Chapter 3. Big Data Ingestion

Exercise Workbook

Contents

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 |
+--------------------+
6 rows in set (0.00 sec)
```

Figure 1. List databases in MariaDB

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

```
MariaDB [labs]> show tables;
+----------------+
| Tables_in_labs |
+----------------+
| authors        |
| authors_export |
| posts          |
+----------------+
3 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;
```

```
MariaDB [labs]> SELECT id, first_name, last_name, email, added FROM authors limit 5;
+----+------------+------------+-----------------------------+---------------------+
| id | first_name | last_name  | email                       | added               |
+----+------------+------------+-----------------------------+---------------------+
|  1 | Walton     | Adams      | barmstrong@example.com      | 1997-01-02 04:18:41 |
|  2 | Marietta   | Walsh      | hand.stella@example.net     | 2010-08-26 18:20:14 |
|  3 | Lily       | Wintheiser | darren.blanda@example.org   | 1973-06-11 07:28:12 |
|  4 | Estevan    | Gleason    | shanahan.aliyah@example.net | 1995-01-29 16:08:31 |
|  5 | Thaddeus   | Rowe       | bednar.robin@example.net    | 2017-01-05 04:13:48 |
+----+------------+------------+-----------------------------+---------------------+
5 rows in set (0.00 sec)
```

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 use better to '\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
Jarred  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/test --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).

Figure 7. List of Parquet files

Each Parquet file is given a unique name such as 5e44cda6-728c-4912-864a-94d0659930f3.parquet, 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/5e44cda6-728c-4912-864a-94d0659930f3.parquet

$ parquet-tools show 5e44cda6-728c-4912-864a-94d0659930f3.parquet
```

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.

```
MariaDB [test]> select * from authors_export;
+------+------------+-----------+-------------------------+------------+---------------------+
| 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

21. Create a target directory in HDFS to import table data into for labs (/tmp/mylabs).

    21.1. From the posts table, import only the primary key, along with posting title, posting date to HDFS directory /tmp/mylabs/posts_info. Please save the file in text format with tab delimiters.

Hint: You will have to figure out what the name of the table columns are in order to complete this lab using just sqoop command instead of MariaDB access.

    21.2. This time save the same in parquet format with snappy compressing. Save it in /tmp/mylabs/posts_compressed.

    21.3. From the terminal, display some of the records that you just imported.

    21.4. Import the only specified columns for authors in hdfs home directory. The imported columns are id, first_name, last_name, birthdate with tab field delimiter. Save the file in text format.

Hint: If the authors directory exists in the HDFS home directory, delete it and execute the import command.

21.5. Import and save in /tmp/mylabs/posts_NotN only posts which title is not null column and id is in posts table. And the imported columns are id, title, content not all columns. Save the file in parquet format and compressed using snappy codec.

Hint: The SQL command for retrieving data whose field value is not null is "column_name is not null".

SQL usage will be explained in a later lecture.

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/data/spool
agent2.sinks.fileSink.type = file_roll
agent2.sinks.fileSink.sink.directory = /home/student/data/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 scratch

4.1. Create a new flume configuration file with the following:

Source	
Type	Netcat
Bind	localhost

Port	11111
Channel	
Type	Disk
Capacity	1000
transactionCapacity	100
Sink	
Type	logger

4.2. Start the agent

4.3. From another terminal start telnet and connect to port 44444. Start typing and you should see the result from the other terminal.

```
$telnet localhost 11111
...
Hello world! <ENTER>
OK
```

Lab 3: Making your first dataflow

1. Start the NiFi Service

NiFi supports user authentication via client certificates, via username/password, via Apache Knox, or via OpenId Connect. The default option is Single User with username and password. In order to authenticate and synchronize with the Linux user, one of the more sophisticated authentication mechanisms must be installed. However, installing such would be beyond the scope of the labs and beyond the resources available within the virtual machine.

In order to facilitate the labs, the NiFI service will be run as the Linux root user.

1.1. Open a terminal and run the following command to start NiFi services

```
sudo service nifi start
```

```
[student@localhost ~]$ sudo service nifi start

Java home: /opt/jdk1.8.0_291
NiFi home: /usr/local/nifi/nifi-1.14.0

Bootstrap Config File: /usr/local/nifi/nifi-1.14.0/conf/bootstrap.conf
```

1.2. Open the Firefox browser and open the NiFi Web UI. It may take some time for the NiFi service to come up. You may have to wait a few minutes if you cannot connect to the NiFi Web UI right away.

```
https://localhost:8443/nifi
```

1.3. Login to the NiFI service. The current username is student123456789 and password is also student123456789.

2. Explore the NiFi User Interface

2.1. Click and drag a processor icon to the canvas from the components toolbar

2.2. Explore the Add Processor pop up window

2.2.1. Trying clicking on hadoop from the category choices on the left tab. Observe how the selection of available processors in the middle changes. How many processors are related to Hadoop?

You will see on the top of the screen, text saying display nn of NNN. There are NNN processors in total in this version of NiFi. Of those, nn are related to Hadoop.

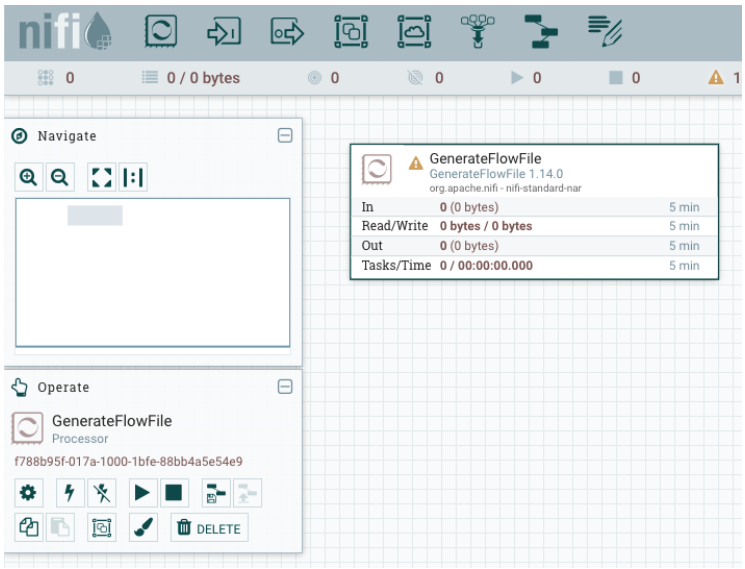
2.2.2. Try combining hadoop with other categories such as put or get. What happens if you select both put and get? Now remove the get? What happens?

When you try to select both put and get, you will notice that there are no matching processors. This is because there are no Hadoop related processors that can perform both get and put operations at the same time.

2.2.3. With hadoop and put selected on the left pane, try entering "Gener" in the filter on the top left of the pop up window. Notice that there are no available selections again. What happened?

2.2.4. The filter and the categories work together. There are no processor that has "Gener" in the name of the processor that is part of both the Hadoop and put category.

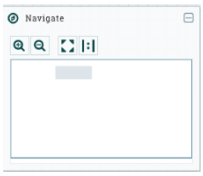
2.2.5. Now, remove Hadoop and put from the category filters and try entering "Gener" in the search filter. You will now see several processors that can be selected. Choose the GenerateFlowFile processor selecting it and Add. Alternatively, you can select by double clicking it. You screen should look similar to below



2.3. Explore navigating the canvas

2.3.1. Move the GenerateFlowFile processor around the canvas by clicking and dragging it.

2.3.2. Move the canvas around by clicking on an empty space within the canvas and dragging the mouse around. Notice that the small box in the Navigate palette screen on the left moves around when you move the canvas.



2.3.3. This time, try moving the box in the Navigate Palette. What happens?

2.3.4. Zoom in and out of the canvas by moving the middle mouse wheel up and down. Alternatively, try using the + and - buttons in the Navigate palette.

2.3.5. Make the processor icon very small by zooming out and then try clicking the fit and actual buttons on the Navigate Palette.

2.4. Explore the Configure Processor pop-up window by either double clicking the GenerateFlowFile processor or right-click and selecting configure.

2.4.1. Observe the SETTINGS, SCHEDULING, PROPERTIES, and COMMENTS tab on the top. Recall setting the processor unit during the lectures.

2.4.2. Change the name of the processor to Lab1-GenerateFlowFile.

2.4.3. From the SCHEDULING tab, change the Run Schedule to 2 sec. The processor will fire up and generate flow files every 2 seconds.

2.4.4. From the PROPERTIES tab, change the File Size to 100 KB. Keep the Batch Size, Data Format and UniqueFlowFiles the same. Click on the question mark for each of these parameters to see what they control.

2.4.5. Finally, put your own personal comment about this processor from the COMMENTS tab

2.4.6. When finished, click APPLY to save the changed parameters.

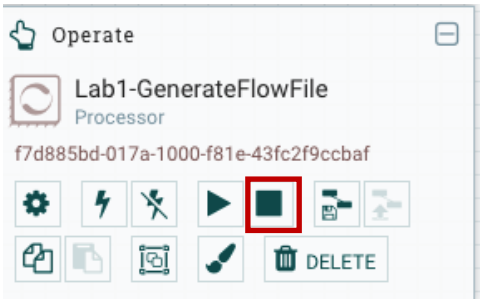
2.5. Explore working with the Processor

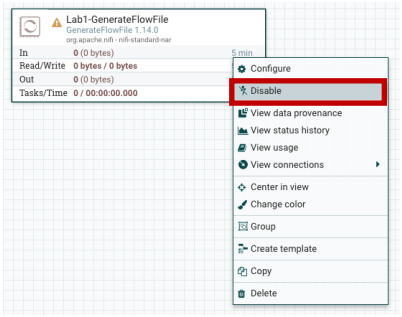
2.5.1. Select the Lab 1-GenerateFlowFile processor by clicking on it.

2.5.2. Make a copy of the processor. There are several ways to accomplish this. You can use the copy followed by the paste button  from the Operate Palette. Alternatively, you can use the context menu by right-clicking the processor and selecting copy and then following with a right-click on the canvas and selecting paste. Each right click brings up a different menu based on the context.

2.5.3. Select both processor by either holding down the shift key and selecting each processor or holding down the shift key and clicking and dragging around the two processors to create a selection box.

2.5.4. Disable the second Lab1-GenerateFlowFile processor. You can do this by using disable button from the Operate Palette or the disable option from the context menu.





2.5.5. Now delete the disabled processor that we just created. How would you do this?

Use the delete button from the Operate Palette or the delete option from the context menu after right-clicking on the processor to delete.

3. Creating a simple dataflow

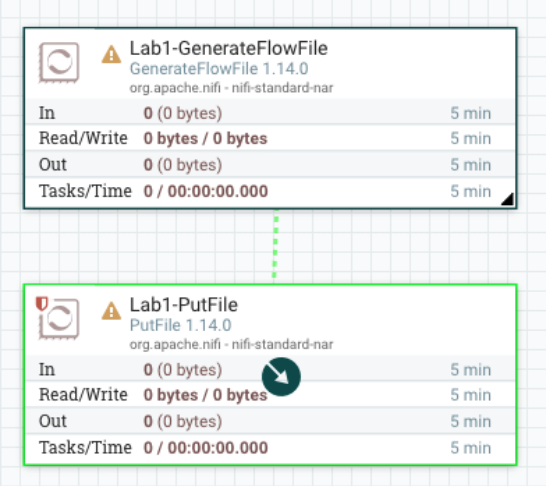
3.1. Add processors and connections

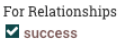
3.1.1. Add a PutFile processor to the canvas

3.1.2. Open the Global Menu from the top left and select Help. From the bottom left, select PutFile and learn what this processor does. Pay attention to the properties as we will need to set them shortly.

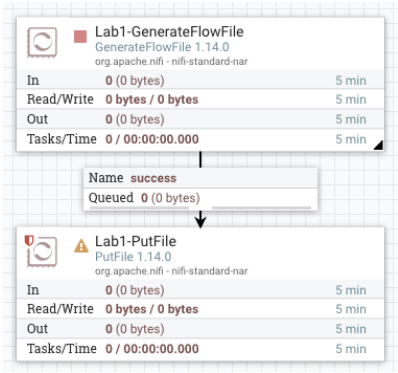
3.1.3. Now that we know what the PutFile processor does, we want it to save flowfiles in the /tmp/nifi/lab1 directory. We want the processor to replace any flowfiles that already exist in the directory. In case the /tmp/nifi/lab1 directory does not already exist, we want the processor to automatically create it. The files should be saved as student so change the Owner and Group to "student." Finally, we want to name this processor Lab1-PutFile.

3.1.4. Create a Connection between the Lab1-GenerateFlowFile and Lab1-PutFile processor by clicking and dragging the icon from the middle of the Lab1-GenerateFlowFile processor to the middle of Lab1-PutFile processor.



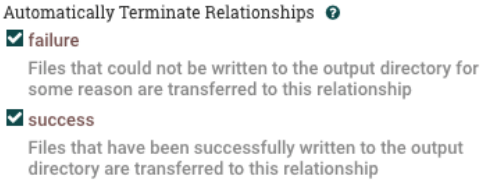
3.1.5. Select the success option for the For Relationships selection 

and ADD the connection. When you are done, your data flow should look similar to below.



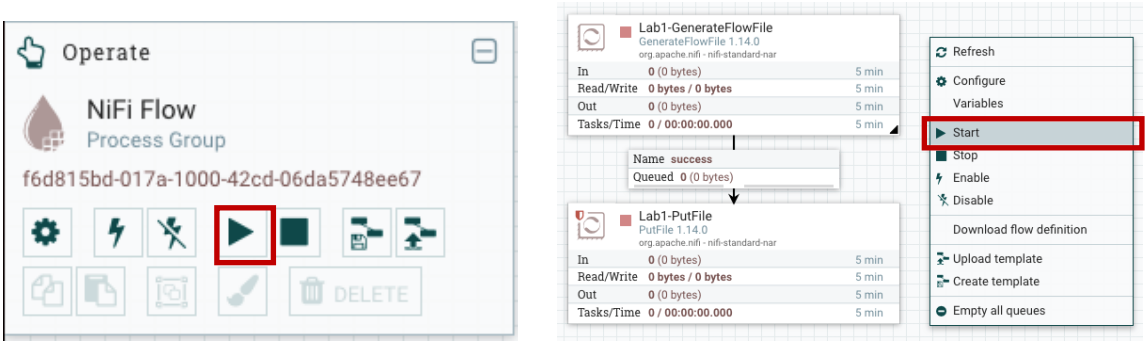
3.1.6. The yellow exclamation triangle in the Lab1-PutFile processor indicates that there is a problem with the setting for this processor. Hover the mouse over the yellow icon to see what the problem is.

3.1.7. Recall from our lectures that all Relationships must be either directed forward to another processor or terminated. There are a couple of relationships that need to be terminated. Since after saving the flowfiles to disk, we will not need to forward them further downstream, lets auto-terminate them from this processor. Go to the SETTING tab in the processor and check both failure and success Relationships for auto-termination.

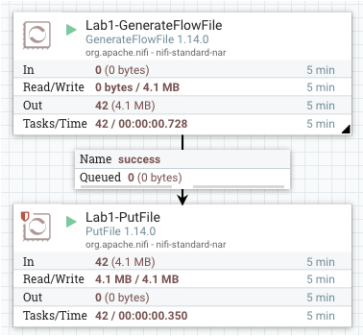


3.2. Run the Processors

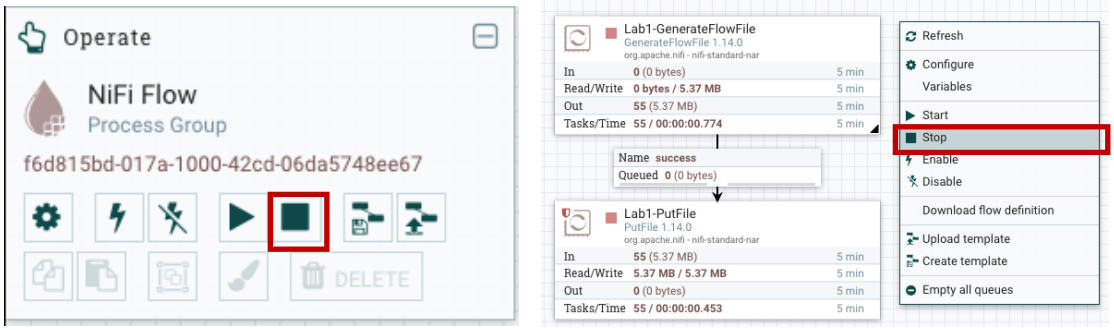
3.2.1. Now both processors should be in a stopped state (red square) and we are ready to run the processors. Select both processors and click on the Run button from the Operate Palette or choose the Start option from the canvas context menu.



3.2.2. Both processors should transition to the run state and you will see data movement from the 5 minute statistics.

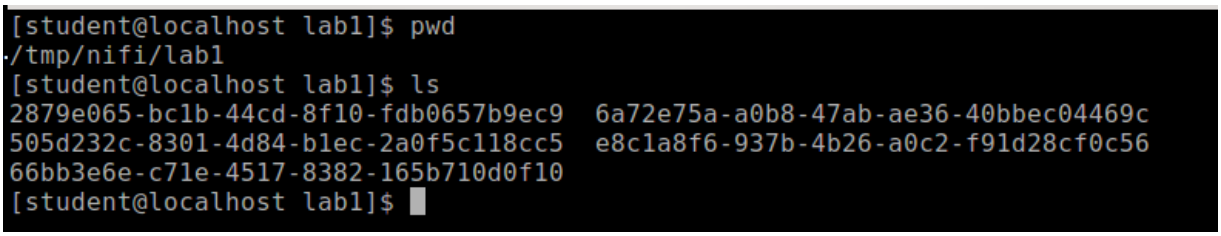


3.2.3. Stop both processors. The procedure is analogous to starting the processors but this time select the stop icons.



3.3. Verify the data pipeline

3.3.1. Open a terminal and navigate to the local /tmp/nifi directory. Get a listing of the directory to verify that the PutFile processor has saved the flowfiles in this directory. Your output will look similar to below.


[student@localhost lab1]$ pwd
/tmp/nifi/lab1
[student@localhost lab1]$ ls
2879e065-bc1b-44cd-8f10-fdb0657b9ec9 6a72e75a-a0b8-47ab-ae36-40bbec04469c
505d232c-8301-4d84-b1ec-2a0f5c118cc5 e8c1a8f6-937b-4b26-a0c2-f91d28cf0c56
66bb3e6e-c71e-4517-8382-165b710d0f10
[student@localhost lab1]$

Lab 4: Creating Connections

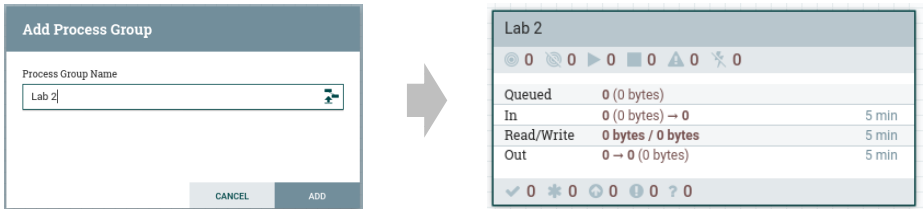
In this lab, we will look at Relationships and their connections between processors. A Relationship in a processor represents a potential outcome of that processor. The available Relationships will be determined by the type of processor as well as attributes in the flowfile. For example, success or failure is a very common Relationship available in most processors. Recall that all Relationships must be either forwarded downstream or auto-terminated.

1. Create a new Processor Group

A Processor Group can be used to organize data flows. Rather that placing all of the data flows on the root canvas, it if often easier to organize and manage our projects by creating Process Groups and creating the data flow within the process group. Process Groups can even be nested creating a hierarchy of processor groups. Using processor groups in this manner is similar to creating directories and subdirectories to organize projects.

1.1. Add a new process group to the root canvas by click and dragging a Processor Group icon to the canvas

1.1.1. Name the process group “Lab 2”



1.1.2. Double click on the newly created processor group. You will be presented with a clean canvas where you can create data flows. Nifi will also let you know where you are in the hierarchy of processor groups from the bread crumb screen. You can move between processor groups by clicking on the desired processor group. In order to return to the root canvas, you can click on the top level “NiFi Flow.” Do not leave the Lab 2 processor group for now. We will create our data flow inside this processor group.

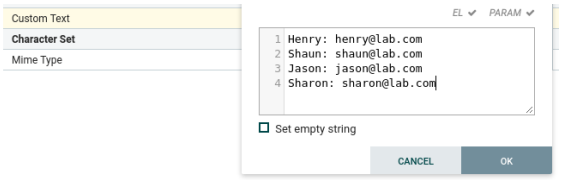
2. Create a data pipeline with multiple paths
   2.1. Add a GenerateFlowFile processor to the canvas
      2.1.1. Change the Name of the processor to Lab2-Good-GenerateFlowFile
      2.1.2. Change the Run Schedule to 10 sec so that we do not generate too many flow files. We want this processor to fire up every 10 seconds.
      2.1.3. In the Properties, leave the File Size to 0B so that we are not restricted by the file size. We will be using some custom text.
      2.1.4. Leave the Batch Size to 1. We want to generate 1 flow file each time this processor fires up.
      2.1.5. We will be using some custom text for the content of the flow file. Select Custom Text and enter the following text. Make sure there is a new line between each entry. In order to add a new line use (shift-enter) :

Henry: henry@lab.com

Shaun: shaun@lab.com

Jason: jason@lab.com

Sharon: sharon@lab.com

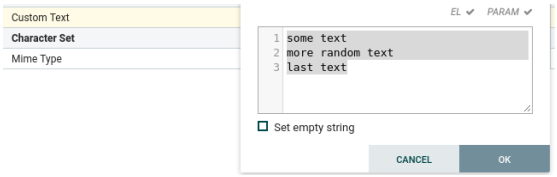


   2.2. Add a second GenerateFlowFile processor to the canvas
      2.2.1. Name the processor Lab2-Bad_GnerateFlowFile
      2.2.2. Run Schedule = 10 seconds
      2.2.3. Custom Text:

some text

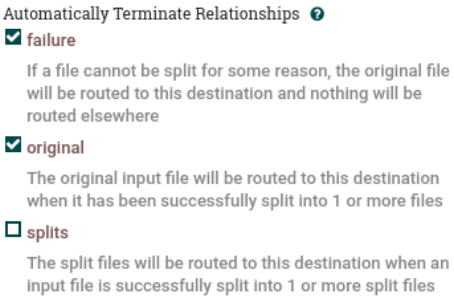
more random text

last text



   2.3. Add a SplitText processor to the canvas
      2.3.1. Name the processor Lab2-SplitText
      2.3.2. Automatically terminate the failure and original Relationships. After the SplitText processor attempts to split the incoming text, there are three potential Relationships. If the attempt to split the text fails, the flow files for that result will be routed to the *failure* Relationship. Similarly, if the split succeeds, it will be routed to the *splits* Relationship. Finally, this processor allows us to also send the original text through the *original* Relationship. By automatically

terminating the *failure* and *original* Relationships, we are effectively preventing any flow files to be further passed downstream for those two situations.

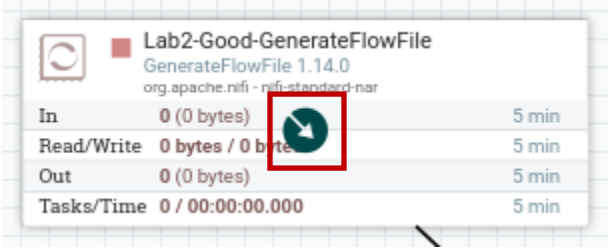


2.3.3. Leave the Run Schedule to 0 sec. This will allow this processor the process any flow files as soon as they become available.

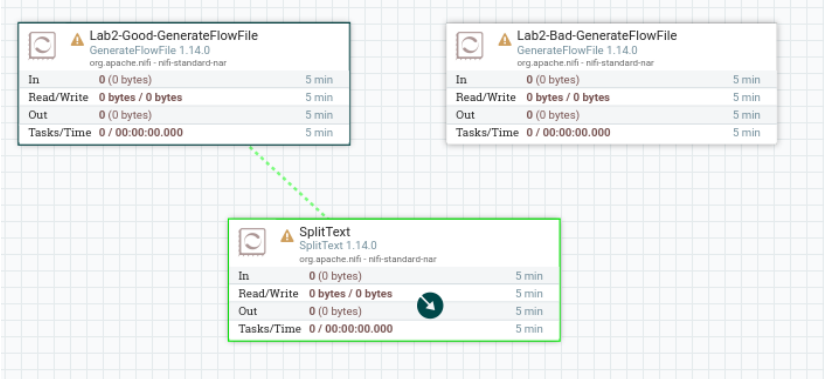
2.3.4. Set the Line Split Count property to 1. This will split the incoming text to a separate flow file for each line of text. Since we have set the upstream GenerateFlowFile processors to send multiple lines of text, this processor will split each line into a new flow file.

2.4. Connect the GenerateFlowFile processors to the SplitText processor

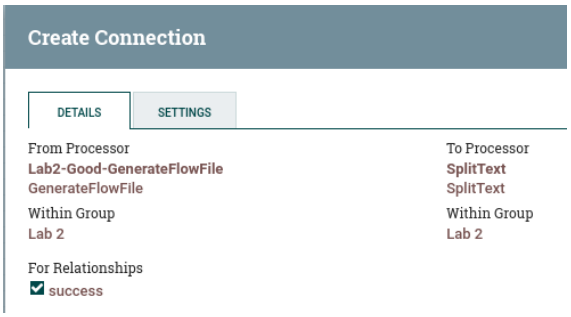
2.4.1. Move the mouse to the middle of Lab2-Good-GenerateFlowFile processor until you see the connection icon



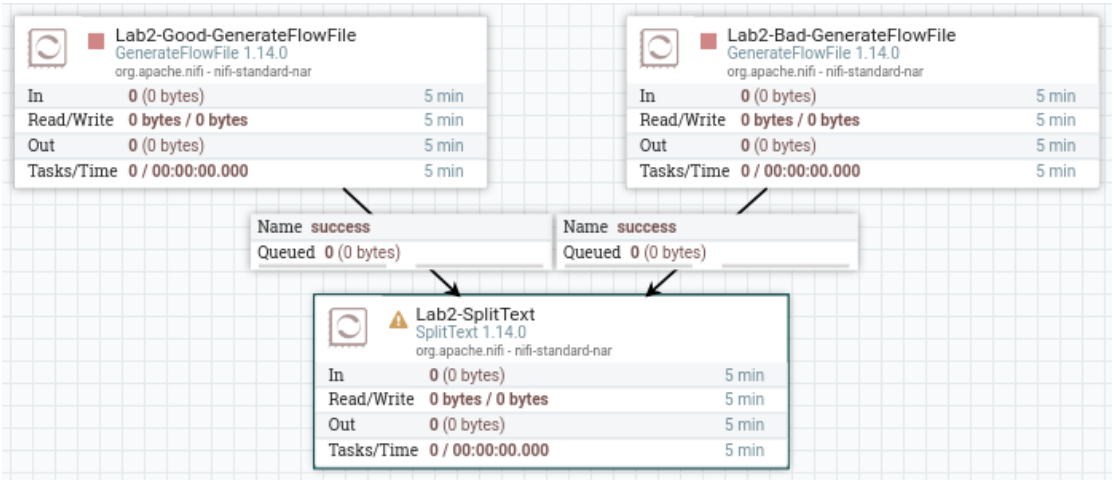
2.4.2. Now, click and drag the connection icon to the middle of the SplitText processor and let go of the mouse



2.4.3. A pop-up window to create the connection will come up. Make sure the *success* Relationship is selected for the connection



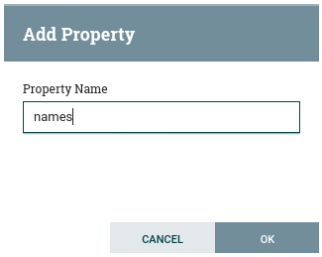
2.5. Connect Lab2-Bad-GenerateFlowFile the same way. Your connection so far should look similar to below.



2.6. Add an ExtractText processor to the canvas

2.6.1. Name the processor, Lab2-ExtractText

2.6.2. From the Properties tab, we are going to add our own custom attribute. We do so by clicking on the plus (+) icon on the top right. A pop up window to name the property will appear. Name the attribute *names*

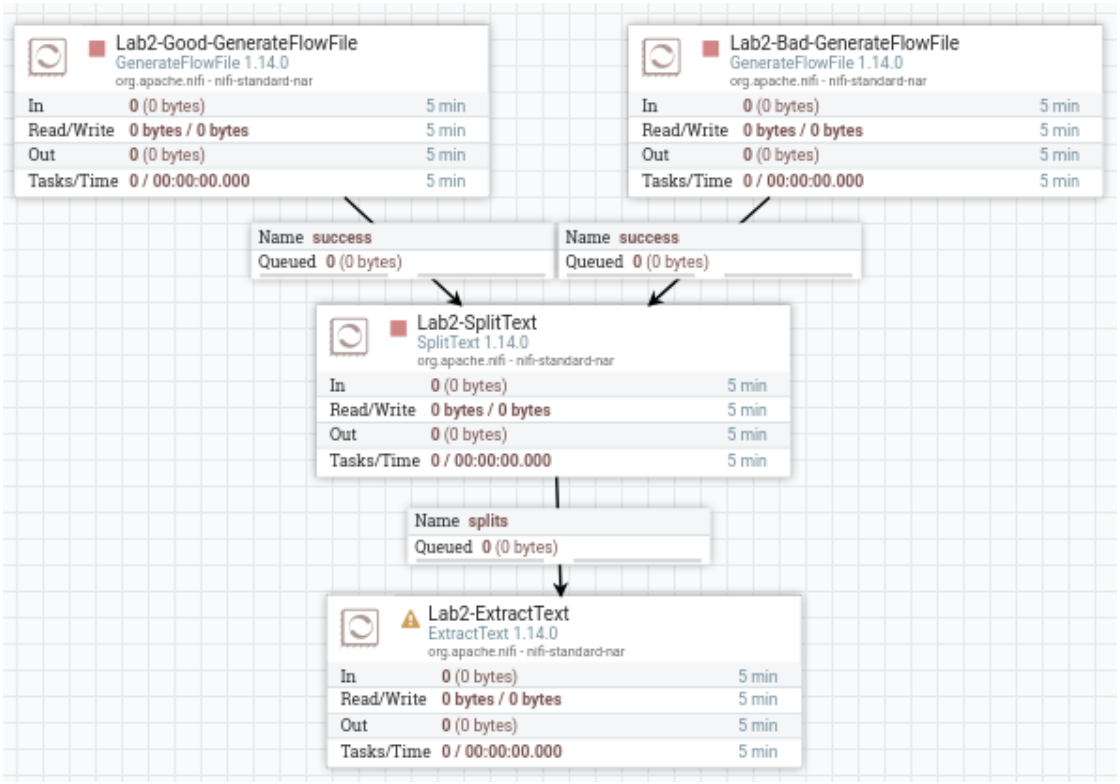


2.6.3. Enter the following as the property:

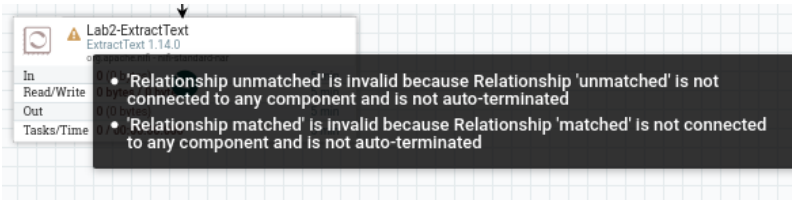
```
(.*?):
```

2.6.4. This is a regular expression that matches zero or more of any character non-greedily until a colon (:) is encountered. Learning regular expressions is not within scope of this lab, but you can easily find resources to learn its syntax. The parenthesis captures a block. The value captured within this block will be assigned to the attribute *names*.

2.7. Connect the Lab2-SplitText processor to the Lab2-Extract processor on the Relationship *splits*.



2.8. Notice the yellow exclamation mark on the Lab2-ExtractText processor. Hover the mouse over the yellow icon to review the error message. What does the error message say? How can you fix it?

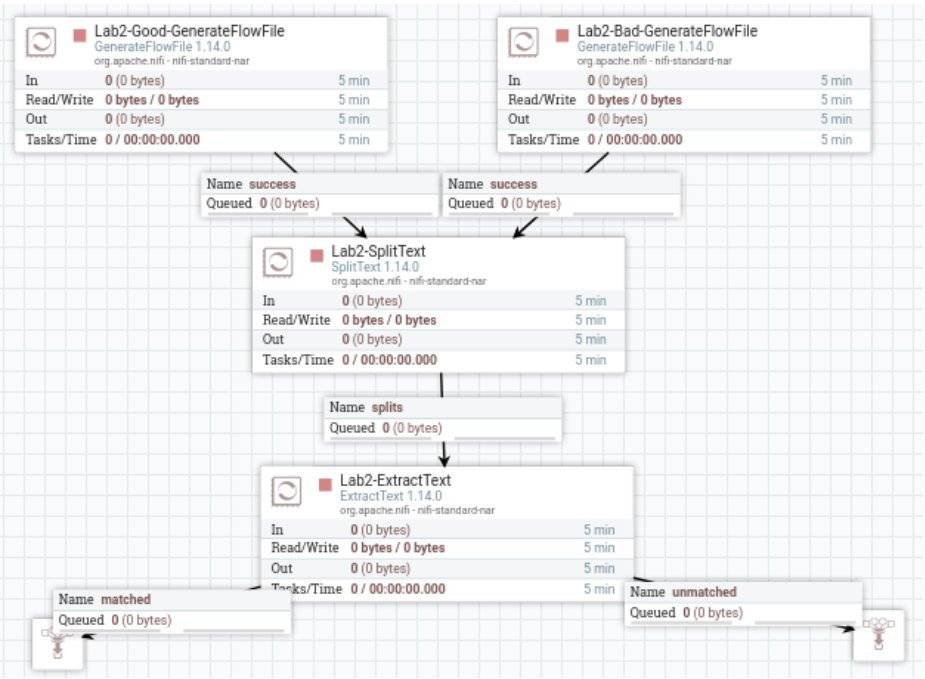


2.9. The problem is that there are two relationships generated from Lab2-ExtractText that is not terminated. We have to either automatically terminate them or pass the flow files for each relationship further downstream.

2.10. Create a temporary downstream destination using a Funnel.

2.10.1. Add two Funnels below the Lab2-ExtractText processor by clicking and dragging the Funnel icon to the canvas.

2.10.2. Connect the *matched* relationship to one of the funnels and the *unmatched* relationship to the other. A funnel allows to take connections from multiple sources and forward it to a downstream processor. However, many NiFi developers use the funnel as a temporary destination for data flows still in the works. We will use the funnel in this context. When you are done, your data flow pipeline should look similar to below.



Lab 5: Navigating Data Flows

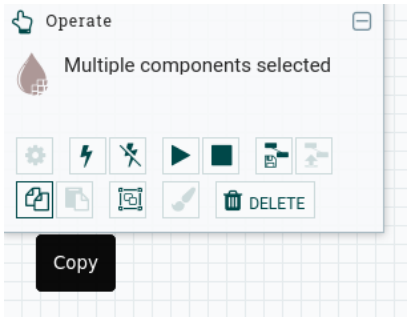
In this lab, we will run the processors that we created in Lab 2 and observe the flow files as they are generated. Finally, we will take part of the data flow and create a template for reuse in other labs.

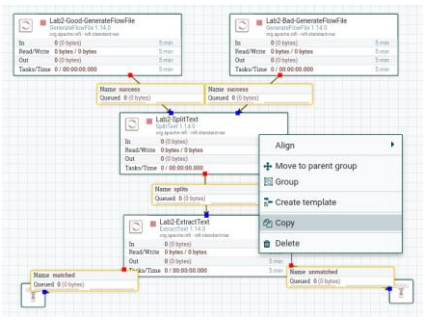
1. Copy data flows between Process Groups
   1.1. Create a new Process Group on the root canvas and name it Lab 3

   1.2. Go the data flow previously created in Lab 2 and select all components, including processors, connections and funnels. Do you recall how to select multiple components?

You can either hold down the shift key and select each of the components or hold down the shift key and click and drag the mouse around all the components that you are trying to select.

   1.3. Copy all the selected components using either the context menu or copy icon from the Operate palette. You can also use the Ctrl-C keyboard shortcut.

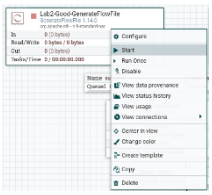




   1.4. Navigate back to Lab 3 processor group and paste the components. You can use the paste icon from the operate palette or right click anywhere on the canvas and select paste. Finally you can also use Ctrl-V keyboard shortcut as well.

2. Start Processors
   2.1. Start the Lab2-Good-GnerateFlowFile processor

      2.1.1. Before any processor can be started, they must be in a stopped state. This is the red square button shown on each of the processors.

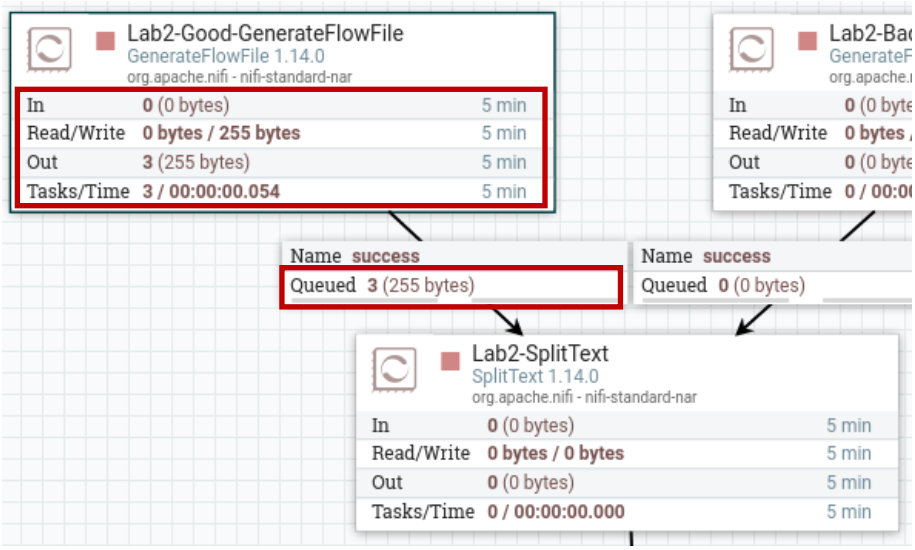


2.1.2. Select just the Lab2-Good-GenerateFlowFile and start it. You can use the start ( ) icon from the Operate Palette. You can also right click on the processor and use the context menu.

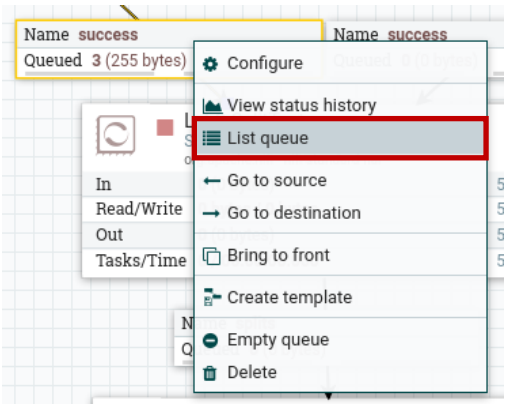
2.2. Stop and observe the generated flowfiles.

2.2.1. Let the Lab2-Good-GenerateFlowFile run for about 30 seconds or so. Recall that we have set this processor to trigger every 10 seconds so you should get 3 or 4 flowfiles generated during that time. Stop the processor using the stop icon ( ) from the Operate palette or using the context menu.

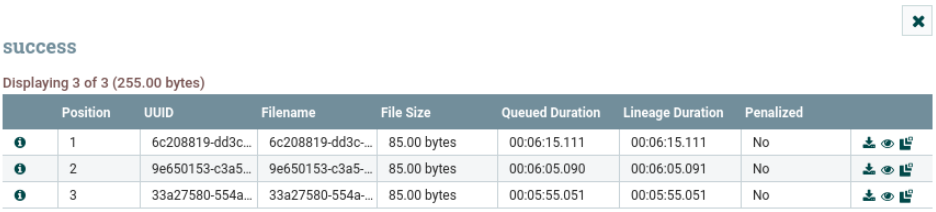
2.2.2. Your connection between Lab2-Good-GenerateFlowFile and Lab2-SplitText should now have some flowfiles queued similar to below. The Lab2-Good-GenerateFlowFile will also have updated and now shows activity.



2.2.3. Right click on the connection where the flowfiles are queued. A context menu will appear. Select the List queue option.



2.2.4. A new pop-up window displaying the flowfiles will appear.



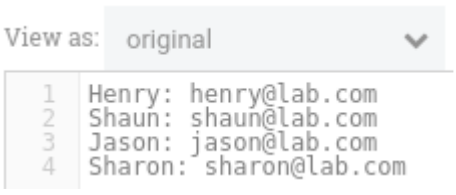
2.2.5. You can get detailed information about flowfiles including information on attributes. Click on any of the ( ) icons on the far left to display the pop-up window shown below.

2.2.6. Every flowfile has a unique UUID associated with it. This allows NiFi to track every flowfile and records its lineage. A linage is information tracking how and when a flowfile was created, modified, copied, cloned, deleted, etc. It tracks each change in time order. This is the provenance feature of NiFi. We also see flowfiles are actual physical files with a Filename, that are saved durably for some configured period of time. This portion of the flowfile does not contain the actual content of the flowfile as that would be very inefficient. The flowfile itself keeps a pointer to another contents file that contains th al contents. We ca r download the content using the download ( ) or view icon ( ) from the details tab.

2.2.7. From the Attributes tab, we can view all attributes associated with this flowfile. Every flowfile will have a few default attributes such as filename, uuid, path, etc. In addition, if the user has created any custom attributes, it will be displayed here.

2.2.8. View the contents of several flowfiles. What do you see?

You should see the text you entered in custom text field from the Lab2-Good-GenerateFlowFile processor. This will be:



2.3. Start only the Lab2-Bad-GenerateFlowFile

processor for about 30 seconds and then stop. View the queued flowfiles from this Relationship. What are the contents of the flowfiles?

You should also see the custom text entered. In our case, we entered "some text, more random text, last text" all on separate lines.

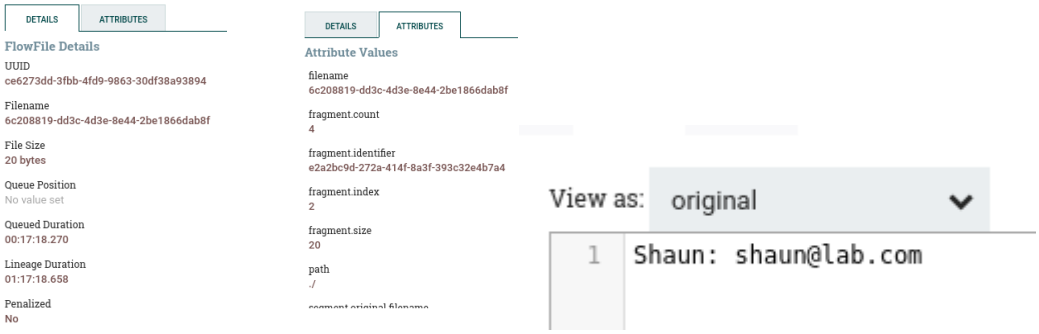
2.4. Start only the Lab2-SplitText processor and let it run until you see all the queued flowfiles from upstream clear. Each flowfile from the Lab2-Good-GenerateFlowFile will be split into a separate flowfile for each line entry. Since there are four lines, this will result in four (4) new flowfiles. Similarly, there are three (3) line entries in each flowfile from Lab2-Bad-GenerateFlowFile. This will result in three (3) new flowfiles. How many new flowfiles do you expect to see queued between the Lab2-SplitText and Lab2-ExtractText processor?

If you had 3 flowfiles on each connection, then you should now see (3x4) +(3x3) = 21 flowfiles queued up.

2.5. As before in steps 2.2.5 and 2.2.8 in Lab 3, view the detail information including the content and attribute of the queued flowfiles. What do you see?

INTENTIONALLY LEFT BLANK

You should see some new attributes automatically created by the system due to the split operation. There is now a fragment.count attribute that corresponds to the number of new flowfiles created from the original flowfile upstream. The new fragment.index attribute corresponds to index number of the current flowfile within the fragments. The flowfile shown below is the second fragment. When we check the contents, we see that it corresponds to the second line item we entered in the custom text field for Lab2-Good-GenerateFlowFile.



2.6. Start the Lab2-ExtractText processor and let it run until all the queued flowfiles are processed and passed downstream to one of the two funnels.

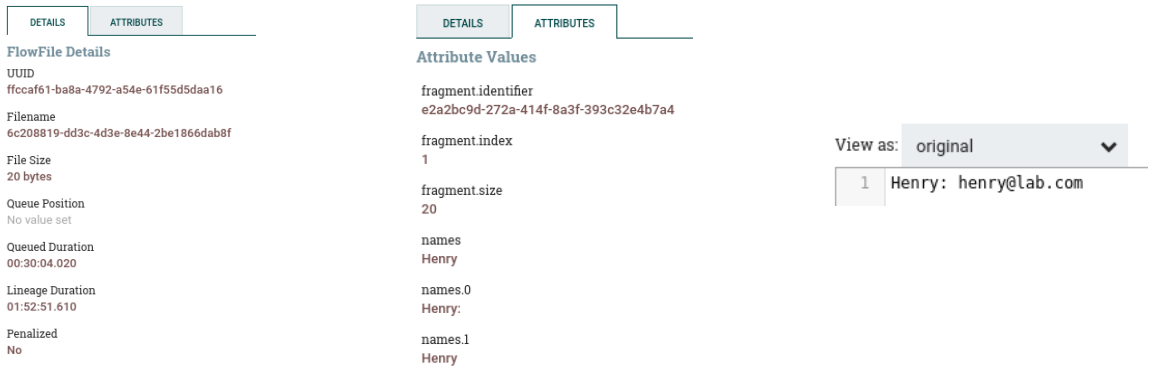
2.7. Open the queued flowfiles from the *matched* queue. What do you observe?

2.8. Similarly, view the flowfiles from the unmatched queue. What do you observe?

INTENTIONALLY LEFT BLANK

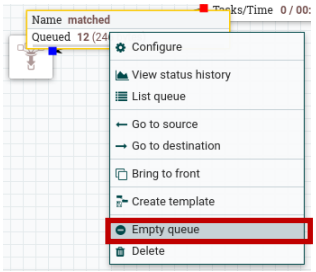
You will see that all of the flowfiles that originated from Lab2-Good-GenerateFlowFile have ended up on the matched queue while all from Lab2-Bad-GenerateFlowFile are in the unmatched queue. This is because of the regular expression we used in the names field of the Lab2-ExtractText processor properties. The regular expression matches all characters prior to seeing a colon. All the text entered in Lab2-Good-GenerateFlowFile was in the form of a key:value pair with a colon between the key and value.

Sample matched flowfile



Sample unmatched flowfile. Notice that in the unmatched flowfile, the names attribute has not been created since there was no match.

2.9. Let's clean up the queues before we finish this lab. Right click on the queue and bring up the context menu. From there select, Empty queue to delete all the flowfiles that are queue in the connection. Do this for both matched and unmatched relationships.



Lab 6: Creating and Using Templates

In this lab, we are going to learn how to create templates from existing data flows. Templates are a way to save often used data flow logics. Many NiFi developers have libraries of templates that they can reuse to create more complex data flows

1. Creating a Template
   1.1. Setup Lab 4 Processor Group
      1.1.1. Add a new processor group to the root canvas and name it Lab 4
      1.1.2. Copy the entire dataflow you created in Lab 3 and paste inside the Lab 4 processor group
      1.1.3. Make sure there are no running processors and all the queues for all the relationships are empty
   1.2. Remove processors not part of the template
      1.2.1. Remove the Lab2-ExtractText processor - Try deleting the Lab2-ExtractText processor. What happens?

NiFI will complain that the processor is the destination of another component. NiFI cannot delete that processor yet, since that will cause the upstream processor to lose one of its destinations. In order to delete a processor, we have to remove any connections entering it. This is not the case for any connections emanating from the processor.

      1.2.2. Remove the connection from the Lab2-SplitText to Lab2-ExtractText processor.
      1.2.3. Now we can delete Lab2-ExtractText processor.
      1.2.4. Delete the two funnels.

Since we are going to save the dataflow as a template, we don't want the name of the processors to be too specific. In our case, all of the processors have a Lab2 prefix.

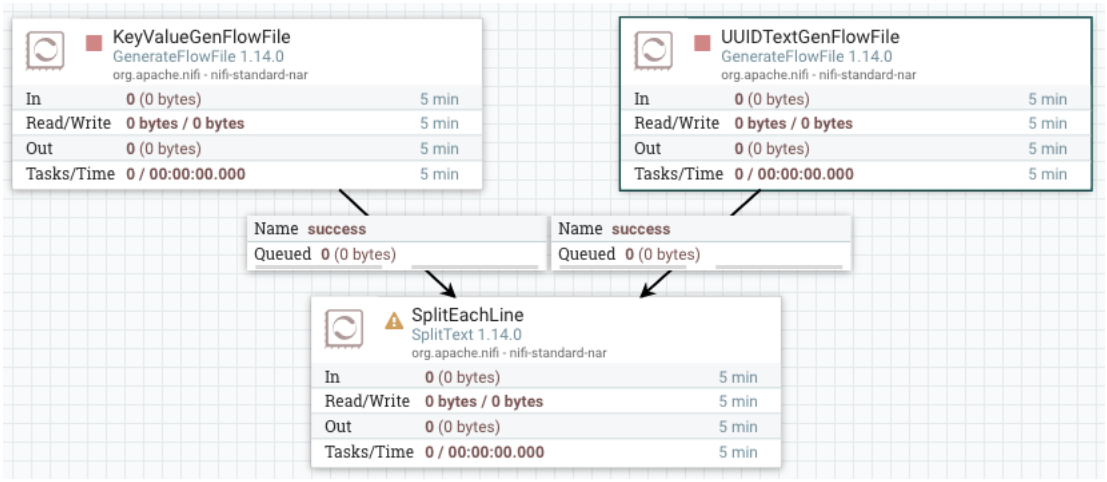
   1.3. Rename the processors to a more generic name.
      1.3.1. Rename Lab2-Good-GenerateFlowFIle to KeyValueGenFlowFile
      1.3.2. Rename Lab2-Bad-GenerateFlowFile to UUIDTextGenFlowFile
      1.3.3. Rename Lab2-SplitText to SplitEachLine
   1.4. Modify the UUIDTextGenFlowFile

In the previous lab, this processor simply generated 3 random lines of text. We will enhance this processor slightly so that each generates random text that are UUID unique.

1.4.1. Delete the previous Custom Text and change to:

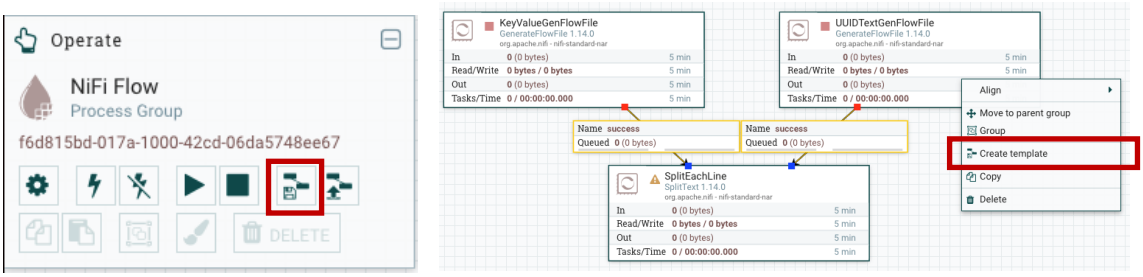
```
${UUID()}
```

1.5. Save as a Template: We are now ready to save the three processors as a template. This small little dataflow can simulate a streaming data source where there is actual key:value data mixed in with random noise. The flow that we will save should look similar to below:

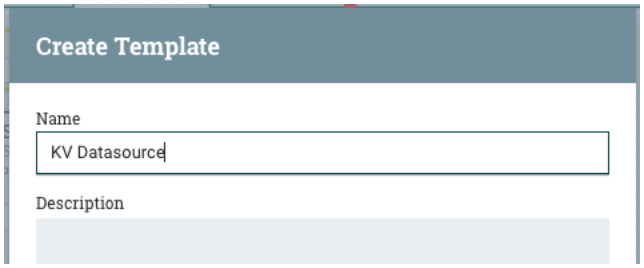


1.5.1. Select all components including the three processors and two connections.

1.5.2. Select the Create Template icon from the Operate Palette or right click on any of the selected components (they have to be already selected) to bring up the context menu. Select Create Template.



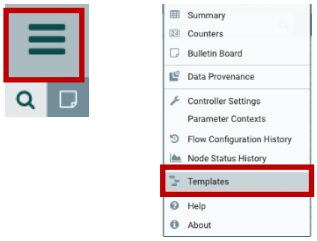
1.5.3. The Create Template pop-up window will open. Name the template KV Datasource and select **Create**


Create Template
Name
KV Datasource
Description

2. Create new dataflow from Templates
   2.1. Create a new dataflow by using the template that we just saved. Select the Template icon from the components toolbar and click and drag to the canvas

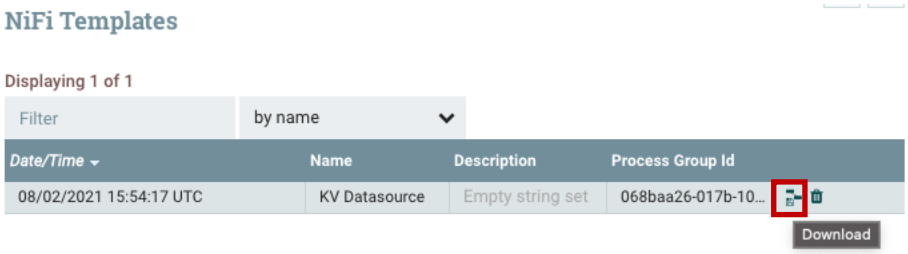
   2.2. From the pop-up menu, select the KV Datasource template that you just saved. A copy of the dataflow will be placed on the canvas. Delete the second copy for now.

3. Exporting Templates
   3.1. We can export templates as XML files to be transported to other NiFi clusters or to simply save permanently as a physical file. Open the Global Menu on the top right of the screen.

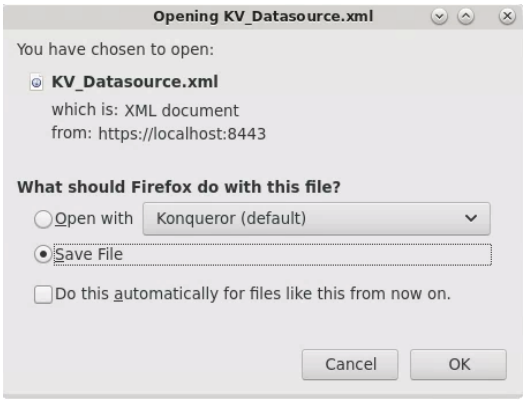


   3.2. Select Templates from the menu.

   3.3. A new pop-up window with all the saved templates in the current NiFi cluster will be displayed. On the far right, there is the download and trashcan icon. Use the trashcan icon to delete any saved templates. Use the download icon to export the selected template as a XML file.

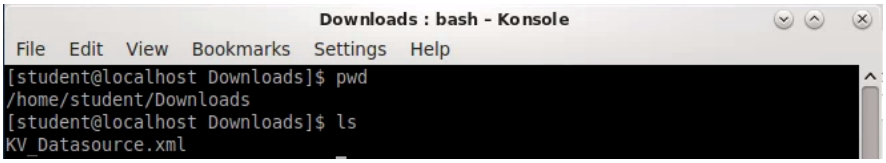


   3.4. Your Firefox browser will prompt you do either save the file or open it. Choose save file.

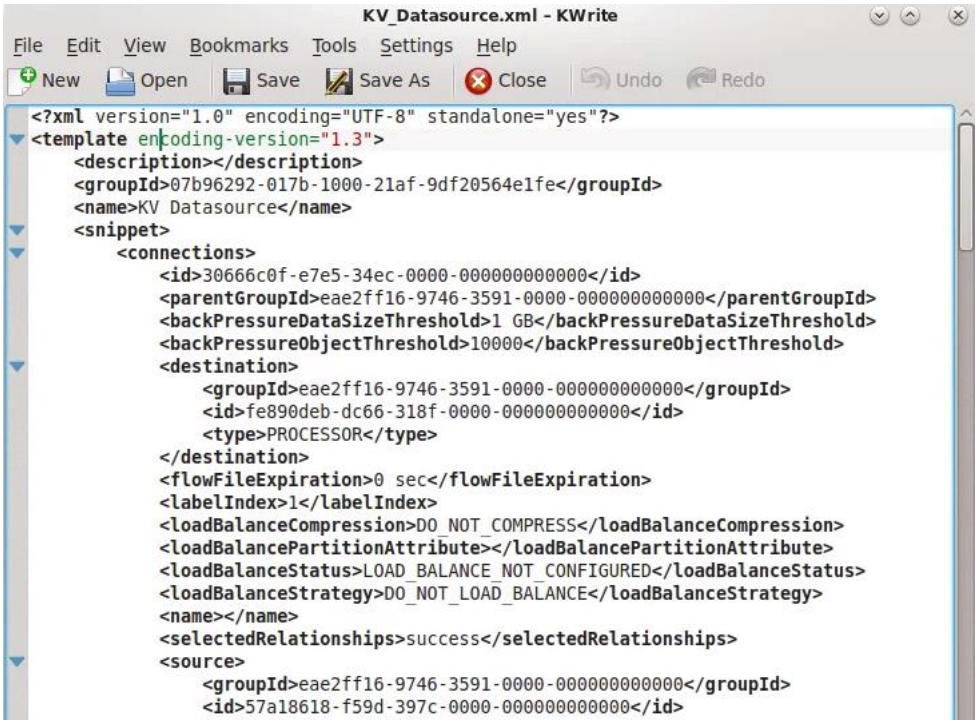


3.5. Open a terminal and navigate to the Downloads directory. List the directory and you will see the saved XML file. You may copy this file and use it on another NiFi cluster.

```
cd ~/Downloads
ls
```



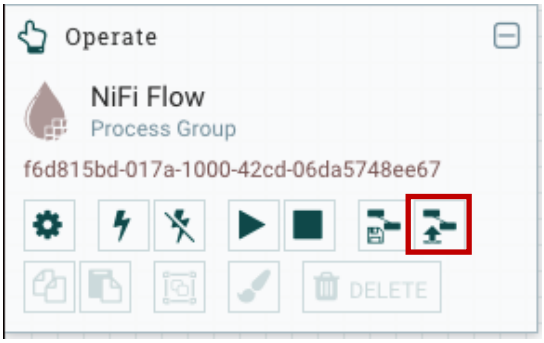
3.6. Open the KV_Datasource.xml file using an editor of your choice. In this example, we will use the KWrite utility available on the bottom right in the panel. Review the XML file. Do not make any changes, however.

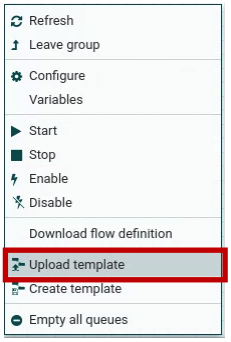


3.7. Delete the template from the NiFI cluster.
Delete the saved template from the current cluster so that we can import it in the next step. If a template with the same name already exists in the cluster, you will not be able to import it. Refer to the instructions in 3.3.

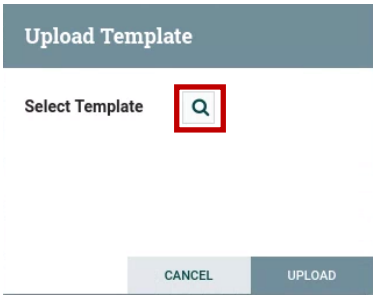
4. Importing Templates

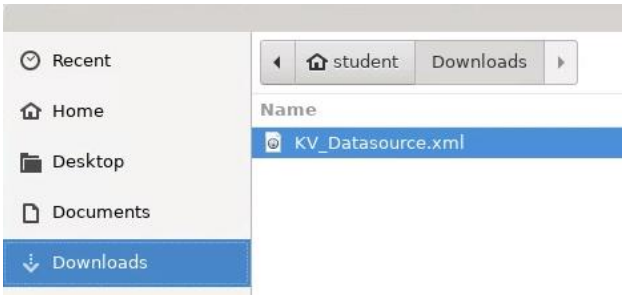
4.1. Click on the Upload Template icon from the Operate Palette or right click on an empty area of the canvas. Make sure that nothing is selected. From the context menu, select Upload template.



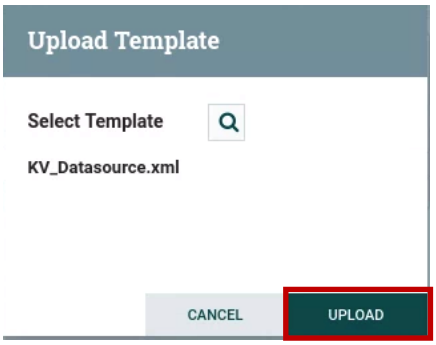


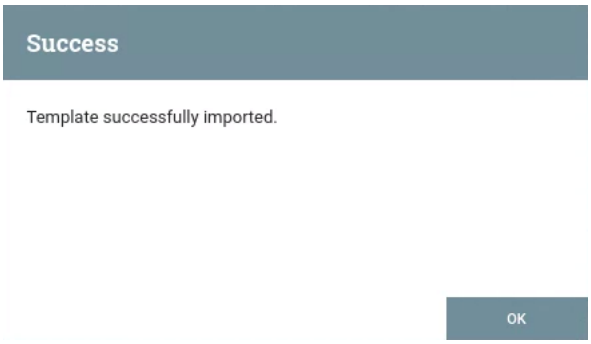
4.2. Select the magnifier icon to browse through the files in your directory. Select the Download directory on the left tab and from there, select he KV_Datasource.xml file that we just saved in the previous step





4.3. Once the XML file has been selected, click on the upload button to complete the operation. A success pop-up window will confirm the success of the operation.





4.4. Return to the Global menu to confirm that the template has been imported and is once again, part of the NiFi cluster.

4.5. Confirm the template by repeating the operations from step **2.1** to make a copy of the KV_Datasource dataflow on the canvas.

4.6. Clean up the canvas by removing any extra dataflows

5. Making a back-up of the entire canvas

It is often useful to make back-up copies of your entire canvas with all the dataflows in it.

5.1. Select all the components in the entire canvas

5.1.1. Navigate to the root canvas using the breadcrumb menu on the bottom left screen

5.1.2. Select all the components using Ctrl-A

5.1.3. Create a template from all the selections as you did in step **1.5.**

Lab 7: Using Processor Groups

In this lab, we will combine using templates and processor groups to simplify the development of dataflows. So far, we have only been using processor groups to organize projects into different directories. Now we will use then to connect smaller logical dataflows into larger ones.

1. Creating a complex dataflow
   1.1. First, set up a new processor group for Lab 5 as we have been doing so far.

   We will create a dataflow where we will take a datasource, merge the contents, compress it and save it to local disk.

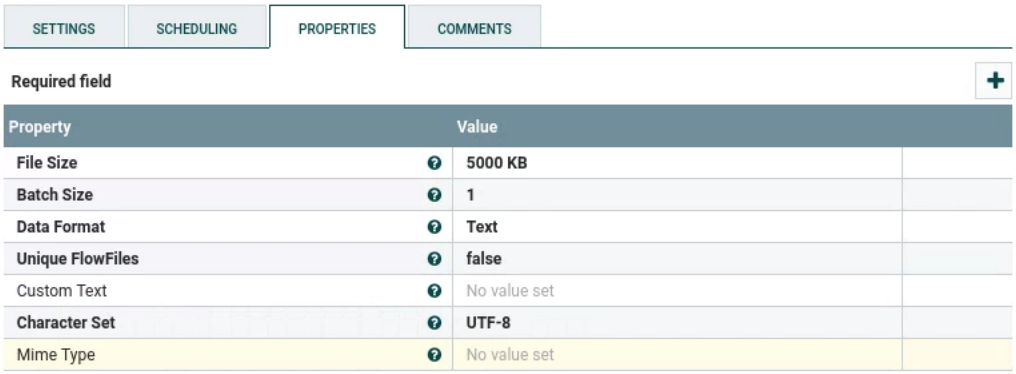
   1.2. Create data source

   1.2.1. Add a GenerateFlowFile to the Lab 5 canvas

   1.2.2. Name the processor Lab5-GenerateFlowFile

   1.2.3. Schedule it to run every 2 seconds

   1.2.4. Change the File Size property to 5000 KB



   1.3. Create Merge-Compress-Save dataflow

   1.3.1. Add a MergeContent processor

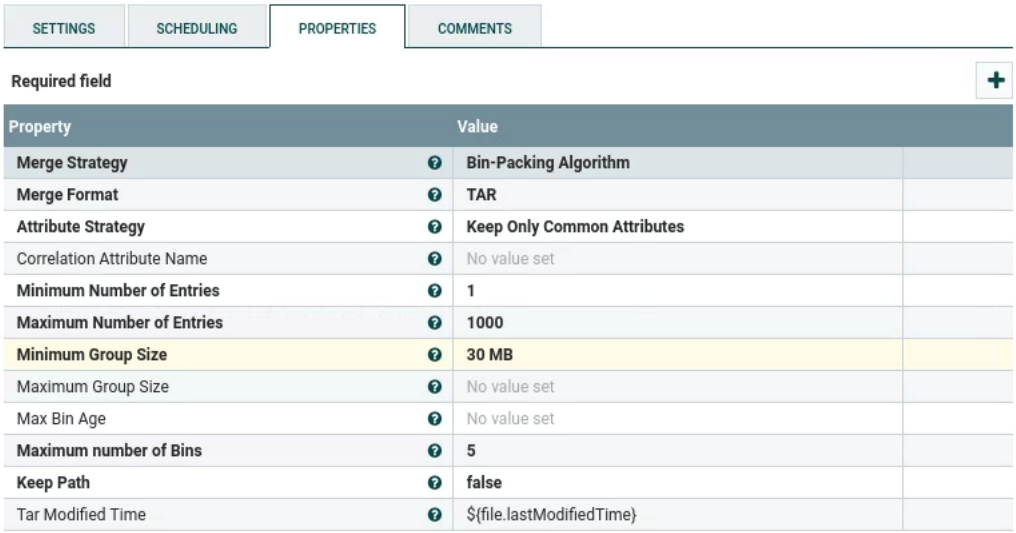
   1.3.2. From settings, automatically terminate the failure and original relationship.

   1.3.3. From properties, change the Merge Format to TAR

   1.3.4. Change the Minimum Group Size to 30 MB

   1.3.5. Make sure you click on the (?) for each of the properties that we modified to understand the changes we have made.

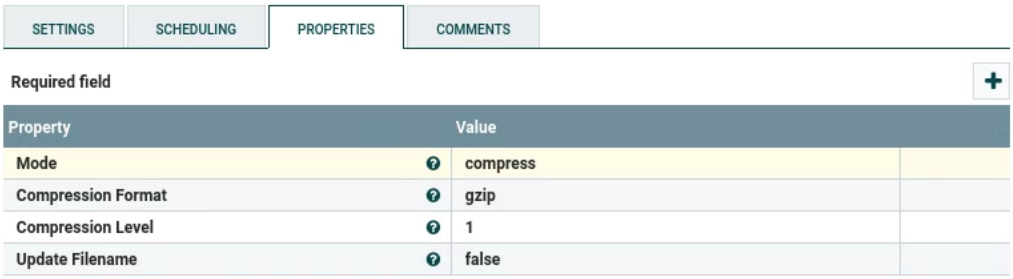
In essence, we are going to wait until the minimum size of the merged content is 30 MB and then package it in tar format.



1.3.6. Add a CompressContent processor

1.3.7. From properties, make sure the Mode is compress

1.3.8. Change the Compression Format to gzip



1.3.9. Add a PutFile processor

1.3.10. From settings, automatically terminate the success relationship. After the file has been saved, we do not need to keep the flowfiles.

1.3.11. From properties, change the Directory to /tmp/nifi/lab5

1.3.12. Change the Conflict Resolution Strategy to replace

1.3.13. Make sure the Create Missing Directories is true

1.3.14. Set the Owner to student

1.3.15. Set the Group to student

Property	Value
Directory	**/tmp/nifi/lab5**
Conflict Resolution Strategy	**replace**
Create Missing Directories	**true**
Maximum File Count	No value set
Last Modified Time	No value set
Permissions	No value set
Owner	No value set
Group	No value set

1.4. Connect the processors

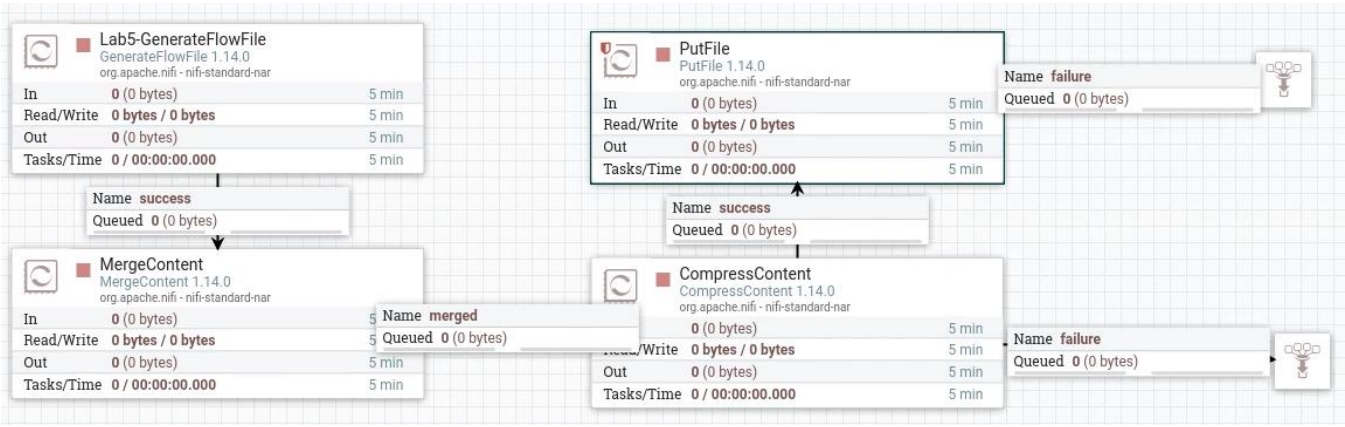
1.4.1. Connect Lab5-GenerateFlowFile to MergeContent for the success relationship.

1.4.2. Connect the MergeContent processor to the CompressContent processor for the merged relationship.

1.4.3. Connect the CompressContent processor to the PutFile processor on the success relationship.

1.4.4. Add a funnel and connect the CompressContent failure relationship to it.

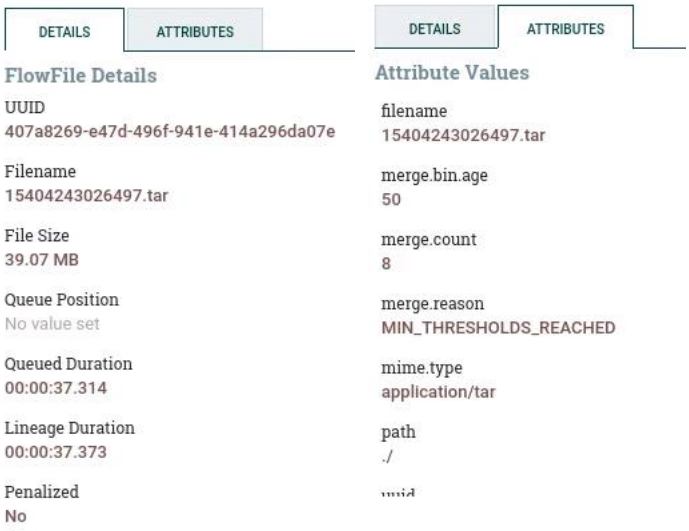
1.4.5. Add a funnel and connect the PutFile failure relationship to it. When you are done, you flow should look similar to below.



1.5. Run the dataflow, processor by processor and observe the queue

1.5.1. Run the Lab5-GenerateFlowFile for about 30 seconds until there are flowfiles queued downstream. Stop the processor and then open the queue and inspect the flowfiles. Take note of the file size.

1.5.2. Run the MergeContent for about 15 seconds until there are flowfiles queued downstream. Stop the processor and then open the queue and inspect the flowfiles. Observe the file size. From the attributes tab, look at the new attributes that have been created by this processor. What do you think those attributes mean?



Let's look at some of the more interesting attributes. The merge.count attribute shows that how many flowfiles have been combined together. If you stopped the processor at 15 seconds, you will have about 8 flow files. Your results may differ slightly and may show more or less Since we had set the Lab5-GenerateFlowFile processor to send 5MB flowfiles, we should expect that the new flowfile sent downstream is approximately 5MB x 8 = 40MB. The actual size is 39.07MB. Once again, this size will depend on the merge.count. If your count is 8, it should be

similar. Your results may differ slightly and may show more or less The merge.reason states that the minimum threshold was reached. Go back to the CompressContent properties tab and confirm that we have set the Minimum Group Size to 30 MB. There is also a maximum threshold that can be set. Another way to trigger the processor to bundle and send forward is the bin age. If a certain amount of time as passed (Max Bin Age), the processor will pack all the flowfiles collected so far and send downstream.

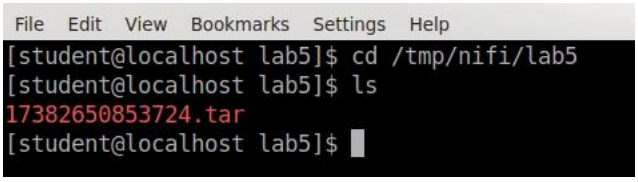
1.5.3. Run the CompressContent until there is/are flowfile(s) queued downstream. Stop the processor. First take a look at the 5 min statistics. It shows 1 in. A Read/Write of 39.07MB/32.28MB and 1 out. This indicates that this processor processed 1 flowfile coming in and sent out 1 flowfile. The amount of Read/Write I/O to complete this is as shown. Since, the output file was 32.28MB, we should expect flowfile in the success relationship connection queue will be of that size and that the gzip compressor was able to reduce the size slightly.

1.5.4. Open the queue and inspect the flowfile. Notice the size of the file matches the output shown from the 5 min statistics above. Also, from the attributes tab, the mime.type has changed from tar to zip because be have compressed using gzip.

1.5.5. Run the PutFile processor until the queue is cleared. Make sure that there was no failure by ensuring that the failure queue to the funnel remains empty. If there is a failure, go back to step 1.3.9 and check the setting.

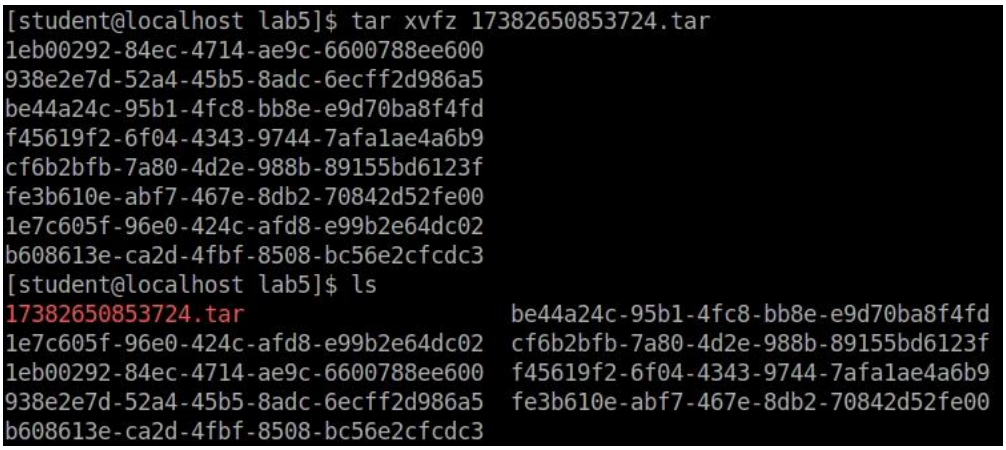
1.5.6. Open up a new terminal and navigate to the /tmp/nifi/lab5 directory. Get a listing of the directory

```
cd /tmp/nifi/lab5
ls
```



1.5.7. Untar the file and list the directory again. You should see the same number of files as merge.count from above step 1.5.2. In our current case, that was 8.

```
tar xvfz <filename>
ls
```



1.6. Use more (less) to inspect the contents of any one of the files.

```
more fe3b610e-abf7-467e-8db2-70842d52fe00
```

```
[student@localhost lab5]$ more fe3b610e-abf7-467e-8db2-70842d52fe00
!x7^!kyf1       3'B=VkBk"x_*gKUq;SbQ.-=a+N6v2+h'<IfL*_'2y!k?%#g8E:PoV9L&xqy     #VK%z4jz-(P7@o:GR&ic4K  8+7i1HwF
hiL$@wZ
Sa)UY=5mwPkRuE'!SqS?Go+2Yk#z<!Ubv/v'!Z'TK6t*zVbE:l5%Ix(.E4_gIq6 N*&r:u)GESvj6?;kNAFh<9HJe^Bqv5Bh#j%TeOJ-PE:?tV4"
2S.&(0HU'/DbE(k>g2o
WRT>m-p0G:fPr (L*%_K;No^LBDzKO
>)
X:"pm;K%Xuh!XEcz<C)?X:@;W       t8I:ldtfet&gNB:z7S      m?l>.njjq 1154!Am=d)wyz
5tZ3Q0dn,b^nxo  Fup^6Q)h%S=c3'EOO&p;#e(E%hryQY:#Gq@kW-W VIXQ6Y_>_We,v4%:&N$?T#M8H&&nBK?A"08ZIcbYJ7ef^0nKHw;,y/k9
i9k"=a
O_*NlFH%_BzN&,8:WP%0RDvK<YoI    4)3v6v"70=jSmvK(';j" ?W4>F?Wf,K.!/9?Q=$?+s<Z^E>h(EzYgW+a,
#n:LIb*4FcW<w-X#&8":JQHENjH?=d?"nj8-^==C:U&8csRwg^$l<F  1L
+?9ULFt@)yd4f   GH7GX+PLt<j$1qStop: Llff@?LUT?ozRVeBhN(1LE*Ohgj0=^_? 3oQx9&wi1n5jq0gShSYW       %t@z*L@Gk5BqijE/
W@oB>&rW601B!cB(ue-bMwC<u
4d-0G_4?8LvHiF9#tzCuy@Dvc&?jg/#(q+76JD>e8R@u=$d3WHDB=5uQkR6n!q_7Uj_s@;KC;P:uUgl"'m^g.wFRwS:!gs J?fXFCYG9&Lj(-G1$
        oyw=82r?0pr!InQkv113>Z
9'w<9K: 4^15@0pMhDlo8nA+Mr7(w&8h?cO%Y 7oSg_t&l:QQ%xVau;4JGP?+6k9 JBAM+Q&<PJH0bo^_v3=EGh5,tNpy@4/nF*bQx=trv?
```

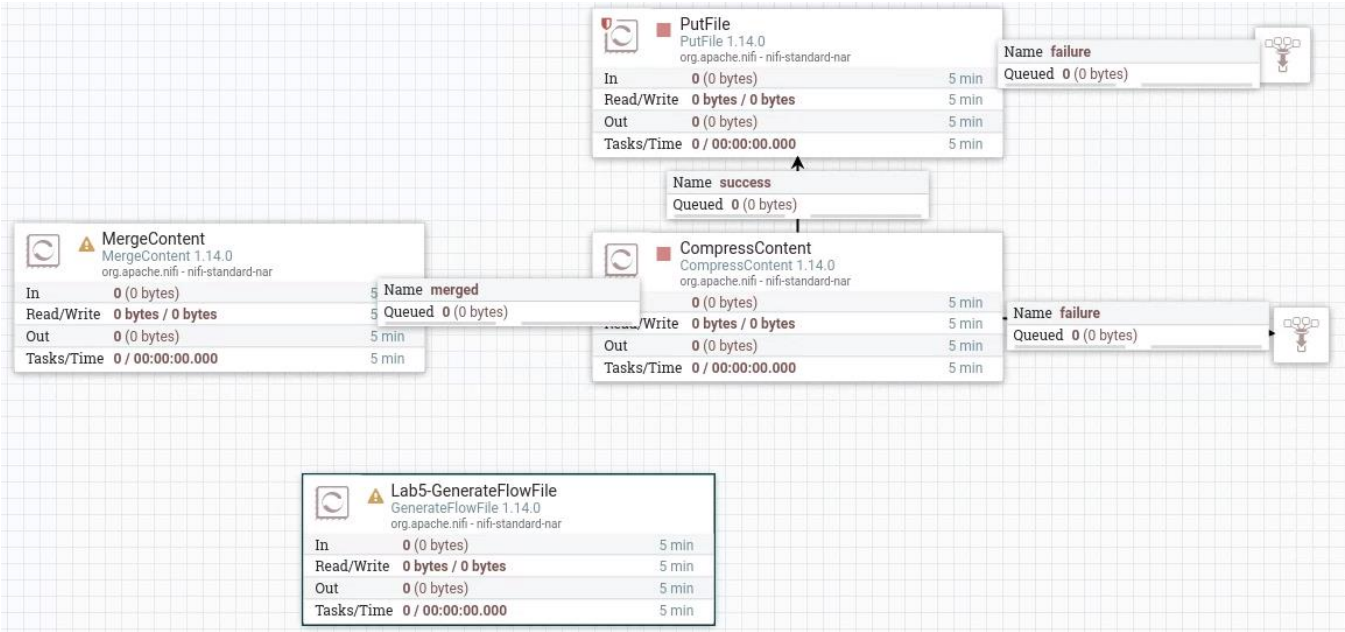
2. Create data flows a Processor Groups

The Merge-Compress-Save portion of our data flow seems like something we can often use. We will now create a process group from it and use it as part of a dataflow.

2.1. Make sure that all the processors are stopped and all the queues are empty

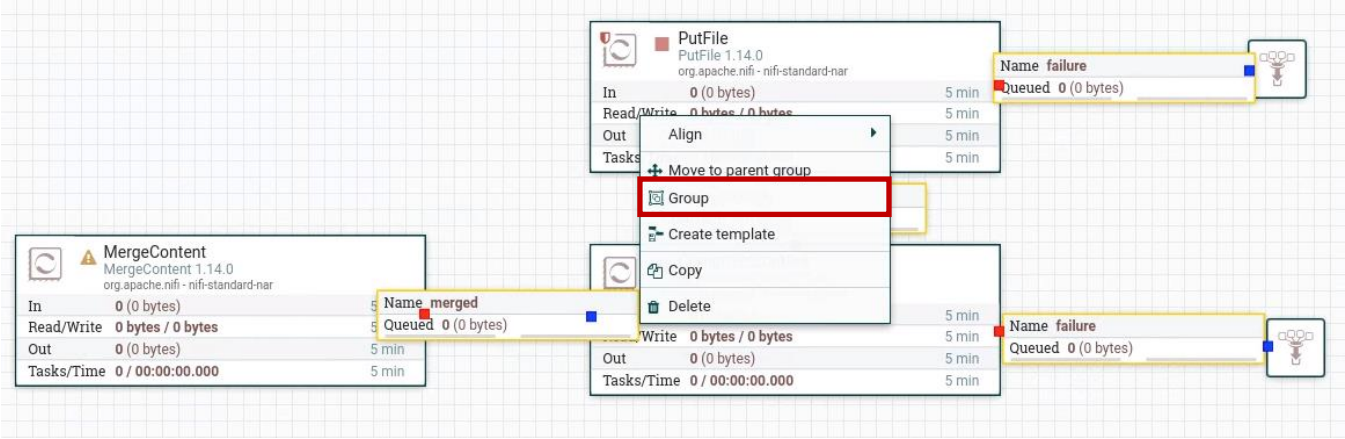
2.2. Remove the relationship between Lab5-GenerateFlowFile processor and MergeContent processor.

2.3. Move the Lab5-GenerateFlowFile out of the way so that we can easily select the rest of the processors and connections.

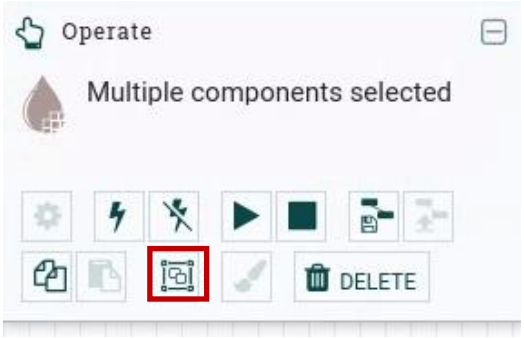


2.4. Create a Processor Group from the rest of the processors

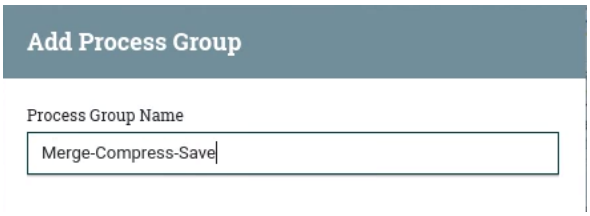
2.4.1. Select all of the components except the Lab5-GenerateFlowFile and right click inside any of the selected components to get the context menu. Choose The Group option. .



Alternatively select all the desired components and click the Group icon from the Operate Palette.



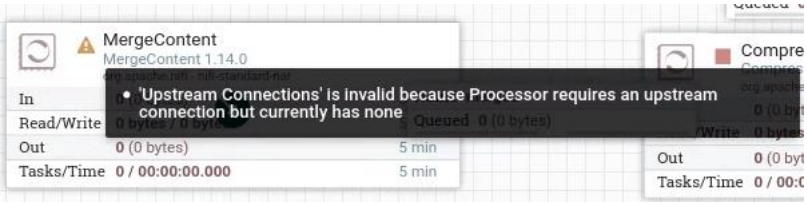
2.4.2. Enter Merge-Compress-Save for the Processor Group name.



2.4.3. Click on the Add to create the processor group. All the selected components will now be inserted into the newly created processor group.

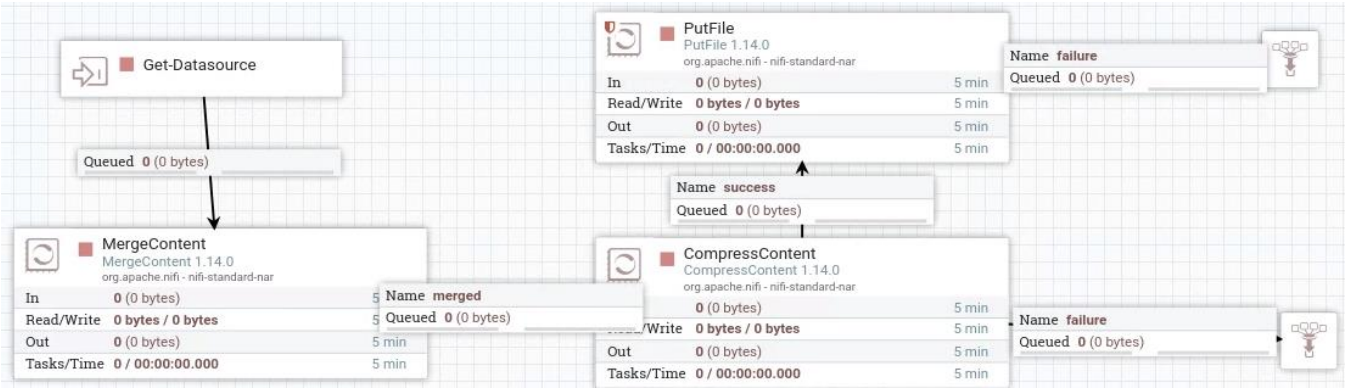
2.4.4. Double click on the processor group to enter it.

2.4.5. Notice that there is now a yellow warning on the MergeContent processor. Hover the mouse over it to read the warning message.



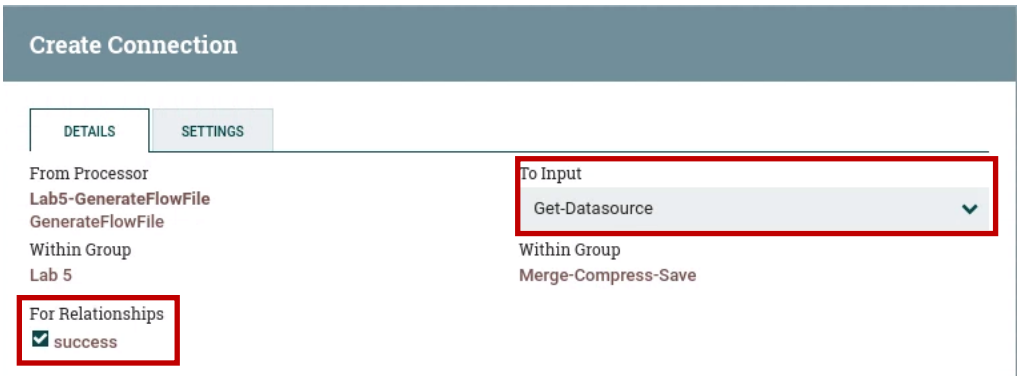
2.4.6. The MergeContent requires an upstream processor but currently does not have one. It used to have the Lab5-GenerateFlowfile processor connected upstream but we have disconnected it. Add an Input Port to the processor group by selecting it from the components toolbar. Name the port, Get-Datasource

2.4.7. Connect the Get-Datasource input port to the Merge-Content processor. Your dataflow should look similar to below.

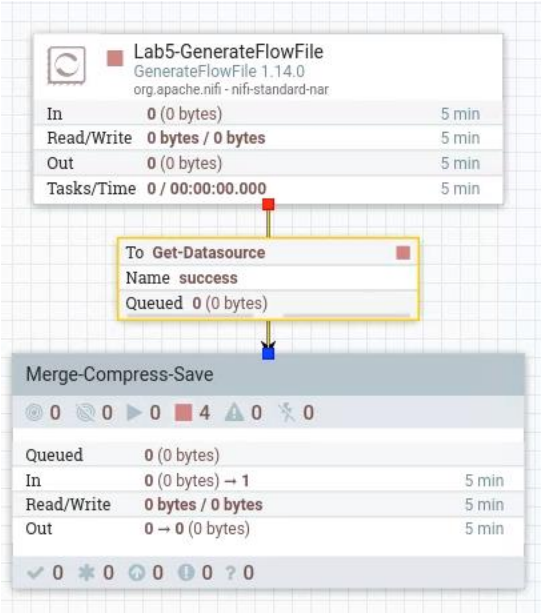


2.4.8. Use the breadcrumb menu on the bottom left to move out of the Merge-Compress-Save processor group and up to the Lab 5 processor group.

2.4.9. Connect the Lab5-GenerateFlowFile processor to the Merge-Compress-Save processor group for the success relationship. Make sure that the To Input on the right side is Get-Datasource

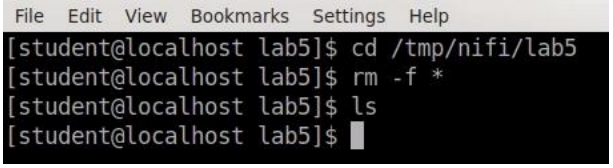


2.4.10. Add the connection. Your dataflow should now look similar to below.

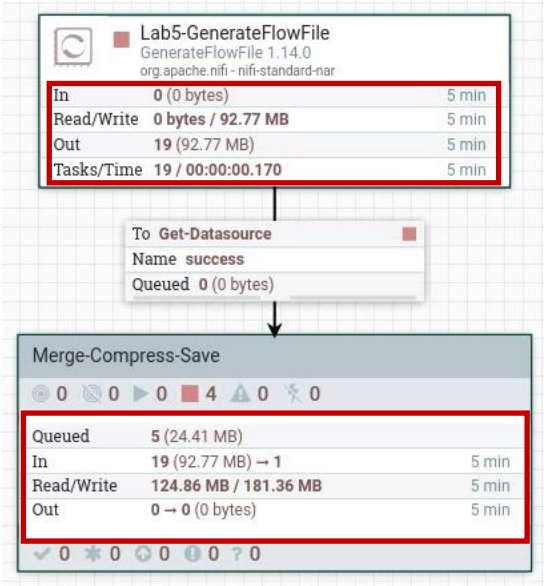


2.4.11. From a terminal, navigate back to /tmp/nifi/lab5. We will be testing the new dataflow. In order to make sure we are adding new files, remove all the files in the directory.

```
cd /tmp/nifi/lab5
rm -f *
ls
```



2.4.12. Return to the Lab 5 canvas and run all processors including the processor group. Let the dataflow run for about a minute and then stop it. You should see activity in the 5 minute statistics for both the Lab5-GenerateFlowFile processor and the Merge-Compress-Save processor group.



2.4.13. Return to the terminal to check that new files have been created.

Lab 8: Setting Back Pressure on your Connections

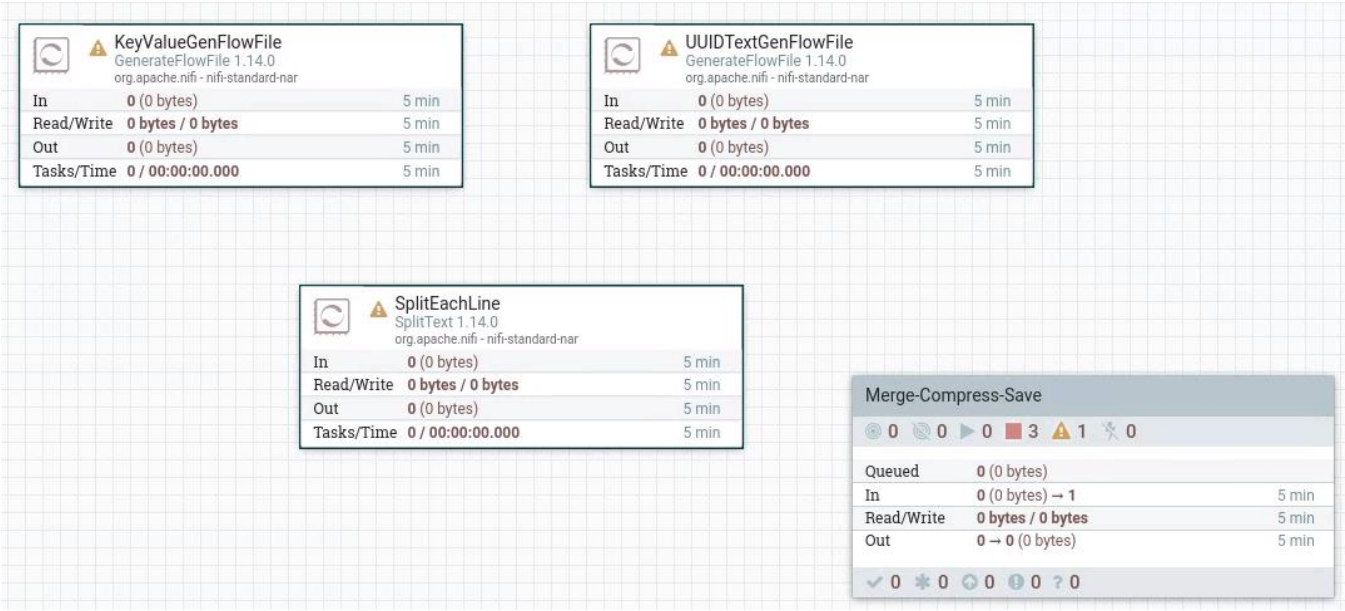
In this lab, we will explore back pressure. Sometimes a processor may experience delays causing the queue from upstream to accumulate. If the queue gets too big, we can setup and configure back pressure such that the upstream processor stops sending new dataflows downstream. This in turn can cause a recursive back pressure to activate. In other words, since the upstream processor has stopped processing flowfiles, its upstream queue, in turn, will begin to accumulate. This accumulation can trigger another back pressure policy, causing the grandparent or two legs upstream processor to stop processing as well.

1. Setting up the scenario
   1.1. As usual, create a processor group and name it Lab 6.

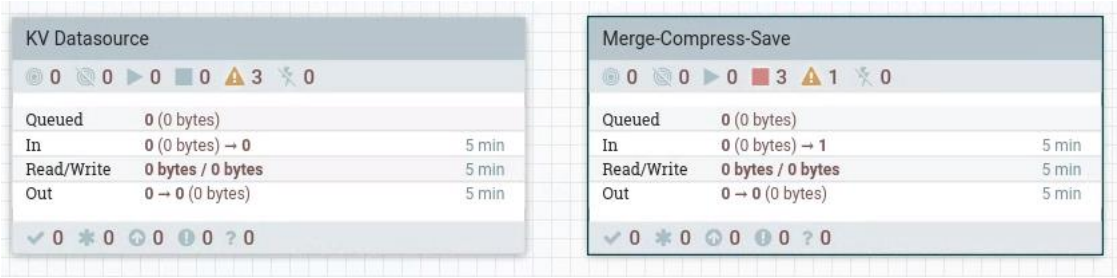
   1.2. From Lab 5, we created a very useful dataflow. The dataflow was made into a processor group named Merge-Compress-Save. Merging and compressing the content and then saving it is a scenario that is often encountered. Go back to Lab 5 and create template from that processor group. Name the template Merge-Compress-Save.

   1.3. Return to the Lab 6 canvas and create a dataflow from the Merge-Compress-Save template

   1.4. We had save another template that generated Key:Value pairs for us in Lab4. Make another dataflow from the KV Datasource template. Your canvas should look similar to below.



1.5. Processor groups makes making dataflow so much easier. Now that we know how to make processor groups, take the three processors from the KV Datasource template and create a new processor group. Name it KV Datasource. Your dataflow should look similar to below.



1.6. Modify the KV Datasource Processor Group

1.6.1. Enter the KV Datasource group

1.6.2. Change the Run Schedule for KeyValueGenFlowFile to 1 sec.

1.6.3. For this lab, we will not need the UUIDTextGenFlowFile. Delete it. Remember, we cannot delete processors until their connections have been deleted

1.6.4. Copy the Lab2-ExtractText processor from Lab2.

1.6.5. Add an Output Port and name it "From KV"

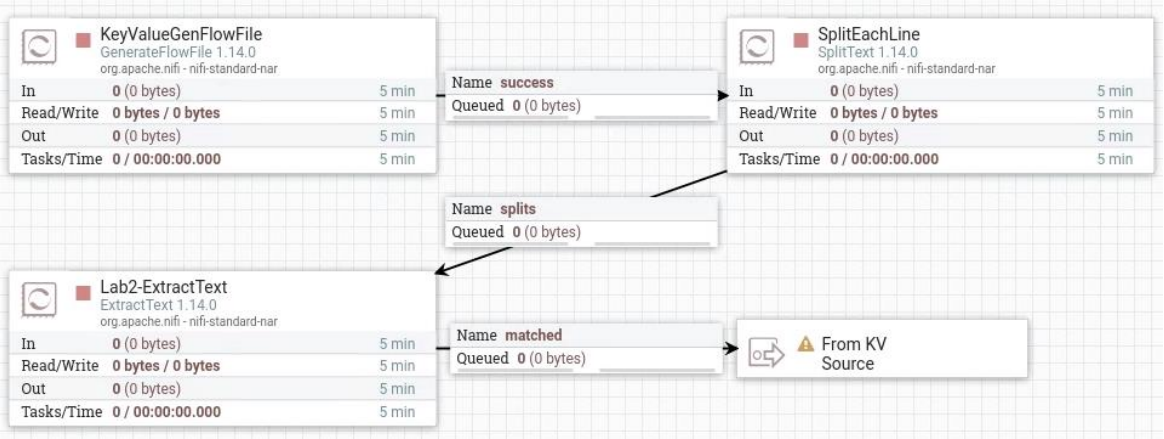
1.7. Connect the components

1.7.1. Connect the KeyValueGenFlowFile to the SplitEachLine processor for the success relationship

1.7.2. Connect the SplitEachLine processor to the Lab2-ExtractText processor for the splits relationship

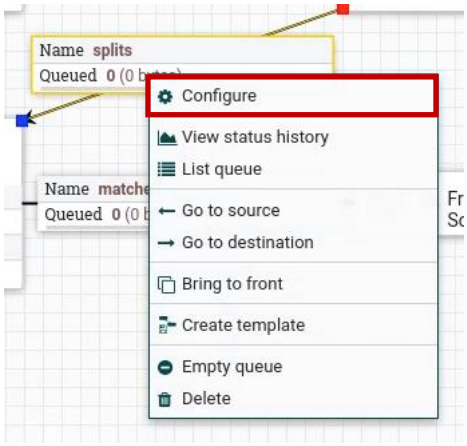
1.7.3. Connect the Lab2-ExtraxtText processor to the KV Source output port

Your dataflow should look similar to below:



1.8. Configure the backpressure settings

1.8.1. Right click on the connection between Lab2-ExtractText and SplitEachLine to bring up the context menu. Select Configure.



1.8.2. From the Setting tab, change the Back Pressure Object Threshold value to 10

1.8.3. Right click on the connection between SplitEachLine and KeyValueGenFlowFile to bring up the context menu. Select Configure.

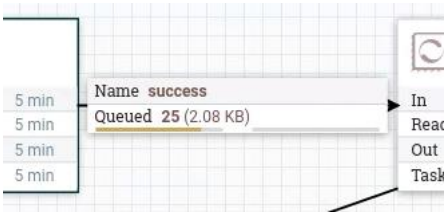
1.8.4. Set the Back Pressure Object Threshold value to 30. This is the maximum number of flowfiles that can be queued before back pressure will be triggered.

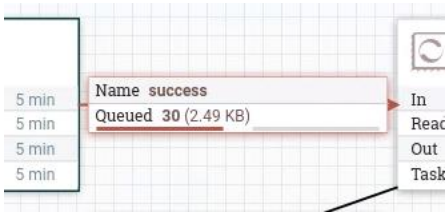
2. Testing the scenario

2.1. Test the backpressure count policy settings

2.1.1. Start the KeyValueGenFlowFile. Observe the downstream queue to the SplitEachLine processor. As more and more flowfiles queue up, the queue will indicate that the allowed buffer is getting full. The queue is color coded to show

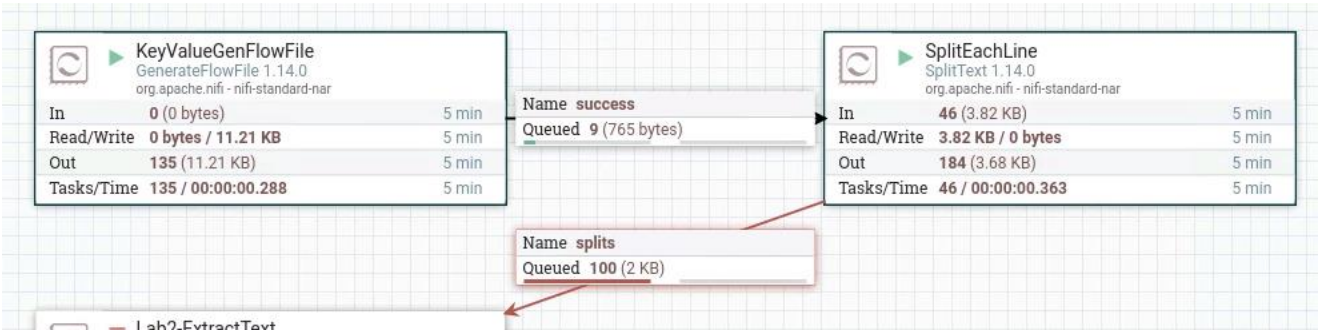
the how much of the buffer is filled before back pressure will be applied. Green is 0-60%, Yellow is61-85% and Red is 86-100%.

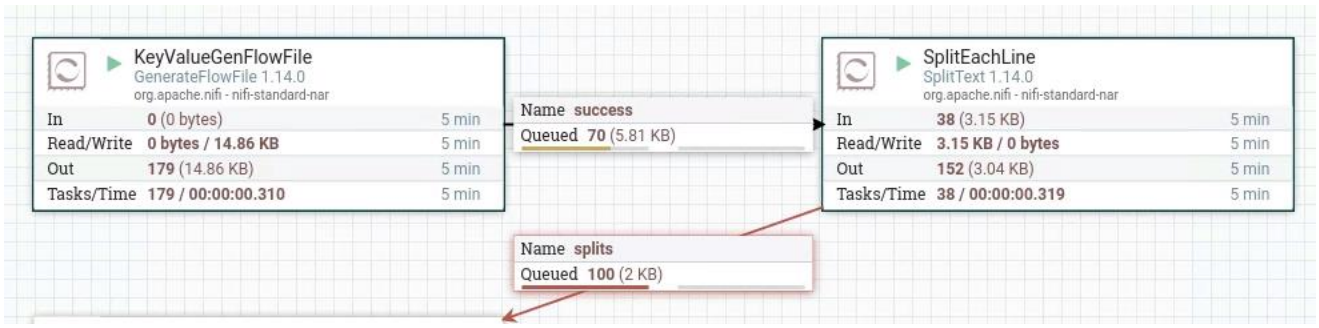


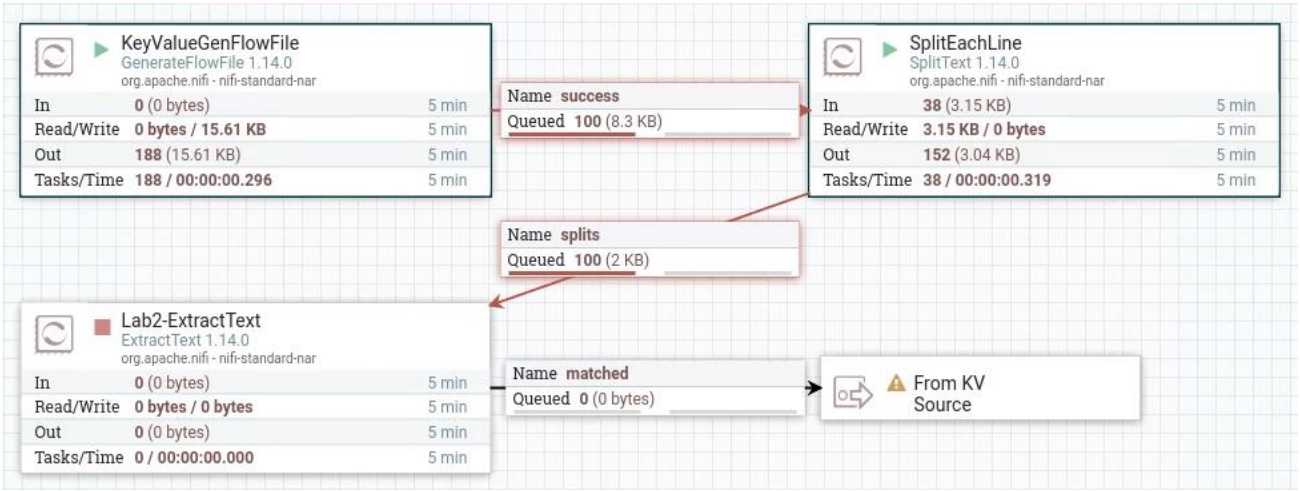


2.1.2. Stop all the processors empty all the queues. the queue.

2.1.3. This time start both the KeyValueGenFlowFile processor and the SplitEachLine processor. Observe both queues.







Notice that there seems to be another threshold marker to the right of the current one. The current marker indicates the threshold for number of flowfiles. The as of yet inactive marker on the right is to show size thresholds.

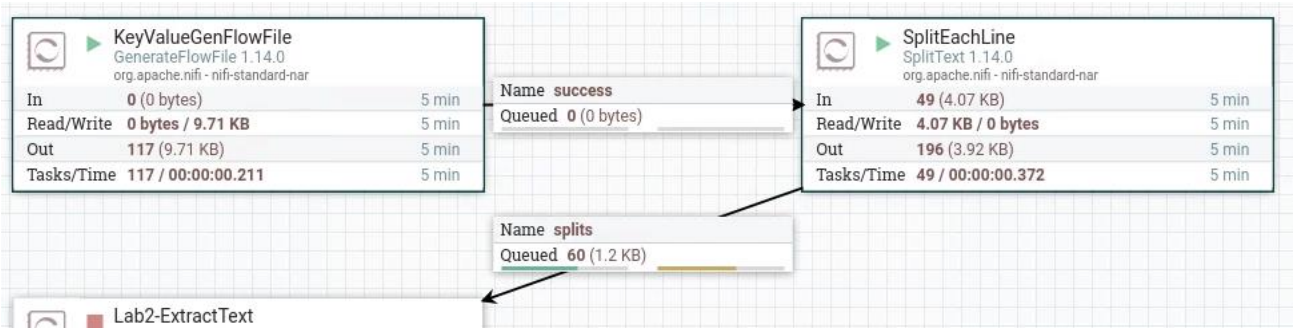
2.2. Test the backpressure size policy setting

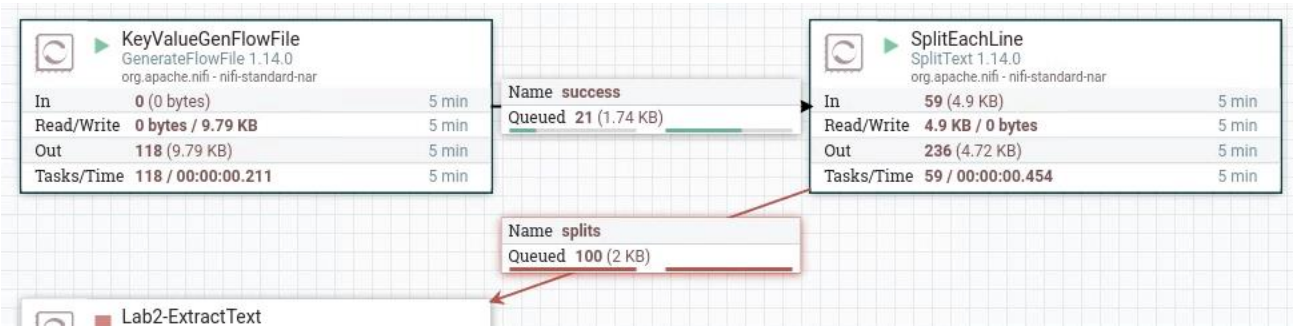
2.2.1. Stop all the processors and empty all the queues.

2.2.2. Configure the "splits" connection and change the Size Threshold to 1000B.

2.2.3. Configure the "success" connection and change the Size Threshold to 3000B.

2.2.4. Restart the KeyValueGenFlowFile and SplitEachLine processors and observer the queuers again.





What is happening? Can you deduct how the backpressure configuration policy works based on the observations?

INTENTIONALLY LEFT BLANK

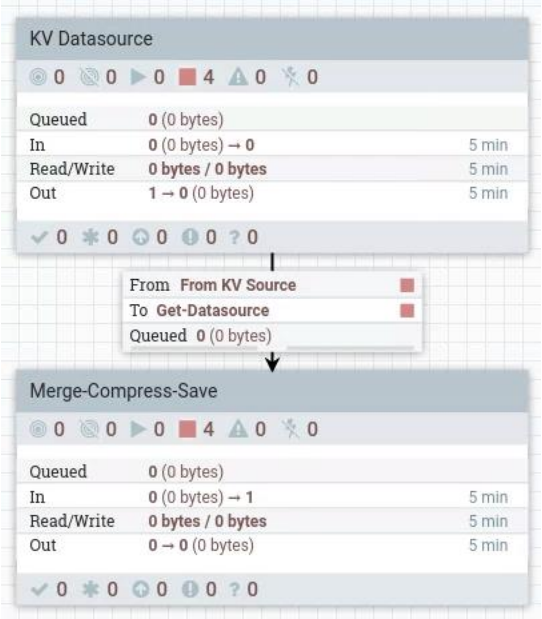
The backpressure policy has two settings. There is a total number of flowfiles queued setting and there is a total size of queued flowfiles settings. The color markers are for number of flowfiles on the left and size of flowfiles on the right. The backpressure policy triggers whenever either one of these thresholds is reached. Notice that in the success queue, the size is reached before the number of flowfiles.

3. Setting a dataflow with backpressure policy
   3.1. Running the dataflow without

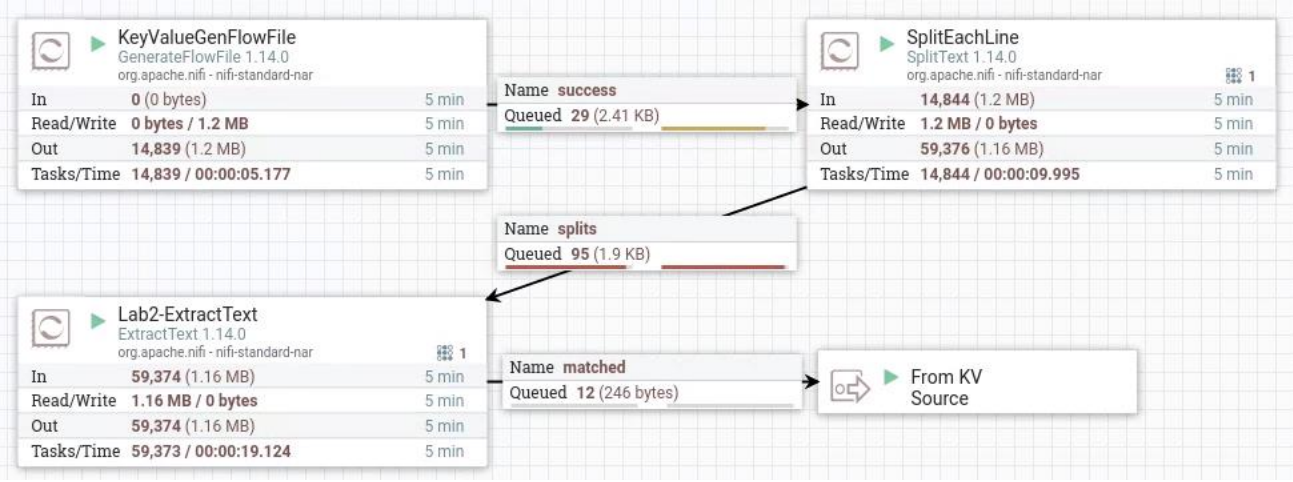
   3.2. Stop all the processor and empty all the queue.

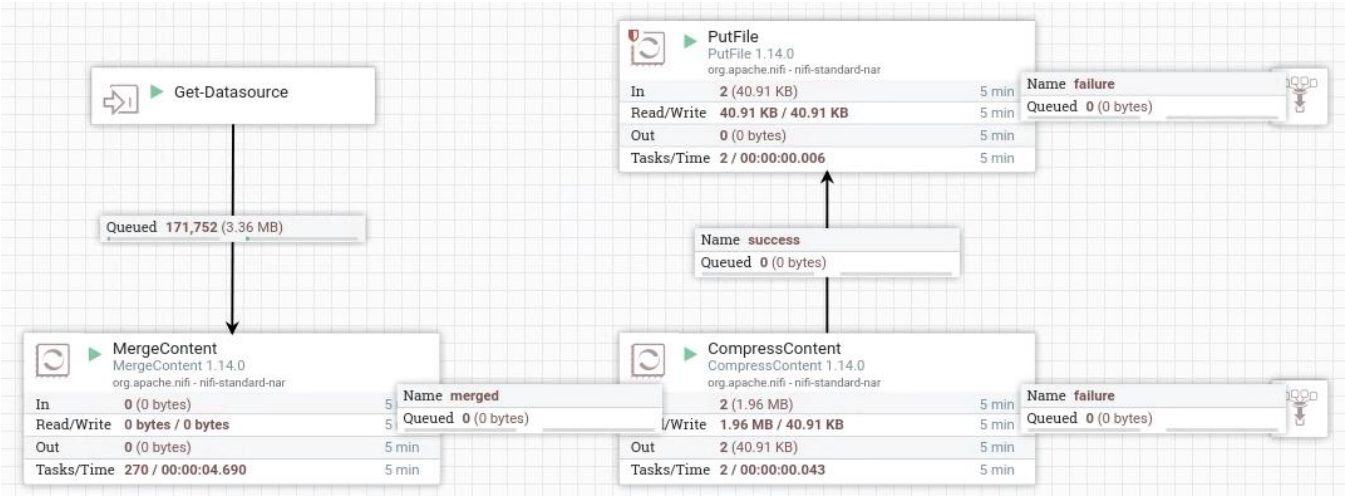
   3.3. Move up to the Lab 3 canvas.

   3.4. Connect the KV Datasource processor group to the Merge-Compress-Save processor group. Your dataflow should look similar to below.



   3.5. Start both processor groups. Observe the queues. Notice that the queues in the KV Datasource keep getting full and the backpressure policy triggers. Also note that the size of the output files is significantly smaller than before? What is happening? Think about the answers before you turn the page.






```
[student@localhost lab5]$ ll
total 120
-rw-r--r--. 1 student student 20830 Aug  4 01:58 0131a973-15ca-4bdf-81c7-f26e0077c6e9.tar
-rw-r--r--. 1 student student 20918 Aug  4 01:57 42f8fd58-9b40-490d-89c8-987a7247f88d.tar
-rw-r--r--. 1 student student 20801 Aug  4 01:58 ab67ed7b-9bb5-4ea1-885c-f21586714b5a.tar
-rw-r--r--. 1 student student 20978 Aug  4 01:59 d83a873d-063e-4607-acb5-a7ad9c08d7e4.tar
-rw-r--r--. 1 student student 20879 Aug  4 01:59 ed2877a3-3493-4b22-9c9d-e16b63cfe399.tar
```

We have setup a Merge Content processor as the receiver processor in the Merge-Compress-Save processor group. This processor tries to merge several flowfiles until criteria are met to trigger. In the previous lab, we had on purpose set things up such that it would trigger after collecting about 30 MB of data. However, now that the data source is coming from KV Datasource, the flowfiles are significantly smaller. So, instead of triggering on the Minimum Group Size, it is triggering on the Maximum Number of Entries of 1000 flowfiles. This is causing the output files to be significantly smaller. It is also causing the upstream processors in KV Source to trigger its backpressure policies because the flowfiles are piling up waiting for their content to be merged.

Lab 9: Working with Hadoop in NiFi

In this lab, we will modify the Merge-Compress-Save processor group and template. So far, we have only been saving to the local Linux drive. Now, we will add HDFS as an additional destination.

1. Create new template from existing one
   1.1. Set up processor group Lab 7

   1.2. Copy the Merge-Compress-Save processor group by adding it to the canvas from the template

   1.3. Enter the Merge-Compress-Save processor group

   1.4. Remove the connection from CompressContent to PutFile

   1.5. Remove the PutFile processor

   1.6. Add a PutHDFS processor and configure it

      1.6.1. Hadoop Configuration Resources: This is the location where the Hadoop configuration XML files are stored.

      Open a terminal and login as hadoop. Enter the following command to navigate to the correct directory. We will use bash terminal's ability to autocomplete directory names as we navigate. When the user enters a tab, bash will autocomplete the rest of the name.exit

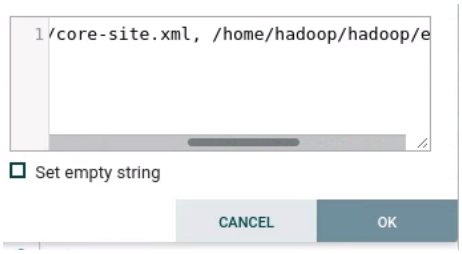
```
su - hadoop
cd ~/
cd had  <and then hit tab>
ls
cd et <and then hit tab>
ls
cd ha <and then hit tab>
ls
```

```
[student@localhost nifi]$ su - hadoop
Password:
Last login: Wed Aug  4 02:12:10 KST 2021 on pts/2
[hadoop@localhost ~]$ cd ~/
[hadoop@localhost ~]$ cd hadoop
[hadoop@localhost hadoop]$ ls
bin  include  libexec          licenses-binary  logs           NOTICE.txt  sbin
etc  lib      LICENSE-binary   LICENSE.txt      NOTICE-binary  README.txt  share
[hadoop@localhost hadoop]$ cd etc/
[hadoop@localhost etc]$ ls
hadoop
[hadoop@localhost etc]$ cd hadoop/
[hadoop@localhost hadoop]$ ls
capacity-scheduler.xml            httpfs-env.sh              mapred-queues.xml.template
configuration.xsl                 httpfs-log4j.properties    mapred-site.xml
container-executor.cfg            httpfs-site.xml            shellprofile.d
core-site.xml                     kms-acls.xml               ssl-client.xml.example
hadoop-env.cmd                    kms-env.sh                 ssl-server.xml.example
hadoop-env.sh                     kms-log4j.properties       user_ec_policies.xml.template
hadoop-metrics2.properties        kms-log4j.properties.bak   workers
hadoop-policy.xml                 kms-site.xml               yarn-env.cmd
hadoop-user-functions.sh.example  log4j.properties           yarn-env.sh
hdfs-rbf-site.xml                 mapred-env.cmd             yarnservice-log4j.properties
hdfs-site.xml                     mapred-env.sh              yarn-site.xml
[hadoop@localhost hadoop]$ pwd
/home/hadoop/hadoop/etc/hadoop
```

```
pwd
```

1.6.2. Hadoop Configuration Resources: Enter the following two entries in a single line with a comma between them

```
/home/hadoop/hadoop/etc/hadoop/core-site.xml,
/home/hadoop/hadoop/etc/hadoop/hdfs-site.xml
```

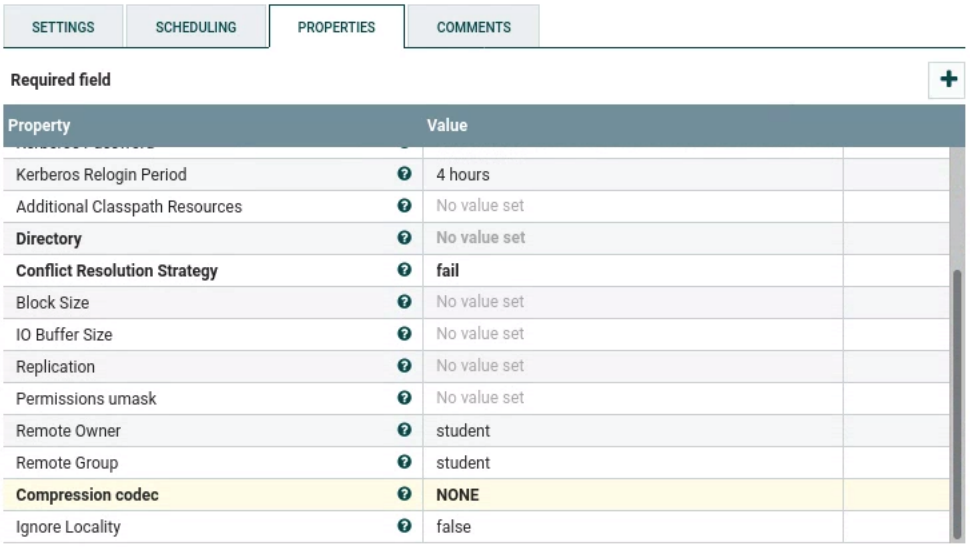


1.6.3. Directory: /user/student/lab7

1.6.4. Conflict Resolution Strategy: fail

1.6.5. Remote Owner: student

1.6.6. Remote Group: student

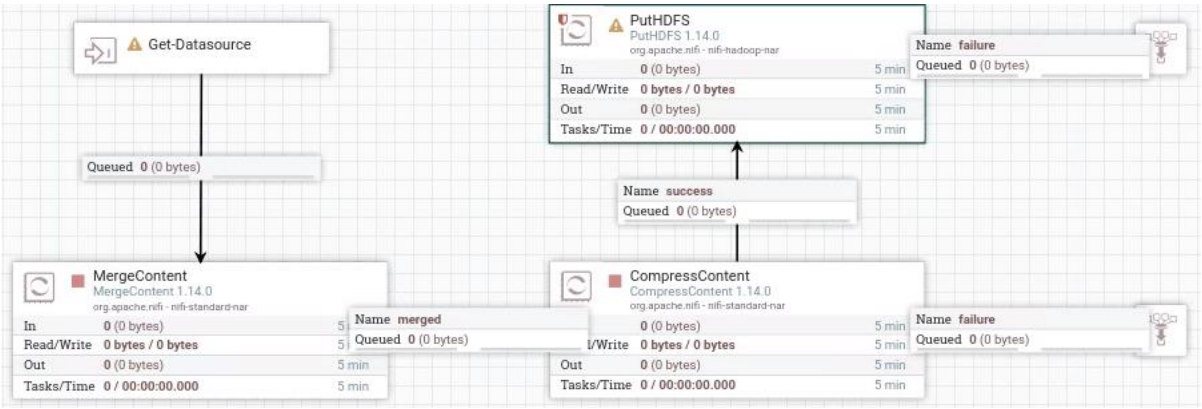


1.6.7. Automatically terminate the success relationship

1.7. Complete the connections

1.7.1. Connect the CompressContent processor to the PutHDFS processor on the success relationship

1.7.2. Connect the PutHDFS failure relationship to the funnel (if you have accidently deleted the funnel, just add a new one)



1.8. Modify the Merge-Compress-Save processor group and save as template

1.8.1. Move up to Lab 7 processor group

1.8.2. Right click on the Merge-Compress-Save processor group and select configure

1.8.3. Change the name of the processor group to Merge-Compress-HDFS

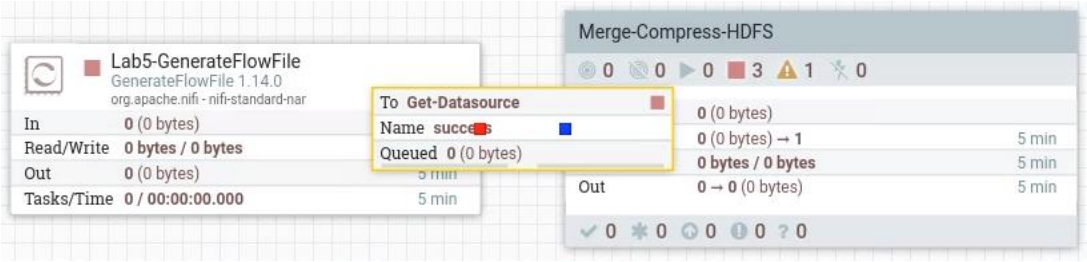
1.8.4. Save the Merge-Compress-HDFS processor group as a new template. Name the template Merge-Compress-HDFS

2. Create a new dataflow

2.1. Add a data source

2.1.1. Copy the Lab5-GenerateFlowfile and paste it to processor group Lab 7

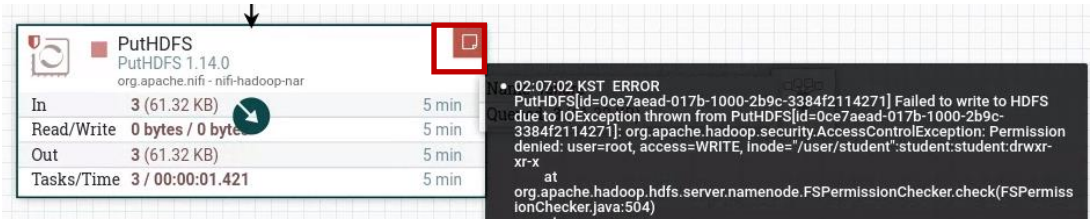
2.1.2. Connect the data source to the Merge-Compress-HDFS processor group.



2.2. Start the dataflow

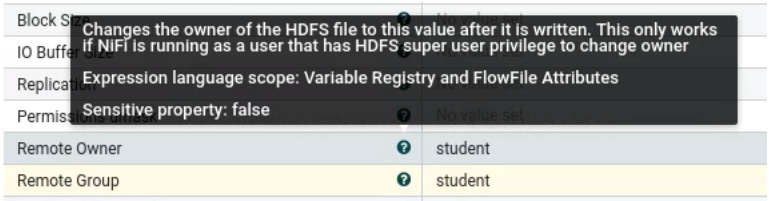
2.2.1. Start both Lab5-GenerateFlowFile and Merge-Compress-HDFS. What happens?

2.2.2. There is an error with the PutHDFS processor and a new bulletin has appeared. Hover the mouse over the bulletin to view the message.



This has happened because NiFI is currently running as the Linux root user. While the root user has complete control over all of the local directories and files, this is not the case for the HDFS file system. The ERROR occurs because the root user is trying to save to a HDFS directory that belongs to user student.

Fortunately, there is a way to work around this. Review the information provided by the setting for Remote Owner and Remote Group. It states that the user running NiFI must have HDFS superuser privileges.



2.3. Fix PutHDFS processor by setting root user as super user for HDFS

2.3.1. open a terminal and login as root using su.

2.3.2. Enter the following commands:

```
su
groupadd supergroup
usermod -aG supergroup root
groups root
hdfs dfsadmin -refreshUserToGroupsMappings
```

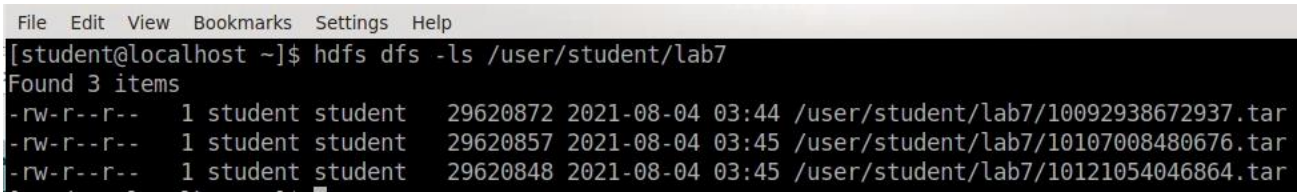
```
[root@localhost student]# groupadd supergroup
[root@localhost student]# usermod -aG supergroup root
```

```
[root@localhost student]# groups root
root : root supergroup
```

```
[root@localhost student]# hdfs dfsadmin -refreshUserToGroupsMappings
Refresh user to groups mapping successful
[root@localhost student]# hdfs dfs -mkdir /testdir
[root@localhost student]# hdfs dfs -ls /
Found 3 items
drwxr-xr-x   - root   supergroup          0 2021-08-04 02:21 /testdir
```

2.4. Restart the dataflow and check that the problem has been fixed. Inspect the 5 min statistics of the processors, and of PutHDFS in particular to see if it was able to successfully Read and more importantly Write.

2.5. Get a listing of the destination directory to check if files have been saved.



2.6. Stop all the processors and clear any queued flowfiles.

Lab 10: Creating a Kafka Topic, Producer, and Consumer

In this lab, you run use kafka services on a command line to create a topic. We use Kafka to create producers and consumers and pass data through them.

1. Setup Apache Kafka

In order to reduce demand to our virtual machine, we shall stop HBase and run only Apache Kafka.

1.1. Stop HBase services

```
sudo stop-hbase.sh
```

1.2. Restart Kafka and Zookeeper

```
sudo systemctl stop kafka
sudo systemctl stop zookeeper
sudo systemctl start zookeeper
sudo systemctl status zookeeper
sudo systemctl start kafka
sudo systemctl status kafka
```

1.3. Make sure that both zookeeper and kaka is running. If not repeat above step.

```
[student@localhost ~]$ sudo systemctl status zookeeper
● zookeeper.service
   Loaded: loaded (/etc/systemd/system/zookeeper.service; disabled; vendor preset: disab
led)
   Active: active (running) since Tue 2021-08-10 03:18:11 KST; 7s ago
 Main PID: 28265 (java)
   CGroup: /system.slice/zookeeper.service
           └─28265 java -Xmx512M -Xms512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=2...

Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Hint: Some lines were ellipsized, use -l to show in full.
[student@localhost ~]$ sudo systemctl start kafka
[student@localhost ~]$ sudo systemctl status kafka
● kafka.service
   Loaded: loaded (/etc/systemd/system/kafka.service; disabled; vendor preset: disabled)
   Active: active (running) since Tue 2021-08-10 03:18:32 KST; 6s ago
 Main PID: 28656 (sh)
   CGroup: /system.slice/kafka.service
           ├─28656 /bin/sh -c /home/kafka/kafka/bin/kafka-server-start.sh /home/kafka...
           └─28657 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -X...
```

2. Creating a Kafka Topic

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

2.1. Use kafka-topics to create a topic

2.1.1. 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.

2.1.2. 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
```

2.2. Review the details of the topic1_logs.

```
$kafka-topics –describe topic1_logs
--bootstrap-server localhost:9092
```

3. Create producers and consumers for a topic

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

3.1.1. 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.

3.1.2. 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
```

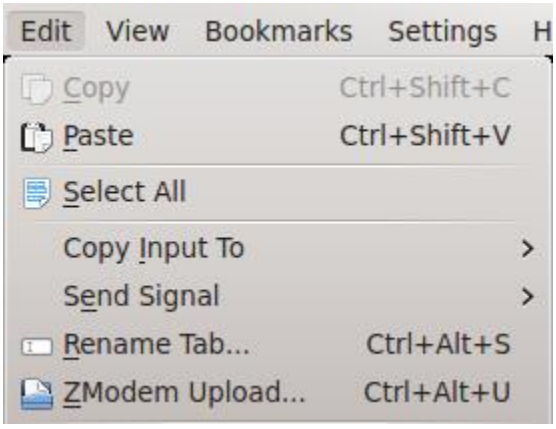
```
[student@localhost ~]$ kafka-console-consumer \
> --bootstrap-server localhost:9092 \
> --topic topic1_logs \
> --from-beginning
```

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

3.2. Rename the terminals.

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

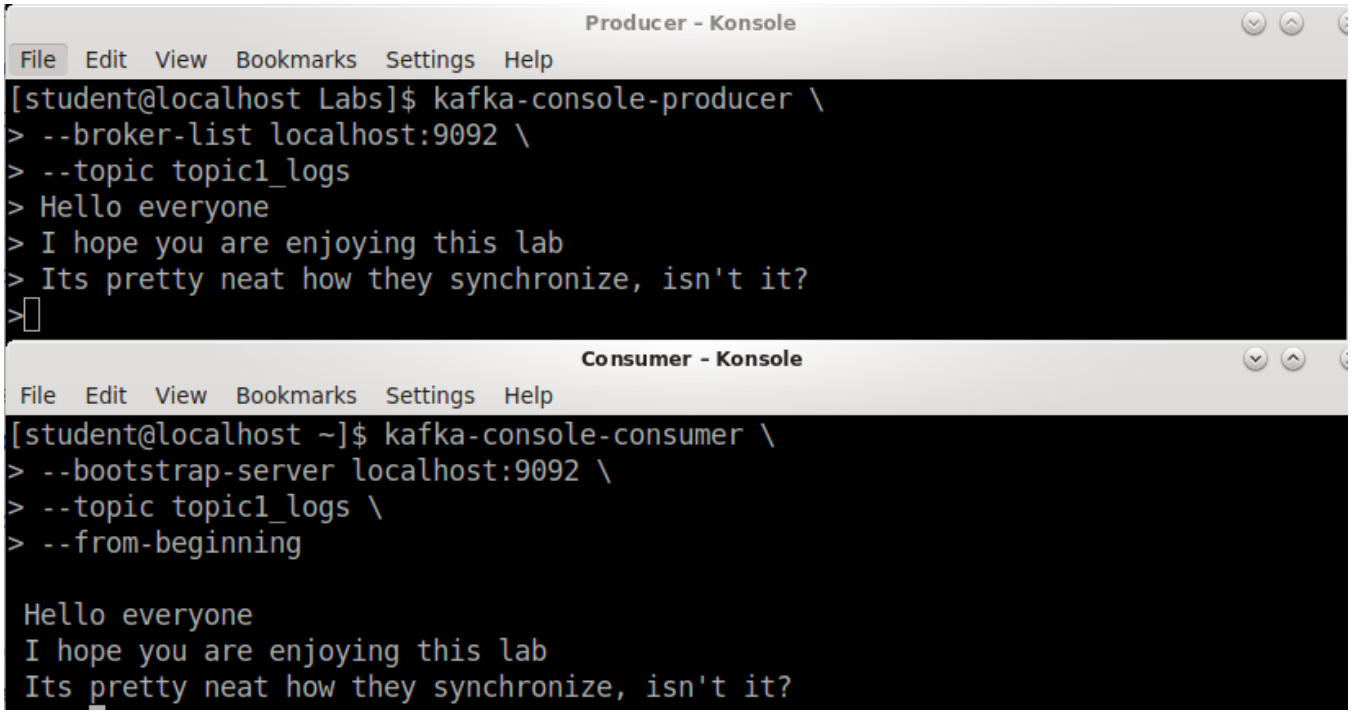
3.2.2. Do the same from the consumer terminal, naming it Consumer.



3.3. Produce messages for topic topic1_logs.

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

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



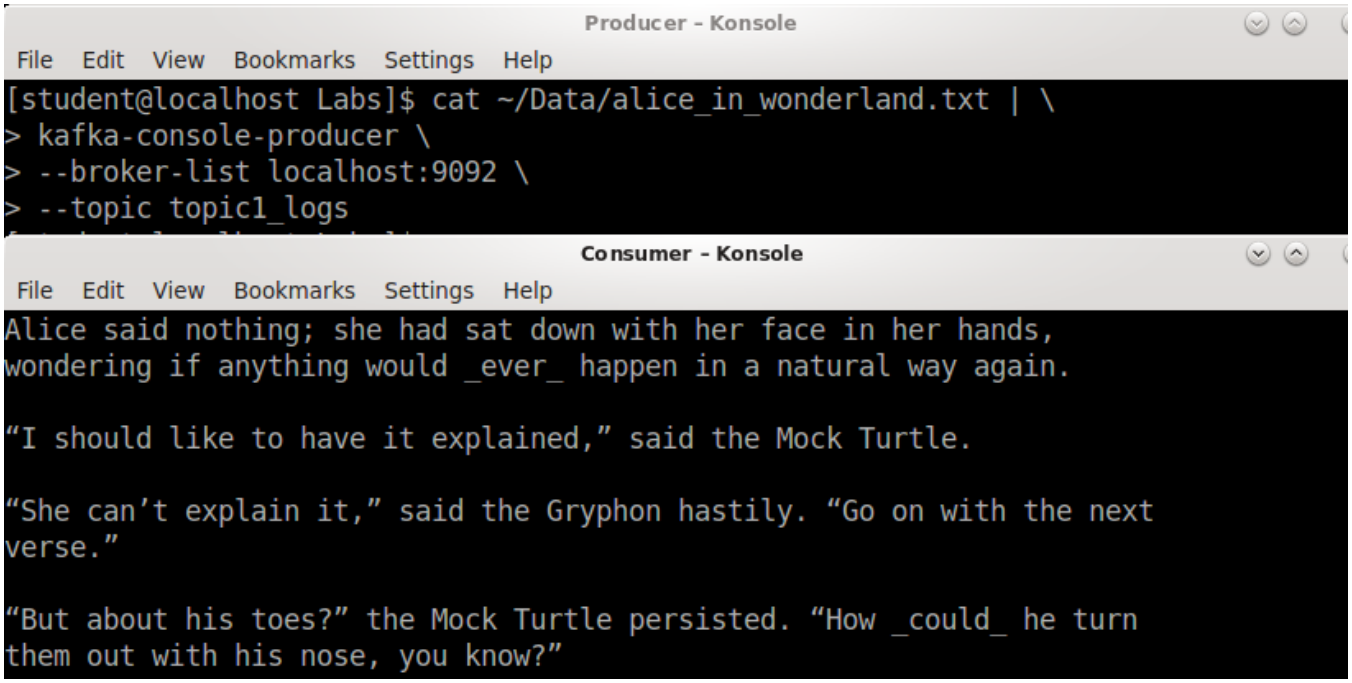
3.4. Create messages in batch mode.

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

3.4.2. 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.



3.5. Clean up

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

Lab 11: Sending Messages from Flume to Kafka

As we have learned, creating Kafka Producers and Consumers require programming them using the Kafka Producer and Consumer API. In other words, Kafka is not a plug-n-play end user tool. Rather, it requires enterprises to develop code. Today, there are many publicly available producers and consumers that organizations can utilize so the need for development has reduced dramatically.

During the early days, using Flume in conjunction with Kafka was proposed. Flume supports and extensive number of sources and sinks. The idea was to develop a Kafka source and Kafka sink for Flume. Then, they could be combined with existing Flume sources and sinks to complete the dataflow pipeline. For example, a Flume dataflow that collects data from Web logs through a NetCat source, Spool Directory source or Syslog source could then designate Kafka as the sink. This would effectively create a Kafka Producer. Another Flume dataflow with a Kafka sink to read the collected Web logs and HDFS sink to store it would effectively act as a Kafka consumer of the Web logs.

In this lab, create a Kafka producer using Flume and a Kafka Sink within. It will collect data from a streaming source that drops file to a spool directory. We will confirm that the Kafka producer is working as expected by reading the topic with the Kafka console consumer.

1. Setup Apache Kafka

In order to reduce resource demand to our virtual machine, we shall stop HBase and run only Apache Kafka

1.1. Stop HBase services

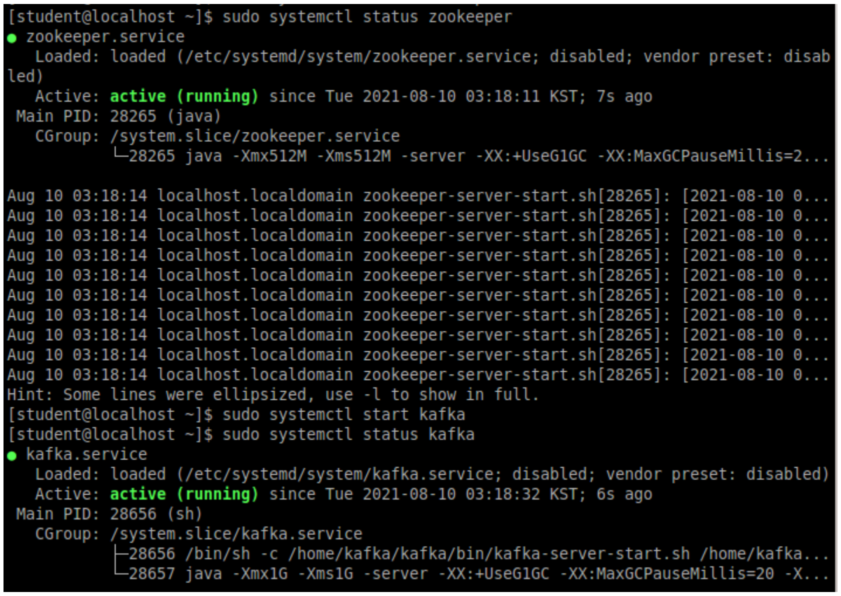
```
sudo stop-hbase.sh
```

1.2. Restart Kafka and Zookeeper

```
sudo systemctl stop kafka
sudo systemctl stop zookeeper
sudo systemctl start zookeeper
sudo systemctl status zookeeper
sudo systemctl start kafka
```

```
sudo systemctl status kafka
```

1.3. Make sure that both zookeeper and kaka is running. If not repeat above step.



```
[student@localhost ~]$ sudo systemctl status zookeeper
● zookeeper.service
   Loaded: loaded (/etc/systemd/system/zookeeper.service; disabled; vendor preset: disab
led)
   Active: active (running) since Tue 2021-08-10 03:18:11 KST; 7s ago
 Main PID: 28265 (java)
   CGroup: /system.slice/zookeeper.service
           └─28265 java -Xmx512M -Xms512M -server -XX:+UseG1GC -XX:MaxGCPauseMillis=2...

Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Aug 10 03:18:14 localhost.localdomain zookeeper-server-start.sh[28265]: [2021-08-10 0...
Hint: Some lines were ellipsized, use -l to show in full.
[student@localhost ~]$ sudo systemctl start kafka
[student@localhost ~]$ sudo systemctl status kafka
● kafka.service
   Loaded: loaded (/etc/systemd/system/kafka.service; disabled; vendor preset: disabled)
   Active: active (running) since Tue 2021-08-10 03:18:32 KST; 6s ago
 Main PID: 28656 (sh)
   CGroup: /system.slice/kafka.service
           ├─28656 /bin/sh -c /home/kafka/kafka/bin/kafka-server-start.sh /home/kafka...
           └─28657 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -X...
```

2. Create a Flume data pipeline to collect streaming text

2.1. Navigate to /home/student/Labs/C3U4 directory

2.2. Create a directory and name it spooldir

```
mkdir /home/student/Labs/C3U4/spooldir
```

2.3. Start with the spooldir-stub.conf stub file provided and complete the Flume configuration.

2.3.1. Rename the file to spooldir.conf

2.3.2. Refer to the following to configure a Spool Directory Source: https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#spooling-directory-source

2.3.3. Configure the **streaming-txt-source** type to spooldir

2.3.4. Set the spool directory to /home/student/Labs/C3U4/spool

2.3.5. Refer to the following to configure a memory channel: https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#memory-channel

2.3.6. Set the **memory-channel** channel type to memory

2.3.7. Configure the memory-channel capacity to 10000

2.3.8. Configure the maximum number of events per transaction to 100

2.3.9. Refer to the following to configure a Kafka Sink
https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#kafka-sink

2.3.10. Set the **kafka-sink** sink type to org.apache.flume.sink.kafka.KafkaSink

2.3.11. Set the Kafka topic to stream_text

2.3.12. Set the list of bootstrap servers to localhost:9092

2.3.13. Set the number of messages to process in one batch to 5

2.3.14. Refer to the following to configure a Logger Sink
https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#logger-sink

2.3.15. Set the **logger-sink** type to logger

2.3.16. Use the **memory-channel** for **streaming-txt-source**, **logger-sink** and **kafka-sink**.

3. Create a Kafka Console Consumer

3.1. Open a new terminal and name it Kafka Consumer Producer as we did in step Error! Reference source not found.

3.2. Start a Kafka console consumer

3.2.1. Execute the kafka-console-consumer command with the following parameters

3.2.2. Set the topic to stream_text

3.2.3. Set the consumer to read from the beginning

4. Test the data pipeline

4.1. Run the Flafka Producer Flume Agent

4.1.1. Open a new terminal and change the name of the terminal to Flafka Producer as we did in step Error! Reference source not found.

4.1.2. Start the Flume dataflow that we created above

```
flume-ng agent --conf $FLUME_HOME/conf \
```

```
--conf-file /home/student/Labs/C3U4/spooldir.conf \
--name agent1 -Dflume.root.logger=INFO,console
```

4.2. Run the streaming source simulator

4.2.1. Open a new terminal and name it Streaming Source

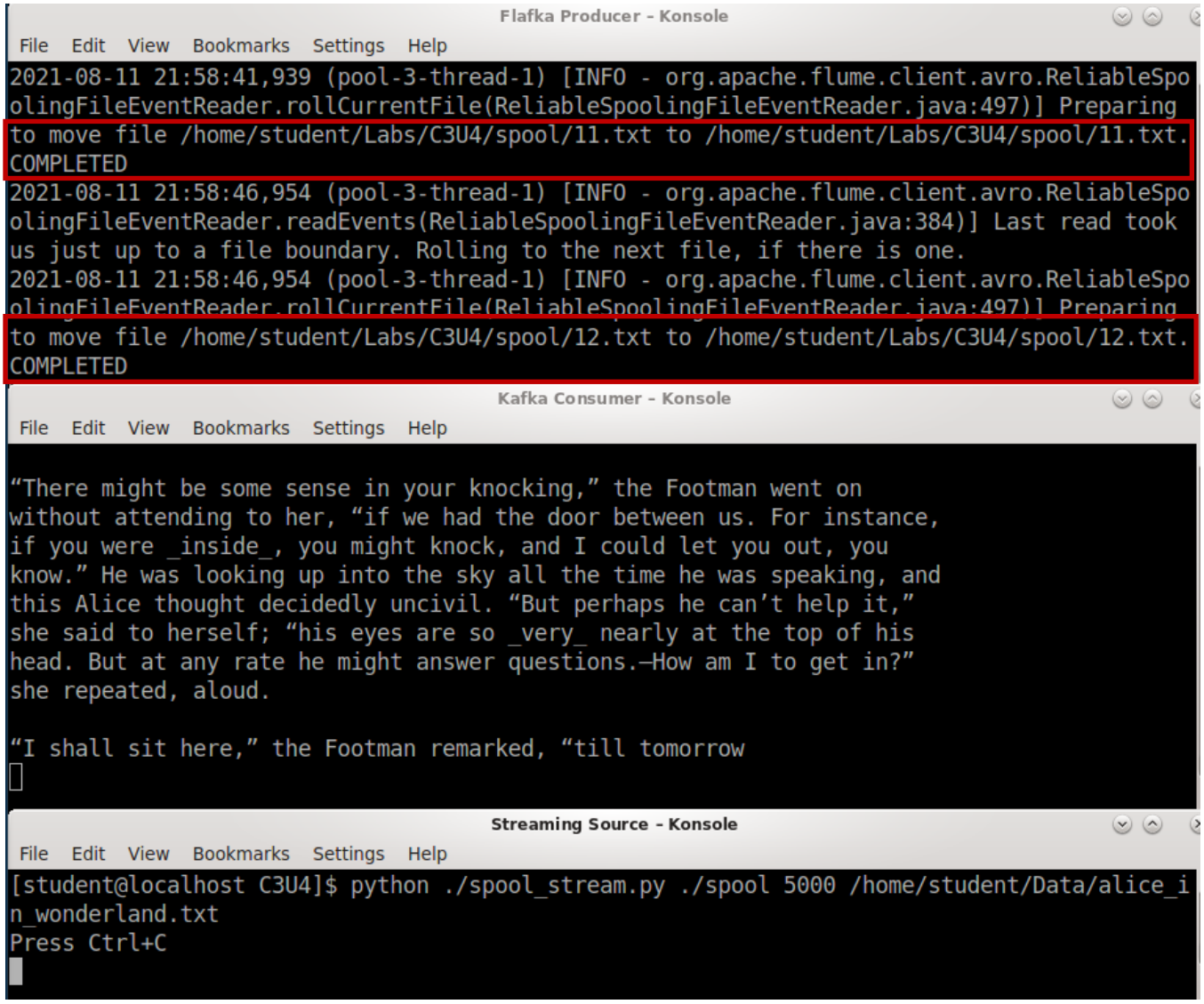
4.2.2. Navigate to /home/student/Labs/C3U4

4.2.3. Create a directory and name it spool. If the directory exists from previous attempts of the lab, completely remove the directory and recreate it. In order to quickly delete the content and directory, you may use the following command:

```
rm -rf ./spool
```

4.2.4. Execute the spool_stream.py python program. This command takes 5000 characters from alice_in_wonderland.txt, creates a temporary file to stage it, and then moves the file to the spool directory. It produces one file every 5 seconds (hard-coded).

```
python ./spool_stream.py ./spool 5000 ~/Data/alice_in_wonderland.txt
```



4.2.5. The output from the three terminals should look similar to above. In the Flafka Producer terminal, the logger is displaying the file from the spool directory that it is sending to Kafka topic. The Kafka Consumer screen will update every 5 seconds with new content.

> Note: There is a chance that the Flafka Producer may fail while running. This is due to the copy move script taking too much time. You many ignore this.

4.2.6. From the Streaming Source terminal, press Ctrl+C to stop the streaming simulator.

4.2.7. Stop the Flafka producer.

4.2.8. Stop the Kafka consumer.

END OF LAB