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 will be provided)
3. Creating an RDS instance
4. Connecting to the RDS instance:
    a. You will use 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.

    (In the demonstration, Ganesh will be connecting to the RDS using the EMR cluster, since EMR comes default with SQL installed on it.)
5. Creating tables on the RDS instance
6. Loading these tables with data present in the CSV files
7. Moving the data from the DBMS tables to the HDFS using Sqoop:
    a. Explore different ways to use the sqoop command
    b. How is the data stored 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 different steps in the process
12. Performing cleanup:
    a. Dropping Hive tables and the Hive database
    b. Terminating the EMR cluster

Solution

Creating the EMR Cluster [Use Version 5.24.0] with hive,hue, oozie and sqoop service 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 instance

Creation of the RDS instance

All the 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
```

You may not get connected to the RDS instance, because the RDS instance may not have privileges to connect to this cluster. To enable this you need to edit the security groups by adding a new rule which enables an SQL connection to the EMR ip address.
For eg: 172.31.93.189/32

Creating a database of the name telco

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

Table Creation

```
 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
);
```

```
show tables;
```

Loading data into these tables

a. Downloading necessary data on your local file system

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	wget *[Link to the dataset]*

b. Loading this data onto the tables SQL tables using our EMR instance

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

```
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’s always good to try a few commands to make sure that your tables have indeed been loaded.

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

Validate this count with the devices1 file records count in EMR

wc device1.csv

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

--Source Setup is done---

7. With the RDS fully loaded, we can directly Sqoop the data into the HDFS. We will be storing it in the Avro format

```
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 see the transferred data by running this 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, let’s experiment by 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
```

Let us check the number of partitions

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

You will find four partition files being created at the destination folder.

Now try the below command and check the results in your destination folder

```
sqoop import --connect
jdbc:mysql://telcodb.cqsesz6h9yjg.us-east-1.rds.amazonaws.com:3306/telco
--table crm --target-dir /user/hadoop/telco/crm_split4_where/ --username
admin -P -m 4 --split-by year_of_birth --where 'gender = "MALE"'
```

Let us carry on with our case study. We haven’t imported the device and revenue table so let’s do that.

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

```
-m 1

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

Always remember to check that your files have indeed been imported. The cat command will help you read the contents of the file.

```
hadoop  fs -cat /user/hadoop/telco/devices/* | head
```

---End of Data Ingestion---------------------------------------
Let’s move on to Hive querying. So first let us create the Hive tables. We will be using HiveServer2 for this purpose and hence connecting to it using beeline.You will find the detailed description on the platform.

In order to open the Beeline CLI, type the following command

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

In Hive, we have created a database of the name, telco_db with three tables

1. crm_stg
2. device_stg
3. revenue_stg

```
Create database telco_db;
use telco_db;
```

Creating the tables

```
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';
```

Now if you type the below command, you should get an empty table

```
select * from crm_stg;
```

Time to load the data into our tables

```
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;
```

And never forget to check that your data has indeed been loaded.

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

So what about the data in the HDFS. Let's go back to the terminal and check for that data.

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

You're right, the data no longer exists since the Hive table was an internal table. So let's jump back into our Hive CLI and try some Hive queries.

Executing Hive Queries

Let's check for discrepancies in the data

```
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;
```

Storing this cleaned data in the 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');
```

So let's see if this new table has indeed been cleaned. We can also find the number of Males or Females for different ages

```
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;
```

Next, we will explore further into the performance of the Avro and Parquet file formats. You will also learn about building Oozie workflow and using the front end Hue service.

So let's import our crm data but this time store it as an Avro file.

```
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
```

Getting back into our Hive CLI.

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

Creating and loading the crm_avro table

```
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;
```

Let us clean the data and store it into two separate formats, Avro and Parquet

```
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");

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

Now we can do a performance comparison by checking the time it takes to execute a given query.

```
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;
```

You would have noticed that the parquet file executed much faster than the Avro file. Try to think as to why this happened.

For the next exercise, we will have to first import the device data into 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
```

We will create an Oozie workflow on Hue. Click on the Hue service and login as a super user. As an additional exercise, explore the different services available in Hue.

So let's build a workflow. We will compile three HQL files and integrate them in a single pipeline. We will write these queries on the vi editor and store them in the HDFS.

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;
```

Put 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/
```

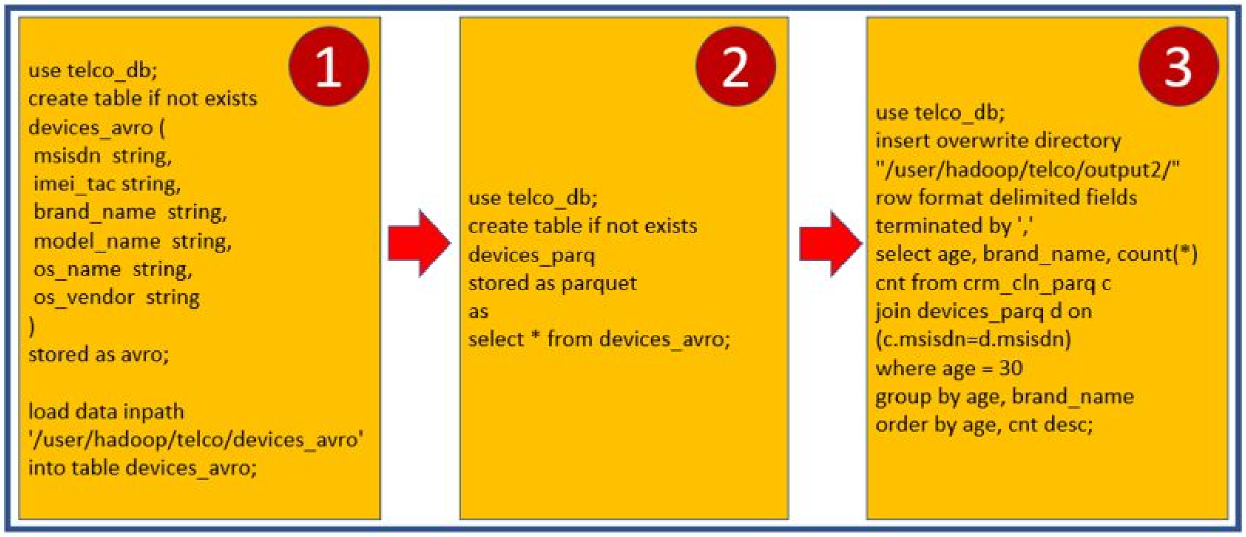
Action 3:

```
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/
```

So our workflow should work like this:



```
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;
```

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

```
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;
```

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