SMOKE TEST

Confluent Kafka 5

Date Prepared: Oct 2019

Document Information

Project Name	**EPIC Accelerator Deployment & Integration Services**		
Project Owner		Document Version No	0.1
Quality Review Method			
Prepared By	Priyanka	Preparation Date	Oct 2019
Reviewed By		Review Date	

Table of Contents

Table of Tables

NO TABLE OF FIGURES ENTRIES FOUND.

1 PREREQUISITES

In this section, we will see pre-requisites:

- This installation includes a Kafka broker, KSQL, Control Center, Zookeeper, Schema Registry, REST Proxy, and Kafka Connect.
- If you installed Confluent Platform via TAR or ZIP, navigate into the installation directory. The paths and commands used throughout this tutorial assume that you are in this installation directory.
- Java: Minimum version 1.8. Install Oracle Java JRE or JDK >= 1.8 on your local machine.

Create and produce data to the Kafka topics pageviews and users. These steps use the KSQL datagen that is included Confluent Platform.

- Create the pageviews topic and produce data using the data generator. The following example continuously generates data with a value in DELIMITED format.

```
<path-to-confluent>/bin/ksql-datagen quickstart=pageviews
format=delimited topic=pageviews maxInterval=500
```

- Produce Kafka data to the user's topic using the data generator. The following example continuously generates data with a value in JSON format.

```
<path-to-confluent>/bin/ksql-datagen quickstart=users format=json
topic=users maxInterval=100
```

2 LAUNCH THE KSQL CLI

After KSQL is started, your terminal should resemble this:

```
                 ===========================================
                  =        _  __ _____  ____  _             =
                  =       | |/ // ____|/ __ \| |            =
                  =       | ' /| (___ | |  | | |            =
                  =       |  <  \___ \| |  | | |            =
                  =       | . \ ____) | |__| | |____        =
                  =       |_|\_\_____/ \___\_\______|       =
                  =                                         =
                  =  Streaming SQL Engine for Apache Kafka® =
                  ===========================================

Copyright 2018 Confluent Inc.

CLI v5.1.1, Server v5.1.1 located at http://localhost:8088

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

3 INSPECT KAFKA TOPICS BY USING SHOW AND PRINT STATEMENTS

KSQL enables inspecting Kafka topics and messages in real time.

- Use the SHOW TOPICS statement to list the available topics in the Kafka cluster.
- Use the PRINT statement to see a topic's messages as they arrive.

In the KSQL CLI, run the following statement:

```
SHOW TOPICS;
```

Output:

```
Kafka Topic       | Registered | Partitions | Partition Replicas | Consumers |
ConsumerGroups
-------------------------------------------------------------------------------------------
 _confluent-metrics | false      | 12         | 1                  | 0         | 0
 _schemas           | false      | 1          | 1                  | 0         | 0
 pageviews          | false      | 1          | 1                  | 0         | 0
 users              | false      | 1          | 1                  | 0         | 0
-------------------------------------------------------------------------------------------
```

1. Inspect the users topic by using the PRINT statement:

   ```
   PRINT 'users';
   ```

   Output:

   ```
   Format:JSON
   {"ROWTIME":1540254230041,"ROWKEY":"User_1","registertime":1516754966866,"user
   id":"User_1","regionid":"Region_9","gender":"MALE"}
   {"ROWTIME":1540254230081,"ROWKEY":"User_3","registertime":1491558386780,"user
   id":"User_3","regionid":"Region_2","gender":"MALE"}
   {"ROWTIME":1540254230091,"ROWKEY":"User_7","registertime":1514374073235,"user
   id":"User_7","regionid":"Region_2","gender":"OTHER"}
   ^C{"ROWTIME":1540254232442,"ROWKEY":"User_4","registertime":1510034151376,"u
   serid":"User_4","regionid":"Region_8","gender":"FEMALE"}
   Topic printing ceased
   ```

2. Inspect the pageviews topic by using the PRINT statement:

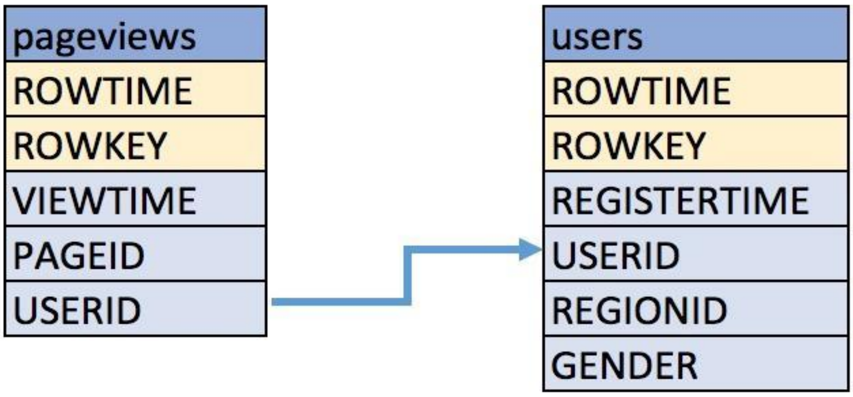
   ```
   PRINT 'pageviews';
   ```

Output:

```
Format:STRING
10/23/18 12:24:03 AM UTC , 9461 , 1540254243183,User_9,Page_20
10/23/18 12:24:03 AM UTC , 9471 , 1540254243617,User_7,Page_47
10/23/18 12:24:03 AM UTC , 9481 , 1540254243888,User_4,Page_27
^C10/23/18 12:24:05 AM UTC , 9521 , 1540254245161,User_9,Page_62
Topic printing ceased
ksql>
```

4 CREATE A STREAM AND TABLE

These examples query messages from Kafka topics called pageviews and users using the following schemas:



pageviews
ROWTIME
ROWKEY
VIEWTIME
PAGEID
USERID

users
ROWTIME
ROWKEY
REGISTERTIME
USERID
REGIONID
GENDER

1. Create a stream pageviews_original from the Kafka topic pageviews, specifying the value_format of DELIMITED.
2. Describe the new STREAM. Notice that KSQL created additional columns called ROWTIME, which corresponds to the Kafka message timestamp, and ROWKEY, which corresponds to the Kafka message key.

```
CREATE STREAM pageviews_original (viewtime bigint, userid
varchar, pageid varchar) WITH \
(kafka_topic='pageviews', value_format='DELIMITED');
```

Output:

```
Message
---------------
Stream created
---------------
```

3. Create a table users_original from the Kafka topic users, specifying the value_format of JSON.

```
CREATE TABLE users_original (registertime BIGINT, gender
VARCHAR, regionid VARCHAR, userid VARCHAR) WITH \
(kafka_topic='users', value_format='JSON', key = 'userid');
```

Output:

```
Message
---------------
Table created
---------------
```

4. Show all streams and tables.

```
ksql> SHOW STREAMS;
```

```
 Stream Name              | Kafka Topic             | Format
----------------------------------------------------------------
 PAGEVIEWS_ORIGINAL       | pageviews               | DELIMITED
```

```
ksql> SHOW TABLES;
```

```
 Table Name        | Kafka Topic       | Format    | Windowed
-------------------------------------------------------------
 USERS_ORIGINAL    | users             | JSON      | false
```

5 WRITE QUERIES

These examples write queries using KSQL.

Note: By default KSQL reads the topics for streams and tables from the latest offset.

1. Use SELECT to create a query that returns data from a STREAM. This query includes the LIMIT keyword to limit the number of rows returned in the query result. Note that exact data output may vary because of the randomness of the data generation.

```
SELECT pageid FROM pageviews_original LIMIT 3;
```

Output:

```
Page_24
Page_73
Page_78
LIMIT reached
Query terminated
```

2. Create a *persistent query* by using the CREATE STREAM keywords to precede the SELECT statement. The continual results from this query are written to the PAGEVIEWS_ENRICHED Kafka topic. The following query enriches the pageviews STREAM by doing a LEFT JOIN with the users_original TABLE on the user ID.

```
CREATE STREAM pageviews_enriched AS \
SELECT users_original.userid AS userid, pageid, regionid,
gender \
FROM pageviews_original \
LEFT JOIN users_original \
  ON pageviews_original.userid = users_original.userid;
```

Output:

```
Message
----------------------------
Stream created and running
----------------------------
```

3. Use SELECT to view query results as they come in. To stop viewing the query results, press <ctrl-c>. This stops printing to the console but it does not terminate the actual query. The query continues to run in the underlying KSQL application.

```
SELECT * FROM pageviews_enriched;
```

Output:

```
1519746861328 | User_4 | User_4 | Page_58 | Region_5 | OTHER
1519746861794 | User_9 | User_9 | Page_94 | Region_9 | MALE
1519746862164 | User_1 | User_1 | Page_90 | Region_7 | FEMALE
^CQuery terminated
```

4. Create a new persistent query where a condition limits the streams content, using WHERE. Results from this query are written to a Kafka topic called PAGEVIEWS_FEMALE.

```
CREATE STREAM pageviews_female AS \
SELECT * FROM pageviews_enriched \
WHERE gender = 'FEMALE';
```

Output:

```
Message
----------------------------
 Stream created and running
----------------------------
```

5. Create a new persistent query where another condition is met, using LIKE. Results from this query are written to the pageviews_enriched_r8_r9 Kafka topic.

```
CREATE STREAM pageviews_female_like_89 \
  WITH (kafka_topic='pageviews_enriched_r8_r9') AS \
SELECT * FROM pageviews_female \
WHERE regionid LIKE '%_8' OR regionid LIKE '%_9';
```

Output:

Message

Stream created and running

----

6. Create a new persistent query that counts the pageviews for each region and gender combination in a tumbling window of 30 seconds when the count is greater than one. Results from this query are written to the PAGEVIEWS_REGIONS Kafka topic in the Avro format. KSQL will register the Avro schema with the configured Schema Registry when it writes the first message to the PAGEVIEWS_REGIONS topic.

```
CREATE TABLE pageviews_regions \
  WITH (VALUE_FORMAT='avro') AS \
SELECT gender, regionid , COUNT(*) AS numusers \
FROM pageviews_enriched \
  WINDOW TUMBLING (size 30 second) \
GROUP BY gender, regionid \
HAVING COUNT(*) > 1;
```

Output:

Message

Table created and running

----

7. **Optional:** View results from the above queries using SELECT.

```
SELECT gender, regionid, numusers FROM pageviews_regions LIMIT
5;
```

Output:

FEMALE | Region_6 | 3

```
FEMALE | Region_1 | 4

FEMALE | Region_9 | 6

MALE | Region_8 | 2

OTHER | Region_5 | 4

LIMIT reached

Query terminated

ksql>
```

8. **Optional:** Show all persistent queries.

```
SHOW QUERIES;
```

Output:

```
Query ID                    | Kafka Topic            | Query String
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
CSAS_PAGEVIEWS_FEMALE_1       | PAGEVIEWS_FEMALE       | CREATE
STREAM pageviews_female AS      SELECT * FROM pageviews_enriched
WHERE gender = 'FEMALE';
CTAS_PAGEVIEWS_REGIONS_3      | PAGEVIEWS_REGIONS      | CREATE
TABLE pageviews_regions       WITH (VALUE_FORMAT='avro') AS      SELECT
gender, regionid , COUNT(*) AS numusers      FROM pageviews_enriched
WINDOW TUMBLING (size 30 second)      GROUP BY gender, regionid
HAVING COUNT(*) > 1;
CSAS_PAGEVIEWS_FEMALE_LIKE_89_2 | PAGEVIEWS_FEMALE_LIKE_89 |
CREATE STREAM pageviews_female_like_89       WITH
(kafka_topic='pageviews_enriched_r8_r9') AS      SELECT * FROM
pageviews_female      WHERE regionid LIKE '%_8' OR regionid LIKE '%_9';
CSAS_PAGEVIEWS_ENRICHED_0      | PAGEVIEWS_ENRICHED      | CREATE
STREAM pageviews_enriched AS      SELECT users_original.userid AS userid,
pageid, regionid, gender      FROM pageviews_original      LEFT JOIN
users_original       ON pageviews_original.userid = users_original.userid;
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
For detailed information on a Query run: EXPLAIN <Query ID>;
```

9. **Optional:** Examine query run-time metrics and details. Observe that information including the target Kafka topic is available, as well as throughput figures for the messages being processed.

```
DESCRIBE EXTENDED PAGEVIEWS_REGIONS;
```

Output:

```
Name                 : PAGEVIEWS_REGIONS
Type                 : TABLE
Key field            : KSQL_INTERNAL_COL_0|+|KSQL_INTERNAL_COL_1
Key format           : STRING
Timestamp field      : Not set - using <ROWTIME>
Value format         : AVRO
Kafka topic          : PAGEVIEWS_REGIONS (partitions: 4, replication: 1)

Field    | Type
-------------------------------------
ROWTIME  | BIGINT           (system)
ROWKEY   | VARCHAR(STRING)  (system)
GENDER   | VARCHAR(STRING)
REGIONID | VARCHAR(STRING)
NUMUSERS | BIGINT
-------------------------------------

Queries that write into this TABLE
-----------------------------------
CTAS_PAGEVIEWS_REGIONS_3 : CREATE TABLE pageviews_regions       WITH (value_format='avro') AS       SELECT gender, regionid , COUNT(*) AS numusers FROM pageviews_enriched       WINDOW TUMBLING (size 30 second)       GROUP BY gender, regionid       HAVING COUNT(*) > 1;

For query topology and execution plan please run: EXPLAIN <QueryId>
```

```
Local runtime statistics
------------------------
messages-per-sec:      3.06   total-messages:      1827     last-message: 7/19/18 4:17:55 PM UTC
failed-messages:         0 failed-messages-per-sec:         0      last-failed:       n/a
```

(Statistics of the local KSQL server interaction with the Kafka topic PAGEVIEWS_REGIONS)

```
ksql>
```

6 KSQL REST API TESTING

1. GET the status of KSQL Server

```
curl -sX GET "http://172.18.0.25:8088/info";
```

Output:

```
{"KsqlServerInfo":{"version":"5.0.1","kafkaClusterId":"sNmrikrMSGGxZO3elbvQig","ksqlServiceId":"default_"}}
```

2. Show all streams and tables using rest API

```
curl -X "POST" "http://172.18.0.25:8088/ksql" \
-H "Content-Type: application/vnd.ksql.v1+json; charset=utf-8" \
-d $'{
"ksql": "LIST STREAMS;",
"streamsProperties": {}
}'
```

Output:

```
[
{"@type":"streams","statementText":"LIST STREAMS;","streams":[
{"type":"STREAM","name":"PAGEVIEWS","topic":"_confluent-metrics","format":"DELIMITED"},
{"type":"STREAM","name":"TEST_STREAM","topic":"_confluent-metrics","format":"DELIMITED"}]}
]
```

7 TESTING CONFLUENT KAFKA BROKER AND KAFKA ZOOKEEPRE SERVICES

1. Create a Kafka Topic

   To create a topic called *Test* in running cluster, use the following command:

   ```
   /bin/kafka-topics --create --zookeeper
   172.18.0.26:2181,172.18.0.19:2181,172.18.0.23:2181 --
   replication-factor 1 --partitions 1 --topic Test
   ```

   Output:

   Created topic "Test".

   Note: kafka_zookeper service will be used to create Kafka Topic

2. List the Kafka topic

   To list the topics in running cluster, use the following command:

   ```
   /bin/kafka-topics --list --zookeeper
   172.18.0.26:2181,172.18.0.19:2181,172.18.0.23:2181
   ```

   Output:

   TEST-STREAMING

   Test

   __confluent.support.metrics

   __consumer_offsets

   _confluent-command

   _confluent-ksql-default__command_topic

   _confluent-metrics

   _confluent-monitoring

   _schemas

   connect-configs

   connect-offsets

   connect-status

Note: This command returns the list with the names of all of the running topics in the cluster.

3. Produce the data in the Kafka topic

   To create a Kafka Producer, run the following:

```
/bin/kafka-console-producer --broker-list
172.18.0.21:9092,172.18.0.20:9092,172.18.0.27:9092 --topic Test
```

Now we can start sending the messages to the Kafka cluster from the console. The messages will be published to the Kafka Topic, "test".

Output:

>1, First Entry

>2, Second Entry

>3, Third Entry

>4, Fourth Entry

>5, Fifth Entry

Note: The output is just an example you can send message according to your Requirement.

4. Read data from a Kafka topic

   To read data from Kafka topic, run the following command:

```
/bin/kafka-console-consumer --bootstrap-server
172.18.0.20:9092,172.18.0.21:9092,172.18.0.27:9092 --topic Test
--from-beginning
```

Start typing messages in the producer. Consumer would get the messages via Kafka Topic

Output:

1, First Entry

2, Second Entry

```
3, Third Entry
4, Fourth Entry
5, Fifth Entry
Processed a total of 5 messages
```

8 TESTING SCHEMA REGISTRY

1. Registering a New Version of a Schema under the subject “Kafka-key”

```
curl -X POST -H "Content-Type:
application/vnd.schemaregistry.v1+json" \
  --data '{"schema": "{\"type\": \"string\"}"}' \
  http://<Schema_registry_IP_address>:8081/subjects/Kafka-key/versions
```

Output:

{"id":1}

2. Registering a New Version of a Schema under the subject "Kafka-value"

```
curl -X POST -H "Content-Type:
application/vnd.schemaregistry.v1+json" \
  --data '{"schema": "{\"type\": \"string\"}"}' \
   http://<Schema_registry_IP_address>:8081/subjects/Kafka-value/versions
```

Output:

{"id":1}

3. Listing all Subjects

```
curl -X GET http://<Schema_Registry_IP_address>:8081/subjects
```

Output:

["Kafka-value","Kafka-key"]

4. Fetching a Schema by Globally Unique ID

```
curl -X GET http://<Schema_Registry_IP_address>:8081/schemas/ids/1
```

Output:

{"schema":"\"string\""}

5. Listing all Schema Versions registered under the subject “Kafka-value”

```
curl -X GET http://<Schema_registry_IP_address>:8081/subjects/Kafka-value/versions
```

Output:

[1]

6. Fetch Version 1 of the Schema registered under subject “Kafka-value”

```
curl -X GET http://<Schema_Registry_IP_address>:8081/subjects/Kafka-value/versions/1
```

Output:

{"subject":"Kafka-value","version":1,"id":1,"schema":"\"string\""}

7. Checking if a Schema is registered under subject “Kafka-key”

```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
  --data '{"schema": "{\"type\": \"string\"}"}' \
  http://<Schema_Registry_IP_address>:8081/subjects/Kafka-key
```

Output:

{"subject":"Kafka-key","version":1,"id":1,"schema":"\"string\""}

8. Testing compatibility of a Schema with the latest schema under subject “Kafka-value”

```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json" \
  --data '{"schema": "{\"type\": \"string\"}"}' \
  http://<Schema_Registry_IP_address>:8081/compatibility/subjects/Kafka-
value/versions/latest
```

Output:

{"is_compatible":true}

9. Getting the Top level Config

```
curl -X GET http://<Schema_Registry_IP_address>:8081/config
```

Output:

{"compatibilityLevel":"BACKWARD"}

10. Updating Compatibility requirements globally

```
curl -X PUT -H "Content-Type:
application/vnd.schemaregistry.v1+json" \
  --data '{"compatibility": "NONE"}' \
  http://<Schema_Registry_IP_address>:8081/config
```

Output:

{"compatibility":"NONE"}

11. Deleting all Schema Versions registered under subject “Kafka-value”

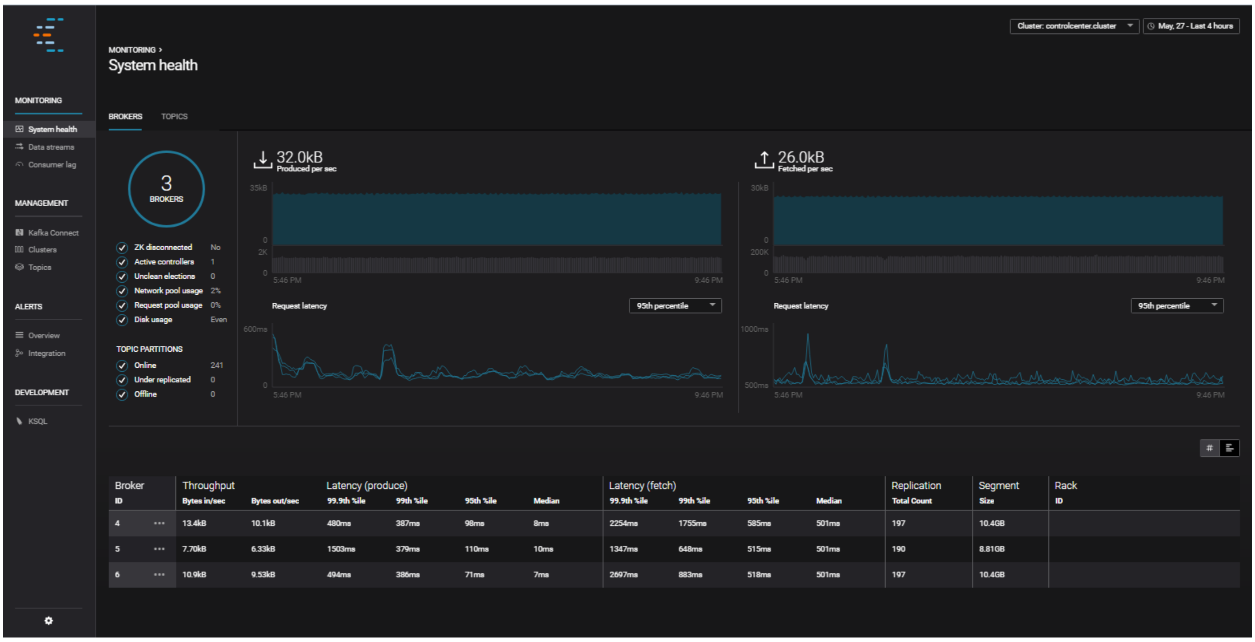
```
curl -X DELETE http://<Schema_registry_IP_address>:8081/subjects/Kafka-value
```

Output:

[1]

9 MONITOR AND MANAGE – CONTROL CENTER

Open your web browser and navigate to http://<IP_Address>:9021 (by default). As you will enter you will see the Confluent dashboard, as shown below:



3 BROKERS

✓ ZK disconnected	No
✓ Active controllers	1
✓ Unclean elections	0
✓ Network pool usage	2%
✓ Request pool usage	0%
✓ Disk usage	Even

TOPIC PARTITIONS

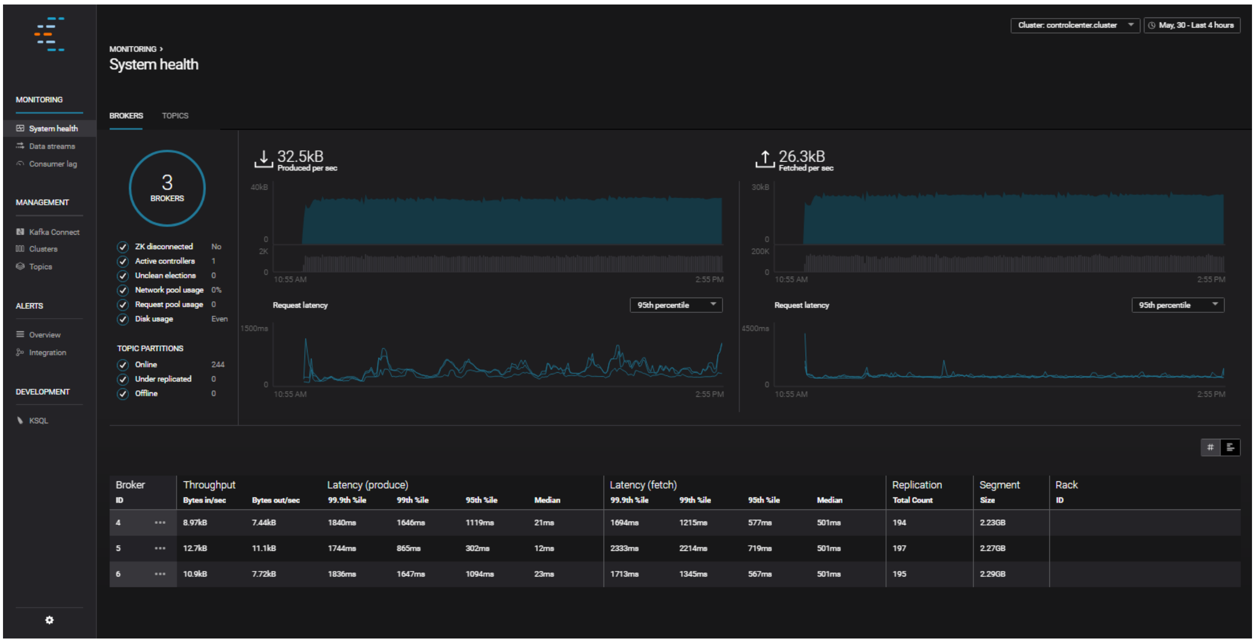
✓ Online	241
✓ Under replicated	0
✓ Offline	0

Broker		Throughput		Latency (produce)				Latency (fetch)				Replication	Segment	Rack
ID		Bytes in/sec	Bytes out/sec	99.9th %ile	99th %ile	95th %ile	Median	99.9th %ile	99th %ile	95th %ile	Median	Total Count	Size	ID
4	•••	13.4kB	10.1kB	480ms	387ms	98ms	8ms	2254ms	1755ms	585ms	501ms	197	10.4GB	
5	•••	7.70kB	6.33kB	1503ms	379ms	110ms	10ms	1347ms	648ms	515ms	501ms	190	8.81GB	
6	•••	10.9kB	9.53kB	494ms	386ms	71ms	7ms	2697ms	883ms	518ms	501ms	197	10.4GB	

9.1.1 System Health

From the left-hand menu, click on **System Health**. System Health page will appear and it will display the health of Broker and Topic.

Broker tab:



3 BROKERS

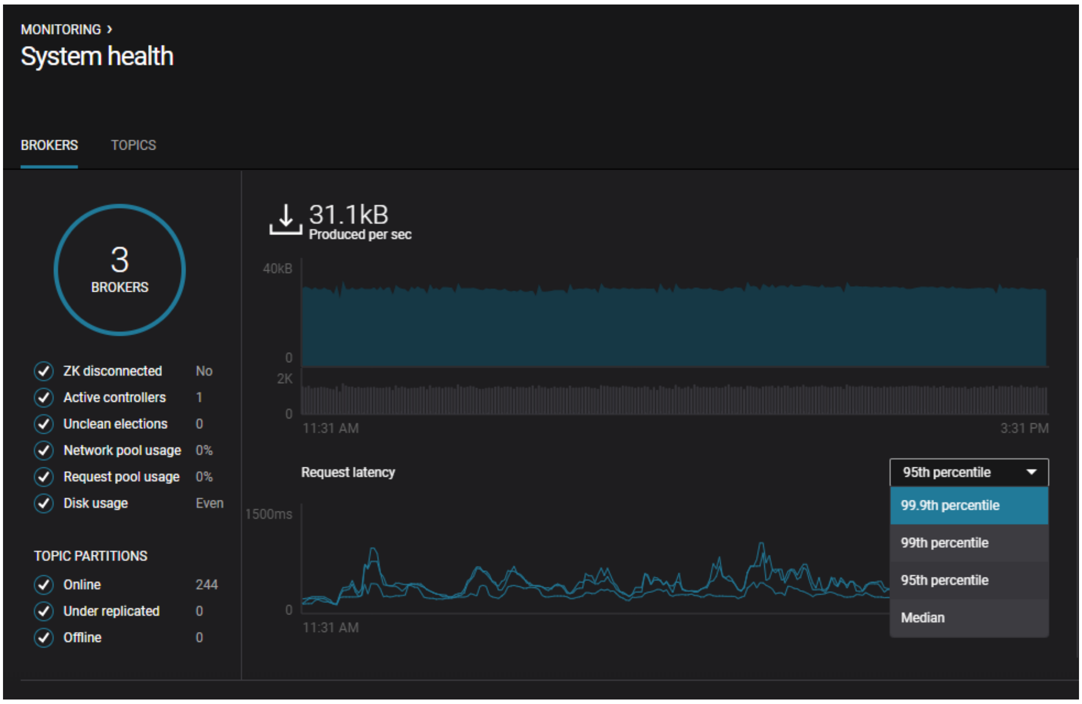
✓ ZK disconnected	No
✓ Active controllers	1
✓ Unclean elections	0
✓ Network pool usage	0%
✓ Request pool usage	0
✓ Disk usage	Even

TOPIC PARTITIONS

✓ Online	244
✓ Under replicated	0
✓ Offline	0

Broker		Throughput		Latency (produce)				Latency (fetch)				Replication	Segment	Rack
ID		Bytes in/sec	Bytes out/sec	99.9th %ile	99th %ile	95th %ile	Median	99.9th %ile	99th %ile	95th %ile	Median	Total Count	Size	ID
4	•••	8.97kB	7.44kB	1840ms	1646ms	1119ms	21ms	1694ms	1215ms	577ms	501ms	194	2.23GB	
5	•••	12.7kB	11.1kB	1744ms	865ms	302ms	12ms	2333ms	2214ms	719ms	501ms	197	2.27GB	
6	•••	10.9kB	7.72kB	1836ms	1647ms	1094ms	23ms	1713ms	1345ms	567ms	501ms	195	2.29GB	

- From the Request latency percentile drop-down list, select any percentile and view the change



MONITORING ›

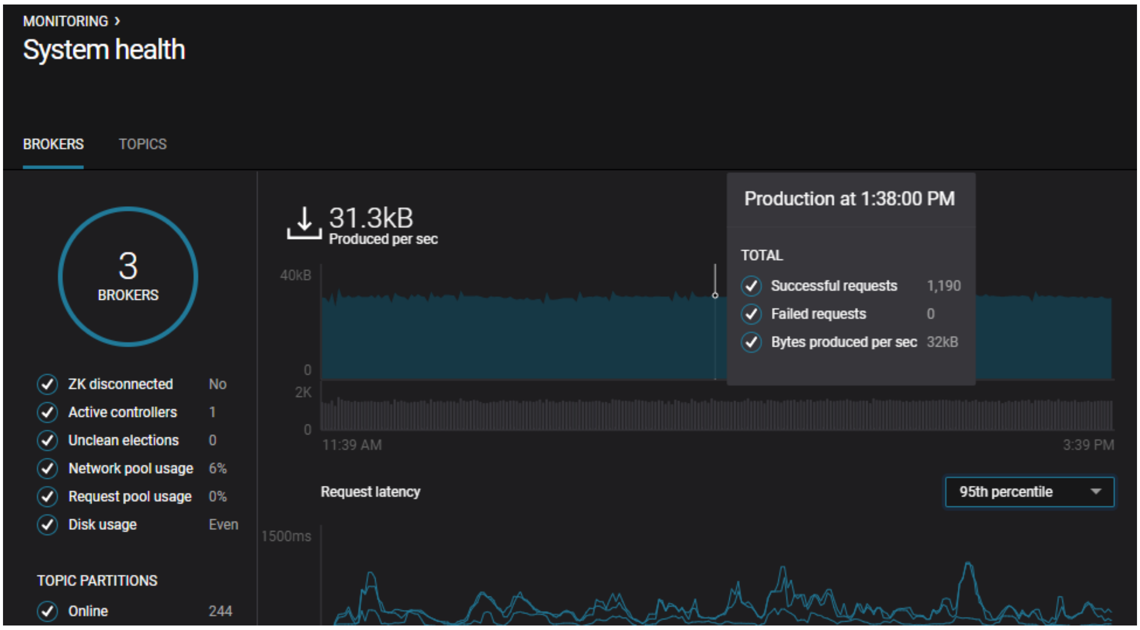
System health

✓ ZK disconnected	No
✓ Active controllers	1
✓ Unclean elections	0
✓ Network pool usage	0%
✓ Request pool usage	0%
✓ Disk usage	Even

TOPIC PARTITIONS

✓ Online	244
✓ Under replicated	0
✓ Offline	0

- Hover your mouse over the charts and you can view details like Successful requests, Failed requests and Bytes produced/fetched per sec



MONITORING ›

System health

Production at 1:38:00 PM

TOTAL

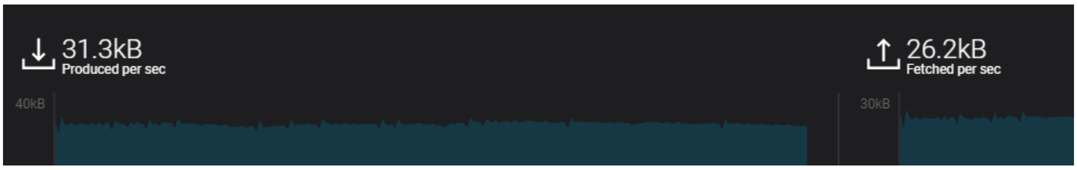
✓ Successful requests	1,190
✓ Failed requests	0
✓ Bytes produced per sec	32kB

✓ ZK disconnected	No
✓ Active controllers	1
✓ Unclean elections	0
✓ Network pool usage	6%
✓ Request pool usage	0%
✓ Disk usage	Even

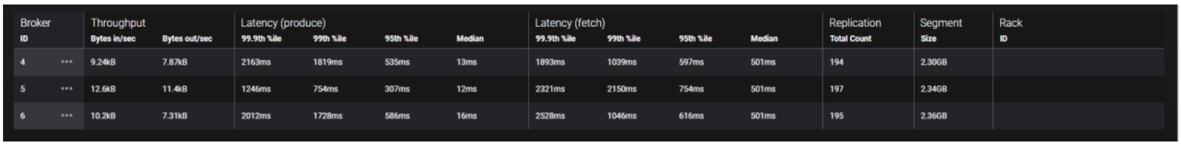
TOPIC PARTITIONS

✓ Online	244

- Summary of the Produce and Fetch requests are given like below

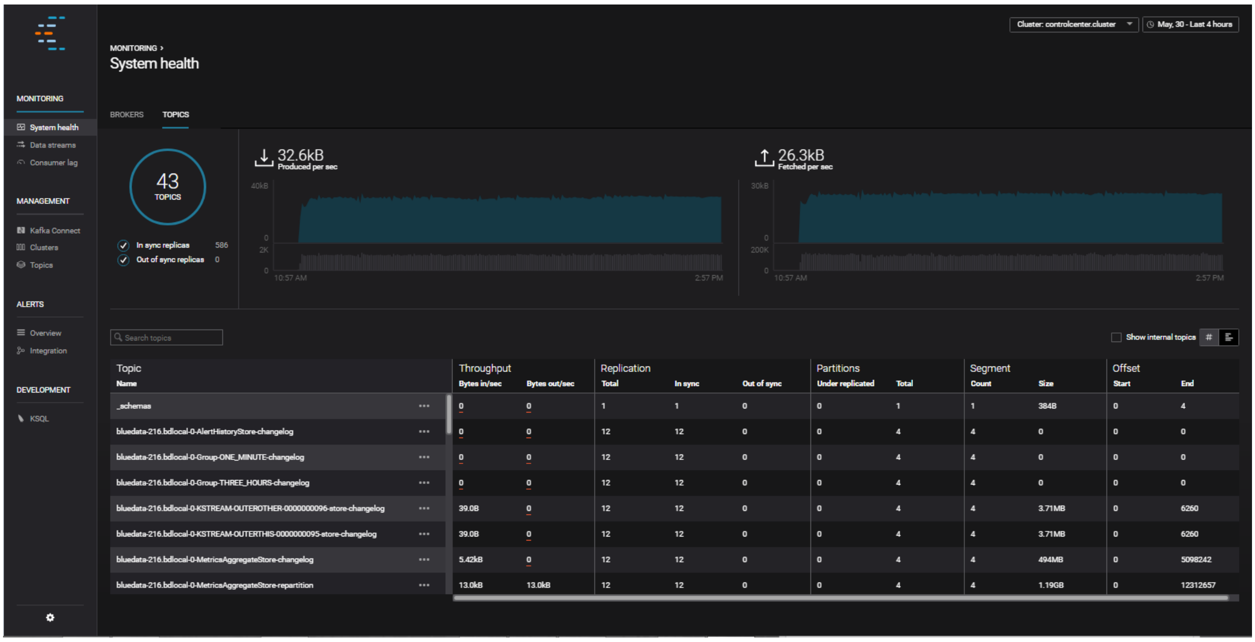


- At the bottom, you will see the Broker Metrics table



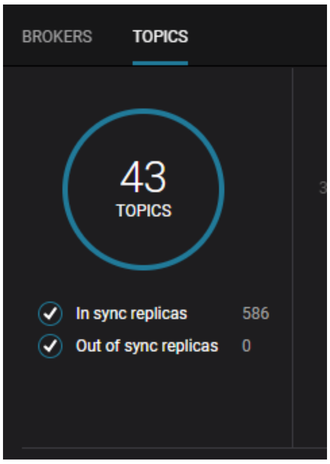
Broker		Throughput		Latency (produce)				Latency (fetch)				Replication	Segment	Rack
ID		Bytes in/sec	Bytes out/sec	99.9th %ile	99th %ile	95th %ile	Median	99.9th %ile	99th %ile	95th %ile	Median	Total Count	Size	ID
4	•••	9.24kB	7.87kB	2163ms	1819ms	535ms	13ms	1893ms	1039ms	597ms	501ms	194	2.30GB	
5	•••	12.6kB	11.4kB	1246ms	754ms	307ms	12ms	2321ms	2150ms	754ms	501ms	197	2.34GB	
6	•••	10.2kB	7.31kB	2012ms	1728ms	586ms	16ms	2528ms	1046ms	616ms	501ms	195	2.36GB	

Topic tab:

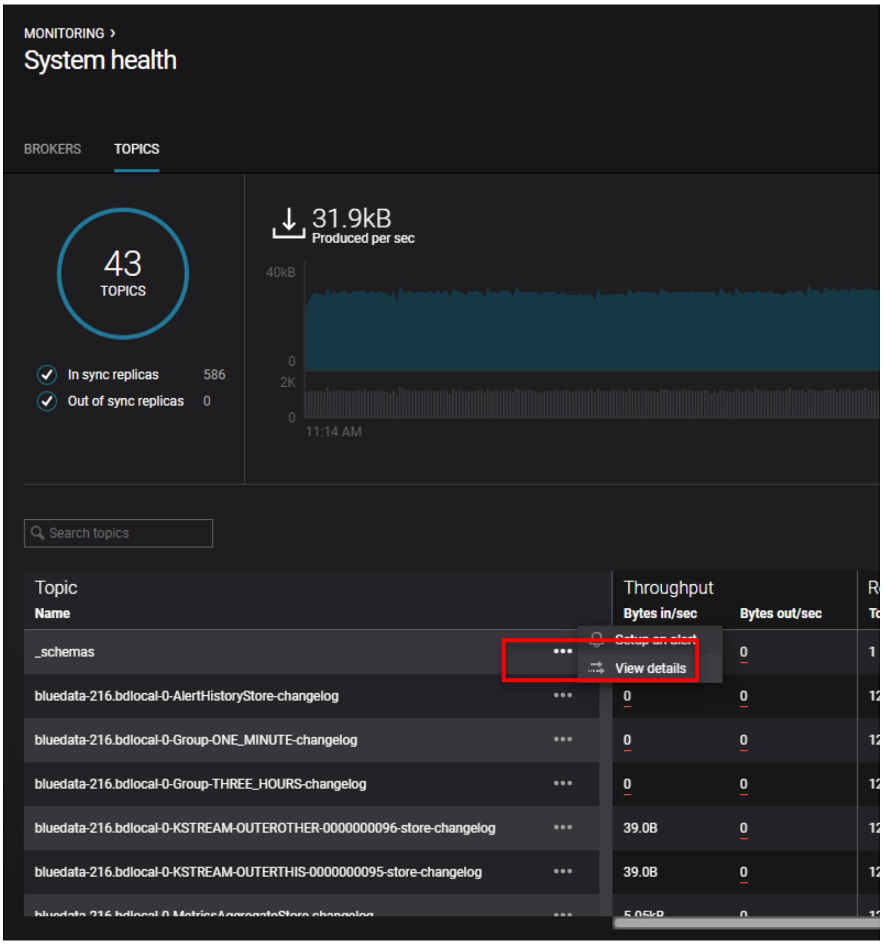


Topic		Throughput		Replication			Partitions		Segment		Offset	
Name		Bytes in/sec	Bytes out/sec	Total	In sync	Out of sync	Under replicated	Total	Count	Size	Start	End
_schemas	•••	0	0	1	1	0	0	1	1	384B	0	4
bluedata-216.bdlocal-0-AlertHistoryStore-changelog	•••	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog	•••	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-Group-THREE_HOURS-changelog	•••	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-KSTREAM-OUTEROTHER-0000000096-store-changelog	•••	39.0B	0	12	12	0	0	4	4	3.71MB	0	6260
bluedata-216.bdlocal-0-KSTREAM-OUTERTHIS-0000000095-store-changelog	•••	39.0B	0	12	12	0	0	4	4	3.71MB	0	6260
bluedata-216.bdlocal-0-MetricsAggregateStore-changelog	•••	5.42kB	0	12	12	0	0	4	4	494MB	0	5098242
bluedata-216.bdlocal-0-MetricsAggregateStore-repartition	•••	13.0kB	13.0kB	12	12	0	0	4	4	1.19GB	0	12312657

- Topic Aggregate Metrics shows the total no. of topics, In sync replicas and Out of sync replicas

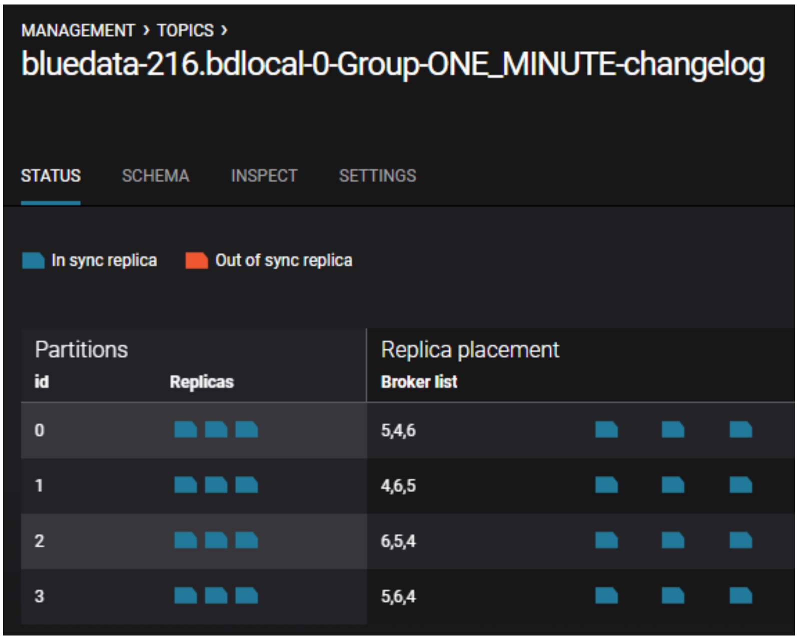


- Click on the ellipsis of any Topic, under Topics section and click on **View details**



Topic		Throughput		R
Name		Bytes in/sec	Bytes out/sec	To
_schemas	•••		0	1
bluedata-216.bdlocal-0-AlertHistoryStore-changelog	•••	0	0	12
bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog	•••	0	0	12
bluedata-216.bdlocal-0-Group-THREE_HOURS-changelog	•••	0	0	12
bluedata-216.bdlocal-0-KSTREAM-OUTEROTHER-0000000096-store-changelog	•••	39.0B	0	12
bluedata-216.bdlocal-0-KSTREAM-OUTERTHIS-0000000095-store-changelog	•••	39.0B	0	12
bluedata-216.bdlocal-0-MetricsAggregateStore-changelog	•••	5.05kB	0	12

- You will get screen like below, showing the In sync replica and Out of sync replica (if any)



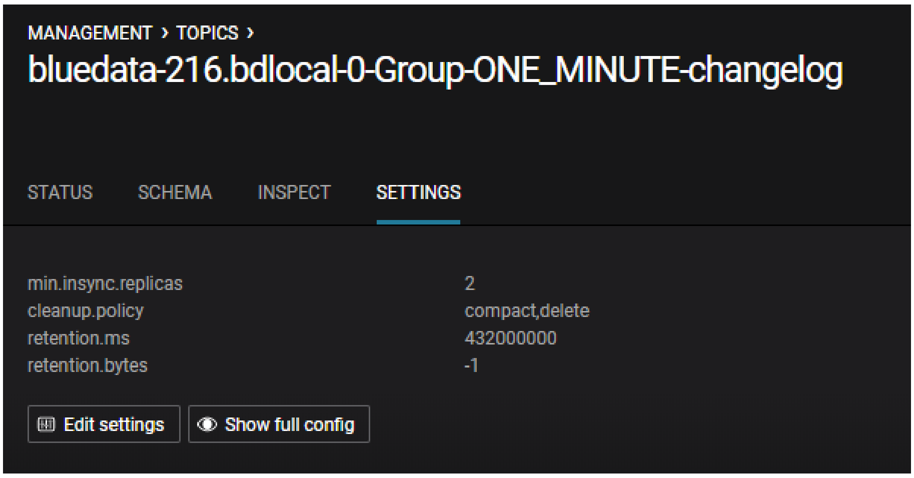
MANAGEMENT › TOPICS ›

bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog

🟦 In sync replica 🟥 Out of sync replica

Partitions		Replica placement			
id	**Replicas**	**Broker list**			
0	🟦🟦🟦	5,4,6	🟦	🟦	🟦
1	🟦🟦🟦	4,6,5	🟦	🟦	🟦
2	🟦🟦🟦	6,5,4	🟦	🟦	🟦
3	🟦🟦🟦	5,6,4	🟦	🟦	🟦

- Click on **SETTINGS**, to **Edit settings** or **Show full config** button, as per requirement

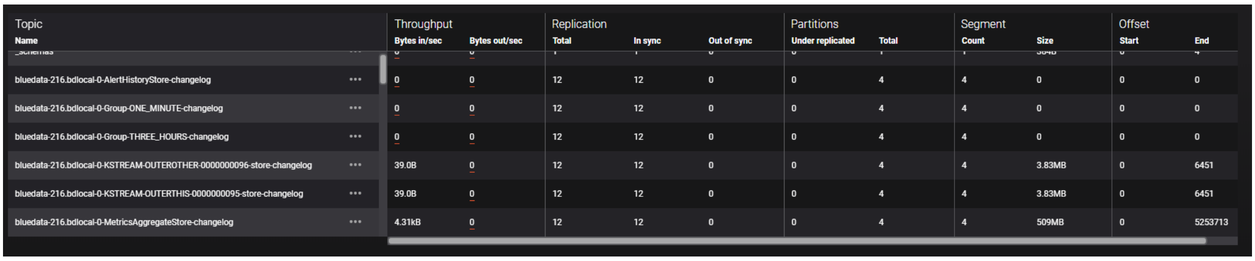


MANAGEMENT › TOPICS ›

bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog

min.insync.replicas	2
cleanup.policy	compact,delete
retention.ms	432000000
retention.bytes	-1

- At the bottom, you will see Topic Metric table

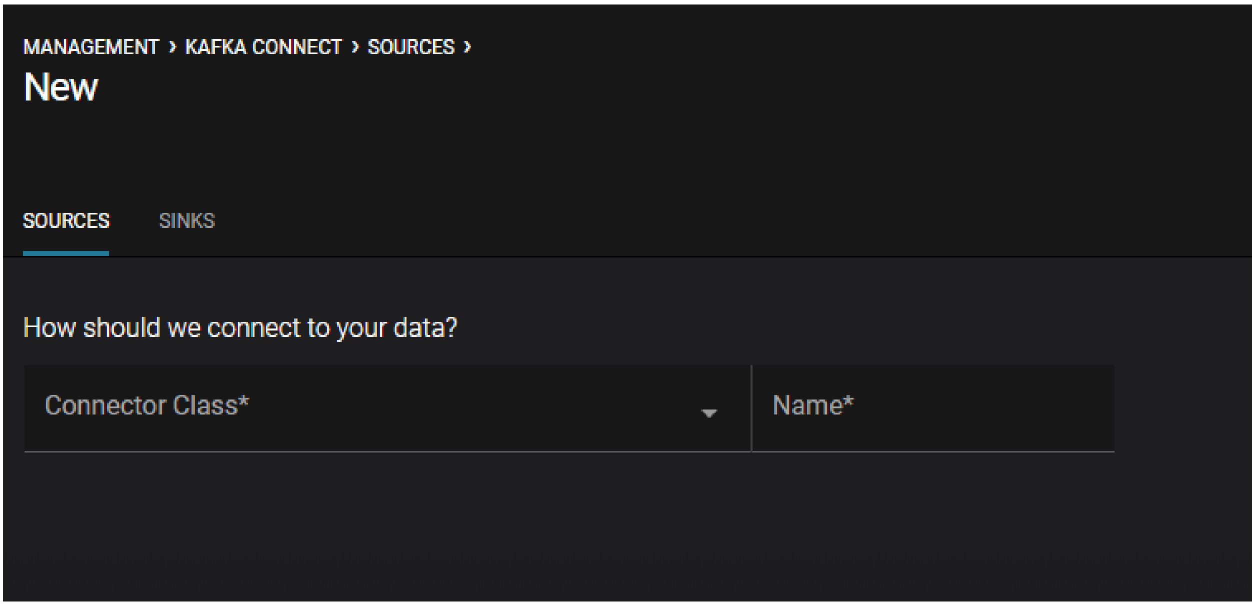


Topic	Throughput		Replication			Partitions		Segment		Offset	
Name	**Bytes in/sec**	**Bytes out/sec**	**Total**	**In sync**	**Out of sync**	**Under replicated**	**Total**	**Count**	**Size**	**Start**	**End**
_schemas	0	0	1	1	0	0	1	1	384B	0	4
bluedata-216.bdlocal-0-AlertHistoryStore-changelog	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-Group-THREE_HOURS-changelog	0	0	12	12	0	0	4	4	0	0	0
bluedata-216.bdlocal-0-KSTREAM-OUTEROTHER-0000000096-store-changelog	39.0B	0	12	12	0	0	4	4	3.83MB	0	6451
bluedata-216.bdlocal-0-KSTREAM-OUTERTHIS-0000000095-store-changelog	39.0B	0	12	12	0	0	4	4	3.83MB	0	6451
bluedata-216.bdlocal-0-MetricsAggregateStore-changelog	4.31kB	0	12	12	0	0	4	4	509MB	0	5253713

- Likewise explore other options to get more information

9.1.2 Kafka Connect

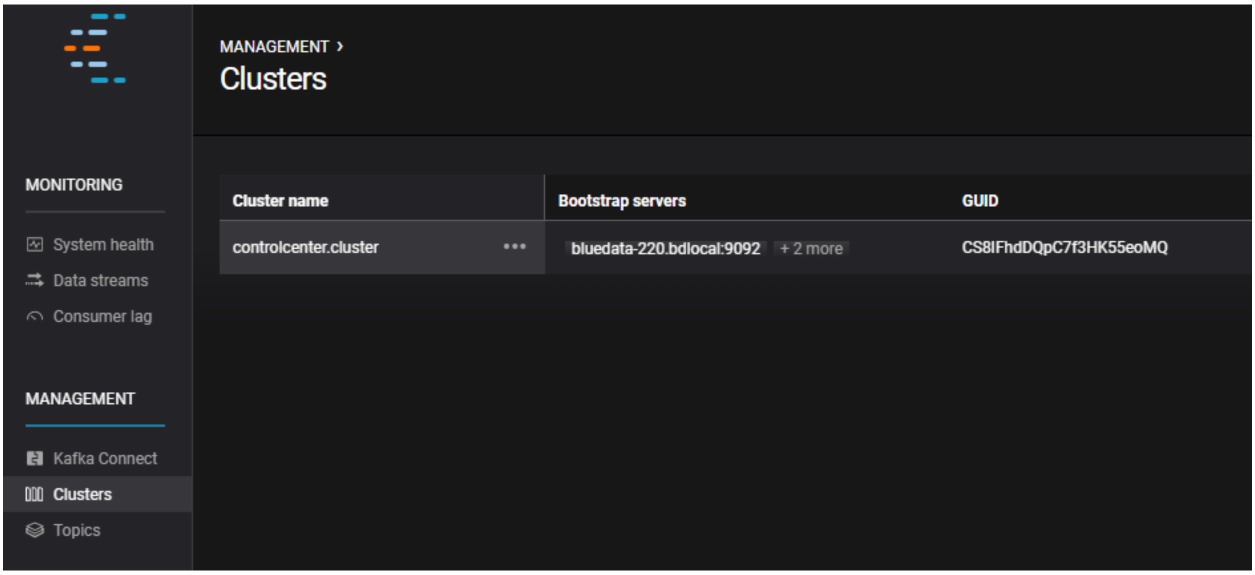
- From the left-hand menu, under **MANAGEMENT** section, click on **Kafka Connect**
- To create a source, provide the value of following parameters and click on **Submit**



MANAGEMENT › KAFKA CONNECT › SOURCES ›
New
SOURCES
SINKS
How should we connect to your data?
Connector Class*
Name*

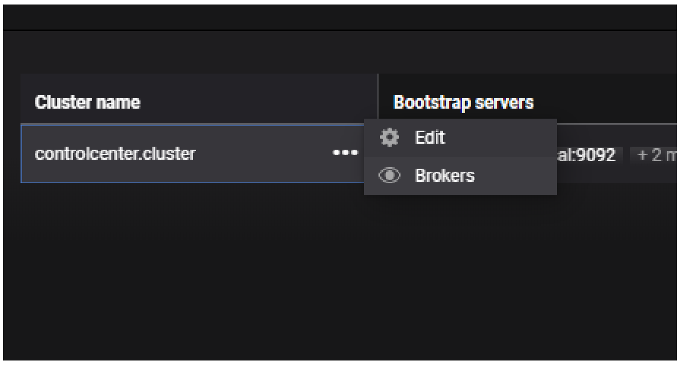
9.1.3 Clusters

- From the left-hand menu, under **MANAGEMENT** section, click on **Clusters**

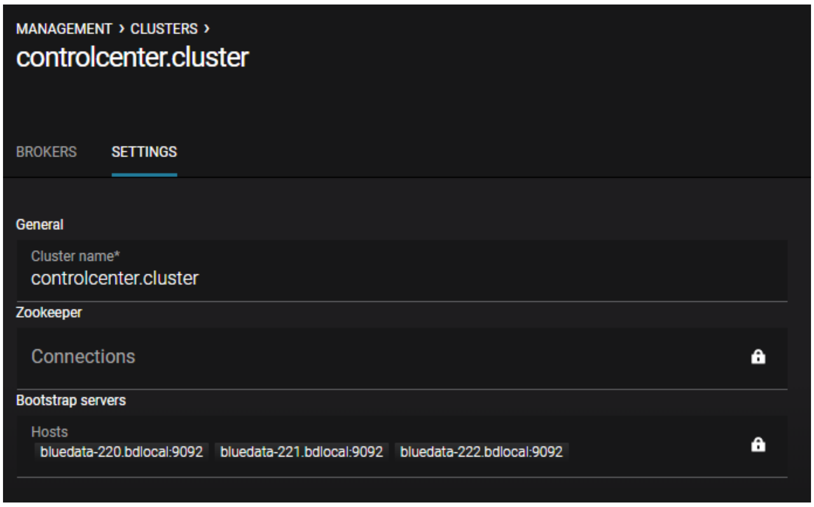


Cluster name	Bootstrap servers	GUID
controlcenter.cluster •••	bluedata-220.bdlocal:9092 + 2 more	CS8IFhdDQpC7f3HK55eoMQ

- Click on ellipsis of the cluster name to **Edit** or **Brokers** button as per requirement



- The Edit window looks like this below



General

Cluster name*
controlcenter.cluster

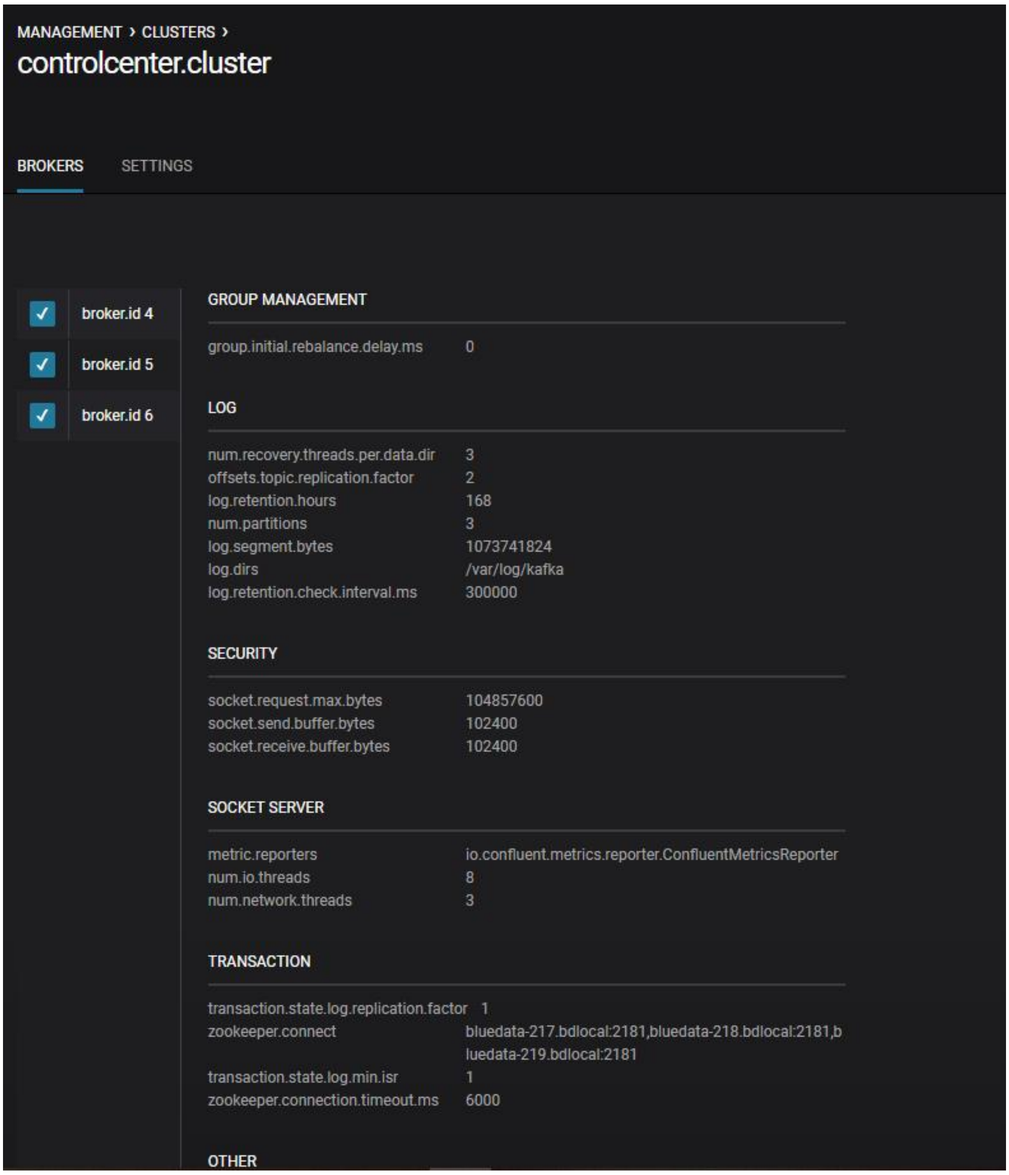
Zookeeper

Connections

Bootstrap servers

Hosts
bluedata-220.bdlocal:9092 bluedata-221.bdlocal:9092 bluedata-222.bdlocal:9092

- The Broker information is listed as below, it can also be downloaded



controlcenter.cluster

☑ broker.id 4
☑ broker.id 5
☑ broker.id 6

GROUP MANAGEMENT

group.initial.rebalance.delay.ms	0

LOG

num.recovery.threads.per.data.dir	3
offsets.topic.replication.factor	2
log.retention.hours	168
num.partitions	3
log.segment.bytes	1073741824
log.dirs	/var/log/kafka
log.retention.check.interval.ms	300000

SECURITY

socket.request.max.bytes	104857600
socket.send.buffer.bytes	102400
socket.receive.buffer.bytes	102400

SOCKET SERVER

metric.reporters	io.confluent.metrics.reporter.ConfluentMetricsReporter
num.io.threads	8
num.network.threads	3

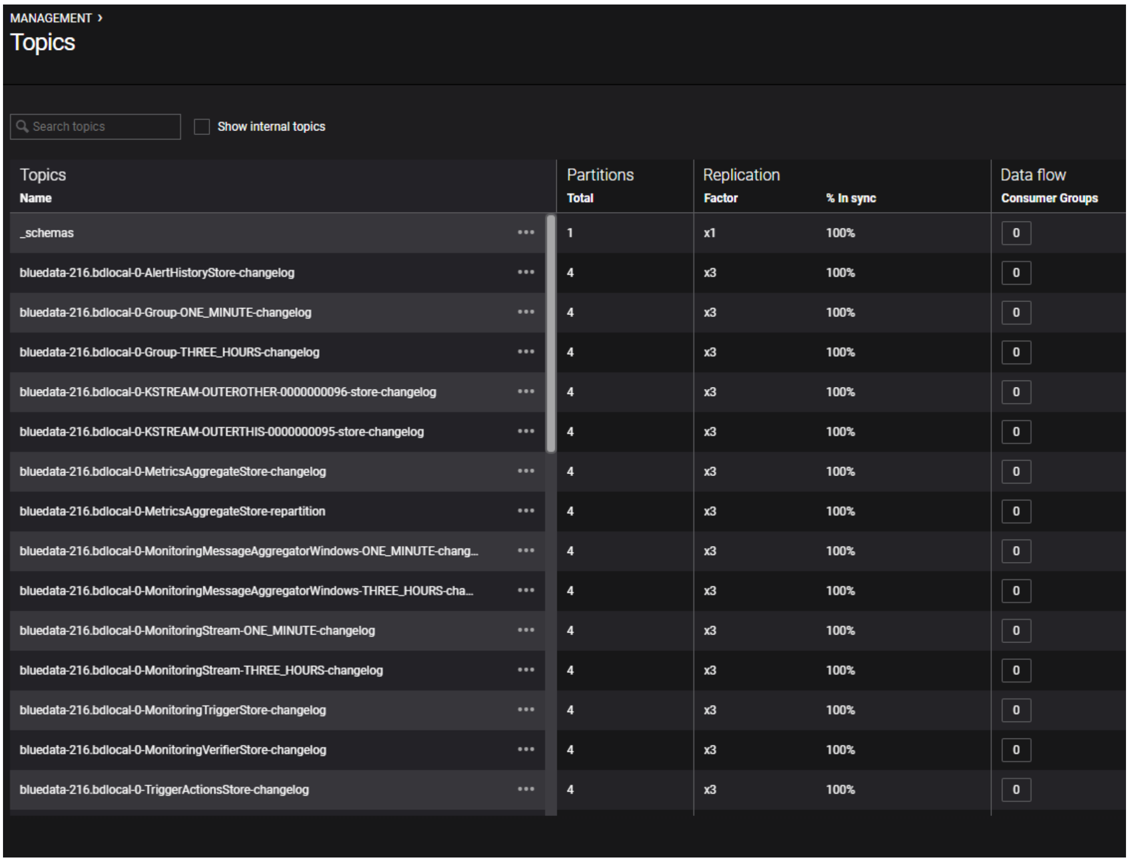
TRANSACTION

transaction.state.log.replication.factor	1
zookeeper.connect	bluedata-217.bdlocal:2181,bluedata-218.bdlocal:2181,bluedata-219.bdlocal:2181
transaction.state.log.min.isr	1
zookeeper.connection.timeout.ms	6000

OTHER

9.1.4 Topics

- From the left-hand menu, under **MANAGEMENT** section, click on **Topics**



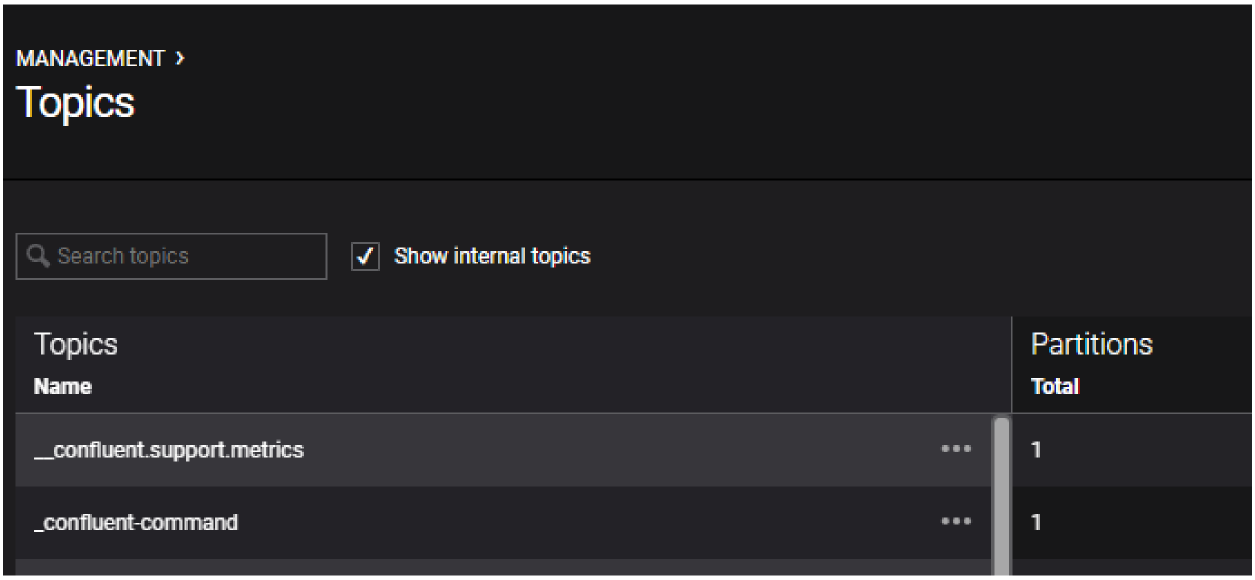
MANAGEMENT ›

Topics

□ Show internal topics

Topics		Partitions	Replication		Data flow
Name		Total	Factor	% In sync	Consumer Groups
_schemas	•••	1	x1	100%	0
bluedata-216.bdlocal-0-AlertHistoryStore-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-Group-THREE_HOURS-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-KSTREAM-OUTEROTHER-0000000096-store-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-KSTREAM-OUTERTHIS-0000000095-store-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MetricsAggregateStore-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MetricsAggregateStore-repartition	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringMessageAggregatorWindows-ONE_MINUTE-chang...	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringMessageAggregatorWindows-THREE_HOURS-cha...	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringStream-ONE_MINUTE-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringStream-THREE_HOURS-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringTriggerStore-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-MonitoringVerifierStore-changelog	•••	4	x3	100%	0
bluedata-216.bdlocal-0-TriggerActionsStore-changelog	•••	4	x3	100%	0

- Check **Show internal topics**, to view all the internal topics



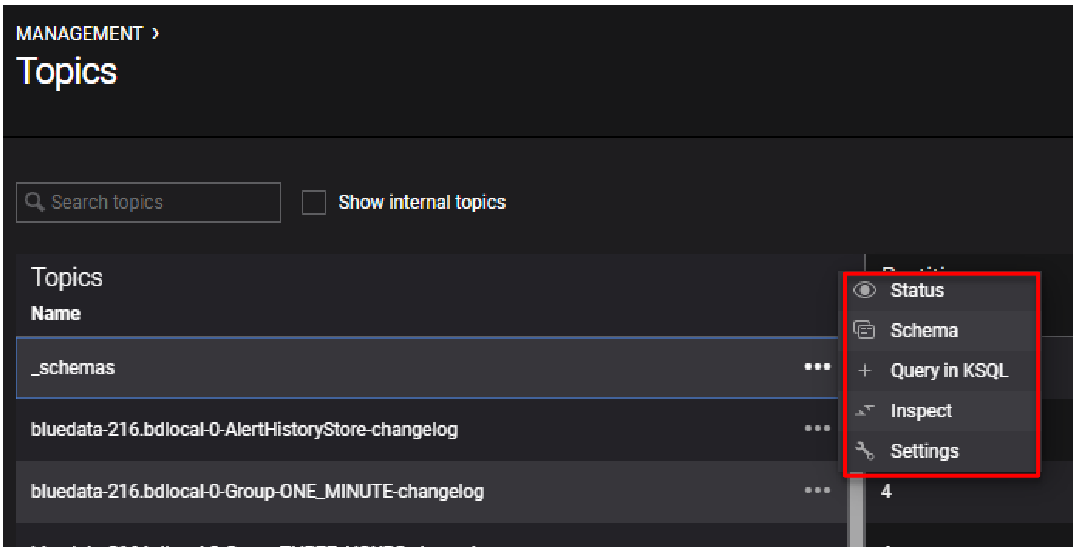
MANAGEMENT ›

Topics

☑ Show internal topics

Topics		Partitions
Name		Total
__confluent.support.metrics	•••	1
_confluent-command	•••	1

- Click on the topics ellipsis to view the multiple options related to that topic



MANAGEMENT ›

Topics

Search topics

□ Show internal topics

Topics

Name

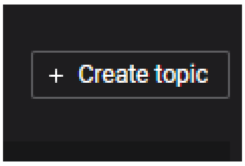
_schemas

bluedata-216.bdlocal-0-AlertHistoryStore-changelog

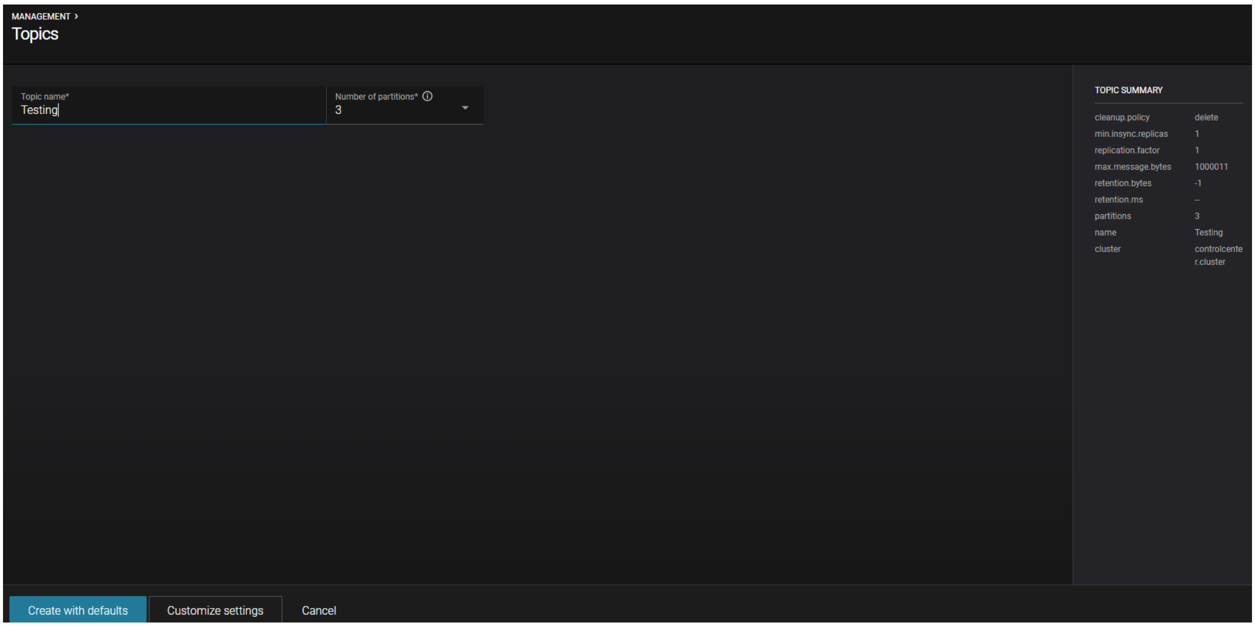
bluedata-216.bdlocal-0-Group-ONE_MINUTE-changelog

4

- To create a new Topic, click on **Create topic**, at right-hand side



- Provide **Topic name** and **Number of partition**s for Topic creation and click on **Create with defaults** or **Customize settings**, according to requirement



MANAGEMENT ›

Topics

Topic name*
Testing

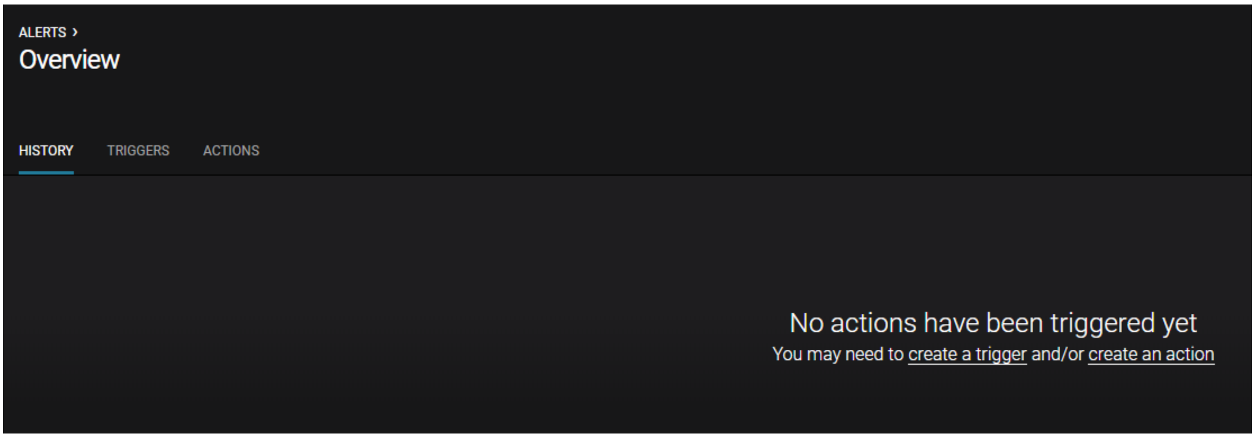
Number of partitions* ⓘ
3

TOPIC SUMMARY

cleanup.policy	delete
min.insync.replicas	1
replication.factor	1
max.message.bytes	1000011
retention.bytes	-1
retention.ms	–
partitions	3
name	Testing
cluster	controlcenter.cluster

9.1.5 Alerts

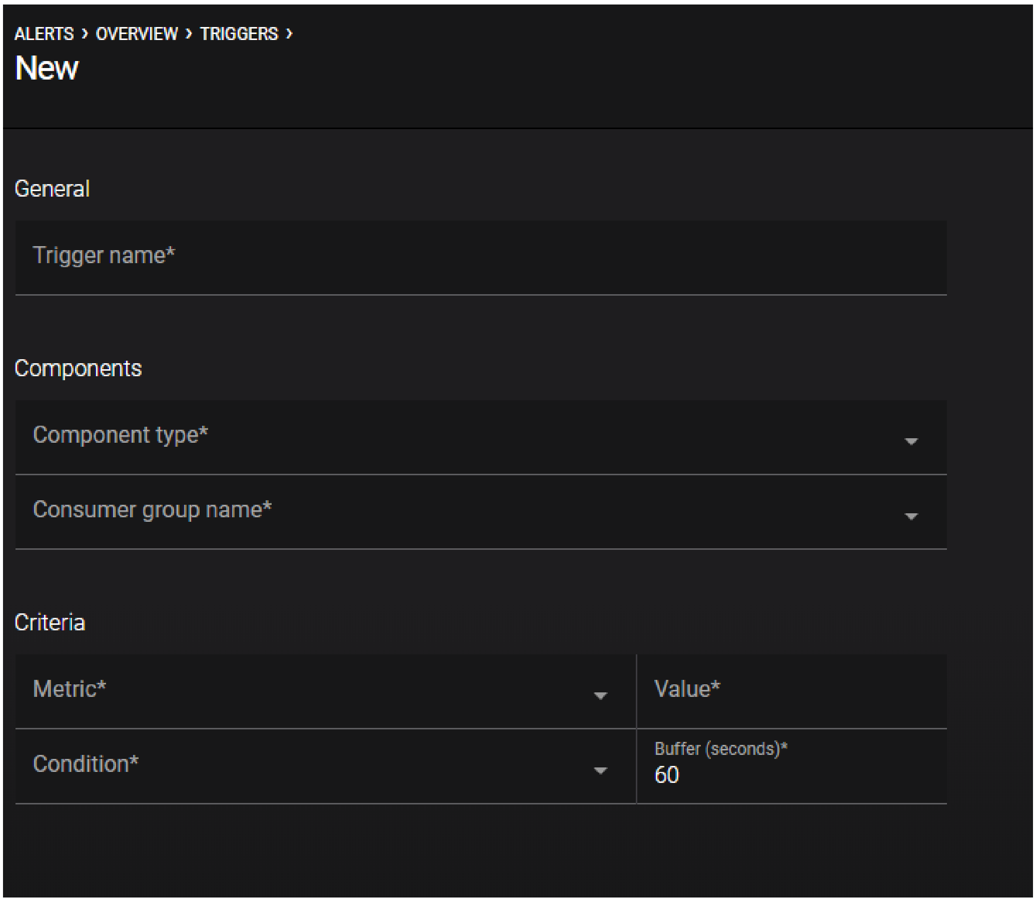
- From the left-hand menu, under **ALERTS** section, click on **Overview**



No actions have been triggered yet

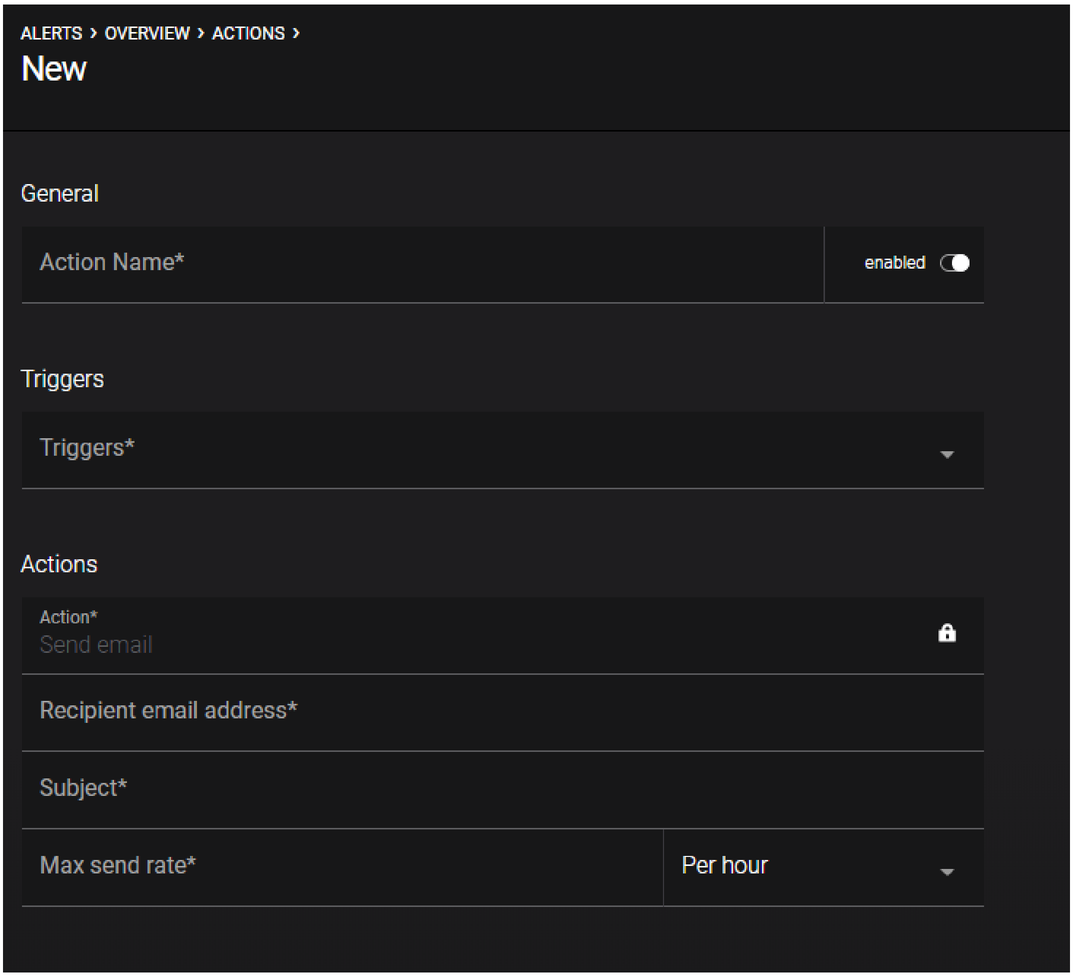
You may need to create a trigger and/or create an action

- It allows you to create a Tigger and/or an Action
- To create a trigger, provide the values of the following parameters and click on **Submit**



Buffer (seconds)* 60

- To create an Action, provide the value for the following parameters and click on **Submit**



ALERTS › OVERVIEW › ACTIONS ›

New

General

Action Name*

enabled

Triggers

Triggers*

Actions

Action*
Send email

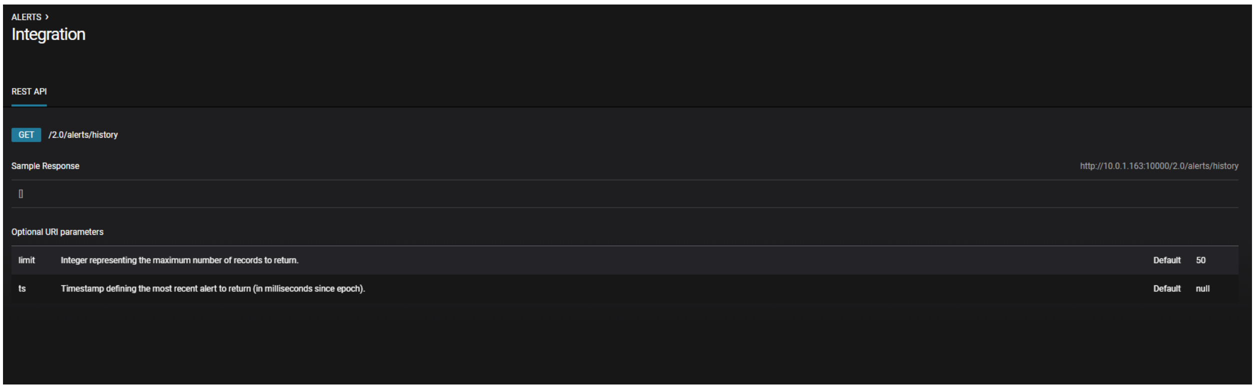
Recipient email address*

Subject*

Max send rate*

Per hour

- Click on **Integration**, under the ALERTS section, it provides REST API endpoints



ALERTS ›

Integration

REST API

GET /2.0/alerts/history

Sample Response

http://10.0.1.163:10000/2.0/alerts/history

```
[]
```

Optional URI parameters

limit	Integer representing the maximum number of records to return.	Default	50
ts	Timestamp defining the most recent alert to return (in milliseconds since epoch).	Default	null

10 TESTING KAFKA CONNECT

1. Get a list of active connectors (Initially empty)

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/connectors
```

Output:

```
[]
```

2. Create a new connector
   a. HDFS Sink Connector

```
curl -H "Content-Type: application/json" \
--request POST \
--data '{
    "name": "hdfs-sink-connector",
    "config": {
        "connector.class":
"io.confluent.connect.hdfs.HdfsSinkConnector",
        "tasks.max": "10",
        "topics": "test-topic",
        "hdfs.url": "hdfs://<IP_Address>:9000",
        "hadoop.conf.dir": "/opt/hadoop/conf",
        "hadoop.home": "/opt/hadoop",
        "flush.size": "100",
        "rotate.interval.ms": "1000"
    }
}'  \
http://<Kafka_Connect_IP_Address>:8083/connectors
```

Output:

```
{"name":"hdfs-sink-connector","config":{"connector.class":"io.confluent.connect.hdfs.HdfsS
inkConnector","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","hadoop.conf.dir":"/opt/hadoop/conf","hadoop.home":"/opt/hadoop","flush.size":"100","rotate.interval.ms":"1000","name":"hdfs-sink-connector"},"tasks":[{"connector":"hdfs-sink-connector","task":0},{"connector":"hdfs-sink-connector","task":1},{"connector":"hdfs-sink-connector","task":2},{"connector":"hdfs-sink-connector","task":3},{"connector":"hdfs-sink-connector","task":4},{"connector":"hdfs-sink-connector","task":5},{"connector":"hdfs-sink-connector","task":6},{"connector":"hdfs-sink-connector","task":7},{"connector":"hdfs-sink-connector","task":8},{"connector":"hdfs-sink-connector","task":9}],"type":"sink"}
```

b. Local File Sink

```
curl -H "Content-Type: application/json" \
--request POST \
--data '{
    "name": "local-file-sink",
    "config": {
        "connector.class": "FileStreamSinkConnector",
        "tasks.max": "2",
        "topics": "connect-test",
        "hdfs.url": "hdfs://<IP_Address>:9000",
        "hadoop.conf.dir": "/opt/hadoop/conf",
        "hadoop.home": "/opt/hadoop",
        "flush.size": "100",
        "rotate.interval.ms": "1000"
    }
}'   \
http://<Kafka_Connect_IP_Address>:8083/connectors
```

Output:

```
{"name":"local-file-sink","config":{"connector.class":"FileStreamSinkConnector","tasks.max":"2","topics":"connect-test","hdfs.url":"hdfs://10.39.250.16:9000","hadoop.conf.dir":"/opt/hadoop/conf","hadoop.home":"/opt/hadoop","flush.size":"100","rotate.interval.ms":"1000","name":"local-file-sink"},"tasks":[],"type":"sink"}
```

3. Verify the connector is added by executing

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/connectors
```

Output:

```
["local-file-sink","hdfs-sink-connector"]
```

Note: **hdfs-sink-connector** and **local-file-sink** are the connector name here, which will be used in later commands. Replace it with any other string name (connector name) according to your environment.

4. Get connector status

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/connectors/local-file-sink/status
```

Output:

```
{"name":"local-file-sink","connector":{"state":"RUNNING","worker_id":"bluedata-4424.bdlocal:8083"},"tasks":[{"id":0,"state":"RUNNING","worker_id":"bluedata-4424.bdlocal:8083"},{"id":1,"state":"RUNNING","worker_id":"bluedata-4424.bdlocal:8083"}],"type":"sink"}
```

5. Get worker's version

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/
```

Output:

```
{"version":"2.0.1-cp4","commit":"49da0fef3e389dc2","kafka_cluster_id":"CS8IFhdDQpC7f3HK55eoMQ"}
```

6. Get the list of connector plugins available on the worker

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/connector-plugins
```

Output:

```
[{"class":"io.confluent.connect.activemq.ActiveMQSourceConnector","type":"source","version":"5.0.3"},{"class":"io.confluent.connect.elasticsearch.ElasticsearchSinkConnector","type":"sink","version":"5.0.3"},{"class":"io.confluent.connect.hdfs.HdfsSinkConnector","type":"sink","version":"5.0.3"},{"class":"io.confluent.connect.hdfs.tools.SchemaSourceConnector","type":"source","version":"2.0.1-cp4"},{"class":"io.confluent.connect.ibm.mq.IbmMQSourceConnector","type":"source","version":"5.0.3"},{"class":"io.confluent.connect.jdbc.JdbcSinkConnector","type":"sink","version":"5.0.3"},{"class":"io.confluent.connect.jdbc.JdbcSourceConnector","type":"source","version":"5.0.3"},{"class":"io.confluent.connect.jms.JmsSourceConnector","type":"source","version":"5.0.3"},{"class":"io.confluent.connect.replicator.ReplicatorSourceConnector","type":"source","version":"5.0.3"},{"class":"io.confluent.connect.s3.S3SinkConnector","type":"sink","version":"5.0.3"},{"class":"io.confluent.connect.storage.tools.SchemaSourceConnector","type":"source","version":"2.0.1-cp4"},{"class":"org.apache.kafka.connect.file.FileStreamSinkConnector","type":"sink","version":"2.0.1-
cp4"},{"class":"org.apache.kafka.connect.file.FileStreamSourceConnector","type":"source","version":"2.0.1-cp4"}]
```

7. Get tasks for a connector

```
curl -X GET http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector/tasks
```

Output:

```
[{"id":{"connector":"hdfs-sink-connector","task":0},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":1},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":2},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":3},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-
```

connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":4},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":5},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":6},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":7},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-connector","task":8},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}},{"id":{"connector":"hdfs-sink-

```
connector","task":9},"config":{"connector.class":"io.confluent.connect.hdfs.HdfsSinkConnector","task.class":"io.confluent.connect.hdfs.HdfsSinkTask","hadoop.conf.dir":"/opt/hadoop/conf","flush.size":"100","tasks.max":"10","topics":"test-topic","hdfs.url":"hdfs://10.39.250.64:9000","name":"hdfs-sink-connector","rotate.interval.ms":"1000","hadoop.home":"/opt/hadoop"}}]
```

8. Restart connector and its tasks

```
curl -X POST http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector/restart
```

Output:

<No response when successful>

9. Pause the connector and its tasks

```
curl -X PUT http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector/pause
```

Output:

<No response when successful>

10. Resume the connector and its tasks

```
curl -X PUT http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector/resume
```

Output:

<No response when successful>

11. Restart an individual task

```
curl -X POST http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector/tasks/0/restart
```

Output:

<No response when successful>

12. Delete a connector

```
curl -X DELETE http://<Kafka_Connect_IP_Address>:8083/connectors/hdfs-sink-connector
```

Output:

<No response when successful>

11 CONFIGURING CONFLUENT KAFKA WITH HDFS AND DTAP USING CONNECTORS

Here in this section, we will see how to receive message from a Kafka Topic and store it into HDFS and DTAP, using connector.

1. SSH into Kafka connect node

   Note: It is assumed that Cloudera Manager Repo is installed in the /etc/yum.repos.d/ directory and Hadoop-client is installed on Kafka connect node.

2. Create a Kafka Topic

   ```
   /bin/kafka-topics --create --zookeeper
   <Zookeeper_1_IP_Address>,<Zookeeper_2_IP_Address>,<Zookeeper_3_
   IP_Address>:2181 --replication-factor 1 --partitions 1 --topic
   blue
   ```

   Output:

   Created topic "blue".

3. Produce some data in the created Topic

   ```
   /bin/kafka-console-producer --broker-list
   <Broker_1_IP_Address>:9092,<Broker_2_IP_Address>:9092,<Broker_3
   _IP_Address>:9092 --topic blue
   ```

   Note: When prompted for data, enter some data for the Topic and return to command prompt, by pressing **Ctrl+Z**.

4. Consume the generated data

   ```
   /bin/kafka-console-consumer --bootstrap-server
   <Broker_1_IP_Address>:9092,<Broker_2_IP_Address>:9092,<Broker_3
   _IP_Address>:9092 --topic blue --from-beginning
   ```

   Output:

   kafka

   message

here

^CProcessed a total of 3 messages.

5. Open /**etc/schema-registry/connect-avro-standalone.properties** file (with sudo privilege), edit the following, save and exit the file

```
bootstrap.servers=<Broker_1_IP_Address>:9092,<Broker_2_IP_Address>:9092,<Broker_3_IP_Address>:9092
# The converters specify the format of data in Kafka and how to translate it into Connect data.
# Every Connect user will need to configure these based on the format they want their data in
# when loaded from or stored into Kafka
key.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schema.registry.url=http://<Schema_Registry_IP_Address>:8081
value.converter=org.apache.kafka.connect.storage.StringConverter
value.converter.schema.registry.url=http://<Schema_Registry_IP_Address>:8081
```

Note: Update the highlighted parameters with the values according to your Confluent Kafka cluster.

6. Navigate to **/etc/kafka-connect-hdfs/quickstart-hdfs.properties** (with sudo privelege), update the HDFS URL and Topic details, save and exit the file

```
topics=blue
hdfs.url=hdfs://<CDH_Controller_IP_Address>:8020
```

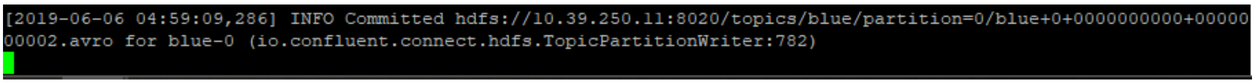
Note: Update the highlighted parameters.

7. Run the HDFS Connector

```
sudo -u hdfs /bin/connect-standalone /etc/schema-
registry/connect-avro-standalone.properties /etc/kafka-connect-
hdfs/quickstart-hdfs.properties
```

Note: Kill any connector service, if running. Use ***ps aux | grep "Connect"*** to get the list of running connector service. Use ***sudo kill -9 process_id*** to kill the service.

Output:



```
[2019-06-06 04:59:09,286] INFO Committed hdfs://10.39.250.11:8020/topics/blue/partition=0/blue+0+0000000000+00000
00002.avro for blue-0 (io.confluent.connect.hdfs.TopicPartitionWriter:782)
```

Note: The snapshot shows the path where the data is stored in HDFS.

8. SSH into the HDFS cluster Controller node and look for the path

```
hdfs dfs -ls /topics/blue/partition=0
```

Output:

Found 1 items

-rw-r--r-- 3 hdfs supergroup 108 2019-06-06 05:00 /topics/blue/partition=0/blue+0+0000000000+0000000002.avro

Note: Here **blue+0+0000000000+0000000002.avro** file contains the data sent from Kafka Connect node.

9. Copy the file to **/tmp** directory

```
hdfs dfs -copyToLocal
/topics/blue/partition=0/blue+0+0000000000+0000000002.avro /tmp
```

10. Convert the generated .avro file to .json file

```
java -jar avro-tools-1.8.2.jar tojson
/tmp/blue+0+0000000000+0000000002.avro > kdata.json
```

Output:

log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).

log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Note: Ignore the Warning messages. Assuming, Avro tools is installed in the HDFS node.

11. View the content of the generated .json file

```
cat kdata.json
```

Output:

{"string":"kafka"}

{"string":"message"}

{"string":"here"}

12. From the Kafka Connect node, open the **/etc/hadoop/conf.empty/core-site.xml** file (with sudo privilege), add given content to it, save and exit the file

```
<configuration>
<property>
  <name>fs.s3a.access.key</name>
  <value></value>
</property>
<property>
  <name>fs.s3a.secret.key</name>
  <value></value>
</property>
<property>
  <name>fs.s3a.impl</name>
  <value>org.apache.hadoop.fs.s3a.S3AFileSystem</value>
```

```
</property>
<property>
    <name>fs.dtap.impl</name>
    <value>com.bluedata.hadoop.bdfs.Bdfs</value>
    <description>The FileSystem for BlueData dtap:
URIs.</description>
  </property>
</configuration>
```

13. Open **/etc/kafka-connect-hdfs/quickstart-hdfs.properties** file (with sudo privilege), update the following details for DTAP connection, save and exit the file

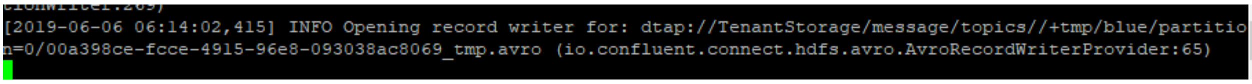
```
topics=blue
hdfs.url=dtap://TenantStorage/kafka
hadoop.conf.dir=/etc/hadoop/conf.empty/
```

Note: The hdfs.url should be the path from EPIC DTAP.

14. Run the executor

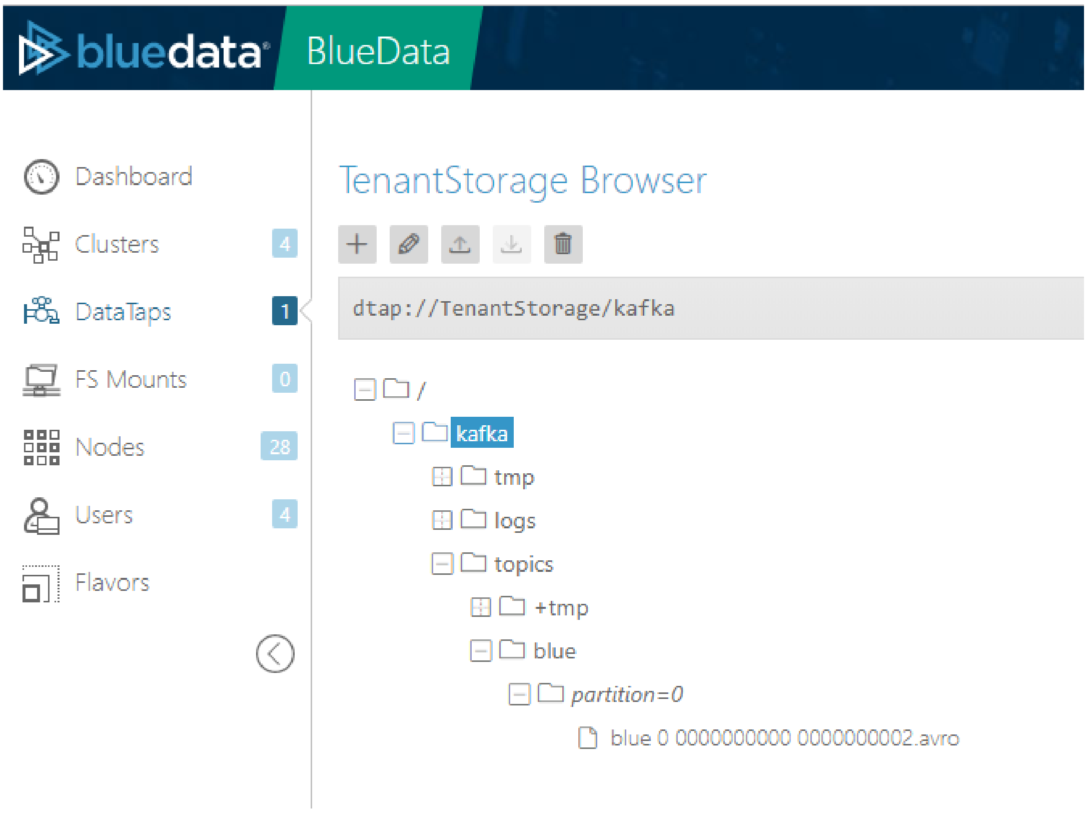
```
sudo -u hdfs /bin/connect-standalone /etc/schema-registry/connect-avro-standalone.properties /etc/kafka-connect-hdfs/quickstart-hdfs.properties
```

Output:



```
tionWriter:269)
[2019-06-06 06:14:02,415] INFO Opening record writer for: dtap://TenantStorage/message/topics//+tmp/blue/partitio
n=0/00a398ce-fcce-4915-96e8-093038ac8069_tmp.avro (io.confluent.connect.hdfs.avro.AvroRecordWriterProvider:65)
```

15. Verify in the EPIC UI



TenantStorage Browser

dtap://TenantStorage/kafka

```
/
  kafka
    tmp
    logs
    topics
      +tmp
      blue
        partition=0
          blue 0 0000000000 0000000002.avro
```