, here are filepaths to two files in this dataset.

```
song_data/A/B/C/TRABCEI128F424C983.json
song_data/A/A/B/TRAABJL12903CDCF1A.json
```

And below is an example of what a single song file, TRAABJL12903CDCF1A.json, looks like.


```
{"num_songs": 1, "artist_id": "ARJIE2Y1187B994AB7", "artist_latitude": null, "artist_longitude": null, "a
```


Fact Table

1. **songplays** - records in log data associated with song plays i.e. records with page `NextSong`
   - *songplay_id, start_time, user_id, level, song_id, artist_id, session_id, location, user_agent*

Dimension Tables

2. **users** - users in the app
   - *user_id, first_name, last_name, gender, level*
3. **songs** - songs in music database
   - *song_id, title, artist_id, year, duration*
4. **artists** - artists in music database
   - *artist_id, name, location, lattitude, longitude*
5. **time** - timestamps of records in **songplays** broken down into specific units
   - *start_time, hour, day, week, month, year, weekday*

Log Dataset

The second dataset consists of log files in JSON format generated by this event simulator based on the songs in the dataset above. These simulate app activity logs from an imaginary music streaming app based on configuration settings.

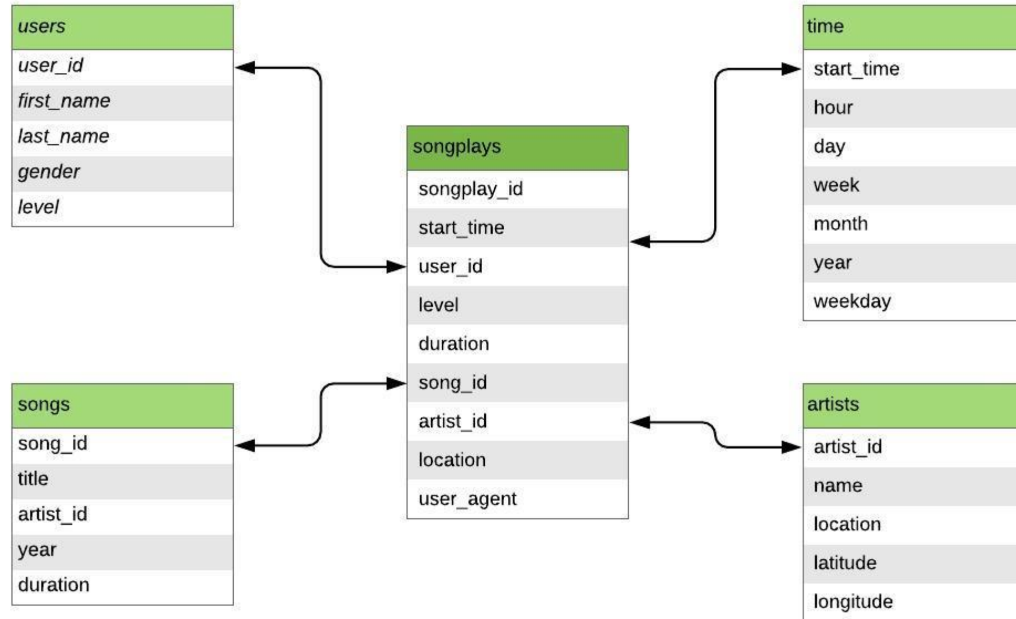
The log files in the dataset you'll be working with are partitioned by year and month. For example, here are filepaths to two files in this dataset.

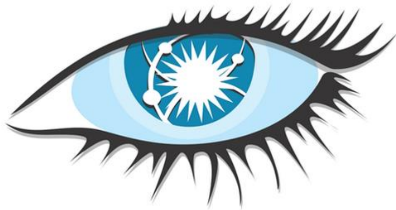
```
log_data/2018/11/2018-11-12-events.json
log_data/2018/11/2018-11-13-events.json
```

And below is an example of what the data in a log file, 2018-11-12-events.json, looks like.

	artist	auth	firstName	gender	itemInSession	lastName	length	level	location	method	page	registration	sessionId	song	status	ts	userAgent	userId
0	None	Logged In	Celeste	F	0	Williams	NaN	free	Klamath Falls, OR	GET	Home	1.541078e+12	438	None	200	1541990217796	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	53
1	Pavement	Logged In	Sylvie	F	0	Cruz	99.16036	free	Washington-Arlington-Alexandria, DC-VA-MD-WV	PUT	NextSong	1.540266e+12	345	Mercy:The Laundromat	200	1541990258796	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	10
2	Barry Tuckwell/Academy of St Martin-in-the-Fie...	Logged In	Celeste	F	1	Williams	277.15873	free	Klamath Falls, OR	PUT	NextSong	1.541078e+12	438	Horn Concerto No. 4 in E flat K495: II. Romanc...	200	1541990264796	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	53
3	Gary Allan	Logged In	Celeste	F	2	Williams	211.22567	free	Klamath Falls, OR	PUT	NextSong	1.541078e+12	438	Nothing On But The Radio	200	1541990541796	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK...	53
4	None	Logged In	Jacqueline	F	0	Lynch	NaN	paid	Atlanta-Sandy Springs-Roswell, GA	GET	Home	1.540224e+12	389	None	200	1541990714796	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4...	29

Entity Relationship Diagram





Project #1:

Data Modeling with PostgreSQL & Cassandra

Data Modeling with PostgreSQL

A startup called 'Sparkify' wants to analyze the data they've been collecting on songs and user activity on their new music streaming app. The analytics team is particularly interested in understanding what songs users are listening to. Currently, they don't have an easy way to query their data, which resides in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

The task at hand was to create a PostgreSQL database utilizing a star schema with fact and dimension tables designed to optimize queries and create a simple ETL pipeline that transfers data from files in two local directories into these tables for song play analysis and then test our database utilizing SQL queries provided by the analytics teams, to see if we can return the expected results.

Data Modeling with PostgreSQL

PostgreSQL

```
12
13  def process_song_file(cur, filepath):
14      # open song file
15      df = pd.read_json(filepath, lines=True)
16
17      # insert song record
18      song_data = df[['song_id', 'title', 'artist_id', 'year', 'duration']].values[0].tolist()
19      song_data = (song_data[0], song_data[1], song_data[2], song_data[3], song_data[4])
20      try:
21          cur.execute(song_table_insert, song_data)
22      except:
23          pass
24
25      # insert artist record
26      artist_data = df[['artist_id','artist_name','artist_location', 'artist_latitude', 'artist_longitude']].values[0].tolist()
27      artist_data = (artist_data[0], artist_data[1], artist_data[2], artist_data[3], artist_data[4])
28      try:
29          cur.execute(artist_table_insert, artist_data)
30      except:
31          pass
32
33      """ ^^^ This function reads in our json data from the song_data folder and using our sql_queries.py script,
34              inserts columns into the song and artist tables.
35
36              song_data= ('song_id', 'title', 'artist_id', 'year', 'duration')
37              artist_data= ('artist_id','artist_name','artist_location', 'artist_latitude', 'artist_longitude') ^^^ """
38
```

```
df = pd.read_json(filepath,lines=True)
df.head()
```

artist	auth	firstName	gender	itemInSession	lastName	length	level	location	method	page	registration	sessionId	song	statu
Sydney Youngblood	Logged In	Jacob	M	53	Klein	238.07955	paid	Tampa-St. Petersburg-Clearwater, FL	PUT	NextSong	1.540558e+12	954	Ain't No Sunshine	200
Gang Starr	Logged In	Layla	F	88	Griffin	151.92771	paid	Lake Havasu City-Kingman, AZ	PUT	NextSong	1.541057e+12	984	My Advice 2 You (Explicit)	200
3OH!3	Logged In	Layla	F	89	Griffin	192.52200	paid	Lake Havasu City-Kingman, AZ	PUT	NextSong	1.541057e+12	984	My First Kiss (Feat. Ke$ha) [Album Version]	200
RÃ□Â¶yksopp	Logged In	Jacob	M	54	Klein	369.81506	paid	Tampa-St. Petersburg-Clearwater, FL	PUT	NextSong	1.540558e+12	954	The Girl and The Robot	200
Kajagoogoo	Logged In	Layla	F	90	Griffin	223.55546	paid	Lake Havasu City-Kingman, AZ	PUT	NextSong	1.541057e+12	984	Too Shy	200

[*Go to project Github]

Data Modeling with Cassandra

A startup called ‘Sparkify’ wants to analyze the data they've been collecting on songs and user activity on their new music streaming app. The analytics team is particularly interested in understanding what songs users are listening to. Currently, they don't have an easy way to query their data, which resides in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

In this project, I was tasked with creating a NoSQL database using Apache Cassandra and complete an ETL pipeline with Python. Again, we will be able to test our database model using provided SQL queries to make sure we return the expected results.

Data Modeling with Cassandra

Data Modeling with Apache Cassandra

Udacity Data Engineering Nanodegree | Project 2 | Data Modeling with Cassandra | Manny Brar

In [ ]:

Import Python packages

In [33]:
```
# Import Python packages
import pandas as pd
import cassandra
import re
import os
import glob
import numpy as np
import json
import csv
```

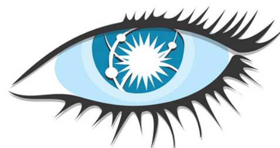
Creating list of filepaths to process original event csv data files

In [34]:
```
# checking your current working directory
print(os.getcwd())

# Get your current folder and subfolder event data
filepath = os.getcwd() + '/event_data'

# Create a for loop to create a list of files and collect each filepath
for root, dirs, files in os.walk(filepath):

# join the file path and roots with the subdirectories using glob
    file_path_list = glob.glob(os.path.join(root,'*'))
    #print(file_path_list)
```

```
/home/workspace
```

Processing the files to create the data file csv that will be used for Apache Casssandra tables

In [35]:
```
# initiating an empty list of rows that will be generated from each file
full_data_rows_list = []

# for every filepath in the file path list
for f in file_path_list:

# reading csv file
    with open(f, 'r', encoding = 'utf8', newline='') as csvfile:
        # creating a csv reader object
        csvreader = csv.reader(csvfile)
        next(csvreader)

 # extracting each data row one by one and append it
        for line in csvreader:
            #print(line)
            full_data_rows_list.append(line)


# creating a smaller event data csv file called event_datafile_full csv that will be used to insert data into the \
# Apache Cassandra tables
csv.register_dialect('myDialect', quoting=csv.QUOTE_ALL, skipinitialspace=True)

with open('event_datafile_new.csv', 'w', encoding = 'utf8', newline='') as f:
    writer = csv.writer(f, dialect='myDialect')
    writer.writerow(['artist','firstName','gender','itemInSession','lastName','length',\
                'level','location','sessionId','song','userId'])
    for row in full_data_rows_list:
        if (row[0] == ''):
            continue
        writer.writerow((row[0], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[12], row[13], row[16]))
```



In [40]:
```
# CREATE TABLES
query1="CREATE TABLE IF NOT EXISTS songplay_"
query1= query1 + "(sessionId int, itemInSession int, artist_name text, song text, length decimal,  PRIMARY KEY (sessionId, itemInSessio
n))"
try:
    session.execute(query1)
except Exception as e:
    print(e)
    """ ^^^^ Here we create a new table if it does not currently exist and we assign it as songplay_,
    we then specify our columns and data types for each column as well as setting our
    PRIMARY KEY and Partition key ^^^"""
```

In [41]:
```
file = 'event_datafile_new.csv'

with open(file, encoding = 'utf8') as f:
    csvreader = csv.reader(f)
    next(csvreader) # skip header
    for line in csvreader:
        query1 = "INSERT INTO songplay_ (sessionId, itemInSession, artist_name, song, length)"
        query1 = query1 + "VALUES (%s, %s, %s, %s, %s)"
        session.execute(query1, (int(line[8]), int(line[3]), line[0], line[9], float(line[5])))

    """ ^^^^ Next we read in the CSV file and Insert the data into the songplay_ table we created ^^^"""
```

In [42]:
```
query1="SELECT artist_name, length, song FROM songplay_ WHERE sessionId = 338 AND itemInSession = 4"
try:
    rows=session.execute(query1)
except Exception as e:
    print(e)


print('Artist Name','|', 'Length', '|', 'Song Title')
for row in rows:
    print(row.artist_name,'|', row.length,'|', row.song)

    """ ^^^^ Here we execute our SELECT statement to answer question 1 and we print the results below ^^^"""
```

```
Artist Name | Length | Song Title
Faithless | 495.3073 | Music Matters (Mark Knight Dub)
```

*Go to project Github

Project #2:

Data Warehousing with Amazon Redshift

Data Warehouse with Amazon Redshift

A music streaming startup 'Sparkify', has grown their user base and song database and want to move their processes and data onto the cloud. Their data resides in S3, in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

As the data engineer I built an ETL pipeline on AWS that extracts their data from S3, stages them into a Redshift database, and transforms data into a set of dimensional tables for their analytics team to continue finding insights in what songs their users are listening to. To test the database and ETL pipeline I run SQL queries provided by the analytics team from Sparkify and compare your results with their expected results.

Data Warehouse with Amazon Redshift

```python
import configparser
import psycopg2
from sql_queries import copy_table_queries, insert_table_queries


def load_staging_tables(cur, conn):
    """Load log_data & song_data from S3 Bucket and insert into staging_events &   staging_songs"""
    for query in copy_table_queries:
        print('Loading query data: '+query)
        cur.execute(query)
        conn.commit()


def insert_tables(cur, conn):
    """INSERT data from staging tables to
the star schema, dimension and fact tables"""
    for query in insert_table_queries:
        print('Processing query:'+query)
        cur.execute(query)
        conn.commit()


def main():
    config = configparser.ConfigParser()
    config.read('dwh.cfg')

    print('Connecting to redshift')
    conn = psycopg2.connect("host={} dbname={} user={} password={} port={}".format(*config['CLUSTER'].values()))
    print('Connected to redshift')
    cur = conn.cursor()

    print('Loading staging tables')
    #load_staging_tables(cur, conn)

    print('Transform from staging')
    insert_tables(cur, conn)

    conn.close()
    print('ETL Ended')


if __name__ == "__main__":
    main()
```

amazon
REDSHIFT

*Go to project Github

Project #3:

Data Lakes with AWS & Spark

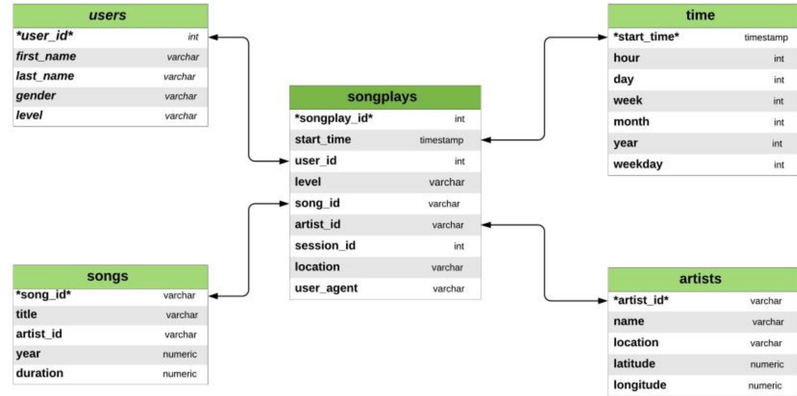
A music streaming startup, Sparkify, has grown their user base and song database even more and want to move their data warehouse to a data lake. Their data resides in S3, in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

As the data engineer, I built an ETL pipeline for a data lake hosted on S3, that extracts the data from S3, processes it using Apache Spark, and loads the data back into S3 as a set of dimensional tables. This will allow the analytics team to continue finding insights in what songs their users are listening to.

Next I'll be able to test the database and ETL pipeline by running queries given by the analytics team from Sparkify and compare the results with their expected results.

Data Lakes with AWS & Apache Spark

```
26
27 def create_spark_session():
28     spark = SparkSession \
29         .builder \
30         .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:2.7.0") \
31         .getOrCreate()
32     return spark
33 print('Creating Spark session: COMPLETE')
34
35 """^^^Creating Spark session for data processing, if it does not currently exist^^^"""
36
37
38 def process_song_data(spark, input_data, output_data):
39     print('Processing song_data from S3 bucket...')
40     """
41     This function will:  Extract and process song data to
42     create 2 dimensional tables (songs_table & artists_table)
43     """
44     song_data = f'{input_data}/song_data/A/A/A/*.json'
45     df = spark.read.json(song_data)
46     print('Reading song_data from S3 bucket: COMPLETE')
47
48     """^^^Read in song_data from Sparkify S3 bucket and assign it as df^^^"""
49
50     songs_table = df.select('song_id', 'title', 'artist_id',
51                             'year', 'duration').dropDuplicates()
52     songs_table.printSchema()
53     songs_table.show(5)
54     songs_table.write.parquet(f'{output_data}/songs_table',
55                               mode='overwrite',
56                              partitionBy=['year','artist_id'])
57     print('Write songs_table to parquet files & partition by year & artist_id: COMPLETE')
58
59     """^^^Extract columns('song_id', 'title', 'artist_id', 'year', 'duration')
60     from df and assign it to songs_table.
61     Write songs_table to parquet files for output and partition by 'year','artist_id'^^^"""
62
```



[*Go to project Github]

Project #4:

Data Pipelines with Apache Airflow

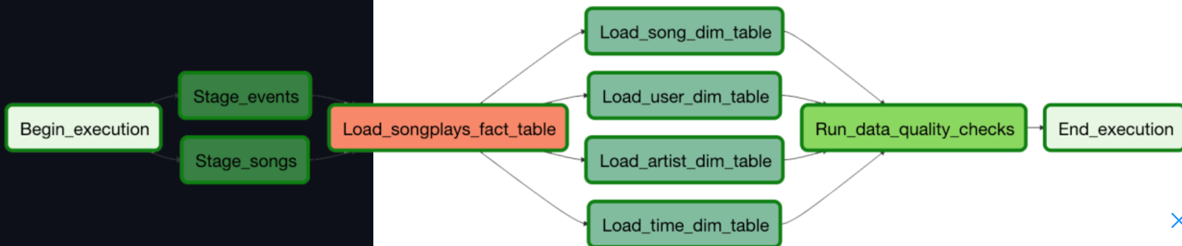
A music streaming company, Sparkify, has decided that it is time to introduce more automation and monitoring to their data warehouse ETL pipelines and concluded that the best tool to achieve this is Apache Airflow.

In this project I create high grade data pipelines that are dynamic and built from reusable tasks, can be monitored, and allow easy backfills. Sparkify have also noted that data quality plays a big part when analyses are executed on top the data warehouse and want to run tests against their datasets after the ETL steps have been executed to catch any discrepancies in the datasets.

The source data resides in S3 and needs to be processed in Sparkify's data warehouse in Amazon Redshift. The source datasets consist of JSON logs of user activity in the application and JSON metadata about the songs the users listen to.

Data Pipelines with Apache Airflow

```python
import os
from datetime import datetime, timedelta

from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators import (StageToRedshiftOperator, LoadFactOperator,
                                LoadDimensionOperator, DataQualityOperator)
from helpers import SqlQueries

default_args = {
    "owner": "Manny",
    "depends_on_past": False,
    "retries": 3,
    "retry_delay": timedelta(minutes=5),
    "start_date": datetime(2020, 8, 21),
    "end_date": datetime(2020, 8, 23),
    "email_on_retry": False
}

dag = DAG("S3_to_Redshift",
          default_args=default_args,
          description="Load and transform data in Redshift with Airflow",
          schedule_interval="0 * * * *",
          catchup=False
        )

```



*Go to project Github

VERIFIED CERTIFICATE OF COMPLETION

September 07, 2020

Manny Brar

Has successfully completed the

Data Engineering Nanodegree

NANODEGREE PROGRAM

Sebastian Thrun
Founder, Udacity

Udacity has confirmed the participation of this individual in this program.
Confirm program completion at confirm.udacity.com/DEK3K7NP