upGrad

Summary

Case Study Part I

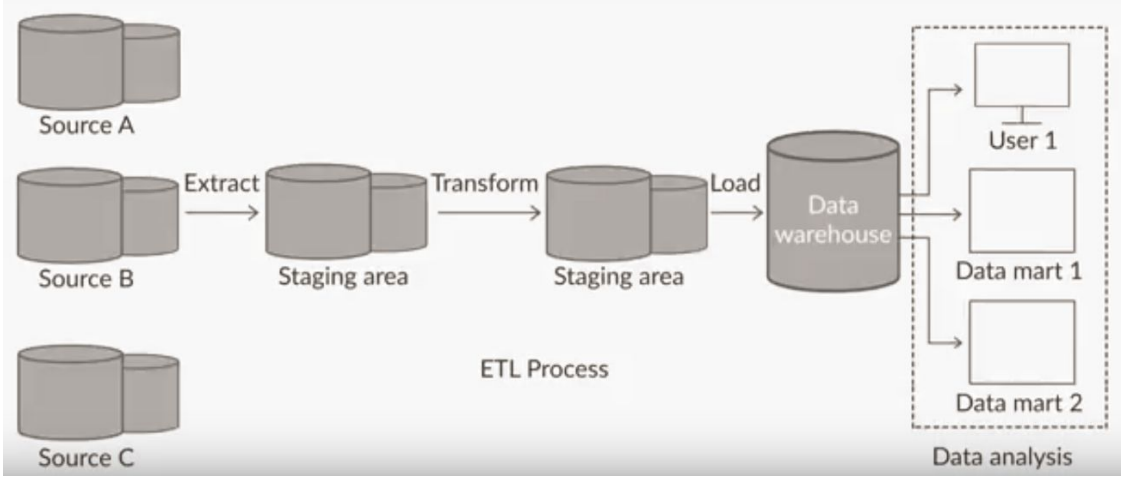
You began this session by learning about the ETL (Extract, Transform, Load) process. Then you learned about data ingestion and its common tools. Finally, you launched an EMR cluster wherein you loaded a telecommunications data set to an RDS instance.

Understanding ETL

In the ETL process, data is first extracted from source systems and then stored in a staging area. It is then subjected to a couple of steps through which it is cleaned and put in the required format, and, finally, it is loaded into a data warehouse. ETL is generally used as a data ingestion process for loading data into a warehouse.

The ETL process essentially comprises of three stages:

1. **Extract**: This is the stage in which relevant data is extracted from various sources.
2. **Transform**: This is the stage where the data is cleaned and converted in the required format. Some of the tasks carried out in this stage include:
    - Calculating a new measure
    - Cleaning the data
    - Filtering the data
    - Splitting a column into multiple columns
    - Merging data from different sources
3. **Load**: This is the stage in which the cleaned data is loaded into the data warehouse for consumption purposes.



upGrad

Data Ingestion

Data ingestion is the process of accessing and importing data from one database to another for the purpose of immediate use or storage. Data ingestion is necessary, since traditional systems that majorly store transactional data, such as MySQL databases, Teradata, Netezza, Oracle, etc., are often unable to handle growing volumes of data, and you need to import data from these systems to those that handle big data.

Data Set Exploration

The database, which included the following three data sets, belonged to a telecommunications company:

1. CRM data
2. Devices data
3. Revenue data

Telco CRM data:

msisdn	gender	yearOfBirth	status	mobileType	Segment
660c3	Male	1969	Active	prepaid	Tier_3
660c4	Female	1981	Active	prepaid	Tier_3

Device data :

msisdn	imei_tac	brand_name	model_name	os_name	os_vendor
660c3	XXX	SAMSUNG	GALAXY J1 ACE	Android	Google
660c4	YYY	LG	W1500	LG OS	LG

Revenue:

msisdn	weekNumber	Revenue_usd
660c3	10	20.26
660c4	11	5.36

CRM stands for customer relationship management. It deals with data about customers who are associated with an msisdn. **msisdn** stands for **Mobile Station International Subscriber Directory Number**. It is used to identify an international mobile number. The second data set includes mobile or device information, using which you can identify the brand_name or model name associated with a particular msisdn. Finally, the third data set shows the revenue generated for a particular msisdn across different weeks.

The links to the data sets are as follows:
https://telecom-case-study-ml-hive-sqoop.s3.amazonaws.com/rev1.csv
https://telecom-case-study-ml-hive-sqoop.s3.amazonaws.com/device1.csv
https://telecom-case-study-ml-hive-sqoop.s3.amazonaws.com/crm1.csv

upGrad

Case Study Approach

The steps involved in building the pipeline are as follows:

1. Launching an EMR (Elastic MapReduce) cluster with Hive, Oozie and Sqoop services
2. Downloading the data sets (link to the data sets have been provided)
3. Creating an RDS instance
4. Connecting to the RDS instance:
    a. You used an EMR cluster to connect to the RDS instance.
    b. You can also do this from your local machine if you have SQL installed on it, or use MySQLWorkbench and connect to the RDS instance.
5. Creating tables on the RDS instance
6. Loading the data in the CSV files into these tables with
7. Moving the data from the DBMS tables to the HDFS using Sqoop:
    a. Explore different ways to use the sqoop command and
    b. Store the data in different file formats
8. Creating a Hive database and Hive tables, and loading data into these tables
9. Understanding the Avro and Parquet file formats, and their impact on the performance while running queries
10. Analysing the data by running Hive queries.
11. Starting the Hue service to run oozie jobs for orchestrating the different steps in the process
12. Performing cleanup:
    a. Dropping Hive tables and the Hive database
    b. Terminating the EMR cluster

Source Setup I - Creating MySQL Tables on the RDS

Creating the EMR cluster [use version 5.24.0] with Hive, Hue, Oozie and Sqoop services running on it

Connecting to EMR

a. **Mac Users**
   ssh -i ~/XXXXXX.pem hadoop@ec2-3-227-21-159.compute-1.amazonaws.com

b. **Windows Users**
   Use putty to connect to the instance

Creating the RDS Instance

All commands have been executed using the following RDS specifications:

Database Name: telcoDb
User: admin
Password: Mysore123

Connecting with the RDS

```
mysql -h telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
```

Here, **database-1.cisk8vkhoaoc.us-east-1.rds.amazonaws.com** is the endpoint that is required to establish a connection between the EMR cluster and the RDS instance. You may not be able to connect to the RDS instance, because it may not have the privileges to connect to this cluster. To enable this, you need to edit the security groups by adding a new rule that enables an SQL connection to the EMR's IP address.

Important Note
Although you select free tier while creating the RDS instance, it does not mean that AWS will not charge any money. In fact, it will charge around $0.5 per day. Hence, terminate the instance when you are not using it.

Creating a database named telco

```
show databases;
create  database telco;
use telco;
```

Creating tables within the database

```
 create table crm
(
 msisdn  VARCHAR(255),
 gender  VARCHAR(255),
 year_of_birth  INT,
 system_status  VARCHAR(255),
 mobile_type  VARCHAR(255),
 value_segment  VARCHAR(255)
);

show tables;


create table device
(
 msisdn  VARCHAR(255),
```

```
 imei_tac VARCHAR(255),
 brand_name VARCHAR(255),
 model_name  VARCHAR(255),
 os_name  VARCHAR(255),
 os_vendor  VARCHAR(255)
);


create table revenue
(
 msisdn  VARCHAR(255),
weekNumber INT,
Revenue_usd FLOAT
);
```

Source Setup II - Loading data into the tables

Loading data into the tables

a. After creating the necessary source tables, the next step is to load the associated data into them. The first step is to have the necessary data in your EMR cluster.

Mac User	scp -i C:\User\Downloads\XXXXX.pem ~/Downloads/crm1.csv hadoop@ec2-34-203-220-27.compute-1.amazonaws.com:/home/hadoop
Windows User	Use the following documentation for loading the data into the EMR instance.

b. Now that you have the necessary data in your EMR cluster, the next step is to load this data into the tables.

Once again, connect with your RDS instance:

```
mysql -h telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
```

Go to your database and load the tables that you have created. The data present in a CSV file will be comma-separated, where each line represents a row. You will need to provide this information while loading the data:

```
use telco;

LOAD DATA LOCAL INFILE '/home/hadoop/crm1.csv'
```

```
INTO TABLE crm
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/hadoop/device1.csv'
INTO TABLE device
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

LOAD DATA LOCAL INFILE '/home/hadoop/rev1.csv'
INTO TABLE revenue
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

It is always good to try a few commands to make sure your tables have indeed been loaded:

```
select * from crm limit 5;
select COUNT(*) from device ;
```

Compare this count with the devices1 file records count in EMR:

wc device1.csv

To see more about wc use ***--help***

You can take a logical break here and terminate the instance. Now, it is not necessary to have the same EMR instance as you move forward. However, do not terminate the RDS instance, as it now contains the necessary tables with the data loaded into them.

upGrad

Summary

Case Study Part II

In this session, you learnt how to import data into the HDFS directly using Sqoop and then perform Hive queries over the data. Finally, you created an Oozie workflow where you integrated the different steps involving Hive and Sqoop, and created a workflow.

Data Ingestion with Apache Sqoop

The steps involved in a sqoop import command are as follows:

1. Sqoop collects metadata from the RDBMS (e.g., MySQL).
2. It launches a MapReduce job (the Sqoop command will be converted automatically to a MapReduce job, using the metadata information).
3. Mappers are created against the specified key ranges of the RDBMS, and the data is written into the Hadoop ecosystem.

The basic idea behind importing large data sets is to divide them into multiple parts and then import these parts in parallel using a MapReduce-like paradigm. As is the standard case, Sqoop runs the map phase first and divides the data into multiple smaller parts, each of which is assigned to a mapper. However, there is no need for a reduce phase, since no aggregation is required - we are writing into an HDFS-type destination.

With the RDS fully loaded, we can directly ingest the data into HDFS using the sqoop command.

```
sqoop import --connect jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --table crm --target-dir /user/hadoop/telco/crm/ --username admin -P -m 1
```

After firing this command on your terminal, data from the ‘crm’ table from the database in MySQL is transferred to the HDFS. You can view the transferred data by running the following command on your console:

```
hadoop fs -cat /user/hadoop/telco/crm/part-m-* | head
```

You will find only one partition file after running this command. Next, run the same command after changing the number of mappers to 4:

```
sqoop import --connect jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --table crm --target-dir /user/hadoop/telco/crm_split4/ --username admin -P -m 4 --split-by year_of_birth
```

You can run the following command to check the number of partitions in this directory:

```
hadoop fs -ls /user/hadoop/telco/crm_split4/
```

You will see that by changing the number of mappers to 4 and specifying year_of_birth as the key to split the data, four part files are created. Of these, two do not have any record. This means that the jobs submitted are distributed irregularly. Hence, the correct column that should be specified is the primary key column.

Sqoop follows the procedure below for data ingestion when you use multiple mappers and a where clause:

1. It looks at the range of the splitting key (from the splitting columns/primary key column).
2. It sets the lower value of the splitting key to some variable.
3. It sets the higher value of the splitting key to some other variable.
4. It generates SQL queries to fetch data in parallel.

If the values of the primary key column/splitting columns are not distributed uniformly across the range of the splitting key, then it can result in unbalanced tasks. In such cases, you are advised to choose a different column by using the '--split-by' clause explicitly.

For example, to specify that Sqoop should use the ‘msisdn’ column for splitting, you can write "--split-by msisdn".

You can import data in various ways:

a. Complete extract
b. Where extract
c. Incremental extract, for instance, based on the Last modified Value

Some of the other industry cases of sqoop are as follows:

1. Exporting data to RDBMSs
2. Interactive analytics on RDBMSs

The export operation is similar to the import operation, except it works in the opposite direction. Now, in the export operation too, the number of maps is specified, and then, the respective data blocks are assigned to each mapper program for transfer.

sqoop eval can be used to query MySQL without importing the data:

```
sqoop eval --connect jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --username admin -P --query "show tables;
```

You can go through the following references to understand data migration through Sqoop in more detail:

- Sqoop user guide: This is a detailed documentation on Sqoop by Apache.
- Data migration through Sqoop at IBM: Here, 'IBM developerWorks' demonstrates data transfer using Sqoop.

Data Ingestion with Sqoop - II

Each type of data can be stored in a variety of file formats, and each format has its advantages and disadvantages. Choosing a particular file format is important if you want maximum efficiency in terms of factors such as processing power, network bandwidth and available storage. A file format directly affects the processing power of the system ingesting the data, the capacity of the network carrying the data and the storage available for the ingested data.

Following are some of the widely used file formats:

1. **Text:** This is the most commonly used file format for exchanging huge volumes of data between Hadoop and external systems. Nevertheless, a text file provides limited support for schema evolution. Also, it does not support block compression and is not compact. You can refer to the additional reading section for more details on schema evolution and block compression.

   Note that unless you specifically mention the file format of a table, the data is stored as a text file by default.

2. **Sequence files:** These are binary files, which store data as binary key-value pairs. A binary file is stored in a binary format. Because sequence files are binary files, they are more compact than text files. Each byte in a binary file has 256 possible values compared with pure unextended ASCII, which has only 128 possible values, making the binary file twice as compact. Sequence files support block compression, and can be split and processed in parallel; because of this, they are used extensively in MapReduce jobs.

3. **Avro:** This is a language-neutral data serialisation system developed in Apache's Hadoop project. Serialisation is the process of converting data structures into a format that can be used for either data **storage** or **transmission** over a network. Being language-neutral means an Avro file can be easily read later, even in a language that differs from the one that is used to write the file. These files are self-describing, compressible and splittable, and can also be split and processed in parallel, thus making them suitable for MapReduce jobs. Being binary in nature, these files are **more**

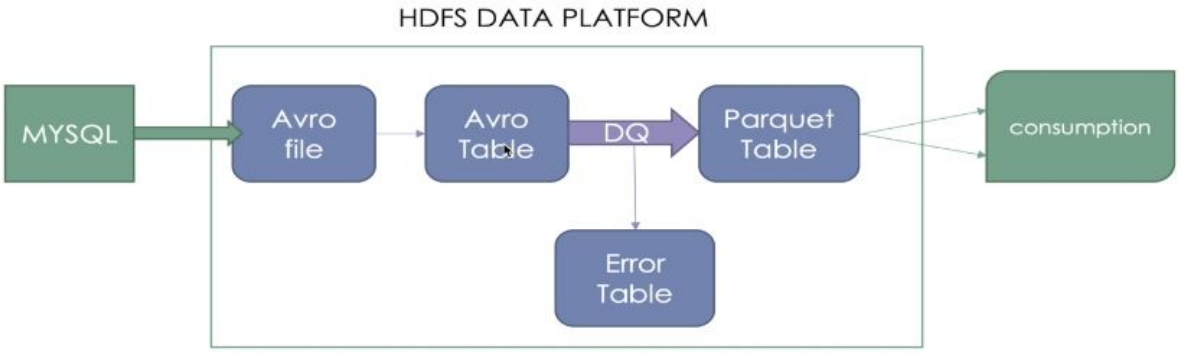
compact than text files. Avro files also support schema evolution, which means the schema used to read the file does not have to match the one used to write the file.

4. **Parquet:** Parquet files support efficient compression and encoding schemes. They also allow compression schemes to be specified at a per-column level and are future-proofed to allow the addition of more encodings as they are invented and implemented.

Based on your requirements, you may choose an appropriate file format for storing your data. In this case study, you first moved the data present in the SQL tables to HDFS platform using sqoop and stored the data in Avro format. This is because Avro file format is a schema embedded file format which preserves the schema along with the data. Later this data stored in Avro format is stored in Avro tables. However, this is just the staging area, and data here is stored without any cleaning. In the next step parquet tables are created to store the cleaned data. This is the transformation phase of the ETL process. Parquet stores the data in a columnar format. This increases the data processing speed when columnar operations are executed.

The main reason to maintain data in two different formats is

- Queries that involve fetching entire data are ideal when data is stored as Avro tables since it is row-based storage.
- When it comes to run analytical queries involving specific columns, i.e., groupby operations, storing data in Parquet tables is ideal since it is columnar-based storage.



So, let us import our CRM data but store it as an Avro file this time

```
sqoop import -Dmapreduce.job.user.classpath.first=true
-Dhadoop.security.credential.provider.path=jceks://x.jceks --connect
jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --table crm
--target-dir /user/hadoop/telco/crm_avro -m 1 --username admin -P --as-avrodatafile
```

In this case study, we have used Beeline as the command-line interface (CLI) by using a jdbc connector to connect to HiveServer2:

```
beeline -u jdbc:hive2://localhost:10000/default  -n hadoop
```

The code snapshot below shows the processes of creating the database telco_db and loading the table crm_avro into it:

```
create database telco_db;

use telco_db;

create table crm_avro
(
 msisdn  string,
 gender  string,
 year_of_birth  INT,
 system_status  string,
 mobile_type  string,
 value_segment  string
)
stored as avro;

load data inpath '/user/hadoop/telco/crm_avro' into table crm_avro;
```

On exploring the data, we find that it contains some discrepancies, for example, inappropriate values in the gender column:

```
Use telco_db;

select distinct(gender) from crm_stg limit 20;

select distinct(gender) from crm_stg where UPPER(gender) in ('MALE', 'FEMALE') limit
10;
```

Next, we store this cleaned data in Parquet format:

```
create table crm_cln_parq
stored as parquet as
select  *,  (YEAR(CURRENT_DATE)  -year_of_birth)  age,  UPPER(gender)  genderUpp  from
crm_stg where UPPER(gender) in ('MALE','FEMALE');
```

Further, we check whether this new table has indeed been cleaned. We can also find the number of Males or Females for different ages as shown in the code snapshot below:

```
select * from crm_cln_parq limit 5;

select genderupp, age, count(*) from crm_cln_parq group by genderupp, age limit 20;

select genderupp, age, count(*) from crm_cln_parq group by genderupp, age order by
genderupp, age limit 20;
```

For performance comparison, we store the same data in avro format as shown below:

```
create table crm_cln_avro
stored as avro
as
select  *,  (YEAR(CURRENT_DATE)-year_of_birth)  age,  UPPER(gender)  genderUpp  from
crm_avro
where UPPER(gender) in ("MALE", "FEMALE");
```

To compare the performance of the avro and parquet files, we apply two queries to each of these file formats.

```
select genderUpp, age, count(*) from crm_cln_avro group by genderUpp, age order by
genderUpp, age limit 50;

select genderUpp, age, count(*) from crm_cln_parq group by genderUpp, age order by
genderUpp, age limit 50;
```

On executing the queries as shown above, crm_cln_avro (the avro file) took 52 seconds whereas crm_cln_parq (parquet file) took only 23 seconds.This is because columnar operations are executed much faster in the parquet format as compared with the avro format. In the code snapshot below, a row-based operation has been carried out on each of these tables:

```
select count(*) from crm_cln_avro ; -- Performs the query in 0.135 seconds

select count(*) from crm_cln_parq;  -- Performs the query in 0.259 seconds
```

To summarise the performance comparison above, queries that involve fetching entire data are ideal while handling Avro tables. Parquet tables use column-based storage. Hence, while handling parquet tables, it is ideal to run analytical queries, which involves handling specific columns, i.e., grouping by operations.

Schema evolution: Updating the schema without taking into account other applications that are running and gathering data based on the current schema might affect those applications. Therefore, you will have to develop the schema in such a way that it caters to the new requirements and the existing applications. This is called schema evolution.

Block compression: Since Hadoop stores large files by splitting them into blocks, it would be best if individual blocks can be compressed. Block compression is the process of compressing each individual block.

Additional Links

- Various file formats in Hive (Apache docs)
- Overview of SerDes (Apache docs)
- Brief overview of the parquet file format (Apache docs)
- A blog on Twitter explaining parquet and columnar formats in detail

upGrad

Creating Hive Tables

The command used for connecting to Beeline is shown below:

```
beeline -u jdbc:hive2://localhost:10000/default  -n hadoop
```

Once you are connected to Beeline, you can create tables. In Hive, we have created a database named telco_db with the following three tables:

1. crm_stg
2. device_stg
3. revenue_stg

```
use telco_db;
```

```
create table crm_stg
(
 msisdn  string,
 gender  string,
 year_of_birth  int,
 system_status  string,
 mobile_type  string,
 value_segment  string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

```
create table device_stg
(
 msisdn  string,
 imei_tac string,
 brand_name  string,
 model_name  string,
 os_name  string,
 os_vendor  string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

```
create table revenue_stg
(
 msisdn  string,
 Week_number int,
 revenue_usd float
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

After this, we load the existing table:

```
load  data inpath '/user/hadoop/telco/crm' into table crm_stg;
load  data inpath '/user/hadoop/telco/devices' into table device_stg;
load  data inpath '/user/hadoop/telco/revenue' into table revenue_stg;
```

Now, you must always check that your data has indeed been loaded. You can do so using the following command:

```
select * from crm_stg limit 5 ;
```

Now, if you use the following command and check for the data in the respective directory, you will find that it no longer exists there:

```
hadoop  fs -ls /user/hadoop/telco/crm
```

This is because the table was created as an internal table.

Hive Queries

CTEs

As you have learnt before, in the case of nested queries, you can write one query based on the result of another one. A simpler and more common approach is to use Common Table Expressions (CTEs). CTEs enable you to create a temporary result based on which you can run a further query; this is done using the 'with' clause. Note that these queries are not supported in MySQL. The common syntax for using CTEs is as follows:

WITH CTE_Name
AS
(SELECT column_names from table)
SELECT CTE_columns from CTE_Name

Please note that the scope of the CTEs is defined within the execution of the following SELECT (SELECT/INSERT/DELETE/CREATE VIEW) statement. An example of a CTE command is shown below:

```
WITH crm_preprocessed AS
(SELECT msisdn, UPPER(gender) as genderUpp, (YEAR(CURRENT_DATE)-year_of_birth) as
age FROM crm_stg
WHERE UPPER(gender) IN ('MALE', 'FEMALE') AND YEAR(CURRENT_DATE)>year_of_birth)
SELECT genderUpp, COUNT(genderUpp) as gender_distribution FROM crm_preprocessed
GROUP BY genderUpp;
```

Output:

genderupp	gender_distribution
FEMALE	1397042
MALE	9343231

Self-Joins

A self-join is a Join operation that a table performs on itself. An example of a self-join is shown below.

What is the weekly increase in revenue?

```
with weekly_sales
as
(select week_number, sum(revenue_usd) as Weekly_Revenue from revenue_stg
group by week_number
order by week_number asc)
select succeeding_week.week_number as week_number,
succeeding_week.Weekly_Revenue - current_week.Weekly_Revenue as revenue_increment
from weekly_sales succeeding_week
inner join weekly_sales current_week
on succeeding_week.week_number = current_week.week_number + 1;
```

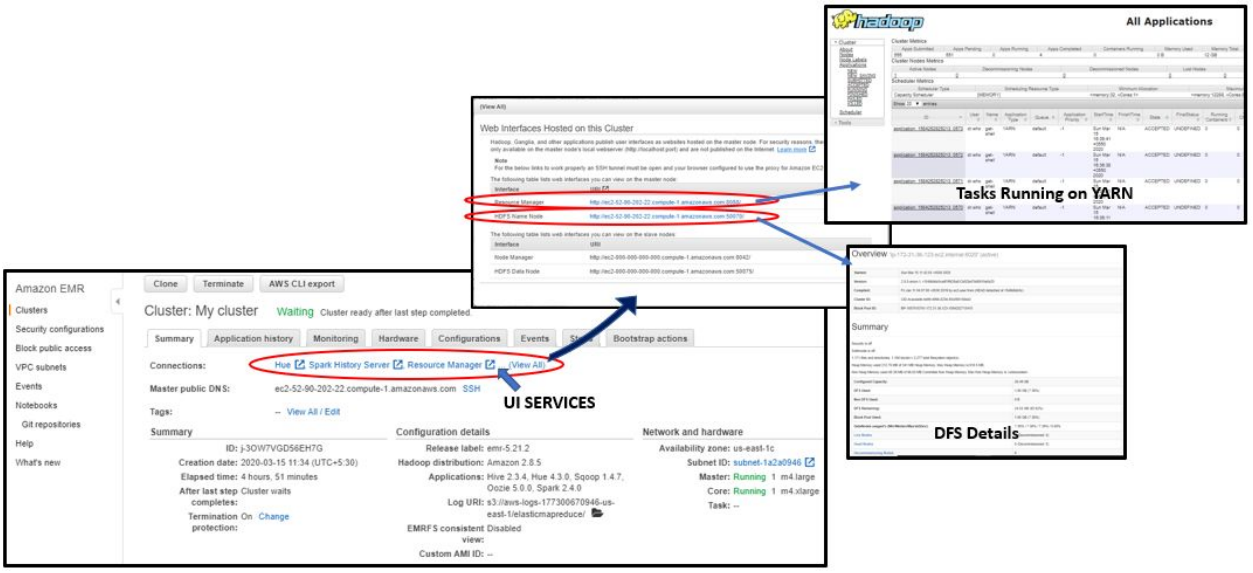
Using the CTE, we have temporarily created a table called weekly_sales over which have we performed a self-join in order to get the difference in revenue between the present week and the next one.

Output:

week_number	revenue_increment
23	214277.45176312048
24	55809.35782143194
25	-197520.0104312105
26	3294.131388515234
27	19594.0649325219
28	13216.291540762875
29	84571.77574238367
30	15687.749238003045
31	156111.35783166625
32	43432.478834591806
33	47297.214860830456
34	57393.82739792764
35	-34844.196724422276

Creating an Oozie Workflow

The Hadoop ecosystem consists of multiple technologies, and different Hadoop jobs are mostly performed sequentially in order to complete data processing based on their dependencies. In such cases, it is important to coordinate the flow of these jobs. Oozie, the workflow scheduler for Hadoop, provides control over such complex multistage Hadoop jobs. Although it is designed for handling Hadoop jobs, it can also manage non-Hadoop jobs such as Java jobs and shell scripts.



Hue is an open-source web-based interface that makes Hadoop much more easy to use. Hue stands for Hadoop User Experience. It contains all the editors that can be used on the Hadoop ecosystem, such as Hive, Impala and Pyspark, and can also connect to different RDBMSs such as MySQL, Oracle, etc. Hue also contains scheduler services such as Oozie.

In the case study, three different tasks are compiled into a single workflow.

Action 1: Creating an avro table named devices_avro for the devices data and loading the data.
Action 2: Creating a Parquet table named devices_parq for the devices table and loading the data from the devices_avro table.
Action 3: Joining the devices data with the crm data to find popular brands for customers of age 30.

Before creating the workflow, the devices data is imported as a devices_avro avro file and stored in the HDFS.

```
sqoop import -Dmapreduce.job.user.classpath.first=true
-Dhadoop.security.credential.provider.path=jceks://x.jceks --connect
jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --table
device --target-dir /user/hadoop/telco/devices_avro -m 1 --username admin -P
--as-avrodatafile
```

Action 1:
Opening the vi editor:

```
vi devices_create_table.hql
```

```
use telco_db;
create table if not exists devices_avro (
 msisdn  string,
 imei_tac string,
 brand_name  string,
 model_name  string,
 os_name  string,
 os_vendor  string
)
stored as avro;

load data inpath '/user/hadoop/telco/devices_avro' into table devices_avro;
```

Storing these files into the HDFS:

```
hadoop fs -put devices_create_table.hql /user/hadoop/
```

Action 2:

```
vi devices_create_parquet.hql
```

```
use telco_db;
create table if not exists devices_parq
stored as parquet
as
select * from devices_avro;
```

```
hadoop fs -put devices_create_parquet.hql /user/hadoop/
```

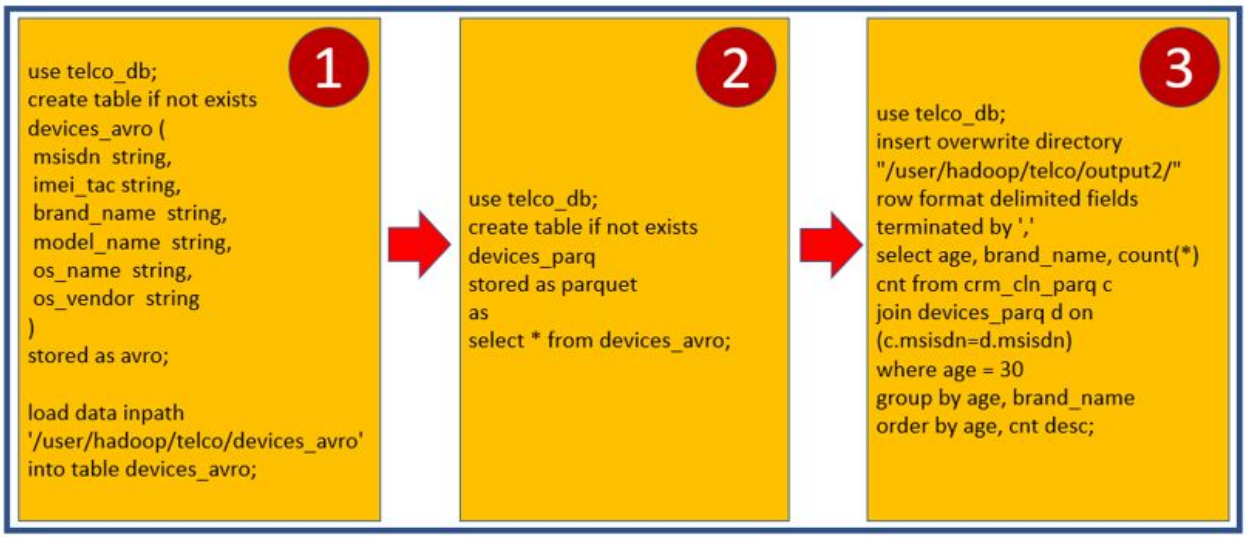
Action 3:

```
vi devices_popular_report.hql
```

```
use telco_db;
insert overwrite directory "/user/hadoop/telco/output2/" row format delimited fields
terminated by ','
select age, brand_name, count(*) cnt from crm_cln_parq c
join devices_parq d on (c.msisdn=d.msisdn)
where age = 30
group by age, brand_name
order by age, cnt desc;
```

```
hadoop fs -put devices_popular_report.hql /user/hadoop/
```

The workflow is summarised below:



You can run the workflow and check your results in the /user/hadoop/telco/output2/ directory.

Creating an Oozie Workflow - II

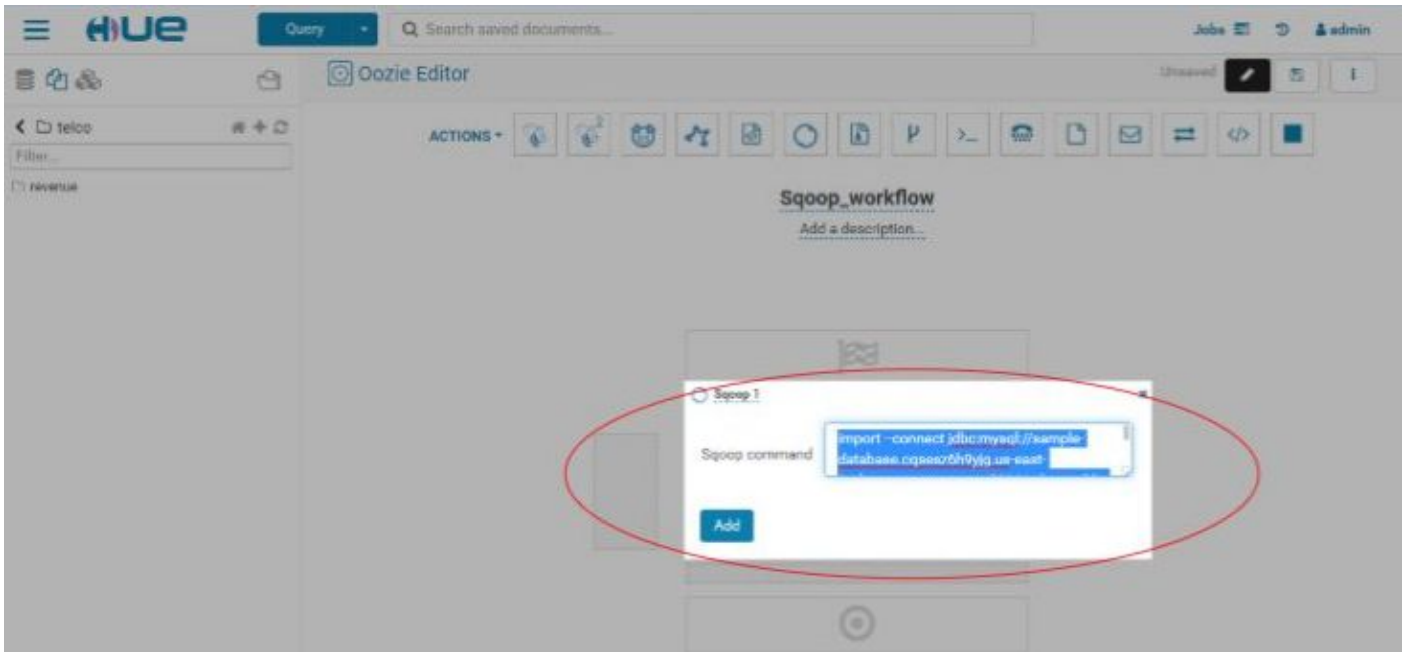
Points to ensure before running the sqoop job:

1. Make sure the EMR version you are running is 5.24.0. The latest version of EMR is not stable and causes problems while creating a workflow
2. Ensure you have an RDS instance with necessary data loaded into the tables
3. Make sure the sample sqoop job is running fine through the CLI

Download the jdbc driver from the link given below and copy it to your EMR cluster using the WINSCP tool
https://www.microsoft.com/en-us/download/details.aspx?id=54671

Given below are the steps for running a sqoop job in a workflow:

1. Launch the hue service by clicking on hue.
2. After creating the hue user, copy the driver to this user. This is to make sure the jdbc driver is available with the hue user to run the sqoop job.
3. On the CLI, run the following command to copy the file to the hue user: ***hadoop fs -put sqljdbc42.jar /user/admin/***.
4. Go back to hue and go to the workflow page.
5. Next, based on the desired action, drag and drop the desired service. Since it is a sqoop command, drop the sqoop service.



6. Enter the following sqoop command:
   import --connect jdbc:mysql://sample-database.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco --table revenue --target-dir /user/admin/telco/revenue/ --username admin --password admin123 --m 1 --driver com.microsoft.sqlserver.jdbc.SQLServerDriver
7. In the Command section, choose the necessary files for running the sqoop job. After selecting a file, run the job.
8. Once your job is submitted, you can view its status under the job browser section, and by clicking on the job, you can view the details of the different tasks involved in the job.
9. After executing the sqoop job successfully, you can see the different files created in the destination folder.