Lab 1: Data Ingestion with Sqoop for RDBMS (MariaDB)

In this lab, you will import and export tables from RDBMS to HDFS(Hadoop Distributed Filesystem) using Sqoop.

Importing Database Table into HDFS with Sqoop.

First, in the next few steps we will use Sqoop to examine the databases and tables in this database before importing them into HDFS.

1. In a terminal window, log in to MariaDB and select the database labs.

Database: labs

```
$ mysql --user=student --password=student labs
```

Note: If you do not enter anything after the password, you will be prompted for the password:

```
$ mysql -u student -p labs
```

Enter the password "student" here.

2. If the login is successful, the "MariaDB [labs]>" prompt appears and a screen waiting for commands is displayed. Enter a command to check which database exists here.

```
MariaDB> show databases;
```

```
MariaDB [labs]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hive               |
| hue                |
| labs               |
| mysql              |
| performance_schema |
| test               |
+--------------------+
7 rows in set (0.00 sec)
```

Figure 1. List databases in MariaDB

3. Next, enter the command to review the table in labs.

```
MariaDB [test]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| authors        |
| posts          |
+----------------+
2 rows in set (0.00 sec)
```

Figure 2. List of table name in Labs DB

The authors and posts tables are displayed on the screen.

This table will be imported/exported through the Sqoop command.

```
MariaDB> desc authors;
MariaDB> describe posts;
```

Note: The `desc` and `describe` commands are the same.

```
MariaDB [test]> desc authors;
+------------+--------------+------+-----+-------------------+----------------+
| Field      | Type         | Null | Key | Default           | Extra          |
+------------+--------------+------+-----+-------------------+----------------+
| id         | int(11)      | NO   | PRI | NULL              | auto_increment |
| first_name | varchar(50)  | NO   |     | NULL              |                |
| last_name  | varchar(50)  | NO   |     | NULL              |                |
| email      | varchar(100) | NO   | UNI | NULL              |                |
| birthdate  | date         | NO   |     | NULL              |                |
| added      | timestamp    | NO   |     | CURRENT_TIMESTAMP |                |
+------------+--------------+------+-----+-------------------+----------------+
6 rows in set (0.00 sec)

MariaDB [test]> describe posts;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| author_id   | int(11)      | NO   |     | NULL    |                |
| title       | varchar(255) | NO   |     | NULL    |                |
| description | varchar(500) | NO   |     | NULL    |                |
| content     | text         | NO   |     | NULL    |                |
| date        | date         | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

Figure 3. Structure of Tables (authors and posts)

4. Review the structure of the authors, posts tables and review some records.

```
MariaDB> SELECT id, first_name, last_name, email, added
            FROM authors limit 5;
```

5. Type `quit` to exit MariaDB and press Enter.

```
MariaDB> quit
```

6. Run the following command to check the basic options of sqoop.

```
$ sqoop help
```

7. To see detailed options for each sub-command, enter the desired subcommand after help. To see detailed options for import, run the command as follows.

```
$ sqoop help import
```

8. Run the list of databases in MariaDB and tables in database labs with the following command.

```
$ sqoop list-databases --connect jdbc:mysql://localhost --username student --
password student
```

The command execution result is the same as the database shown in Figure.1.

Note: an alternative to using the `--password` argument is to use `-P` (capital letter)and let Sqoop prompt you for the password, which is then not visible when you type it.

```
$ sqoop list-tables --connect jdbc:mysql://localhost/labs --username student -P
```

The authors and posts tables are displayed on execution result.

9. Import all tables in labs database using the `import-all-tables` command.

```
$ sqoop import-all-tables --connect jdbc:mysql://localhost/labs \
--username student --password student
```

Note: Sqoop provides import-all-tables, but this command is rarely used in real production.

The reason is that this command tries to accomplish many things with one command. Don't use it this time.

The real environments typically have hundreds of databases and thousands of tables in each database, so use this command to just test your system.

Usually, even importing a single table can take a lot of time, so the command to import all tables is impractical. Most of these cases use the import command.

10. Execute the command to fetch the posts table from the labs database using Sqoop and store it in HDFS.

```
$ sqoop import --connect jdbc:mysql://localhost/labs \
--username student --password student --table posts
```

When this command is executed, the posts directory is created under the /user/student home directory of HDFS and data is stored as follows.

```
[student@localhost ~]$ hdfs dfs -ls /user/student/posts
Found 5 items
-rw-r--r--   1 student student          0 2021-07-31 17:11 /user/student/posts/_SUCCESS
-rw-r--r--   1 student student   10247865 2021-07-31 17:11 /user/student/posts/part-m-00000
-rw-r--r--   1 student student   10260760 2021-07-31 17:11 /user/student/posts/part-m-00001
-rw-r--r--   1 student student   10257979 2021-07-31 17:11 /user/student/posts/part-m-00002
-rw-r--r--   1 student student   10250911 2021-07-31 17:11 /user/student/posts/part-m-00003
[student@localhost ~]$
```

Figure 4. list of posts in HDFS

11. Create a target directory in HDFS to import table data into.

```
$hdfs dfs -mkdir /mywarehouse
```

12. Import the authors table and save it to the HDFS directory we created above using ',' to delimit the fields.

Note: The `--fields-terminated-by ','` option separates the fields in the HDFS file with the tab character. If you want to work with Hive or Spark, it's better to use '\t' instead of ','.

```
$ sqoop import --connect jdbc:mysql://localhost/labs \
```

```
--username student --password student \
--table authors --fields-terminated-by ',' \
--target-dir /mywarehouse/authors
```

13. Review that the command worked with hdfs commands for target-dir.

```
$ hdfs dfs -ls /mywarehouse/authors
$ hdfs dfs -cat /mywarehouse/authors/part-m-00000
```

```
[student@localhost ~]$ hdfs dfs -ls /mywarehouse/authors
Found 5 items
-rw-r--r--   1 student supergroup          0 2021-10-31 17:48 /mywarehouse/authors/_SUCCESS
-rw-r--r--   1 student supergroup     189526 2021-10-31 17:48 /mywarehouse/authors/part-m-00000
-rw-r--r--   1 student supergroup     190441 2021-10-31 17:48 /mywarehouse/authors/part-m-00001
-rw-r--r--   1 student supergroup     190481 2021-10-31 17:48 /mywarehouse/authors/part-m-00002
-rw-r--r--   1 student supergroup     190379 2021-10-31 17:48 /mywarehouse/authors/part-m-00003
```

Figure 5. list of /mywarehouse/authors

If you execute the `cat` command, you can see that each line of data is stored separated by "," unlike the previous posts file (tab delimiter) in hdfs.

14. Import the only specified columns with `--columns` for authors in hdfs home directory. The imported columns are first_name, last_name, email.

```
$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --table authors --fields-terminated-by '\t' --columns "first_name, last_name, email"
```

```
[student@localhost ~]$ hdfs dfs -tail authors/part-m-00000
ed      Mills   yundt.marisa@example.net
Jammie  Wiza    tomas.konopelski@example.org
Khalid  Brekke  stracke.ivah@example.net
Abdullah        Olson   stephanie72@example.com
Laurie  Hammes  nharber@example.org
Mittie  Hoeger  jonatan.swift@example.org
Travis  Smitham peggie.dickinson@example.com
Owen    Walsh   abdiel.frami@example.net
Adam    Keeling ottis45@example.org
Gordon  Thompson        heidi.huel@example.org
Dorthy  Hermann cebert@example.com
Dan     Green   moore.maxime@example.com
Ross    Bergnaum        celine15@example.org
Alexie  Goldner leonie.hansen@example.org
Raymundo        Hirthe  sigurd.marks@example.net
Alta    Thiel   heidenreich.deshaun@example.com
Alexys  Ferry   sam.grant@example.com
Andreane        Lind    johanna52@example.org
Hilario Sipes   zbraun@example.org
Holden  Reinger nolan.christ@example.com
Darred  Skiles  millie.mayert@example.com
Miller  Schumm  uschuppe@example.net
Candida Hamill  fvolkman@example.org
Edmond  Johns   beatty.helga@example.com
Prince  Bartoletti      oswald76@example.com
Maya    Quitzon eldon.flatley@example.org
Alexandre       Ratke   turner.cornelius@example.com
```

Figure 6. Result of Sqoop command

15. Import the only matching row with `--where` statement. The imported rows are the first named 'Dorthy' in the authors table.

```
$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --table authors --fields-terminated-by '\t' --where "first_name='Dorthy'" –-target-dir authors_Dorthy
```

Note: Output of Hadoop jobs is saved as one or more “partition” files. Usually 4 files are created, and query results are stored in arbitrary files.

16. Import a table using an alternate file format instead of text format. Import the authors table to Parquet format.

```
$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --table authors --target-dir /mywarehouse/authors_parquet --as-parquetfile
```

17. view the results of the import commands by listing the contents in HDFS (target-dir).

```
[student@localhost ~]$ hdfs dfs -ls /mywarehouse/authors_parquet
Found 6 items
drwxr-xr-x   - student supergroup          0 2021-10-31 18:00 /mywarehouse/authors_parquet/.metadata
drwxr-xr-x   - student supergroup          0 2021-10-31 18:00 /mywarehouse/authors_parquet/.signals
-rw-r--r--   1 student supergroup      97492 2021-10-31 18:00 /mywarehouse/authors_parquet/5e44cda6-728c-4912-864a-94d0659930f3.parquet
-rw-r--r--   1 student supergroup      97397 2021-10-31 18:00 /mywarehouse/authors_parquet/a6582da0-64b9-4316-ab99-7dcfddf0a9ed.parquet
-rw-r--r--   1 student supergroup      97715 2021-10-31 18:00 /mywarehouse/authors_parquet/b6c558f1-4b36-45ee-a145-708f3eac8f23.parquet
-rw-r--r--   1 student supergroup      97582 2021-10-31 18:00 /mywarehouse/authors_parquet/c42c91a2-79e5-413b-a55b-2f0f85c74a4e.parquet
```

Figure 7. List of Parquet files

Each Parquet file is given a unique name such as 5e44cda6-728c-4912-864a-94d0659930f3.parquet (<parquet_file_name>), and this file format cannot be viewed directly because it is a binary format. Use the parquet-tools `show` command to view the records in the set of data files. First, you have to get parquet file to local and run the parquet-tools command.

```
$ hdfs dfs -get /mywarehouse/authors_parquet/<parquet_file_name>
$ parquet-tools show <parquet_file_name>
```

```
| 4965 | Rae          | Carroll       | bill.kertzmann@example.org        |  441212400000 | 1059685635000 |
| 4966 | Gino         | Murazik       | litzy91@example.com               | 1299942000000 | 1047975239000 |
| 4967 | Axel         | Schneider     | trudie60@example.net              | 1421334000000 | 1076066714000 |
| 4968 | Joyce        | Lakin         | christopher.crist@example.org     |  514911600000 |  601662564000 |
| 4969 | Elias        | Corwin        | aglae49@example.com               | 1305903600000 | 1242686781000 |
| 4970 | Kayli        | Kihn          | stiedemann.cielo@example.org      |  746722800000 |  253598614000 |
| 4971 | Laura        | Wolff         | madge.hirthe@example.net          | 1456671600000 |  219668579000 |
| 4972 | Agustin      | Bahringer     | jayda92@example.net               | 1460214000000 |  771999322000 |
| 4973 | Laurel       | Tremblay      | ericka66@example.org              |  338482800000 |  316857814000 |
| 4974 | Sylvia       | Swift         | prosacco.palma@example.com        | 1131980400000 |  301255295000 |
| 4975 | Abraham      | Kirlin        | kdare@example.org                 | 1008774000000 |  475420713000 |
| 4976 | Katelyn      | Nicolas       | co'hara@example.com               | 1155394800000 | 1069368442000 |
| 4977 | Laverna      | Feest         | fritz.mosciski@example.com        |  730998000000 |  554992578000 |
| 4978 | Manuel       | Bradtke       | laura.hauck@example.net           |   98290800000 |   94076368000 |
| 4979 | Amely        | Effertz       | dweissnat@example.com             | 1248188400000 | 1312011333000 |
| 4980 | Amya         | Champlin      | morar.urban@example.org           |  246553200000 |  629733336000 |
| 4981 | Nathanial    | Prosacco      | jordane70@example.org             |  508863600000 |   43328576000 |
| 4982 | Marilie      | Kiehn         | emmett.aufderhar@example.org      | 1291561200000 | 1402507545000 |
| 4983 | Marie        | Hessel        | mariano.connelly@example.com      | 1013958000000 | 1157309893000 |
| 4984 | Abdiel       | Stiedemann    | dmorar@example.org                | 1335020400000 |  787219649000 |
| 4985 | Luna         | Tromp         | schneider.lora@example.net        |  922892400000 | 1531190553000 |
| 4986 | Payton       | Blick         | johanna67@example.net             |  648486000000 |  736460842000 |
| 4987 | Ismael       | Rohan         | macejkovic.catharine@example.net  |   31330800000 |  414412061000 |
| 4988 | Jaunita      | Durgan        | trantow.neil@example.org          |  106758000000 | 1429898230000 |
| 4989 | Javier       | Lindgren      | corbin.wolf@example.net           | 1506783600000 |   38242464000 |
| 4990 | Larissa      | Thiel         | tbuckridge@example.com            | 1364482800000 | 1203953980000 |
| 4991 | Miles        | Schowalter    | sheldon.rolfson@example.net       |   92934000000 |  131942054000 |
| 4992 | Breanna      | Metz          | parisian.randi@example.org        |  353430000000 |  165863586000 |
| 4993 | Garrett      | Little        | ashley50@example.net              |  257439600000 |  774366161000 |
| 4994 | Izabella     | Hilll         | xmacejkovic@example.org           |  947170800000 | 1225902378000 |
| 4995 | Newell       | Ledner        | delphia70@example.org             |  256662000000 |  133328982000 |
| 4996 | Marta        | Homenick      | purdy.titus@example.org           | 1385391600000 | 1482346861000 |
| 4997 | Esteban      | Lehner        | louisa.fritsch@example.org        |  333903600000 | 1255941968000 |
| 4998 | Loyce        | Nikolaus      | mariela.wyman@example.org         |  184777200000 |  399152685000 |
| 4999 | Jackeline    | Huel          | elaina33@example.org              |  126630000000 |   86069565000 |
| 5000 | Maudie       | Gutkowski     | viola34@example.net               |  892911600000 |  120857921000 |
+------+--------------+---------------+-----------------------------------+---------------+---------------+
```

18. Import a table using a compression option `--compress` or `-z` for authors table.

```
$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --table authors --target-dir /mywarehouse/authors_compressed –compress
```

```
[student@localhost works]$ hdfs dfs -ls /mywarehouse/authors_compressed
Found 5 items
-rw-r--r--   1 student supergroup          0 2021-10-31 18:10 /mywarehouse/authors_compressed/_SUCCESS
-rw-r--r--   1 student supergroup      72776 2021-10-31 18:10 /mywarehouse/authors_compressed/part-m-00000.gz
-rw-r--r--   1 student supergroup      72745 2021-10-31 18:10 /mywarehouse/authors_compressed/part-m-00001.gz
-rw-r--r--   1 student supergroup      72689 2021-10-31 18:10 /mywarehouse/authors_compressed/part-m-00002.gz
-rw-r--r--   1 student supergroup      72760 2021-10-31 18:10 /mywarehouse/authors_compressed/part-m-00003.gz
[student@localhost works]$
```

Figure 8. List of compressed files

19. First, import the rows whose first name is "Dorthy" performed in step 15, and save it as dorthy folder in the hdfs home directory.

```
$ sqoop import --connect jdbc:mysql://localhost/labs --username student --password student --table authors --fields-terminated-by '\t' --where "first_name='Dorthy'" --target-dir dorthy
```

Export the saved dorthy folder as a table to the labs DB of RDBMS.

```
$sqoop export --connect jdbc:mysql://localhost/labs --username student --password student --table authors_export --fields-terminated-by '\t' --export-dir dorthy
```

20. Review the contents of the exported records in MariaDB.

id	first_name	last_name	email	birthdate	added
1298	Dorthy	Dietrich	ibrekke@example.com	1983-02-12	1999-08-07 10:35:08
2484	Dorthy	Hermann	cebert@example.com	1999-06-14	2017-10-31 07:50:47
3377	Dorthy	West	mayer.braden@example.com	2001-02-08	2008-05-10 20:54:13

Figure 9. exported records in MariaDB

Lab 2: Data Ingestion with Apache Flume

In this lab, you run the Flume agent to collect data from various data sources and store it as HDFS or local filesystem.

1. Simple Data Transfer

This Agent allows the user to generate events and subsequently log them to the console. This configuration defines a single agent named agent1.

1.1. Create configuration file

```
mkdir flume
cd flume
vi transfer.conf
```

1.2. Agent1 configuration file

The agent1 has a source that listens for data on port 3333, a channel that buffers event data in memory, and a sink that logs event data to the console.

```
agent1.sources = netcatSrc
agent1.channels = memChannel
agent1.sinks = log

agent1.sources.netcatSrc.channels = memChannel
agent1.sinks.log.channel = memChannel
agent1.sources.netcatSrc.type = netcat
agent1.sources.netcatSrc.bind = 0.0.0.0
agent1.sources.netcatSrc.port = 3333

agent1.sinks.log.type = logger
agent1.channels.memChannel.type = memory
agent1.channels.memChannel.capacity = 100
```

1.3. Flume agent1 execution

```
flume-ng agent -name agent1 –conf-file transfer.conf
```

1.4. Open another terminal window and execute the telent command.

```
telnet localhost 3333
Typing whatever you want...
Hadoop
..
```

```
[student@localhost flume]$ telnet localhost 3333
Trying ::1...
Connected to localhost.
Escape character is '^]'.
this is a test message.
OK
hadoop
OK
speak
```

1.5. Check that the message sent to telnet in step 4 is output from the terminal where the flume agent was executed in step 3

```
2021-08-01 18:15:14,834 INFO node.Application: Starting Sink log
2021-08-01 18:15:14,842 INFO node.Application: Starting Source netcatSrc
2021-08-01 18:15:14,842 INFO source.NetcatSource: Source starting
2021-08-01 18:15:14,851 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerS
ocketChannelImpl[/0:0:0:0:0:0:0:0:3333]
2021-08-01 18:15:37,067 INFO sink.LoggerSink: Event: { headers:{} body: 74 68 69 73 20 69
 73 20 61 20 74 65 73 74 20 6D this is a test m }
2021-08-01 18:16:01,024 INFO sink.LoggerSink: Event: { headers:{} body: 68 61 64 6F 6F 70
 0D                              hadoop. }
2021-08-01 18:16:06,953 INFO sink.LoggerSink: Event: { headers:{} body: 73 70 65 61 6B 0D
                                 speak. }
```

Note: If the telnet is not closed properly, the port is not closed properly, and when you try to connect again, an error that the port is already open may occur.

1.6. telnet close with command after ctrl + quit.

```
^] (ctrl+])
telnet> close
```

2. Basic Data Transfer with spool directory

This agent 2 is to save the files coming into the spool directory to the local directory.

2.1. Create configuration file

```
vi transfer_spool.conf
```

2.2. Agent2 configuration file

```
agent2.sources = dirSrc
agent2.channels = memChannel
agent2.sinks = fileSink
agent2.sources.dirSrc.channels = memChannel
agent2.sinks.fileSink.channel = memChannel
agent2.sources.dirSrc.type = spoolDir
agent2.sources.dirSrc.spoolDir = /home/student/flume/incoming
agent2.sinks.fileSink.type = file_roll
agent2.sinks.fileSink.sink.directory = /home/student/flume/output
agent2.sinks.fileSink.sink.rollInterval = 0
agent2.channels.memChannel.type = memory
agent2.channels.memChannel.capacity = 100
```

2.3. Flume agent2 execution

```
flume-ng agent -name agent2 –conf-file transfer_spool.conf
```

2.4. Open another terminal window and copy two sql files to spool directory.

```
mkdir -p flume/incoming flume/output
cd /home/student/flume/incoming
cp ~/Data/*.txt .
vi hello.txt
This is test file for Flume.
```

```
[student@localhost incoming]$ pwd
/home/student/flume/incoming
[student@localhost incoming]$ ls -l
total 192
-rwxr-x---. 1 student student 174313 Aug 10 22:40 alice_in_wonderland.txt.COMPLETED
-rw-rw-r--. 1 student student     29 Aug 10 22:40 hello.txt.COMPLETED
-rwxr-x---. 1 student student   4987 Aug 10 22:40 pig_data1.txt.COMPLETED
-rwxr-x---. 1 student student   5240 Aug 10 22:40 pig_data2.txt.COMPLETED
[student@localhost incoming]$
```

2.5. You can check the message that pig_data1.txt, pig_data2.txt, alice_in_wonderland.txt, and hello.txt copied to the spool directory in step 4 are transmitted to the terminal where Agent 2 is running.

```
2021-08-10 22:45:36,236 INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /ho
me/student/flume/incoming/alice_in_wonderland.txt to /home/student/flume/incoming/alice_in_wo
nderland.txt.COMPLETED
2021-08-10 22:45:40,237 INFO avro.ReliableSpoolingFileEventReader: Last read took us just up
to a file boundary. Rolling to the next file, if there is one.
2021-08-10 22:45:40,238 INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /ho
me/student/flume/incoming/pig_data1.txt to /home/student/flume/incoming/pig_data1.txt.COMPLET
ED
2021-08-10 22:45:44,241 INFO avro.ReliableSpoolingFileEventReader: Last read took us just up
to a file boundary. Rolling to the next file, if there is one.
2021-08-10 22:45:44,241 INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /ho
me/student/flume/incoming/pig_data2.txt to /home/student/flume/incoming/pig_data2.txt.COMPLET
ED
2021-08-10 22:45:44,243 INFO avro.ReliableSpoolingFileEventReader: Last read took us just up
to a file boundary. Rolling to the next file, if there is one.
2021-08-10 22:45:44,243 INFO avro.ReliableSpoolingFileEventReader: Preparing to move file /ho
me/student/flume/incoming/hello.txt to /home/student/flume/incoming/hello.txt.COMPLETED
```

Also, you can check the transmission of hello.txt created with vi. The transferred files are saved in the output directory.

2.6. The transferred files are stored as files in the OUTPUT directory

```
[student@localhost output]$ ls -l
total 184
-rw-rw-r--. 1 student student 184569 Aug 10 22:45 1628603135792-1
```

3. Using Interceptor

This agent3 is the role of inserting the IP address of the host where the agent is running into the event header.

3.1. Create configuration file

```
vi interceptor.conf
```

3.2. Agent3 configuration file

```
agent3.sources = netcatSrc
agent3.channels = memChannel
agent3.sinks = log

agent3.sources.netcatSrc.channels = memChannel
agent1.sinks.log.channel = memChannel
agent1.sources.netcatSrc.type = netcat
agent1.sources.netcatSrc.bind = 0.0.0.0
agent1.sources.netcatSrc.port = 3333

agent1.sinks.log.type = logger
agent1.channels.memChannel.type = memory
agent1.channels.memChannel.capacity = 100
agent03.sources.netcatSrc.interceptors = i1
agent03.sources.netcatSrc.interceptors.i1.type = host
agent03.sources.netcatSrc.interceptors.i1.hostHeader = hostname
```

3.3. Flume agent3 execution.

```
flume-ng agent -name agent3 –conf-file interceptor.conf
```

3.4. Open another terminal window and execute the telent command.

```
telnet localhost 3333
This is testing Flume with interceptor.
Hadoop
Spark
```

```
telnet localhost 3333
Trying ::1...
Connected to localhost.
Escape character is '^]'.
This is Flume test.
OK
Hadoop
OK
Spark
OK
```

3.5. The message sent to telnet in step 4 is output from the terminal where the flume agent was executed in step 3, and it is confirmed that the IP address where the agent is currently running is inserted into the event header and transmitted.

```
2021-08-01 19:46:02,589 INFO node.Application: Starting Sink log
2021-08-01 19:46:02,590 INFO node.Application: Starting Source netcatSrc
2021-08-01 19:46:02,590 INFO source.NetcatSource: Source starting
2021-08-01 19:46:02,599 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerS
ocketChannelImpl[/0:0:0:0:0:0:0:0:3333]
2021-08-01 19:46:40,618 INFO sink.LoggerSink: Event: { headers:{hostname=127.0.0.1} body:
 54 68 69 73 20 69 73 20 46 6C 75 6D 65 20 74 65 This is Flume te }
2021-08-01 19:46:50,749 INFO sink.LoggerSink: Event: { headers:{hostname=127.0.0.1} body:
 48 61 64 6F 6F 70 0D                             Hadoop. }
2021-08-01 19:46:59,762 INFO sink.LoggerSink: Event: { headers:{hostname=127.0.0.1} body:
 53 70 61 72 6B 0D                                Spark. }
```

Note: IP is 127.0.0.1

3.6. Delete the temporary directory used for the flume operations.

```
$cd ~/flume
$rm -rf incoming output
```

4. Create a new Flume dataflow from the previous labs.

4.1. Create a Flume dataflow(agent) in lab.conf and show the result from the terminal.

a. The agent name is agent7. The agent7 has a configuration with the following:

Source	
Type	Netcat
Bind	localhost
Port	1111
Channel	
Type	Disk
Capacity	1000
transactionCapacity	100
Sink	
Type	logger

b. List the lab.conf file.
c. List the command to start the agent and type the “Hello world!” in terminal.
d. List the result from the other terminal.

Lab 3: Using Kafka

1. Creating a Kafka Topic

Create a Kafka topic named topic1_logs that will contain messages representing lines in log files.

1.1. Use kafka-topics to create a topic

1.1.1. Execute the following code from a terminal to create topic1_logs topic

```
$kafka-topics –create \
--bootstrap-server localhost:9092 \
--replication-factor 1\
--partitions 1\
--topic topic1_logs
```

You will see the message: Created topic “topic1_logs”.

Note:If you previously worked on an lab that used Kafka, you may get an error here indicating that this topic already exists. You may disregard the error.

1.1.2. Use the –list option to display all kafka topics and confirm that the new topic you just created is listed:

```
$ kafka-topics –list \
--bootstrap-server localhost:9092
```

1.2. Review the details of the topic1_logs.

```
$kafka-topics –describe topic1_log \
--bootstrap-server localhost:9092
```

2. Create producers and consumers for a topic

2.1. Open 2 terminals and create the producer on one and the consumer on the other.

2.1.1. From the first terminal use kafka-console-producer command to start the producer.

```
$kafka-console-producer \
--broker-list localhost:9092 \
--topic topic1_logs
```

```
[student@localhost Labs]$ kafka-console-producer \
> --broker-list localhost:9092 \
> --topic topic1_logs
>
```

Notice that the kafka-console-producer is waiting for text to be typed in. Text that is type here will become a message in the Kafka topic topic1_logs.

2.1.2. From the second terminal, use kafka-console-consumer to create a consumer for the topic

```
kafka-console-consumer \
--bootstrap-server localhost:9092 \
--topic topic1_logs \
--from-beginning
```

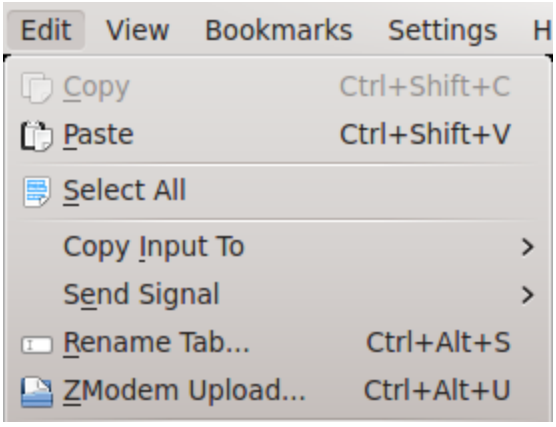
```
[student@localhost ~]$ kafka-console-consumer \
> --bootstrap-server localhost:9092 \
> --topic topic1_logs \
> --from-beginning
```

Notice that the kafka-console-consumer is waiting for messages to arrive at kafka topic topic1_logs.

2.2. Rename the terminals.

2.2.1. From the producer terminal, select Edit > Rename Tab and change the terminal tab Producer.

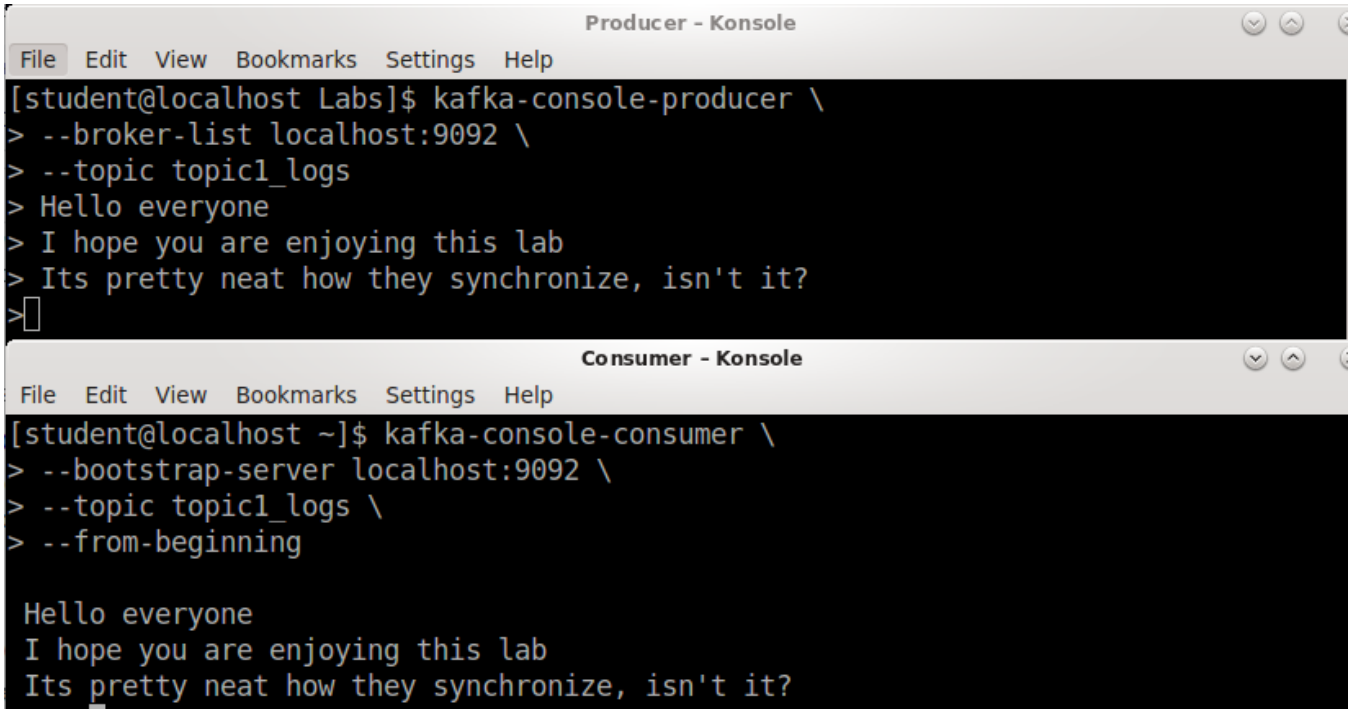
2.2.2. Do the same from the consumer terminal, naming it Consumer.



2.3. Produce messages for topic topic1_logs.

2.3.1. Begin typing something from the **Producer terminal** where the producer is running

2.3.2. Observe that in **Consumer terminal**, the consumer will pull messages that have been pushed by the producer.



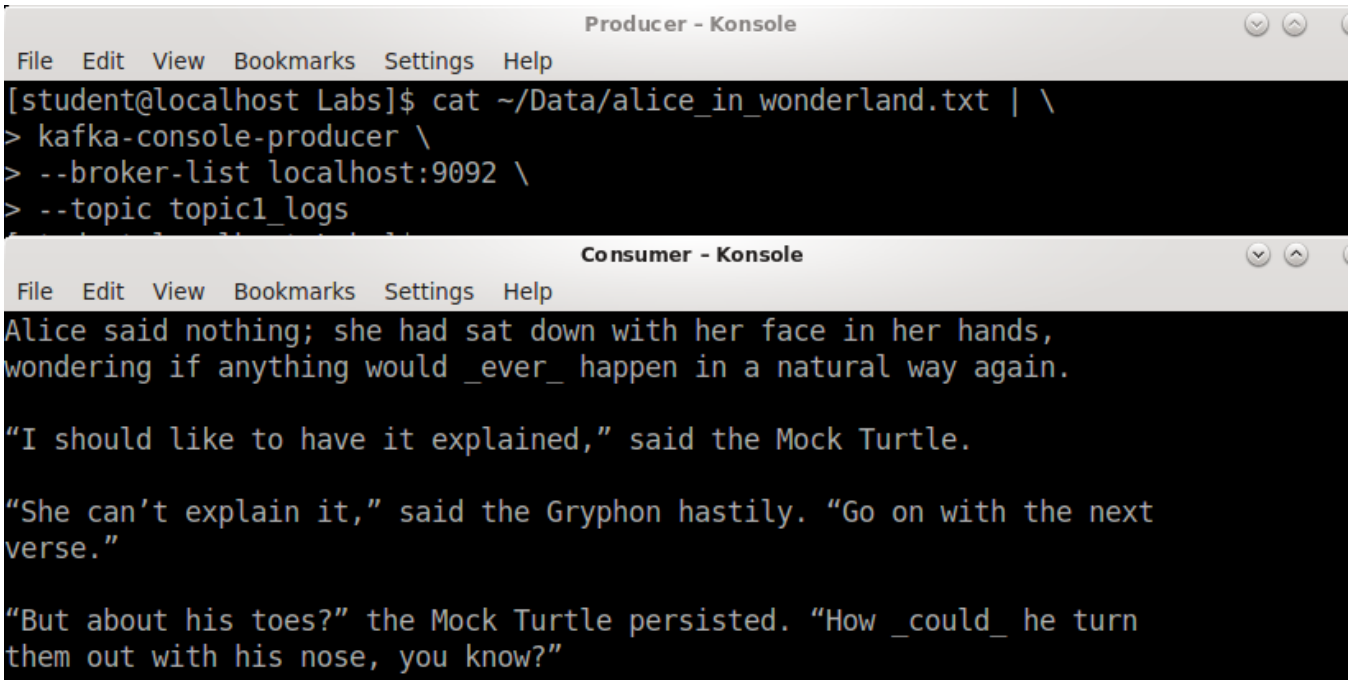
2.4. Create messages in batch mode.

2.4.1. From the Producer terminal, stop the Producer by sending a Ctlr-C KeyboardTerminate signal

2.4.2. Send the entire contents of Alice-in-Wonderland.txt file to the topic1_logs topic

```
cat ~/Data/alice_in_wonderland.txt | \
kafka-console-producer \
--broker-list localhost:9092 \
--topic topic1_logs
```

What happened? It went very fast. I hope you didn't blink. The entire content of the book "Alice in Wonderland" was passes as messages by the Producer and then picked up by the Consumer.



2.5. Clean up

2.5.1. Stop producer and consumer as necessary using Ctrl-C kill signal.

END OF LAB