Project: Data Lake with EMR (Spark & Co. @ AWS)

Creating EMR Cluster (`"emr-5.31.0" - Spark 2.4.6` , `Zeppelin 0.8.2` , `Hadoop 2.10.0`) of AWS
This last EMR Version does have the correct setup up with `JupyterEnterpriseGateway` !

Overview and Steps

- Step 0: Create Base Config (Setup IAM-Role [dl.cfg], Setup EMR Cluster (with Spark), Create Notebook)
- Step 1: Create Spark Session
- Step 2: Define Data Paths and Access Data
- Step 3: Process Song Data (read out, define schemas, write as parquet)
- Step 4: Process Log Data (read out, define schemas, write as parquet)
- Step 5: Run [`etl.py`] script with clean code
- Step 6: Clean up Resources (EMR, Notbook, IAM-Role)

Clear fields at [`dl.cfg`] before submiting project for review!
KEY = \`YOUR_AWS_KEY\` SECRET = \`YOUR_AWS_SECRET\`

*Version 2 **(LOC)** - Revision 06 - 2023/03/20 Mr Morphy - [GitHub Profile (https://github.com/MrMorphy)](https://github.com/MrMorphy)*
GitHub Project - [udacity-course-proj-data-lake (https://github.com/MrMorphy/udacity-course-proj-data-lake)](https://github.com/MrMorphy/udacity-course-proj-data-lake)

Step 0: Create Base Config

- Setup IAM-Role [dl.cfg]
- Setup EMR Cluster (with Spark) at AWS UI
- Create Notebook at AWS UI

Load libraries to execute code

In [1]:

```
import configparser
from datetime import datetime
import os
```

In [2]:

```
from pyspark.sql import SparkSession
```

In [3]:

```
from pyspark.sql.functions import udf, col, monotonically_increasing_id
from pyspark.sql.functions import year, month, dayofmonth, hour, weekofyear, date_format
```

In [4]:

```
from pyspark.sql.types import StructType as R,    StructField as Fld, \
                              DoubleType as Dbl,  StringType as Str, \
                              IntegerType as Int, DateType as Date, TimestampType
```

In [5]:

```
# only for jupyter workbook tests necessary
import pandas as pd
```

In [21]:

```
# AWS: behaviour? Pending!
config = configparser.ConfigParser()
config.read('dl.cfg')
print('>> Read Out Config-Infos from [dl.cfg]')

os.environ['AWS_ACCESS_KEY_ID']     = config['AWS_ACCESS_KEY_ID']
os.environ['AWS_SECRET_ACCESS_KEY'] = config['AWS_SECRET_ACCESS_KEY']
```

```
---------------------------------------------------------------------------
MissingSectionHeaderError                 Traceback (most recent call last)
<ipython-input-21-73b7fd6b2e87> in <module>()
      1 config = configparser.ConfigParser()
----> 2 config.read('dl.cfg')
      3 print('>> Read Out Config-Infos from [dl.cfg]')
      4 
      5 os.environ['AWS_ACCESS_KEY_ID']     = config['AWS_ACCESS_KEY_ID']

/opt/conda/lib/python3.6/configparser.py in read(self, filenames, encoding)
    695             try:
    696                 with open(filename, encoding=encoding) as fp:
--> 697                     self._read(fp, filename)
    698             except OSError:
    699                 continue

/opt/conda/lib/python3.6/configparser.py in _read(self, fp, fpname)
   1078                 # no section header in the file?
   1079                 elif cursect is None:
-> 1080                     raise MissingSectionHeaderError(fpname, lineno,
line)
   1081                 # an option line?
   1082                 else:

MissingSectionHeaderError: File contains no section headers.
file: 'dl.cfg', line: 1
'AWS_ACCESS_KEY_ID=AKIAQ4TUNEC3HLIPRATG\n'
```

INFO: Add [AWS] on dl.cfg File as well at code " =config['AWS'][...] "

In [6]:

```
config = configparser.ConfigParser()
config.read('dl.cfg')
print('>> Read Out Config-Infos from [dl.cfg]')

os.environ['AWS_ACCESS_KEY_ID']     = config['AWS']['AWS_ACCESS_KEY_ID']
os.environ['AWS_SECRET_ACCESS_KEY'] = config['AWS']['AWS_SECRET_ACCESS_KEY']
```

```
>> Read Out Config-Infos from [dl.cfg]
```

In [ ]:

```
# AWS: only for Jupyter Notebook execution @ AWS, but NOT @ [etl.py] required
os.environ['AWS_ACCESS_KEY_ID']     = '<AWS_ACCESS_KEY_ID>'
os.environ['AWS_SECRET_ACCESS_KEY'] = '<AWS_SECRET_ACCESS_KEY>'
```

Step 1: Create Spark Session

In [7]:

```
print('>> START (main)')
```

```
>> START (main)
```

In [8]:

```
#def create_spark_session():
spark = SparkSession \
    .builder \
    .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:2.10.0") \
    .getOrCreate()
#return spark
# @ EMR (AWS): "emr.5.20.0" > "hadopp 2.7.0"
# @ EMR (AWS): "emr-5.31.0" > "hadoop 2.10.0"
print('>> spark session created')
```

```
>> spark session created
```

In [9]:

```
# DEBUG
spark
```

Out[9]:

SparkSession - in-memory

SparkContext

[Spark UI (http://202c6ecc0754:4041)](http://202c6ecc0754:4041)

Version

`v2.4.3`

Master

`local[*]`

AppName

`pyspark-shell`

Step 2: Define Data Paths and Access Data

In [10]:

```
# AWS: Online @ AWS - S3:
input_data = "s3a://udacity-dend/"
output_data = "s3a://udacity-project-data-lake-sparkify/output/"
```

In [10]:

```
# local data ./data/A/A/A/*.json
input_data  = "data/"
output_data = "data_output/"
```

Step 3: Process Song Data

(read out, define schemas, write as parquet)

In [11]:

```
print('>> processing song data')
```

```
>> processing song data
```

In [12]:

```
# AWS:
#def process_song_data(spark, input_data, output_data):

# get filepath to song data file [song_data/A/B/C/TRABCEI128F424C983.json]
song_data = input_data + 'song_data/*/*/*/*.json'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-4a56d3830b03> in <module>()
      2 
      3 # get filepath to song data file [song_data/A/B/C/TRABCEI128F424C98
3.json]
----> 4 song_data = input_data + 'song_data/*/*/*/*.json'

NameError: name 'input_data' is not defined
```

In [23]:

```
# 1) experiment with a subset of the files,
# 2) and with local data (extract before from /data/song-data.zip)
song_data_loc = input_data + 'song_data/A/A/A/*.json'
```

In [28]:

```
# EINMALIG!
# 3) for df.join(df.song == song_df.title) ALL required
# song_data_loc = input_data_local + 'song-data.zip'
import zipfile
with zipfile.ZipFile("data/song-data.zip") as zip_ref:
    zip_ref.extractall("data/song_data_unzipped/")
    # /data/song_data_unzipped/song_data/A/[A|B]/[A|B|C]/*.json
```

In [12]:

```
# get filepath to song data file [song_data/A/B/C/TRABCEI128F424C983.json]
song_data = input_data + 'song_data_unzipped/song_data/*/*/*/*.json'
```

In [13]:

```
# define the song schema (like [staging_songs] table)
songSchema = R([
   Fld("artist_id",        Str()),
   Fld("artist_latitude",  Dbl()),
   Fld("artist_location",  Str()),
   Fld("artist_longitude", Dbl()),
   Fld("artist_name",      Str()),
   Fld("duration",         Dbl()),
   Fld("num_songs",        Int()),
   Fld("song_id",          Str()),
   Fld("title",            Str()),
   Fld("year",             Int()),
])
```

In [18]:

```
# AWS:
# read song data JSON file into data frame
df = spark.read.json(song_data, schema=songSchema)
# ERR, maybe because of S3FullAccess.
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-18-68542580ee9a> in <module>()
      1 # read song data JSON file into data frame
----> 2 df = spark.read.json(song_data, schema=songSchema)

/opt/spark-2.4.3-bin-hadoop2.7/python/pyspark/sql/readwriter.py in json(sel
f, path, schema, primitivesAsString, prefersDecimal, allowComments, allowUn
quotedFieldNames, allowSingleQuotes, allowNumericLeadingZero, allowBackslas
hEscapingAnyCharacter, mode, columnNameOfCorruptRecord, dateFormat, timesta
mpFormat, multiLine, allowUnquotedControlChars, lineSep, samplingRatio, dro
pFieldIfAllNull, encoding)
    272             path = [path]
    273         if type(path) == list:
--> 274             return self._df(self._jreader.json(self._spark._sc._jv
m.PythonUtils.toSeq(path)))
    275         elif isinstance(path, RDD):
    276             def func(iterator):

/opt/spark-2.4.3-bin-hadoop2.7/python/lib/py4j-0.10.7-src.zip/py4j/java_gat
eway.py in __call__(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
-> 1257             answer, self.gateway_client, self.target_id, self.name)
   1258 
   1259         for temp_arg in temp_args:

/opt/spark-2.4.3-bin-hadoop2.7/python/pyspark/sql/utils.py in deco(*a, **k
w)
     61     def deco(*a, **kw):
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:
     65             s = e.java_exception.toString()

/opt/spark-2.4.3-bin-hadoop2.7/python/lib/py4j-0.10.7-src.zip/py4j/protoco
l.py in get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o146.json.
: java.lang.NoClassDefFoundError: org/apache/hadoop/fs/StorageStatistics
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:348)
        at org.apache.hadoop.conf.Configuration.getClassByNameOrNull(Config
uration.java:2134)
        at org.apache.hadoop.conf.Configuration.getClassByName(Configuratio
n.java:2099)
        at org.apache.hadoop.conf.Configuration.getClass(Configuration.jav
a:2193)
        at org.apache.hadoop.fs.FileSystem.getFileSystemClass(FileSystem.ja
va:2654)
        at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.jav
a:2667)
        at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:94)
        at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.jav
a:2703)
        at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:2685)
        at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:373)
        at org.apache.hadoop.fs.Path.getFileSystem(Path.java:295)
        at org.apache.spark.sql.execution.streaming.FileStreamSink$.hasMeta
data(FileStreamSink.scala:45)
        at org.apache.spark.sql.execution.datasources.DataSource.resolveRel
ation(DataSource.scala:332)
        at org.apache.spark.sql.DataFrameReader.loadV1Source(DataFrameReade
r.scala:223)
        at org.apache.spark.sql.DataFrameReader.load(DataFrameReader.scala:
211)
        at org.apache.spark.sql.DataFrameReader.json(DataFrameReader.scala:
391)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessor
Impl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethod
AccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:35
7)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:
132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.ClassNotFoundException: org.apache.hadoop.fs.StorageSt
atistics
        at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        ... 28 more
```

In [14]:

```
# read song data JSON file into data frame
df = spark.read.json(song_data, schema=songSchema)
```

In [15]:

```
# DEBUG
# songSchema given
df.printSchema()
```

```
root
 |-- artist_id: string (nullable = true)
 |-- artist_latitude: double (nullable = true)
 |-- artist_location: string (nullable = true)
 |-- artist_longitude: double (nullable = true)
 |-- artist_name: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- num_songs: integer (nullable = true)
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- year: integer (nullable = true)
```

In [16]:

```
# df.count()
print('>> [' + str(df.count()) + '] songs from song_data read out in JSON-format')
```

```
>> [71] songs from song_data read out in JSON-format
```

In [17]:

```
# DEBUG
df.limit(5).toPandas()
```

Out[17]:

	artist_id	artist_latitude	artist_location	artist_longitude	artist_name	dur
0	ARDR4AC1187FB371A1	NaN		NaN	Montserrat Caballé;Placido Domingo;Vicente Sar...	511.1
1	AREBBGV1187FB523D2	NaN	Houston, TX	NaN	Mike Jones (Featuring CJ_ Mello & Lil' Bran)	173.6
2	ARMAC4T1187FB3FA4C	40.82624	Morris Plains, NJ	-74.47995	The Dillinger Escape Plan	207.7
3	ARPBNLO1187FB3D52F	40.71455	New York, NY	-74.00712	Tiny Tim	43.3
4	ARDNS031187B9924F0	32.67828	Georgia	-83.22295	Tim Wilson	186.4

In [18]:

```
# DEBUG
df1 = df.filter(df.title == 'Young Boy Blues')
df1.limit(5).toPandas()
```

Out[18]:

	artist_id	artist_latitude	artist_location	artist_longitude	artist_name	durati
0	ARGSJW91187B9B1D6B	35.21962	North Carolina	-80.01955	JennyAnyKind	218.775

In [52]:

```
# read song data JSON file into data frame
dfSdLoc = spark.read.json(song_data)
```

In [53]:

```
# DEBUG
# default Schema at read JSON
dfSdLoc.printSchema()
```

```
root
 |-- artist_id: string (nullable = true)
 |-- artist_latitude: double (nullable = true)
 |-- artist_location: string (nullable = true)
 |-- artist_longitude: double (nullable = true)
 |-- artist_name: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- num_songs: long (nullable = true)
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- year: long (nullable = true)
```

In [54]:

```
# DEBUG
dfSdLoc.limit(5).toPandas()
```

Out[54]:

	artist_id	artist_latitude	artist_location	artist_longitude	artist_name	durati
0	ARKFYS91187B98E58F	NaN		NaN	Jeff And Sheri Easter	267.702
1	ARGSJW91187B9B1D6B	35.21962	North Carolina	-80.01955	JennyAnyKind	218.775
2	ARD7TVE1187B99BFB1	NaN	California - LA	NaN	Casual	218.931

In [19]:

```
# extract columns to create songs table
song_columns = ["song_id", "title", "artist_id", "year", "duration"]
songs_table = df.select(song_columns) \
                .dropDuplicates()
```

In [20]:

```
# DEBUG
songs_table.printSchema()6
```

```
root
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- artist_id: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- duration: double (nullable = true)
```

In [21]:

```
# DEBUG
songs_table.limit(5).toPandas()
```

Out[21]:

	song_id	title	artist_id	year	duration
0	SOHUOAP12A8AE488E9	Floating	ARD842G1187B997376	1987	491.12771
1	SOKEJEJ12A8C13E0D0	The Urgency (LP Version)	ARC43071187B990240	0	245.21098
2	SONHOTT12A8C13493C	Something Girls	AR7G5I41187FB4CE6C	1982	233.40363
3	SOHKNRJ12A6701D1F8	Drop of Rain	AR10USD1187B99F3F1	0	189.57016
4	SOOLYAZ12A6701F4A6	Laws Patrolling (Album Version)	AREBBGV1187FB523D2	0	173.66159

In [22]:

```
# [-] VERIFICAR @ AWS...
# write songs table to parquet files partitioned by year and artist
songs_table.write.partitionBy('year', 'artist_id') \
                 .parquet(output_data + 'songs_data/songs_table.parquet', 'overwrite')
```

In [ ]:

```
# ??? Como me puedo alistar el contenido del directorio, para ver todo lo que ha echo?
# .. Que lo hizo correctamente! ;-)
```

In [23]:

```
# extract columns to create artists table
artists_column = ["artist_id", "artist_name as name", "artist_location as location", \
                  "artist_latitude as latitude", "artist_longitude as longitude"]
artists_table = df.selectExpr(artists_column).dropDuplicates()
```

In [24]:

```
# DEBUG
artists_table.printSchema()
```

```
root
 |-- artist_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- location: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
```

In [25]:

```
# DEBUG
artists_table.limit(5).toPandas()
```

Out[25]:

	artist_id	name	location	latitude	longitude
0	ARPBNLO1187FB3D52F	Tiny Tim	New York, NY	40.71455	-74.00712
1	ARXR32B1187FB57099	Gob		NaN	NaN
2	AROGWRA122988FEE45	Christos Dantis		NaN	NaN
3	ARBGXIG122988F409D	Steel Rain	California - SF	37.77916	-122.42005
4	AREVWGE1187B9B890A	Bitter End	Noci (BA)	-13.44200	-41.99520

In [26]:

```
# write artists table to parquet files
artists_table.write.parquet(output_data + 'artists_data/artists_table.parquet', 'overwri
te')
```

In [ ]:

```
# ??? Como me puedo alistar el contenido del directorio, para ver todo lo que ha echo?
# .. Que lo hizo correctamente! ;-)
```

Step 4: Process Log Data

(read out, define schemas, write as parquet)

In [27]:

```
print('>> log data processed')
```

```
>> log data processed
```

In [ ]:

```
# AWS:
#def process_log_data(spark, input_data, output_data):

# get filepath to log data file [log_data/2018/11/2018-11-17-events.json]
log_data = input_data + 'log_data/*/*/*.json'
```

In [45]:

```
# 1) experiment with a subset of the files,
# 2) and with local data (extract before from /data/log-data.zip)
log_data = input_data + 'log_data/2018/11/*.json'
```

In [68]:

```
# EINMALIG >>
# 3) for df.join(df.song == song_df.title) ALL required
# song_data_loc = input_data_local + 'song-data.zip'
import zipfile
with zipfile.ZipFile("data/log-data.zip") as zip_ref:
    zip_ref.extractall("data/log_data_unzipped/")
    # /data/logg_data_unzipped/*.json
```

In [28]:

```
# 3) and with local data (extract before from /data/log-data.zip)
log_data = input_data + 'log_data_unzipped/*.json'
```

In [29]:

```
# read log data file into data frame
df = spark.read.json(log_data)
```

In [30]:

```
# number of lines
# df.count()
print('>> [' + str(df.count()) + '] logs entries read IN, of JSON logs_data')
```

```
>> [8056] logs entries read IN, of JSON logs_data
```

In [31]:

```
# DEBUG
df.printSchema()
```

```
root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: double (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
```

In [32]:

```
# DEBUG
df.limit(5).toPandas()
```

Out[32]:

	artist	auth	firstName	gender	itemInSession	lastName	length	level	location
0	Harmonia	Logged In	Ryan	M	0	Smith	655.77751	free	San Jose-Sunnyvale-Santa Clara, CA
1	The Prodigy	Logged In	Ryan	M	1	Smith	260.07465	free	San Jose-Sunnyvale-Santa Clara, CA
2	Train	Logged In	Ryan	M	2	Smith	205.45261	free	San Jose-Sunnyvale-Santa Clara, CA
3	None	Logged In	Wyatt	M	0	Scott	NaN	free	Eureka-Arcata-Fortuna, CA
4	None	Logged In	Austin	M	0	Rosales	NaN	free	New York-Newark-Jersey City, NY-NJ-PA

In [33]:

```
# filter by actions for song plays
df = df.filter(df.page == 'NextSong')
```

In [34]:

```
# number of reduce amount of lines
# df.count()
print('>> [' + str(df.count()) + '] songs filtered "NextSong" of logs_data')
```

```
>> [6820] songs filtered "NextSong" of logs_data
```

In [35]:

```
# extract columns for USERS table
# artists_table =  # TYPO at resources (*.zip)!
users_columns = ["userId as user_id", "firstName as first_name", \
                 "lastName as last_name", "gender", "level"]
users_table = df.selectExpr(users_columns).dropDuplicates()
```

In [36]:

```
# DEBUG
users_table.printSchema()
```

```
root
 |-- user_id: string (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- level: string (nullable = true)
```

In [37]:

```
# DEBUG
users_table.limit(5).toPandas()
```

Out[37]:

	user_id	first_name	last_name	gender	level
0	26	Ryan	Smith	M	free
1	7	Adelyn	Jordan	F	free
2	71	Ayleen	Wise	F	free
3	81	Sienna	Colon	F	free
4	87	Dustin	Lee	M	free

In [38]:

```
# write USERS table to parquet files
users_table.write.parquet(output_data + 'users_data/users_table.parquet', 'overwrite')
```

In [40]:

```
time_columns = ["ts"]
dfTime = df.selectExpr(time_columns)
dfTime.printSchema()
```

```
root
 |-- ts: long (nullable = true)
```

In [41]:

```
# DEBUG
dfTime.limit(2).toPandas()
```

Out[41]:

	ts
0	1542241826796
1	1542242481796

In [42]:

```
# SOLUTION:
get_datetime = udf(lambda x: str( datetime.fromtimestamp( int(x)/1000.0 )) )
dfTimestamp = dfTime.withColumn("start_time", get_datetime(dfTime.ts))

dfTimestamp.printSchema()
dfTimestamp.show(3)
dfTimestamp.limit(3).toPandas()
```

```
root
 |-- ts: long (nullable = true)
 |-- start_time: string (nullable = true)

+-------------+--------------------+
|           ts|          start_time|
+-------------+--------------------+
|1542241826796|2018-11-15 00:30:...|
|1542242481796|2018-11-15 00:41:...|
|1542242741796|2018-11-15 00:45:...|
+-------------+--------------------+
only showing top 3 rows
```

Out[42]:

	ts	start_time
0	1542241826796	2018-11-15 00:30:26.796000
1	1542242481796	2018-11-15 00:41:21.796000
2	1542242741796	2018-11-15 00:45:41.796000

In [43]:

```
# extract columns to create time table
time_table = dfTimestamp.select("start_time").dropDuplicates() \
               .withColumn("hour",    hour(col("start_time"))) \
               .withColumn("day",     dayofmonth(col("start_time"))) \
               .withColumn("week",    weekofyear(col("start_time"))) \
               .withColumn("month",   month(col("start_time"))) \
               .withColumn("year",    year(col("start_time"))) \
               .withColumn("weekday", date_format(col("start_time"), 'E'))
```

In [44]:

```
# DEBUG
time_table.printSchema()
time_table.limit(5).toPandas()
```

```
root
 |-- start_time: string (nullable = true)
 |-- hour: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- week: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- weekday: string (nullable = true)
```

Out[44]:

	start_time	hour	day	week	month	year	weekday
0	2018-11-15 11:22:06.796000	11	15	46	11	2018	Thu
1	2018-11-15 18:09:32.796000	18	15	46	11	2018	Thu
2	2018-11-15 18:59:14.796000	18	15	46	11	2018	Thu
3	2018-11-15 19:01:55.796000	19	15	46	11	2018	Thu
4	2018-11-21 03:57:19.796000	3	21	47	11	2018	Wed

In [45]:

```
# write time table to parquet files partitioned by year and month
time_table.write.partitionBy("year", "month") \
          .parquet(output_data + 'time_data/time_table.parquet', 'overwrite')
```

Preparation and Code for `songplays_table`

In [46]:

```
get_datetime = udf(lambda x: str( datetime.fromtimestamp( int(x)/1000.0 )) )
df = df.withColumn("start_time", get_datetime(df.ts))
df = df.withColumn("year",       year(col("start_time")))
df = df.withColumn("month",      month(col("start_time")))
```

In [47]:

```
# DEBUG
df.printSchema()
df.limit(10).toPandas()
```

```
root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: double (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
 |-- start_time: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
```

Out[47]:

	artist	auth	firstName	gender	itemInSession	lastName	length	level	location
0	Harmonia	Logged In	Ryan	M	0	Smith	655.77751	free	San Jose-Sunnyvale-Santa Clara, CA
1	The Prodigy	Logged In	Ryan	M	1	Smith	260.07465	free	San Jose-Sunnyvale-Santa Clara, CA
2	Train	Logged In	Ryan	M	2	Smith	205.45261	free	San Jose-Sunnyvale-Santa Clara, CA
3	Sony Wonder	Logged In	Samuel	M	0	Gonzalez	218.06975	free	Houston-The Woodlands-Sugar Land, TX
4	Van Halen	Logged In	Tegan	F	2	Levine	289.38404	paid	Portland-South Portland, ME
5	Magic Sam	Logged In	Tegan	F	3	Levine	132.04853	paid	Portland-South Portland, ME
6	Edward Sharpe & The Magnetic Zeros	Logged In	Tegan	F	4	Levine	306.31138	paid	Portland-South Portland, ME
7	Usher featuring will.i.am	Logged In	Tegan	F	5	Levine	395.72853	paid	Portland-South Portland, ME
8	Helen Reddy	Logged In	Tegan	F	7	Levine	176.50893	paid	Portland-South Portland, ME
9	Taylor Swift	Logged In	Tegan	F	8	Levine	201.06404	paid	Portland-South Portland, ME

10 rows × 21 columns

In [48]:

```
# df.count()
print('>> [' + str(df.count()) + '] songs filtered "NextSong" of logs_data')
```

```
>> [6820] songs filtered "NextSong" of logs_data
```

In [49]:

```
# read in song data to use for songplays table
song_df = spark.read.option("mergeSchema", "true").parquet(output_data + "songs_data/son
gs_table.parquet")
```

In [50]:

```
#song_df.count()
print('>> [' + str(song_df.count()) + '] songs readout from songs_table.PARQUET')
```

```
>> [71] songs readout from songs_table.PARQUET
```

In [51]:

```
# DEBUG
song_df.printSchema()
song_df.limit(5).toPandas()
```

```
root
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- year: integer (nullable = true)
 |-- artist_id: string (nullable = true)
```

Out[51]:

	song_id	title	duration	year	artist_id
0	SOAOIBZ12AB01815BE	I Hold Your Hand In Mine [Live At Royal Albert...	43.36281	2000	ARPBNLO1187FB3D52F
1	SONYPOM12A8C13B2D7	I Think My Wife Is Running Around On Me (Taco ...	186.48771	2005	ARDNS031187B9924F0
2	SODREIN12A58A7F2E5	A Whiter Shade Of Pale (Live @ Fillmore West)	326.00771	0	ARLTWXK1187FB5A3F8
3	SOYMRWW12A6D4FAB14	The Moon And I (Ordinary Day Album Version)	267.70240	0	ARKFYS91187B98E58F
4	SOWQTQZ12A58A7B63E	Streets On Fire (Explicit Album Version)	279.97995	0	ARPFHN61187FB575F6

In [81]:

```
# DEBUG - JOIN - Testing (1/3)
df1 = song_df.filter(song_df.title == 'Setanta matins')
df1.limit(5).toPandas()
```

Out[81]:

	song_id	title	duration	year	artist_id
0	SOZCTXZ12AB0182364	Setanta matins	269.58322	0	AR5KOSW1187FB35FF4

In [82]:

```
# DEBUG - JOIN - Testing (2/3)
df2 = df.filter(df.song == 'Setanta matins') # Riverside / Young Boy Blues
df2.limit(5).toPandas()
```

Out[82]:

	artist	auth	firstName	gender	itemInSession	lastName	length	level	location	me
0	Elena	Logged In	Lily	F	5	Koch	269.58322	paid	Chicago-Naperville-Elgin, IL-IN-WI	

In [92]:

```
# DEBUG - JOIN - Testing (3/3)
df3 = df.join(song_df, song_df.title == df.song ).select(df.song.alias("song (df)"), son
g_df.title.alias("title (song_df)"))
df3.count()
df3.limit(5).toPandas()
```

Out[92]:

	song (df)	title (song_df)
0	Setanta matins	Setanta matins
1	Intro	Intro
2	Intro	Intro
3	Intro	Intro

In [52]:

```
# extract columns from joined song and log datasets to create songplays table
songplays_table = song_df.join(df, (song_df.title == df.song) ) \
                         .select('start_time', \
                                 df.year, \
                                 df.month, \
                                 col('userId').alias("user_id"), \
                                 df.level, \
                                 song_df.song_id, \
                                 song_df.artist_id, \
                                 col('sessionId').alias("session_id"), \
                                 df.location, \
                                 col('userAgent').alias("user_agent") \
                                )
```

In [53]:

```
songplays_table = songplays_table.withColumn("songplay_id", monotonically_increasing_id
())
```

In [54]:

```
#songplays_table.count()
print('>> [' + str(songplays_table.count()) + '] songs found, on JOIN matching for songs
plays_table')
```

```
>> [4] songs found, on JOIN matching for songsplays_table
```

In [55]:

```
# DEBUG
songplays_table.printSchema()
songplays_table.limit(3).toPandas()
```

```
root
 |-- start_time: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- user_id: string (nullable = true)
 |-- level: string (nullable = true)
 |-- song_id: string (nullable = true)
 |-- artist_id: string (nullable = true)
 |-- session_id: long (nullable = true)
 |-- location: string (nullable = true)
 |-- user_agent: string (nullable = true)
 |-- songplay_id: long (nullable = false)
```

Out[55]:

	start_time	year	month	user_id	level	song_id	artist_ic
0	2018-11-21 21:56:47.796000	2018	11	15	paid	SOZCTXZ12AB0182364	AR5KOSW1187FB35FF4
1	2018-11-14 05:06:03.796000	2018	11	10	free	SOGDBUF12A8C140FAA	AR558FS1187FB4565{
2	2018-11-19 09:14:20.796000	2018	11	24	paid	SOGDBUF12A8C140FAA	AR558FS1187FB4565{

In [56]:

```
# write songplays table to parquet files partitioned by year and month
songplays_table.write.partitionBy("year", "month") \
               .parquet(output_data + 'songplays_data/songplays_table.parquet', 'overwri
te')
```

In [57]:

```
print('>> END! (main)')
```

```
>> END! (main)
```

Step 5: Run [etl.py] script

... with working clean code

In [58]:

```
!python 'etl.py' # to be executed at the Jupyter Notebook of the Udacity Workspace
```

```
>> Read Out Config-Infos from [dl.cfg]
>> START (main)
Ivy Default Cache set to: /root/.ivy2/cache
The jars for the packages stored in: /root/.ivy2/jars
:: loading settings :: url = jar:file:/opt/spark-2.4.3-bin-hadoop2.7/jars/i
vy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.hadoop#hadoop-aws added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-91ef8ed6-
174b-40e9-ba64-9264353e7b01;1.0
        confs: [default]
        found org.apache.hadoop#hadoop-aws;2.10.0 in central
        found com.amazonaws#aws-java-sdk-bundle;1.11.271 in central
        found org.apache.commons#commons-lang3;3.4 in central
:: resolution report :: resolve 697ms :: artifacts dl 13ms
        :: modules in use:
        com.amazonaws#aws-java-sdk-bundle;1.11.271 from central in [defaul
t]
        org.apache.commons#commons-lang3;3.4 from central in [default]
        org.apache.hadoop#hadoop-aws;2.10.0 from central in [default]
        -------------------------------------------------------------------
--
        |                  |            modules            ||   artifacts
|
        |       conf       | number| search|dwnlded|evicted|| number|dwnlde
d|
        -------------------------------------------------------------------
--
        |      default     |   3   |   0   |   0   |   0   ||   3   |   0
|
        -------------------------------------------------------------------
--
:: retrieving :: org.apache.spark#spark-submit-parent-91ef8ed6-174b-40e9-ba
64-9264353e7b01
        confs: [default]
        0 artifacts copied, 3 already retrieved (0kB/25ms)
23/03/21 00:02:20 WARN NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.proper
ties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLo
gLevel(newLevel).
23/03/21 00:02:22 WARN Utils: Service 'SparkUI' could not bind on port 404
0. Attempting port 4041.
23/03/21 00:02:22 WARN Utils: Service 'SparkUI' could not bind on port 404
1. Attempting port 4042.
>> spark session created
>> processing song data
>> [71] songs from song_data read out in JSON-format
>> song data processed
>> processing log data
>> [8056] logs entries read IN, of JSON logs_data
>> [6820] songs filtered page("NextSong") of logs_data
>> [6820] songs filtered "NextSong" of logs_data
>> [71] songs readout from songs_table.PARQUET
>> [4] songs found, on JOIN matching for songsplays_table
>> log data processed
>> END!
```

Step 6: Clean up Resources

(EMR, Notbook, IAM-Role)

- Export Jupyter Notebook
- Terminate EMR Cluster
- Delete IAM-Role

In [ ]:

```

```