Exercise #2:

CAPTURE REAL TIME EVENT STREAM WITH APACHE KAFKA

INTRODUCTION

Apache Kafka can be used on the Hortonworks Data Platform to capture real-time events. We will begin with showing you how to configure Apache Kafka and Zookeeper. Next, we will show you how to capture truck event data from Apache NiFi using Kafka.

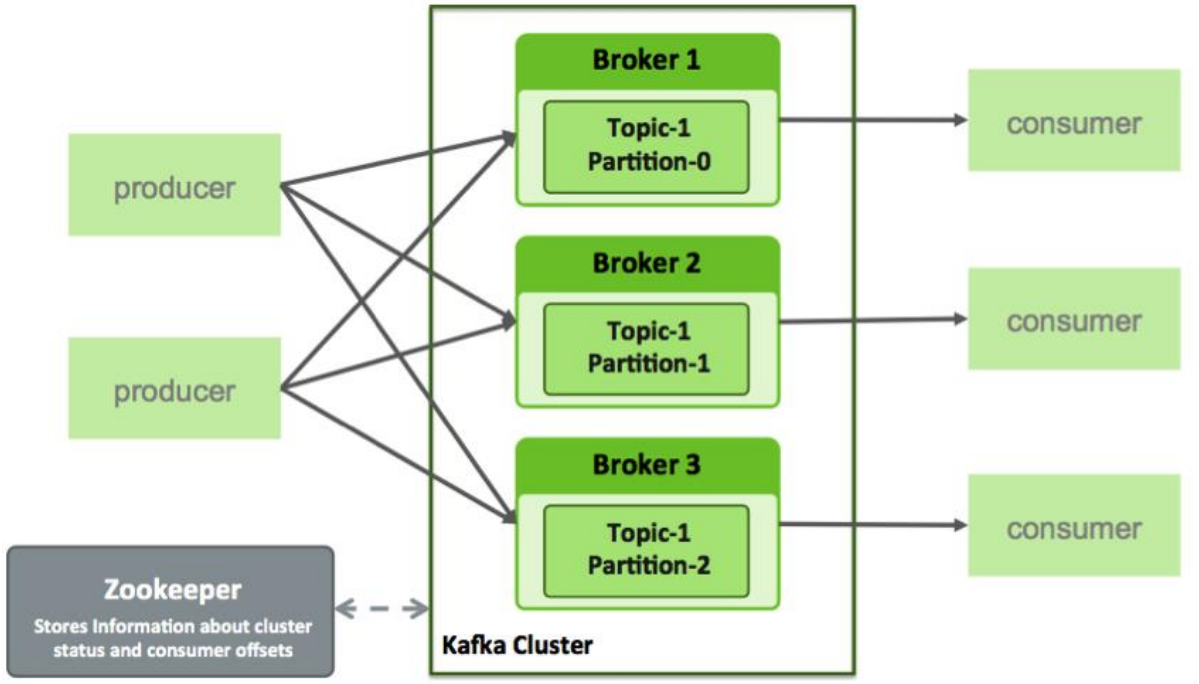
PRE-REQUISITES

- Exercise 0: Set Up Simulator, Apache Services and IDE Environment
- Exercise 1: Ingest, Route and Land Real Time Events with Apache NiFi

APACHE KAFKA

Apache Kafka is an open source messaging system designed for:

- Persistent messaging
- High throughput
- Distributed
- Multi-client support
- Real time



Kafka Producer-Broker-Consumer

EXERCISE OVERVIEW

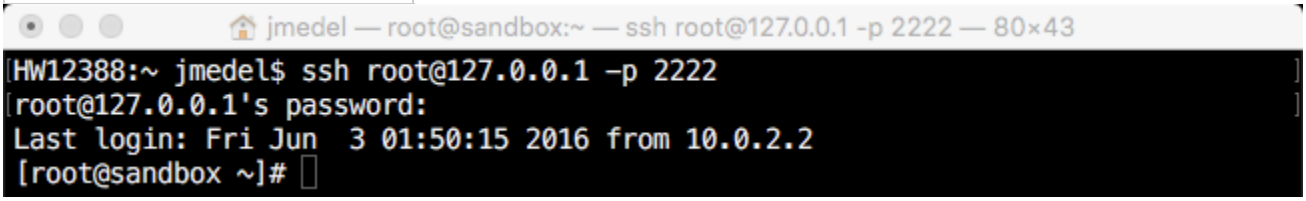
- Create Kafka topics for Truck events.
- Write Kafka Producers for Truck events.

STEP 1: CREATE A KAFKA TOPIC

1.1 SSH INTO THE SANDBOX

SSH into the Sandbox to define the Kafka topic. Type the following command:

```
ssh root@127.0.0.1 -p 2222
```



NOTE: You can also SSH using a program like Putty for Windows or the Terminal application on a Mac.

1.2 CREATE A NEW TOPIC

Use the kafka-topics.sh script (which should be in your PATH), create a new topic named truck_events:

```
[root@sandbox ~]# kafka-topics.sh --create --zookeeper 127.0.0.1:2181 --
replication-factor 1 --partitions 2 --topic truck_events
```

If the kafka-topics.sh script is not in your PATH and you get a command not found error, then change directories to where the Kafka scripts are installed:

```
[root@sandbox ~]# cd /usr/hdp/current/kafka-broker/bin/
```

You will need to **add a dot and a slash (./)** to the beginning of the commands:

```
[root@sandbox bin]# ./kafka-topics.sh --create --zookeeper 127.0.0.1:2181 --
replication-factor 1 --partitions 2 --topic truck_events
```

Also note that sometimes ZooKeeper does not listen on localhost, so you may need to use 127.0.0.1 or the Sandbox's IP address instead.

The output should show your topic was created:

```
[root@sandbox bin]# ./kafka-topics.sh --create --zookeeper localhost:2181 --repl
ication-factor 1 --partitions 2 --topic truck_events
WARNING: Due to limitations in metric names, topics with a period ('.') or under
score ('_') could collide. To avoid issues it is best to use either, but not bot
h.
Created topic "truck_events".
[root@sandbox bin]#
```

1.3 VERIFY THE TOPIC WAS CREATED SUCCESSFULLY

Check if topic truck_events was created successfully with the following command:

```
[root@sandbox ~]# ./kafka-topics.sh --list --zookeeper 127.0.0.1:2181
```

You should see truck_events in the list of topics (and probably your only topic):

```
[root@sandbox bin]# ./kafka-topics.sh --list --zookeeper localhost:2181
truck_events
[root@sandbox bin]#
```

STEP 2: CREATE NIFI PUTKAFKA PROCESSOR

In the previous Exercise, we stored the truck event data into a file. Now we can use the PutKafka processor since the Kafka service is running, we have access to the "Known Broker", "Topic Name" and "Client Name." We will send the truck event contents of a FlowFile to Kafka as a message. Similar to the Kafka Producer, NiFi acts as a producer since it creates messages and publishes them to the Kafka broker for further consumption.

1. If not already open, navigate to the NiFi Web Interface at http://127.0.0.1:9090/nifi/. For vmware and azure, the port may be different.

2. If your data flow is still running, click on the stop button  in the **actions** toolbar to stop the flow.

2.1 ADD PUTKAFKA PROCESSOR

1. Drag the processor icon onto your graph. Add the **PutKafka** processor.

2. Right click on the Putkafka processor, click on **configure**.

3. The warning message tells us, we need to add “Known Broker”, “Topic Name” and “Client Name” values to our Properties section. Enter the following information into the **Properties** section in the **Configure Processor** window:

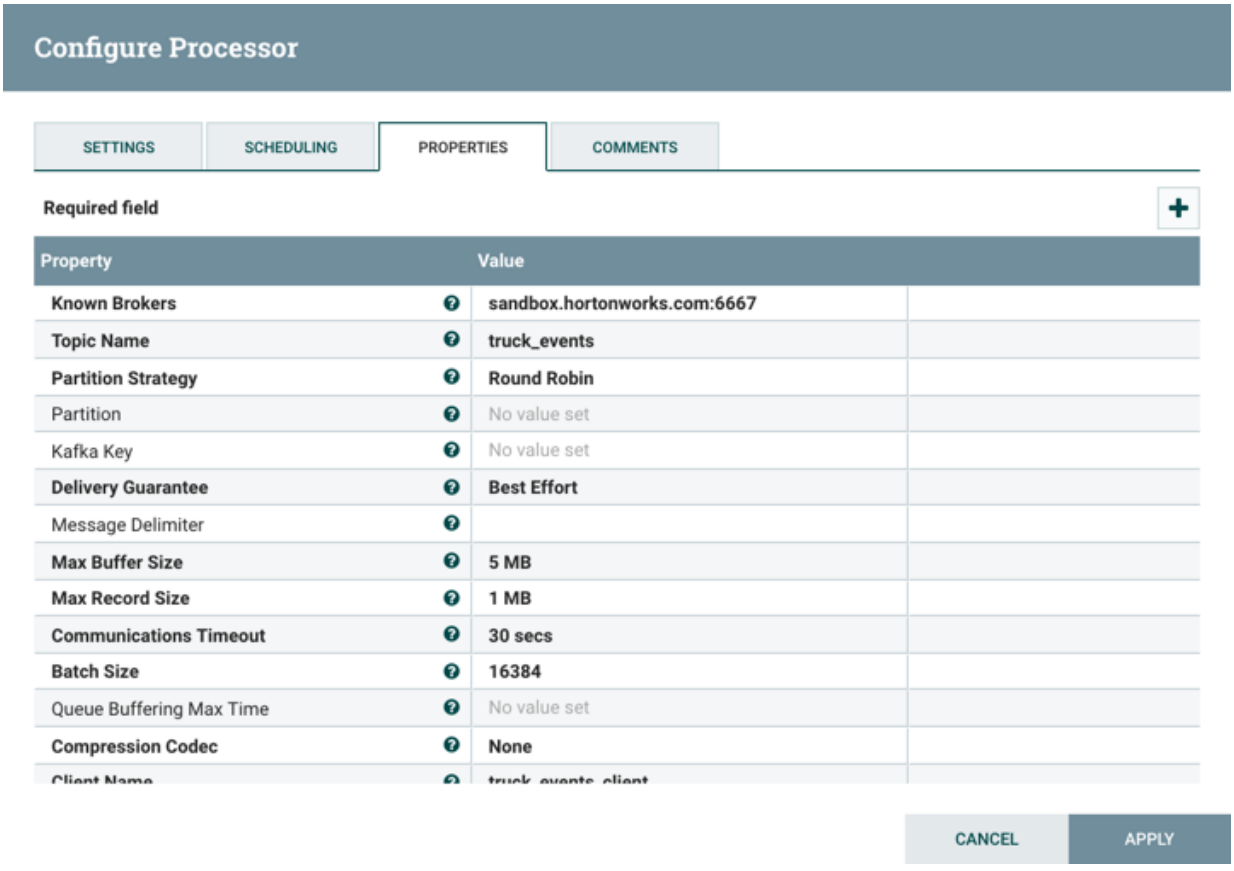
Property = Value

Known Brokers = 127.0.0.1:6667

Topic Name = truck_events

Message Delimiter = press "Shift+enter"

Client Name = truck_events_client



Note: Every property above is required except **Message Delimiter**, but this property is helpful with splitting apart the contents of the FlowFile.

Known Brokers can be found in **Kafka** configs under listeners

Topic Name is the name you created earlier for Kafka. Type the following command to see your topic name: `./kafka-topics.sh –list –zookeeper 127.0.0.1:2181`.

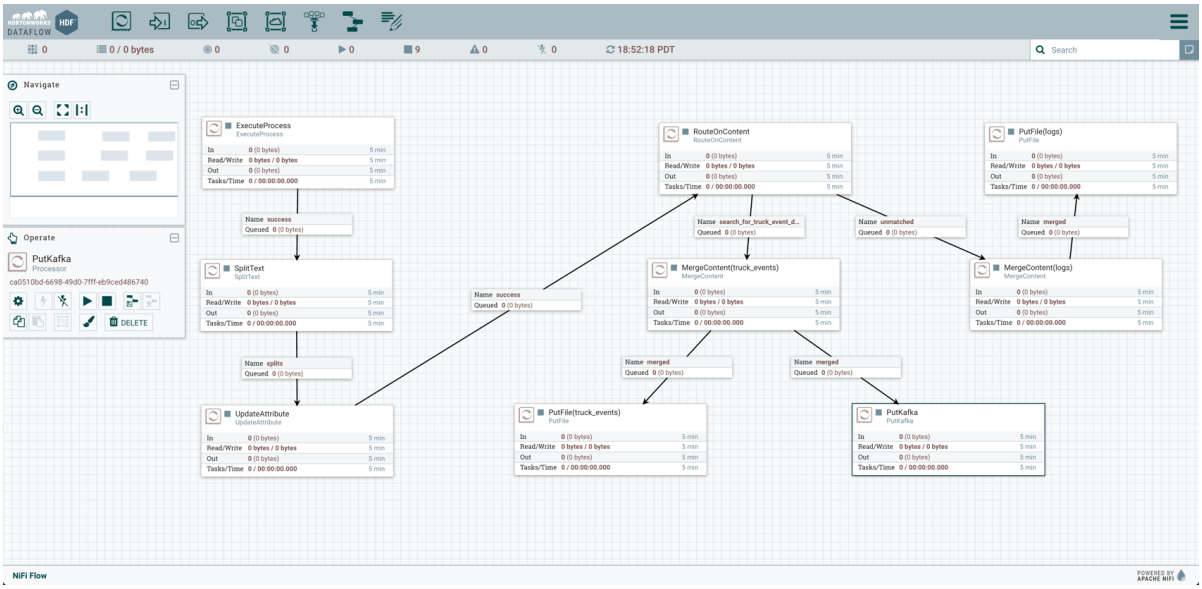
Message Delimiter set as “Shift+enter” in the value field makes each line of incoming FlowFile a single message, so kafka does not receive an enormous flowfile as a single message.

Client Name can be named to your liking, it is the name that is used when communicating with kafka.

4. Open the Configure Processor window again, navigate to the **Settings** tab. Set the **Auto termination relationship** to success and failure. Click **apply**.

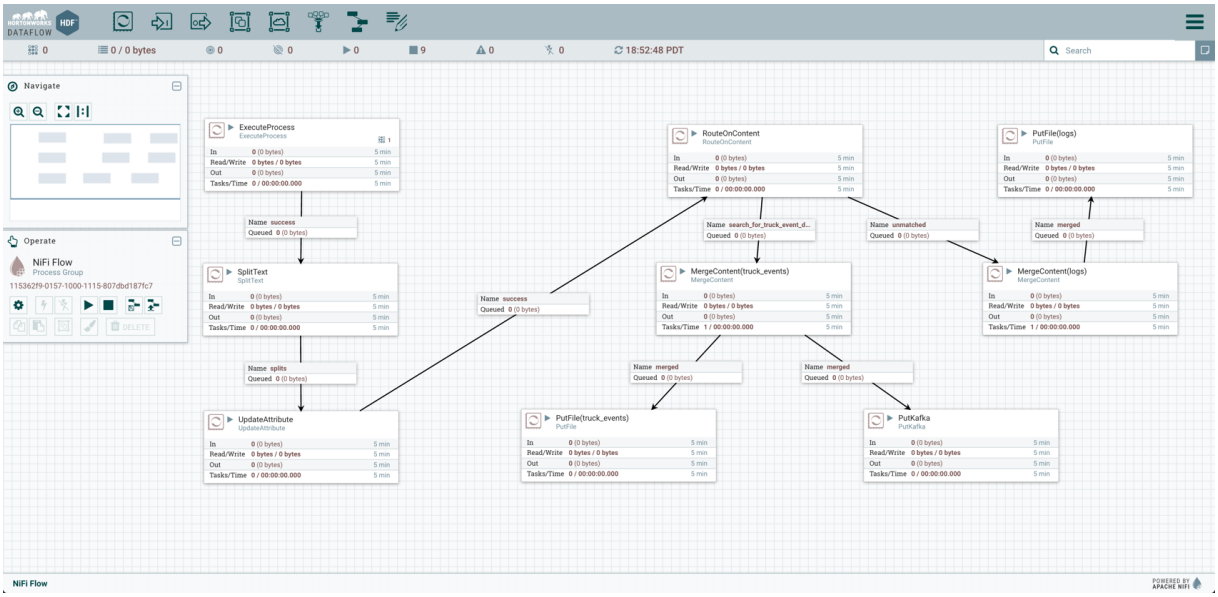
5. Connect the **MergeContent(truck_events)** processor to **PutKafka**. A Create Connection window will appear, set the **For relationship** to merged.

You should obtain a similar dataflow as the following:



Note: If there is a warning symbol after updating the PutKafka, verify that the property values are correct. Check **3**. in case you need to review the values changed.

6. Let’s start our Hortonworks DataFlow to see a real live stream of truck event data be read from NiFi and written to a Kafka cluster. In the **actions** toolbar, hit the **start** button.



Dataflow generates data, filtering truck events from the dataflow and sending those events to kafka.

2.2 VERIFY PUTKAFKA PUBLISHED MESSAGE TO KAFKA

After a few seconds, stop the NiFi DataFlow using the **stop** button in the **actions** toolbar.
To verify that the PutKafka processor successfully published messages to the Kafka cluster, execute the following command to start a consumer to see the produced events:

```
[root@sandbox ~]# cd /usr/hdp/current/kafka-broker/bin/
```

```
[root@sandbox bin]# ./kafka-console-consumer.sh --zookeeper 127.0.0.1:2181 --topic
truck_events --from-beginning
```

Your terminal should show that messages successfully published to Kafka:

```
jmedel — root@sandbox:/usr/hdp/current/kafka-broker/bin — ssh root@127.0.0.1 -p 2222 — 143×36
2016-06-01 11:20:23.709|33|11|Jamie Engesser|160779139|Des Moines to Chicago Route 2|Normal|41.89|-87.66|1000
2016-06-01 11:20:23.728|51|31|Rommel Garcia|1594289134|Memphis to Little Rock Route 2|Normal|38.67|-94.38|1000
2016-06-01 11:20:23.758|95|28|Olivier Renault|137128276|Springfield to KC Via Hanibal Route 2|Lane Departure|39.53|-94.28|1000
2016-06-01 11:20:23.758|68|24|Don Hilborn|1961634315|Saint Louis to Memphis|Unsafe tail distance|37.47|-89.71|1000
2016-06-01 11:20:23.779|107|16|Tom McCuch|1384345811|Joplin to Kansas City|Normal|40.7|-89.52|1000
2016-06-01 11:20:23.859|85|20|Chris Harris|371182829|Memphis to Little Rock|Normal|42.21|-88.64|1000
2016-06-01 11:20:23.979|16|19|Ajay Singh|1962261785|Wichita to Little Rock.kml|Normal|35.47|-94.88|1000
2016-06-01 11:20:23.979|42|32|Ryan Templeton|1594289134|Memphis to Little Rock Route 2|Normal|35.21|-90.37|1000
2016-06-01 11:20:23.979|26|29|Teddy Choi|803014426|Wichita to Little Rock Route 2|Unsafe following distance|35.37|-94.57|1000
2016-06-01 11:20:24.048|109|30|Dan Rice|1567254452|Saint Louis to Memphis Route2|Unsafe tail distance|41.92|-89.03|1000
2016-06-01 11:20:24.088|98|18|Grant Liu|1565885487|Springfield to KC Via Hanibal|Lane Departure|39.53|-94.28|1000
2016-06-01 11:20:24.099|92|21|Jeff Markham|160779139|Des Moines to Chicago Route 2|Normal|34.78|-92.31|1000
2016-06-01 11:20:24.178|84|25|Jean-Philippe Player|1198242881| Saint Louis to Chicago Route2|Normal|37.47|-89.71|1000
2016-06-01 11:20:24.2|54|13|Joe Niemiec|1090292248|Peoria to Ceder Rapids Route 2|Normal|42.21|-88.64|1000
2016-06-01 11:20:24.219|72|10|George Vetticaden|1390372503|Saint Louis to Tulsa|Normal|36.17|-95.99|1000
2016-06-01 11:20:24.289|64|15|Rohit Bakshi|1198242881| Saint Louis to Chicago Route2|Overspeed|35.19|-90.04|1000
```

SUMMARY

This Exercise gave you brief glimpse of how to use Apache Kafka to capture real-time event data as a message and verify that Kafka successfully received those messages. In our next Exercise, you will create HBase and Hive tables to ingest real-time streaming data using Storm.