STREAMSETS DOCUMENTATION

NAME : P.SANJANA REDDY

EMP ID: MT4020

Table of contents

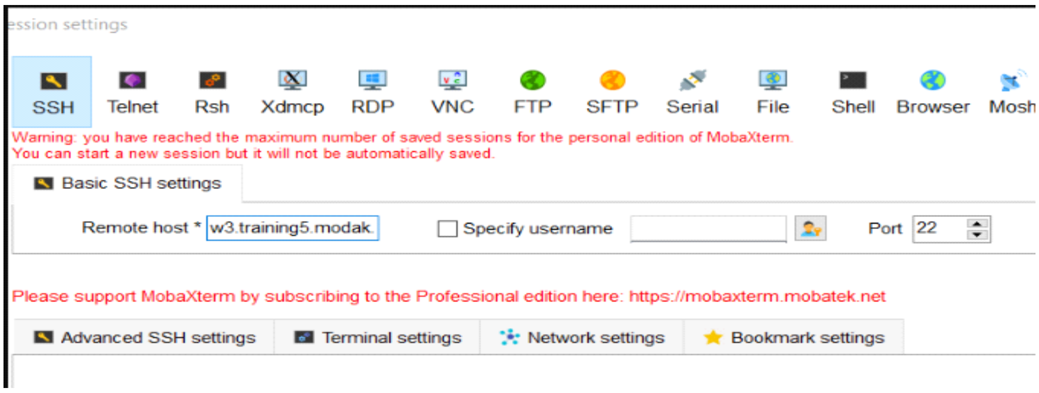
USER STORY – 12

- **Consume messages from StreamSets pipeline(origin- Kafka Multi-topic consumer) and write the data to Mongo DB by performing transformations.**
    - **Description:** As a developer, I need to consume data on a topic(with 3 partitions), transform the messages and store the transformed data in MongoDB.
        - Consume at least 1M records for better analysis.
    - **Acceptance Criteria:** Consume messages with varying thread counts and compare the throughputs observed.
- Data validated at the destination
- User story reviewed by the Technical Lead
- Azure board to be updated
- Approach and Implementation of user story should be documented

EXPLANATION:

Accessed the Apache kafka through CLI:

- **Step 1:** Give the Remote host as "w3.training5.modak.com".



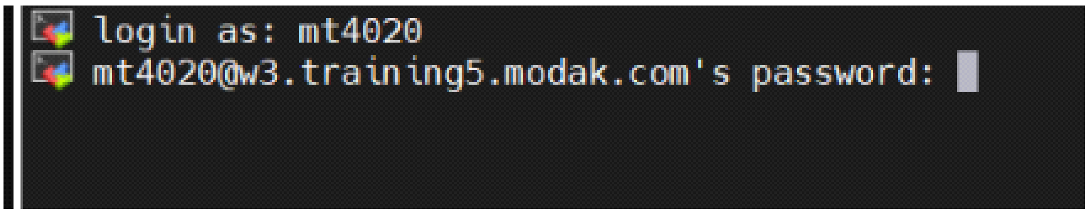
Warning: you have reached the maximum number of saved sessions for the personal edition of MobaXterm.
You can start a new session but it will not be automatically saved.

Basic SSH settings

Remote host * w3.training5.modak ☐ Specify username Port 22

Please support MobaXterm by subscribing to the Professional edition here: https://mobaxterm.mobatek.net

- **Step 2:** Give the "Username: mt4020" and "Password: mt4020&0222my*"

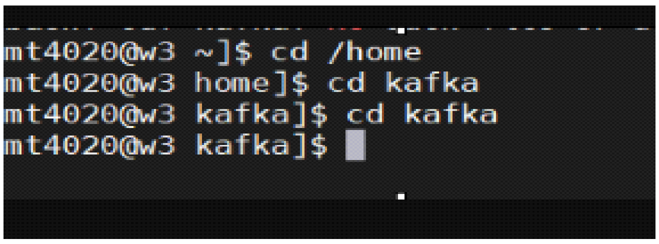


```
login as: mt4020
mt4020@w3.training5.modak.com's password:
```

- **step 3:** It will ask for new password, Set the new password.
- **step 4:** Then login with new password.

Go to kafka directory:

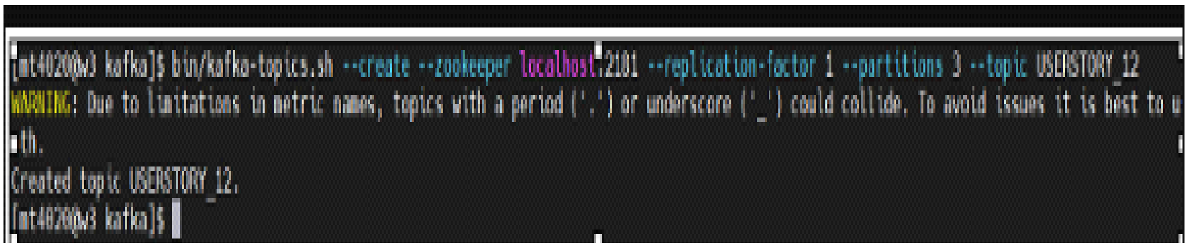
- Used cd command to change the directory.



```
mt4020@w3 ~]$ cd /home
mt4020@w3 home]$ cd kafka
mt4020@w3 kafka]$ cd kafka
mt4020@w3 kafka]$ █
```

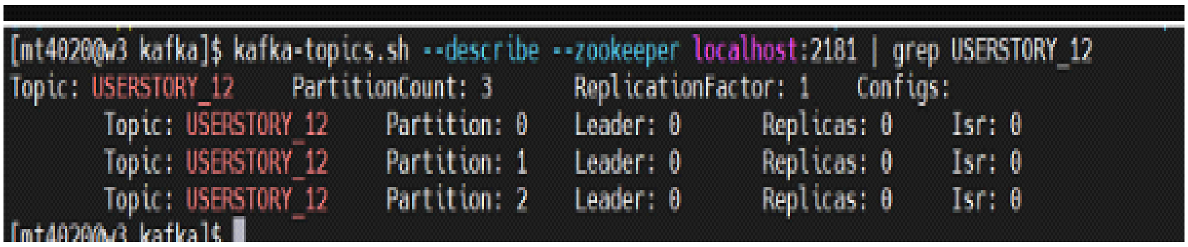
Created partitions using CLI :

- Created partitions using command line interface.
- Created 3 partitions with topic name "USERSTORY_12"
- **SYNTAX:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic [topicname]**
- Zookeeper is like a file system it stores configuration data for topics ,producers in kafka.



```
[mt4020@w3 kafka]$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic USERSTORY_12
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to u
th.
Created topic USERSTORY_12.
[mt4020@w3 kafka]$ █
```

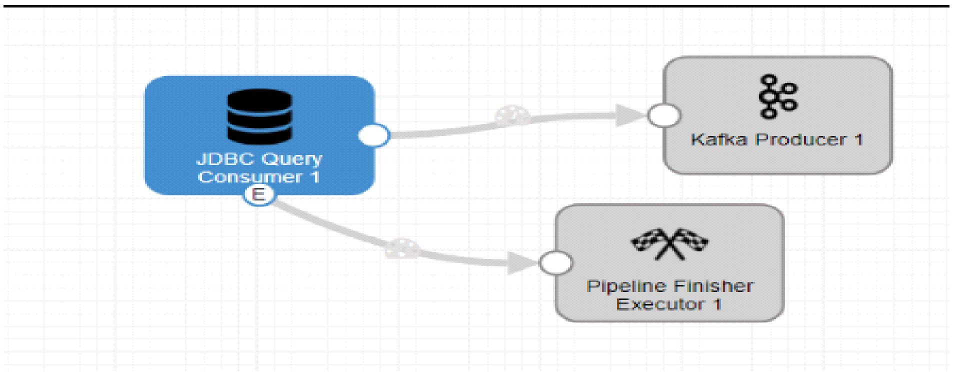
- Checked whether partitions created or not.
- **SYNTAX:kafka-topics.sh --describe --zookeeper localhost:2181 | grep topic_name**



```
[mt4020@w3 kafka]$ kafka-topics.sh --describe --zookeeper localhost:2181 | grep USERSTORY_12
Topic: USERSTORY_12     PartitionCount: 3       ReplicationFactor: 1    Configs:
        Topic: USERSTORY_12     Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: USERSTORY_12     Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: USERSTORY_12     Partition: 2    Leader: 0       Replicas: 0     Isr: 0
[mt4020@w3 kafka]$ █
```

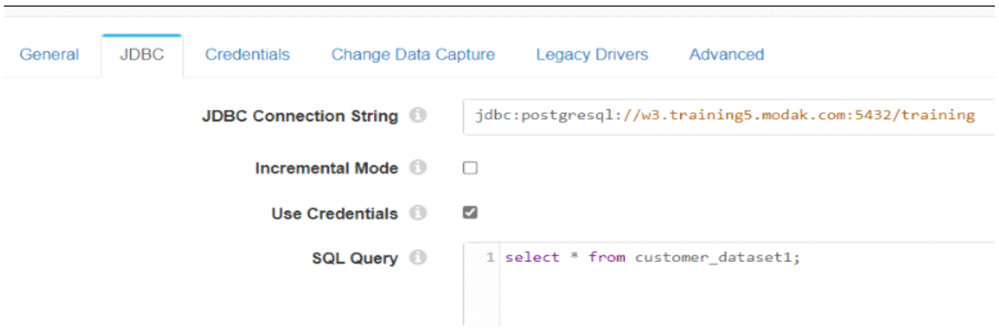
Sent data from jdbc Query Consumer to kafka Producer:

- Sent data from jdbc Consumer to kafka producer by using below pipeline.



Origin JDBC Query Consumer:

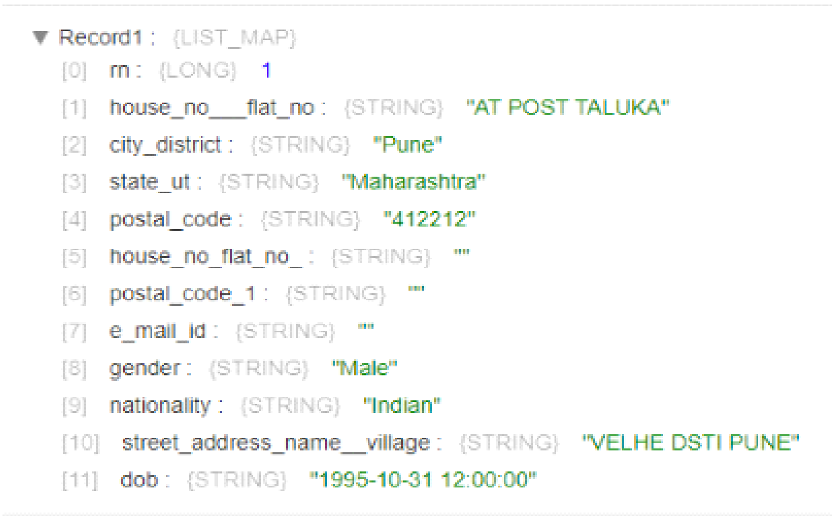
- The JDBC Query Consumer origin reads database data using a user-defined SQL query through a JDBC connection. The origin returns data as a map with column names and field values.
- Connected to postgresql by providing JDBC Connection String and Credentials.
- Used the below sql query to get data from postgresql.



JDBC Connection String: jdbc:postgresql://w3.training5.modak.com:5432/training
Incremental Mode: ☐
Use Credentials: ✓
SQL Query:

```
select * from customer_dataset1;
```

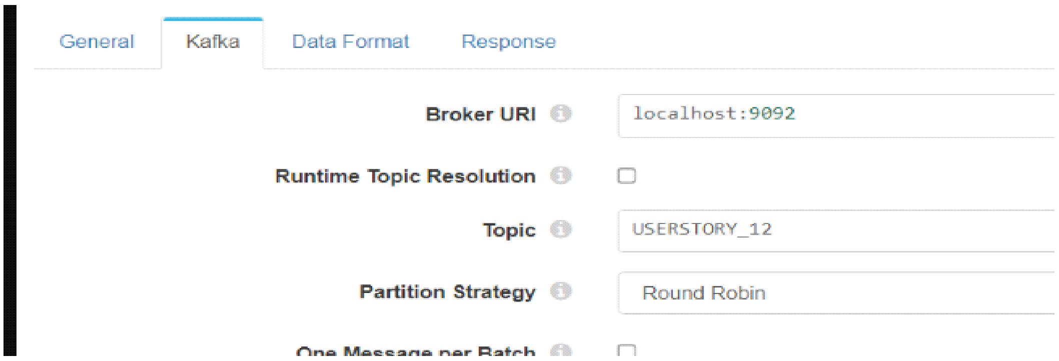
- When previewed the output is as follows.



```
▼ Record1 :  {LIST_MAP}
  [0]   rn :  {LONG}   1
  [1]   house_no___flat_no :  {STRING}   "AT POST TALUKA"
  [2]   city_district :  {STRING}   "Pune"
  [3]   state_ut :  {STRING}   "Maharashtra"
  [4]   postal_code :  {STRING}   "412212"
  [5]   house_no_flat_no_ :  {STRING}   ""
  [6]   postal_code_1 :  {STRING}   ""
  [7]   e_mail_id :  {STRING}   ""
  [8]   gender :  {STRING}   "Male"
  [9]   nationality :  {STRING}   "Indian"
  [10]   street_address_name__village :  {STRING}   "VELHE DSTI PUNE"
  [11]   dob :  {STRING}   "1995-10-31 12:00:00"
```

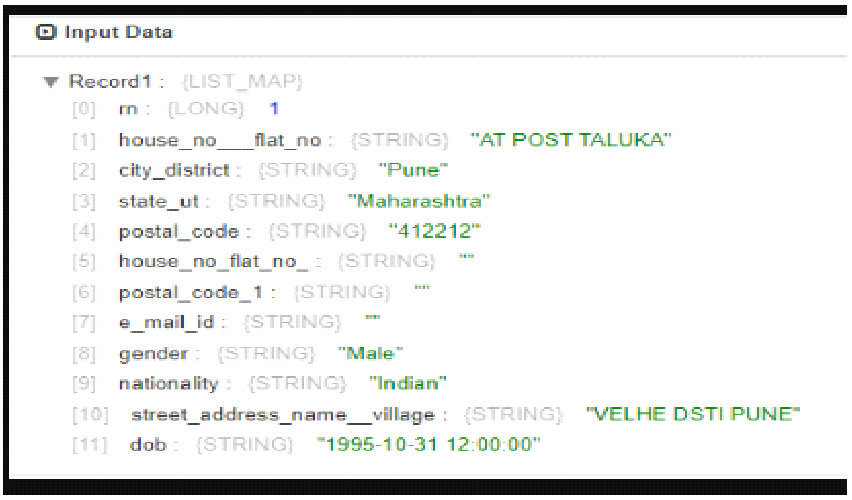
Destination Kafka Producer:

- The Kafka Producer destination writes data to a Kafka cluster.
- Given topic name as "USERSTORY_12" for which we created partitions.



Broker URI: localhost:9092
Runtime Topic Resolution: ☐
Topic: USERSTORY_12
Partition Strategy: Round Robin
One Message per Batch: ☐

- When previewed the input is as follows.

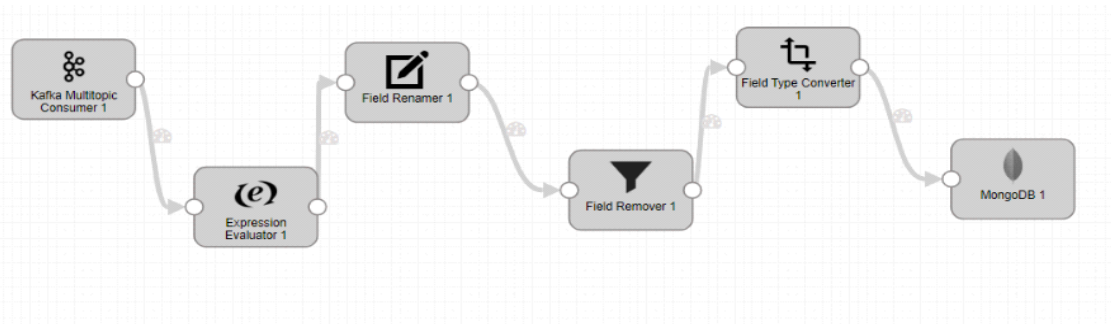


```
Record1 : {LIST_MAP}
  [0]  rn : {LONG}  1
  [1]  house_no___flat_no : {STRING}  "AT POST TALUKA"
  [2]  city_district : {STRING}  "Pune"
  [3]  state_ut : {STRING}  "Maharashtra"
  [4]  postal_code : {STRING}  "412212"
  [5]  house_no_flat_no_ : {STRING}  ""
  [6]  postal_code_1 : {STRING}  ""
  [7]  e_mail_id : {STRING}  ""
  [8]  gender : {STRING}  "Male"
  [9]  nationality : {STRING}  "Indian"
  [10]  street_address_name__village : {STRING}  "VELHE DSTI PUNE"
  [11]  dob : {STRING}  "1995-10-31 12:00:00"
```

Pipeline Finisher Executor: It is used to stop pipeline automatically after completion of execution of sql query once.

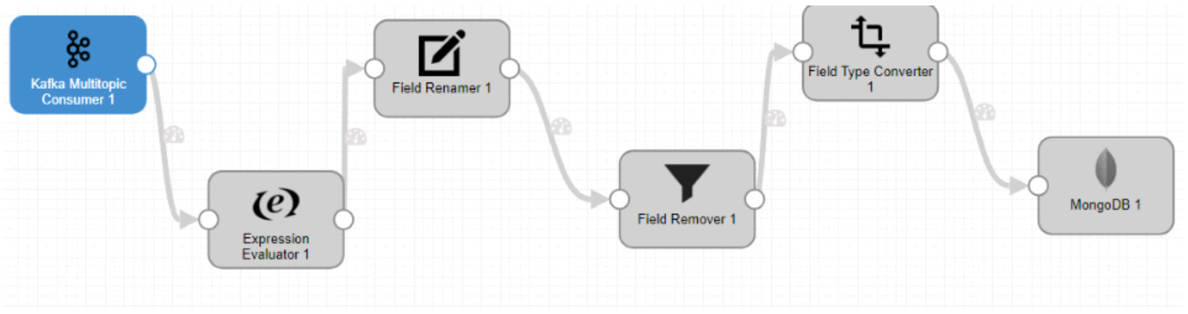
Created a pipeline from kafka Multitopic Consumer to Mongodb:

- Created a pipeline from kafka Multitopic Consumer to Mongodb as shown below.

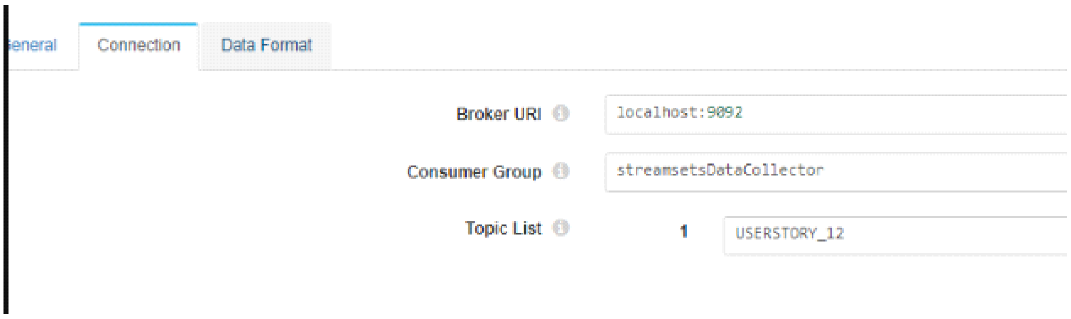


Origin Kafka Multitopic Consumer:

- The Kafka Multitopic Consumer origin reads data from multiple topics in an Apache Kafka cluster.



- Given the topic name as "USERSTORY_12" which created before.

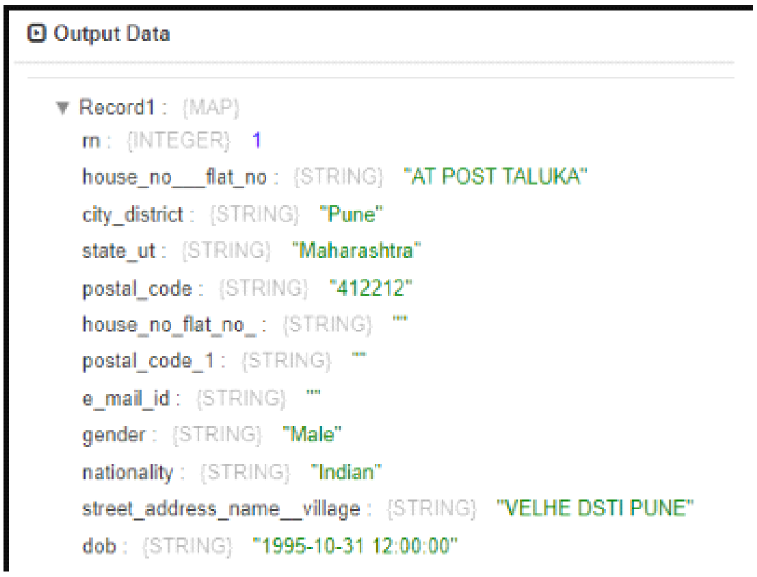


Broker URI: localhost:9092

Consumer Group: streamsetsDataCollector

Topic List: 1 USERSTORY_12

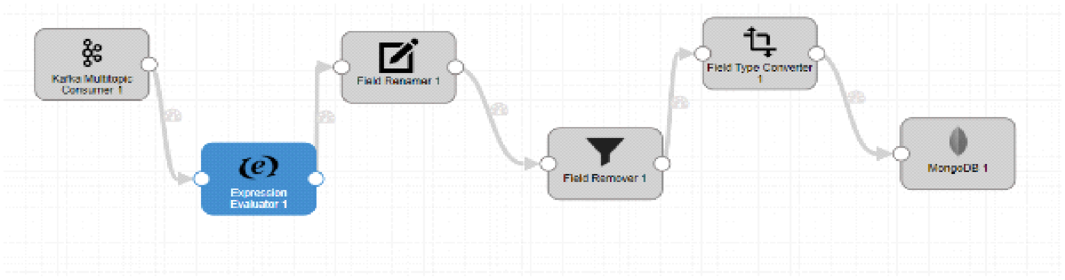
- When previewed the output is as follow.



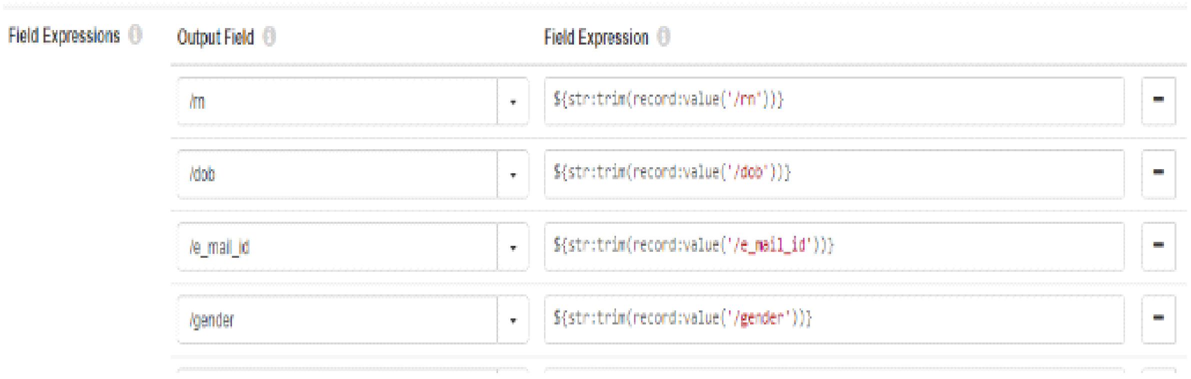
```
Output Data

▼ Record1 : {MAP}
   rn : {INTEGER} 1
   house_no___flat_no : {STRING} "AT POST TALUKA"
   city_district : {STRING} "Pune"
   state_ut : {STRING} "Maharashtra"
   postal_code : {STRING} "412212"
   house_no_flat_no_ : {STRING} ""
   postal_code_1 : {STRING} ""
   e_mail_id : {STRING} ""
   gender : {STRING} "Male"
   nationality : {STRING} "Indian"
   street_address_name__village : {STRING} "VELHE DSTI PUNE"
   dob : {STRING} "1995-10-31 12:00:00"
```

Processor Expression Evaluator:

- The Expression Evaluator performs calculations and writes the results to new or existing fields. You can also use the Expression Evaluator to add or modify record header attributes and field attributes.

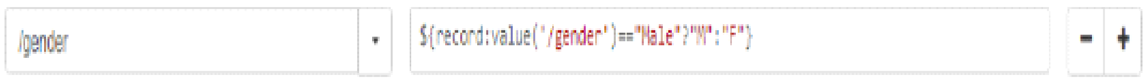


- Used Expression Evaluator to trim values of fields.



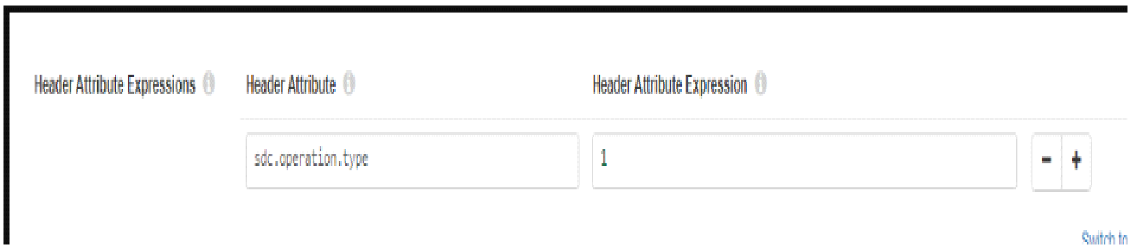
Field Expressions	Output Field	Field Expression
	/rn	${str:trim(record:value('/rn'))}
	/dob	${str:trim(record:value('/dob'))}
	/e_mail_id	${str:trim(record:value('/e_mail_id'))}
	/gender	${str:trim(record:value('/gender'))}

- Used Expression Evaluator to replace "Male" with "M" and "Female" with "F".



/gender	${record:value('/gender')=="Male"?"M":"F"}

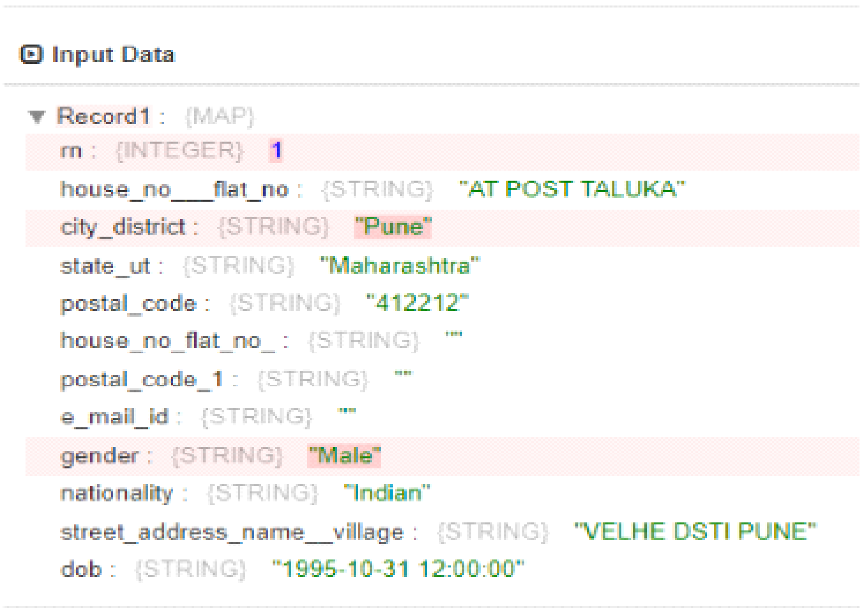
- Used Expression Evaluator to specify sdc.operation.type



Header Attribute Expressions	Header Attribute	Header Attribute Expression
	sdc.operation.type	1

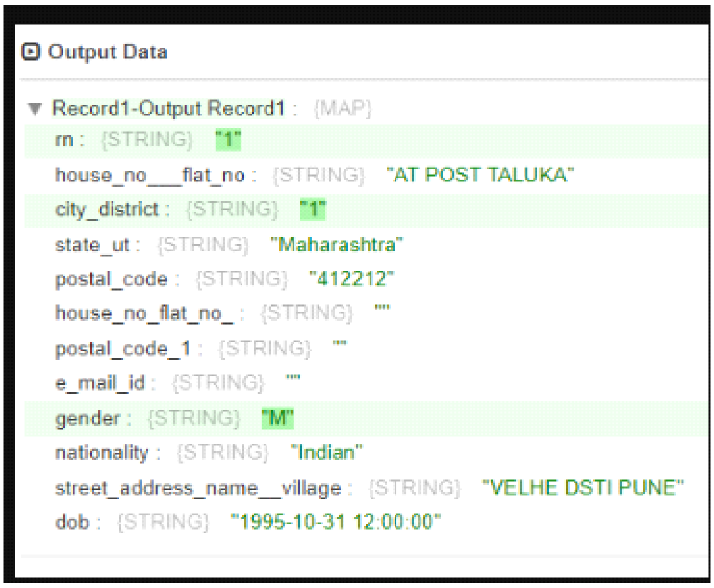
Switch to

- **Input and output:**



Input Data

```
▼ Record1 :  {MAP}
  rn :  {INTEGER}  1
  house_no___flat_no :  {STRING}  "AT POST TALUKA"
  city_district :  {STRING}  "Pune"
  state_ut :  {STRING}  "Maharashtra"
  postal_code :  {STRING}  "412212"
  house_no_flat_no_ :  {STRING}  ""
  postal_code_1 :  {STRING}  ""
  e_mail_id :  {STRING}  ""
  gender :  {STRING}  "Male"
  nationality :  {STRING}  "Indian"
  street_address_name__village :  {STRING}  "VELHE DSTI PUNE"
  dob :  {STRING}  "1995-10-31 12:00:00"
```

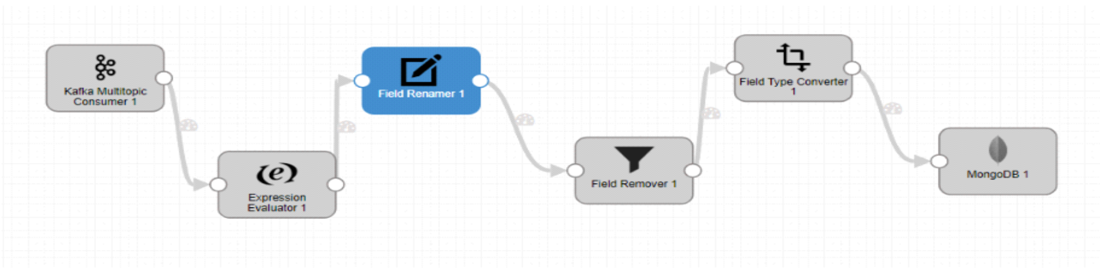


Output Data

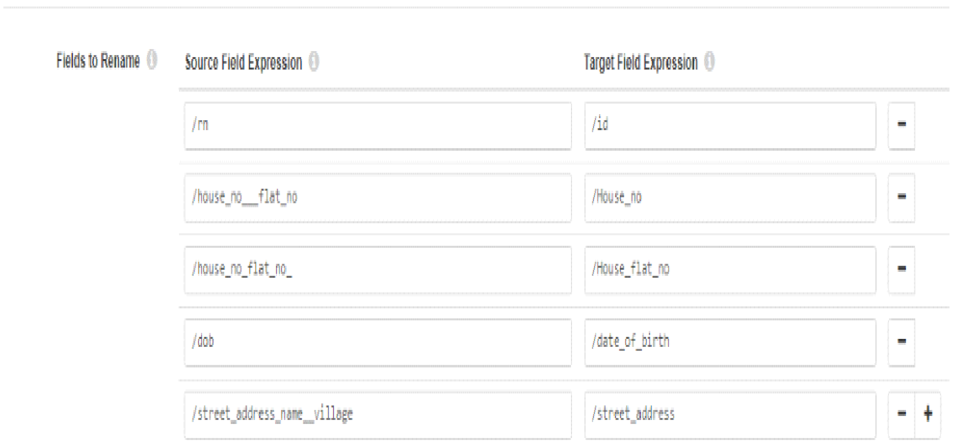
```
▼ Record1-Output Record1 :  {MAP}
  rn :  {STRING}  "1"
  house_no___flat_no :  {STRING}  "AT POST TALUKA"
  city_district :  {STRING}  "1"
  state_ut :  {STRING}  "Maharashtra"
  postal_code :  {STRING}  "412212"
  house_no_flat_no_ :  {STRING}  ""
  postal_code_1 :  {STRING}  ""
  e_mail_id :  {STRING}  ""
  gender :  {STRING}  "M"
  nationality :  {STRING}  "Indian"
  street_address_name__village :  {STRING}  "VELHE DSTI PUNE"
  dob :  {STRING}  "1995-10-31 12:00:00"
```

Processor Field Renamer:

- Use the Field Renamer to rename fields in a record. You can specify individual fields to rename or use regular expressions to rename sets of fields.

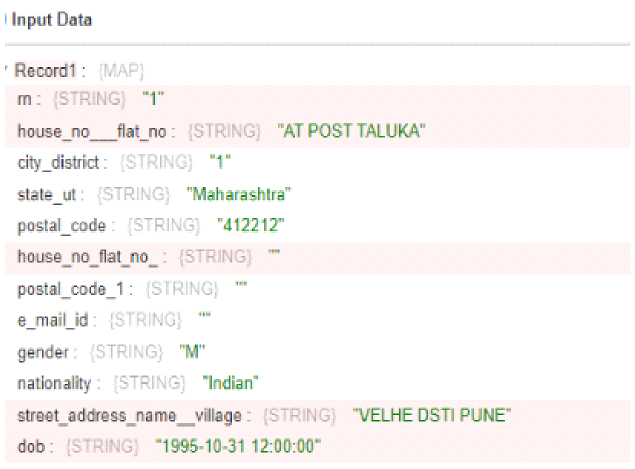


- Used Field Renamer to rename some field names.
- Changed the field names of some fields

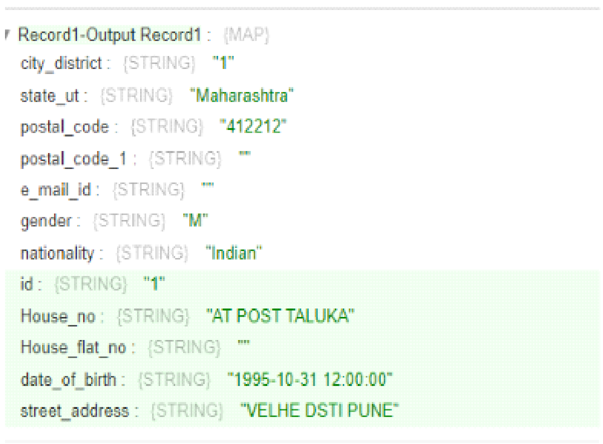


Fields to Rename	Source Field Expression	Target Field Expression
	/rn	/id
	/house_no__flat_no	/House_no
	/house_no_flat_no_	/House_flat_no
	/dob	/date_of_birth
	/street_address_name__village	/street_address

- **Input and output:**



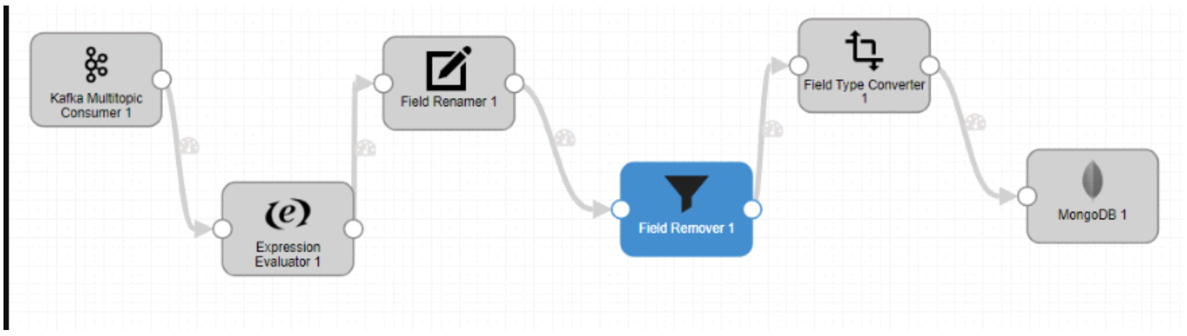
```
Input Data
Record1 : {MAP}
rn : {STRING} "1"
house_no__flat_no : {STRING} "AT POST TALUKA"
city_district : {STRING} "1"
state_ut : {STRING} "Maharashtra"
postal_code : {STRING} "412212"
house_no_flat_no_ : {STRING} ""
postal_code_1 : {STRING} ""
e_mail_id : {STRING} ""
gender : {STRING} "M"
nationality : {STRING} "Indian"
street_address_name__village : {STRING} "VELHE DSTI PUNE"
dob : {STRING} "1995-10-31 12:00:00"
```



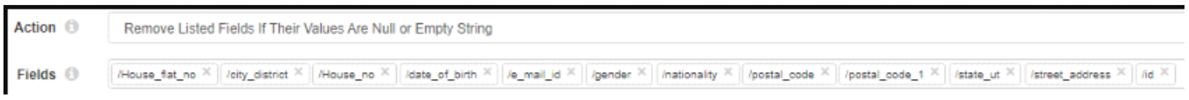
```
Record1-Output Record1 : {MAP}
city_district : {STRING} "1"
state_ut : {STRING} "Maharashtra"
postal_code : {STRING} "412212"
postal_code_1 : {STRING} ""
e_mail_id : {STRING} ""
gender : {STRING} "M"
nationality : {STRING} "Indian"
id : {STRING} "1"
House_no : {STRING} "AT POST TALUKA"
House_flat_no : {STRING} ""
date_of_birth : {STRING} "1995-10-31 12:00:00"
street_address : {STRING} "VELHE DSTI PUNE"
```

Processor Field Remover:

- The Field Remover processor removes fields from records. Use the processor to discard field data that you do not need in the pipeline.

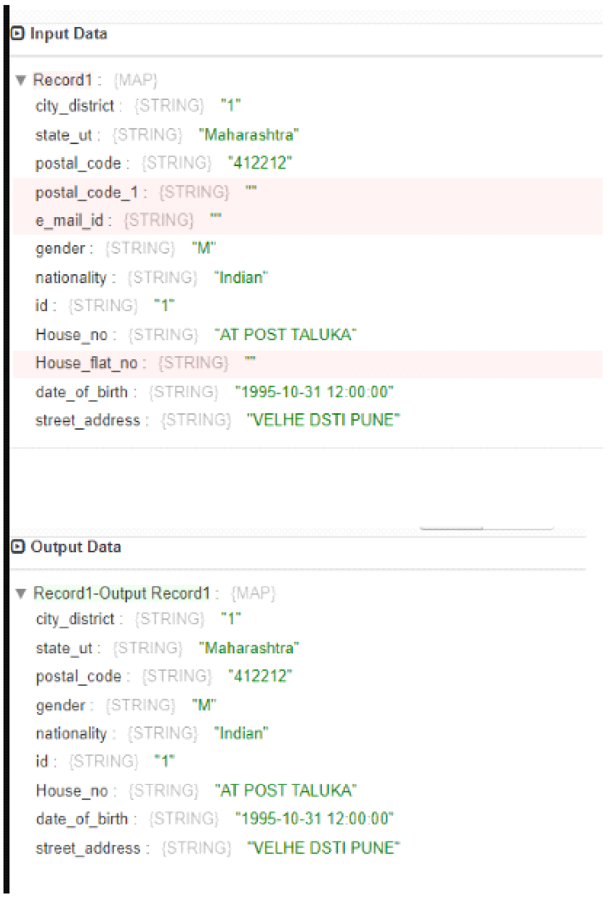


- Used Field Remover to remover null values in the data.



Action	Remove Listed Fields If Their Values Are Null or Empty String
Fields	/House_flat_no ✕ /city_district ✕ /House_no ✕ /date_of_birth ✕ /e_mail_id ✕ /gender ✕ /nationality ✕ /postal_code ✕ /postal_code_1 ✕ /state_ut ✕ /street_address ✕ /id ✕

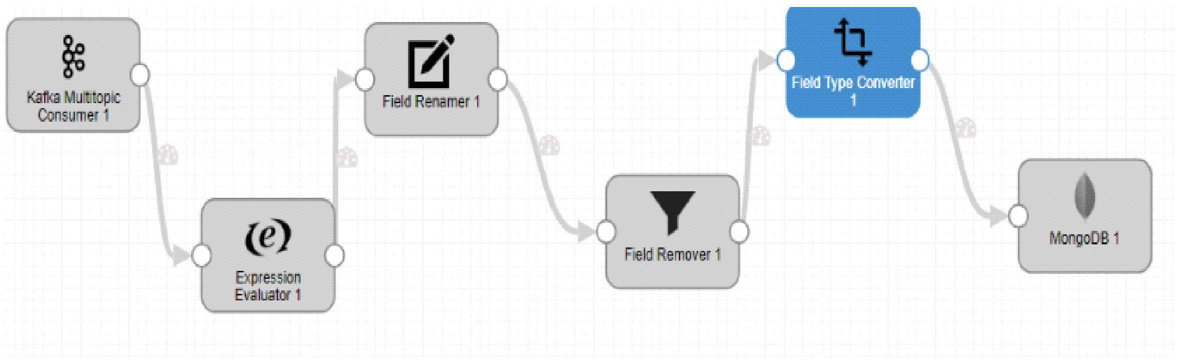
- **Input and output:**



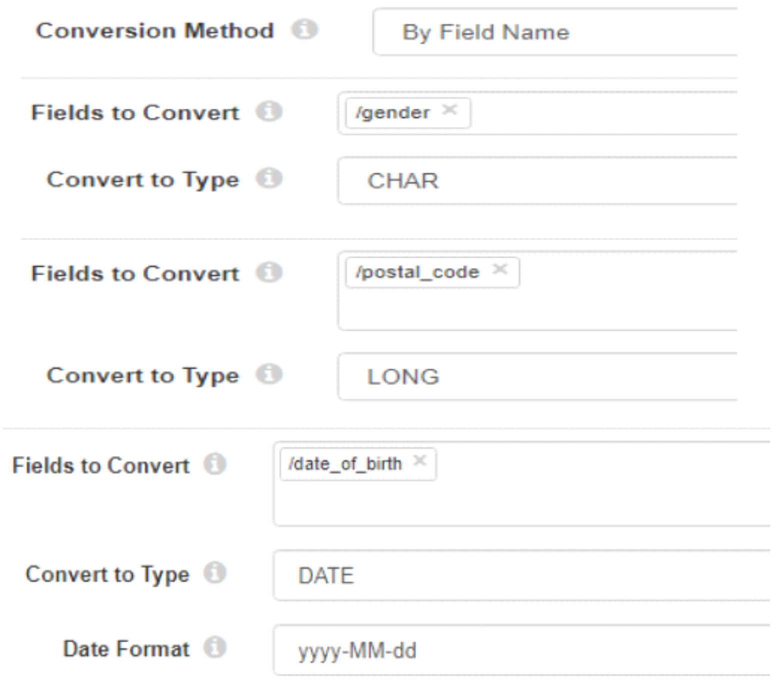
```
⊡ Input Data
▼ Record1 : {MAP}
  city_district : {STRING}  "1"
  state_ut : {STRING}  "Maharashtra"
  postal_code : {STRING}  "412212"
  postal_code_1 : {STRING}  ""
  e_mail_id : {STRING}  ""
  gender : {STRING}  "M"
  nationality : {STRING}  "Indian"
  id : {STRING}  "1"
  House_no : {STRING}  "AT POST TALUKA"
  House_flat_no : {STRING}  ""
  date_of_birth : {STRING}  "1995-10-31 12:00:00"
  street_address : {STRING}  "VELHE DSTI PUNE"

⊡ Output Data
▼ Record1-Output Record1 : {MAP}
  city_district : {STRING}  "1"
  state_ut : {STRING}  "Maharashtra"
  postal_code : {STRING}  "412212"
  gender : {STRING}  "M"
  nationality : {STRING}  "Indian"
  id : {STRING}  "1"
  House_no : {STRING}  "AT POST TALUKA"
  date_of_birth : {STRING}  "1995-10-31 12:00:00"
  street_address : {STRING}  "VELHE DSTI PUNE"
```

Processor Field Type Converter:

- The Field Type Converter processor converts the data types of fields to compatible data types. You might use the processor to convert the data types of fields before performing calculations.
- You can also use the processor to change the scale of decimal data.

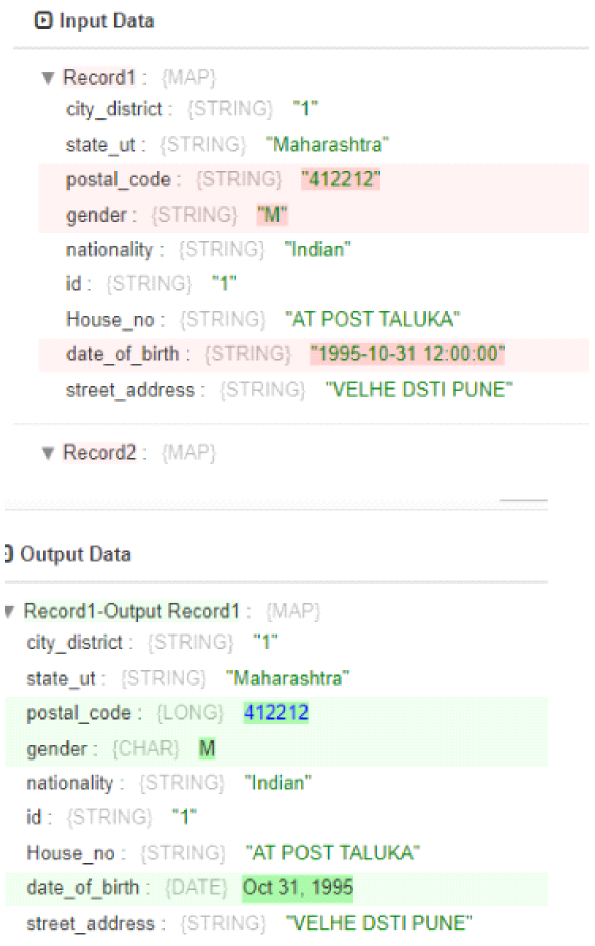


- Used Field Type Converter to change datatype of some fields.



Conversion Method	By Field Name
Fields to Convert	/gender
Convert to Type	CHAR
Fields to Convert	/postal_code
Convert to Type	LONG
Fields to Convert	/date_of_birth
Convert to Type	DATE
Date Format	yyyy-MM-dd

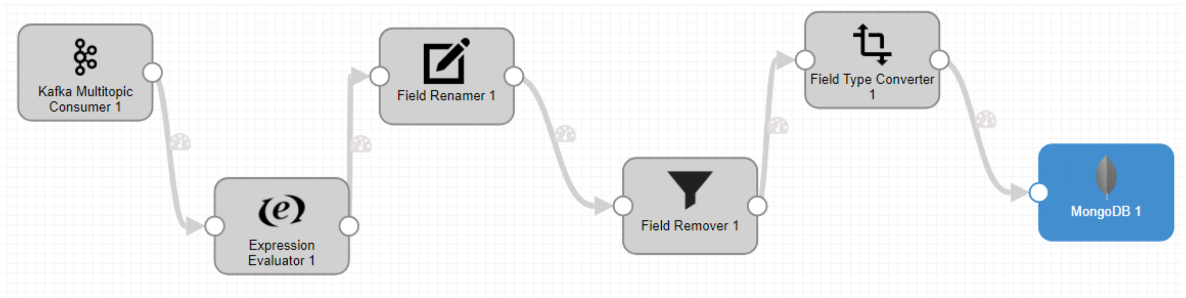
- **Input and output:**



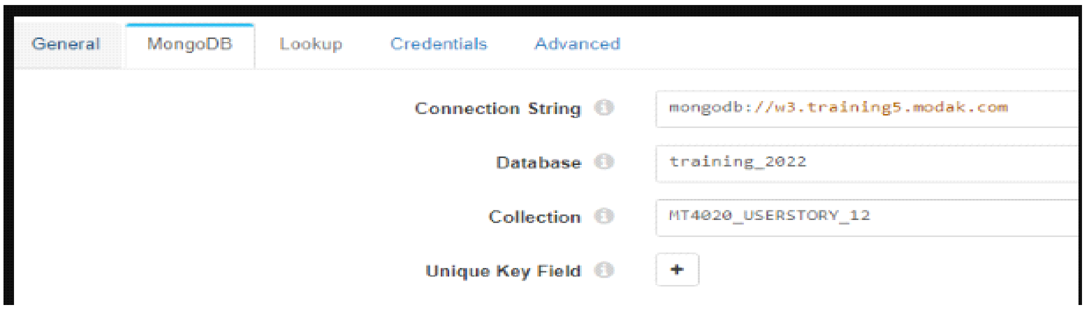
```
Input Data

▼ Record1 :  {MAP}
    city_district :  {STRING}  "1"
    state_ut :  {STRING}  "Maharashtra"
    postal_code :  {STRING}  "412212"
    gender :  {STRING}  "M"
    nationality :  {STRING}  "Indian"
    id :  {STRING}  "1"
    House_no :  {STRING}  "AT POST TALUKA"
    date_of_birth :  {STRING}  "1995-10-31 12:00:00"
    street_address :  {STRING}  "VELHE DSTI PUNE"

▼ Record2 :  {MAP}

Output Data

▼ Record1-Output Record1 :  {MAP}
    city_district :  {STRING}  "1"
    state_ut :  {STRING}  "Maharashtra"
    postal_code :  {LONG}  412212
    gender :  {CHAR}  M
    nationality :  {STRING}  "Indian"
    id :  {STRING}  "1"
    House_no :  {STRING}  "AT POST TALUKA"
    date_of_birth :  {DATE}  Oct 31, 1995
    street_address :  {STRING}  "VELHE DSTI PUNE"
```

Destination MongoDB:

- The MongoDB destination writes data to MongoDB.

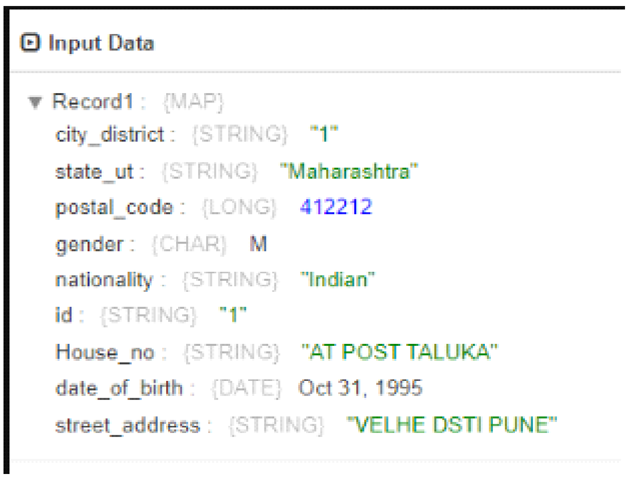


- Created a collection in MongoDB compass "MT4020_USERSTORY_12" in training_2022 database.
- Connected to MongoDB by providing Connection String and Credentials.



Connection String: mongodb://w3.training5.modak.com
Database: training_2022
Collection: MT4020_USERSTORY_12
Unique Key Field: +

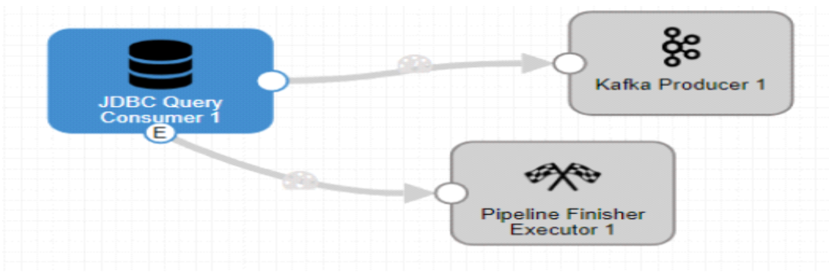
- **Input:**



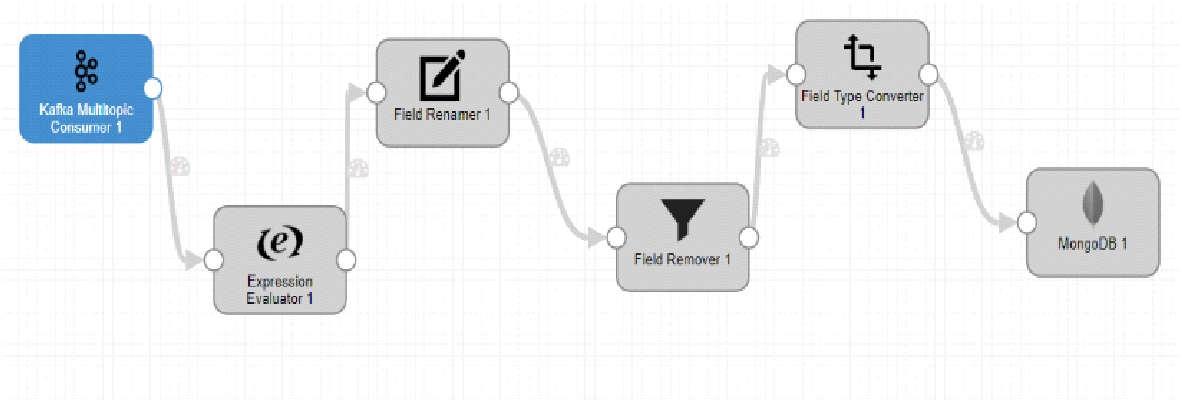
```
Input Data
▼ Record1 : {MAP}
    city_district : {STRING} "1"
    state_ut : {STRING} "Maharashtra"
    postal_code : {LONG} 412212
    gender : {CHAR} M
    nationality : {STRING} "Indian"
    id : {STRING} "1"
    House_no : {STRING} "AT POST TALUKA"
    date_of_birth : {DATE} Oct 31, 1995
    street_address : {STRING} "VELHE DSTI PUNE"
```

Final pipeline:

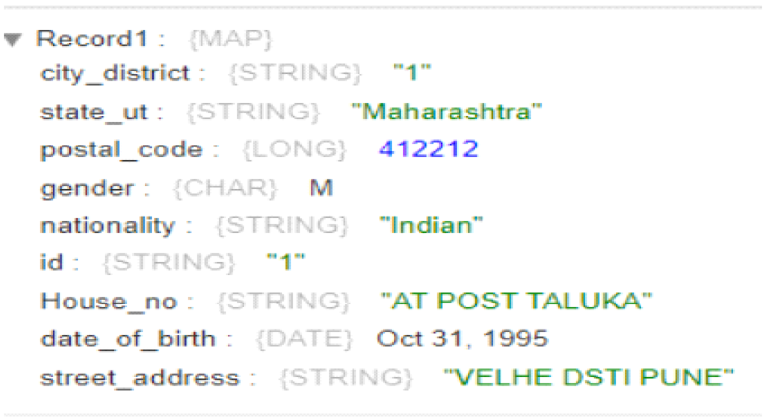
- JDBC Query Consumer to kafka Producer.



- Kafka Multitopic Consumer to MongoDB.



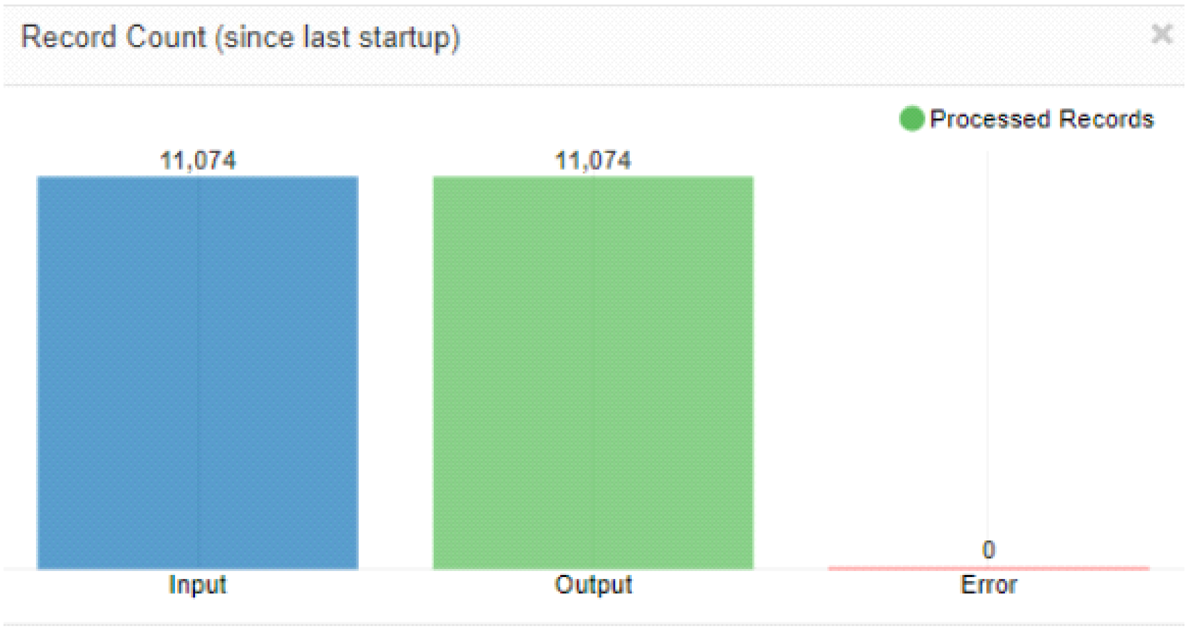
Input and Output:



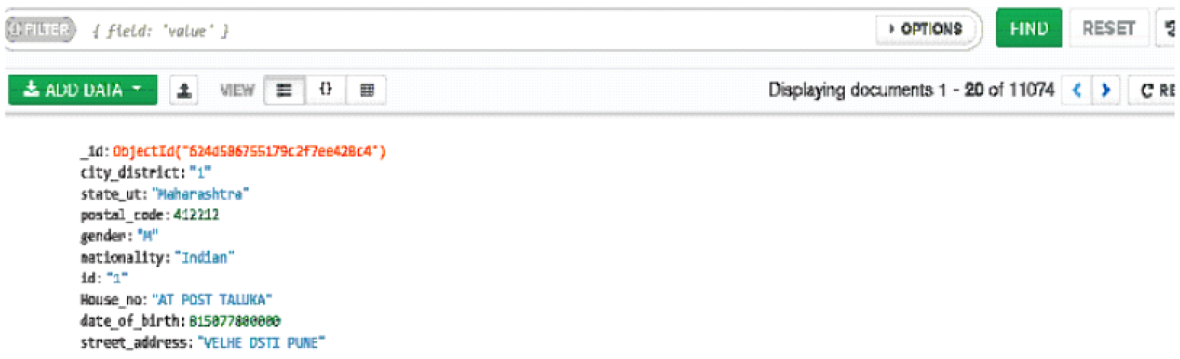
```
▼ Record1 :  {MAP}
  city_district :  {STRING}   "1"
  state_ut :  {STRING}   "Maharashtra"
  postal_code :  {LONG}   412212
  gender :  {CHAR}   M
  nationality :  {STRING}   "Indian"
  id :  {STRING}   "1"
  House_no :  {STRING}   "AT POST TALUKA"
  date_of_birth :  {DATE}   Oct 31, 1995
  street_address :  {STRING}   "VELHE DSTI PUNE"
```

Data in MongoDB:

- Input data record count is 11074 and output record count is 11074.



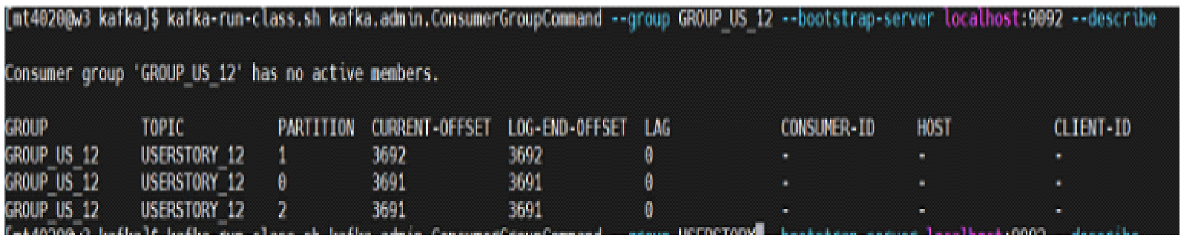
- The count of records went to MongoDB is also 11074.



```
FILTER  { field: 'value' }                                   OPTIONS   FIND   RESET
ADD DATA   VIEW                              Displaying documents 1 - 20 of 11074

_id: ObjectId("624d586755179c2f7ee428c4")
city_district: "1"
state_ut: "Maharashtra"
postal_code: 412212
gender: "M"
nationality: "Indian"
id: "1"
House_no: "AT POST TALUKA"
date_of_birth: 815077800000
street_address: "VELHE DSTI PUNE"
```

Number of Records in each partitions:

- Checked number of records in each partitions .



```
[mt4020@w3 kafka]$ kafka-run-class.sh kafka.admin.ConsumerGroupCommand --group GROUP_US_12 --bootstrap-server localhost:9092 --describe

Consumer group 'GROUP_US_12' has no active members.

GROUP        TOPIC        PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
GROUP_US_12  USERSTORY_12 1          3692            3692            0               -               -               -
GROUP_US_12  USERSTORY_12 0          3691            3691            0               -               -               -
GROUP_US_12  USERSTORY_12 2          3691            3691            0               -               -               -
```

USER_STORY_07

- **Read records from a delimited file and convert it to map & publish**

As a developer, I want to read data from Google bucket and store it in Mongo Collection.

- Consume records from GCS file storage.
- Parse the records & convert into maps.
- Exclude null values in any column.
- Store it in MongoDB with valid datatypes of data for integer and timestamp.
- Records will be in a file path, need to look for a new file & process when received.
- Use dataset2_5
- CDC to be performed based on action codes in the data before dumping to MongoDB.

The values in the flag column can be treated as below.

A- Addition of new row

D- Deletion of a row

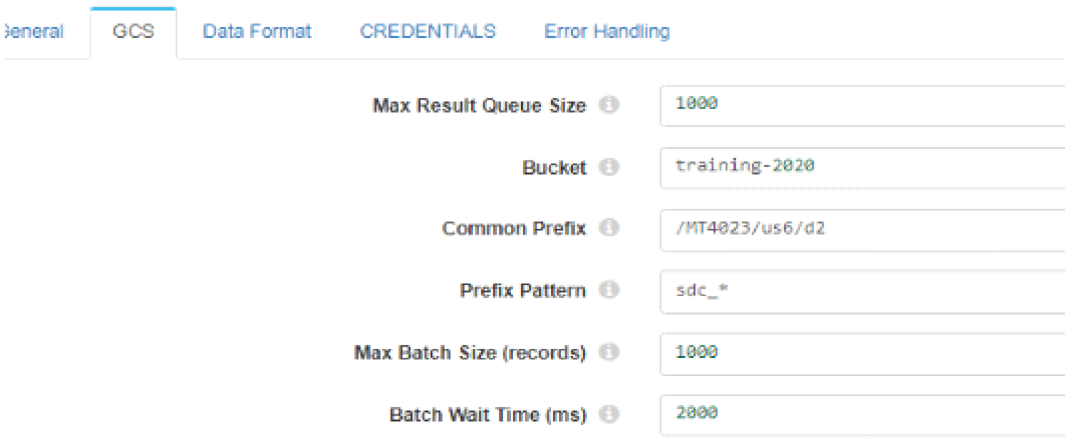
C- Change/Update in a row

- **Acceptance Criteria :**
- Data validated at the destination
- Userstory reviewed by the Technical Lead
- Azure board to be updated
- Approach and Implementation of user story should be documented

EXPLANATION:

Consumed records from GCS file storage which are in JSON format.

- Had dependency on User Story 6. So, the input was taken as output of User Story 6.
- Origin is Google Cloud Storage .
- Provided the Bucket name , Common prefix of data.
- The Google Cloud Storage origin reads objects stored in Google Cloud Storage. The objects must be fully written and reside in a single bucket.



Max Result Queue Size: 1000

Bucket: training-2020

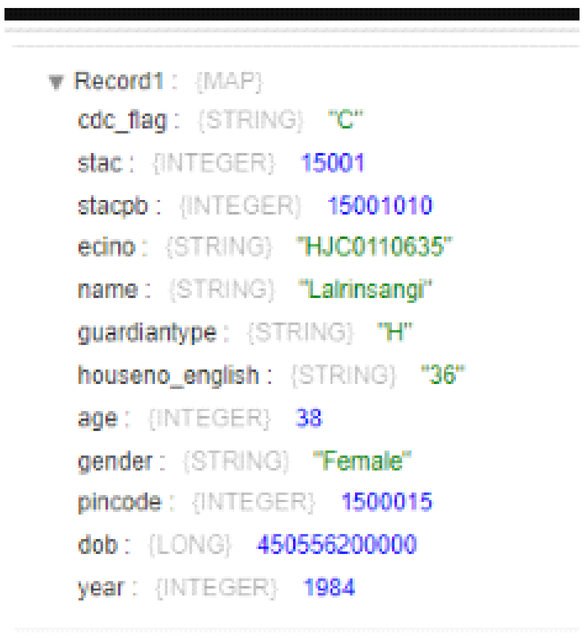
Common Prefix: /MT4023/us6/d2

Prefix Pattern: sdc_*

Max Batch Size (records): 1000

Batch Wait Time (ms): 2000

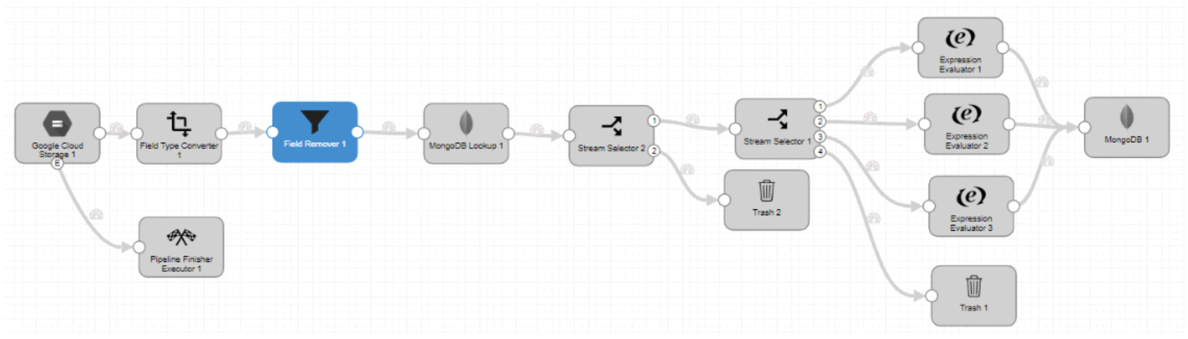
- Output of Google Cloud Storage



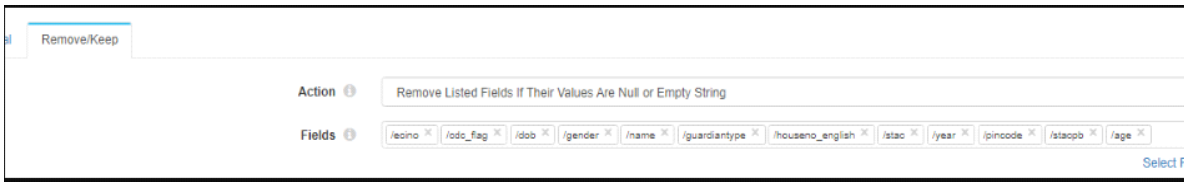
```
▼ Record1 : {MAP}
  cdc_flag : {STRING} "C"
  stac : {INTEGER} 15001
  stacpb : {INTEGER} 15001010
  ecino : {STRING} "HJC0110635"
  name : {STRING} "Lalrinsangi"
  guardiantype : {STRING} "H"
  houseno_english : {STRING} "36"
  age : {INTEGER} 38
  gender : {STRING} "Female"
  pincode : {INTEGER} 1500015
  dob : {LONG} 450556200000
  year : {INTEGER} 1984
```

Excluded Null values in data:

- Used Field Remover to remove null values in data.
- The Field Remover processor removes fields from records. Use the processor to discard field data that you do not need in the pipeline



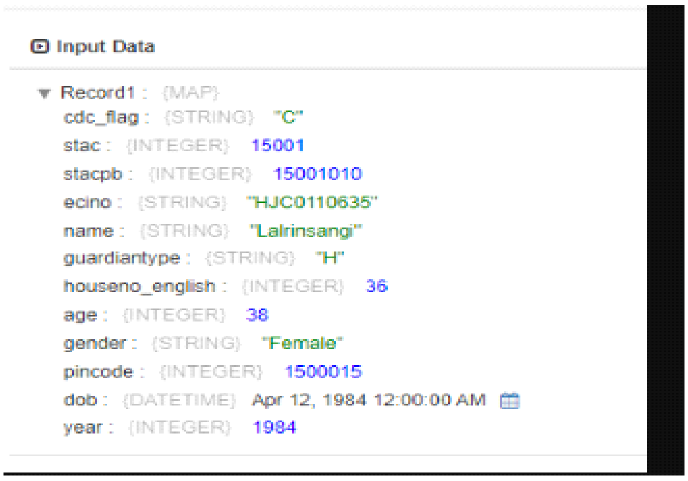
- Provided field names and action to be performed.



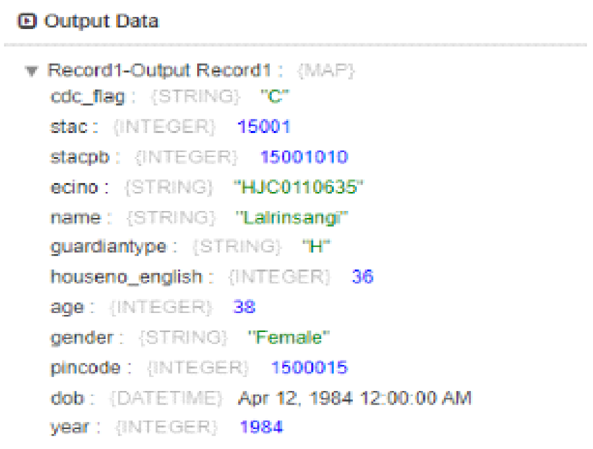
Action: Remove Listed Fields If Their Values Are Null or Empty String

Fields: /ecino, /cdc_flag, /dob, /gender, /name, /guardiantype, /houseno_english, /stac, /year, /pincode, /stacpb, /age

- **Input and output:**



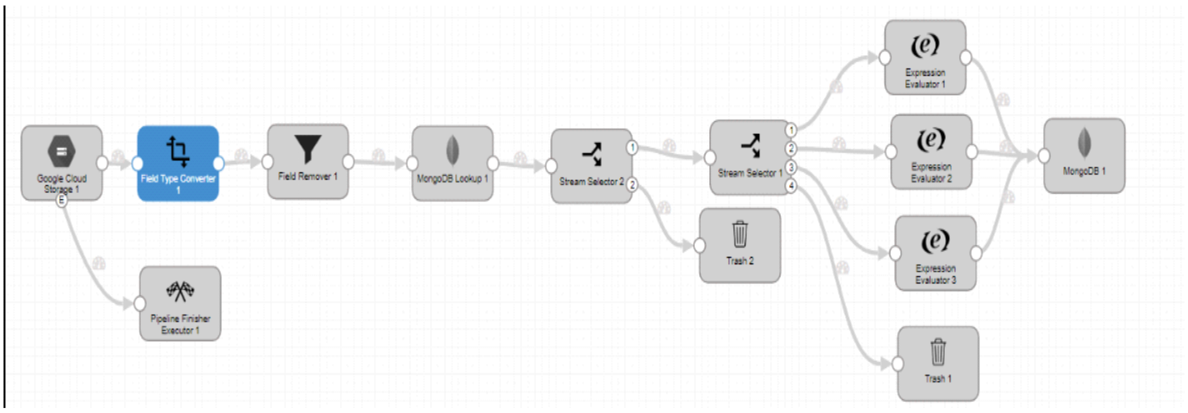
```
Input Data
▼ Record1 : {MAP}
    cdc_flag : {STRING} "C"
    stac : {INTEGER} 15001
    stacpb : {INTEGER} 15001010
    ecino : {STRING} "HJC0110635"
    name : {STRING} "Lalrinsangi"
    guardiantype : {STRING} "H"
    houseno_english : {INTEGER} 36
    age : {INTEGER} 38
    gender : {STRING} "Female"
    pincode : {INTEGER} 1500015
    dob : {DATETIME} Apr 12, 1984 12:00:00 AM
    year : {INTEGER} 1984
```



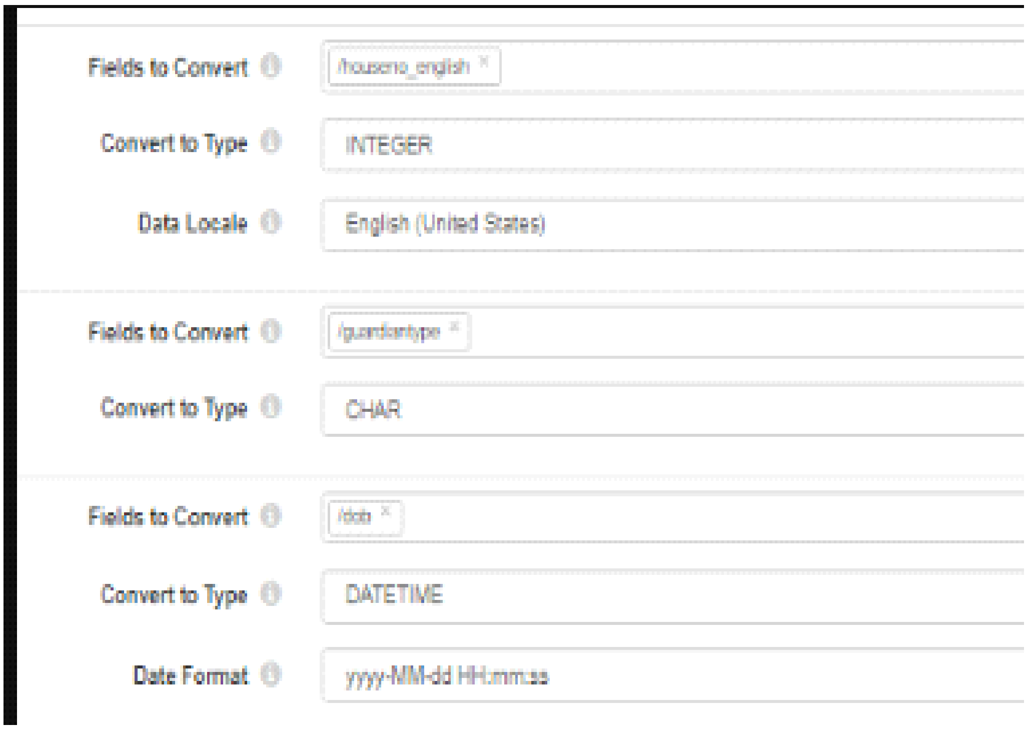
```
Output Data
▼ Record1-Output Record1 : {MAP}
    cdc_flag : {STRING} "C"
    stac : {INTEGER} 15001
    stacpb : {INTEGER} 15001010
    ecino : {STRING} "HJC0110635"
    name : {STRING} "Lalrinsangi"
    guardiantype : {STRING} "H"
    houseno_english : {INTEGER} 36
    age : {INTEGER} 38
    gender : {STRING} "Female"
    pincode : {INTEGER} 1500015
    dob : {DATETIME} Apr 12, 1984 12:00:00 AM
    year : {INTEGER} 1984
```

Changed the datatype of data for Integer and Timestamp:

- Used Field Type Converter to convert datatype of Fields.
- The Field Type Converter processor converts the data types of fields to compatible data types. You might use the processor to convert the data types of fields before performing calculations.

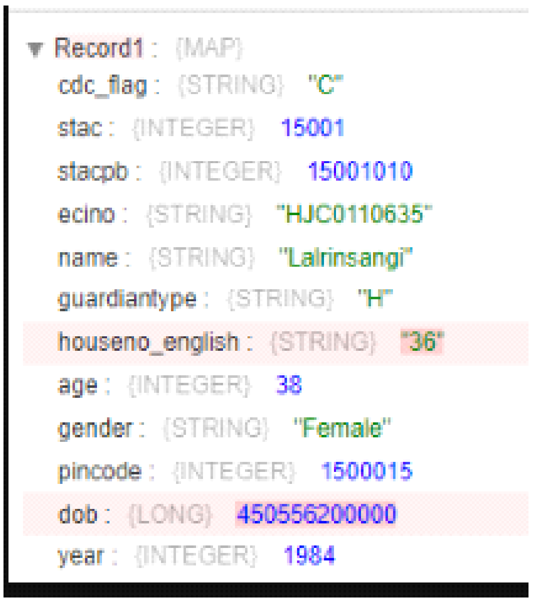


- Provided conversion Method.
- Provided Fields to convert and datatype to convert.

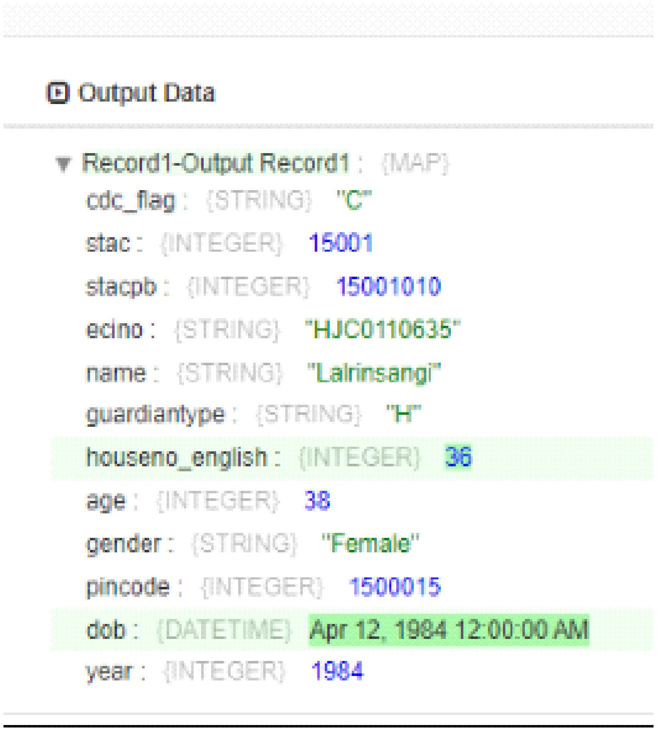


Fields to Convert	/houseno_english
Convert to Type	INTEGER
Data Locale	English (United States)
Fields to Convert	/guardiantype
Convert to Type	CHAR
Fields to Convert	/dob
Convert to Type	DATETIME
Date Format	yyyy-MM-dd HH:mm:ss

- **Input and output:**



```
▼ Record1 : {MAP}
   cdc_flag : {STRING}  "C"
   stac : {INTEGER}  15001
   stacpb : {INTEGER}  15001010
   ecino : {STRING}  "HJC0110635"
   name : {STRING}  "Lalrinsangi"
   guardiantype : {STRING}  "H"
   houseno_english : {STRING}  "36"
   age : {INTEGER}  38
   gender : {STRING}  "Female"
   pincode : {INTEGER}  1500015
   dob : {LONG}  450556200000
   year : {INTEGER}  1984
```

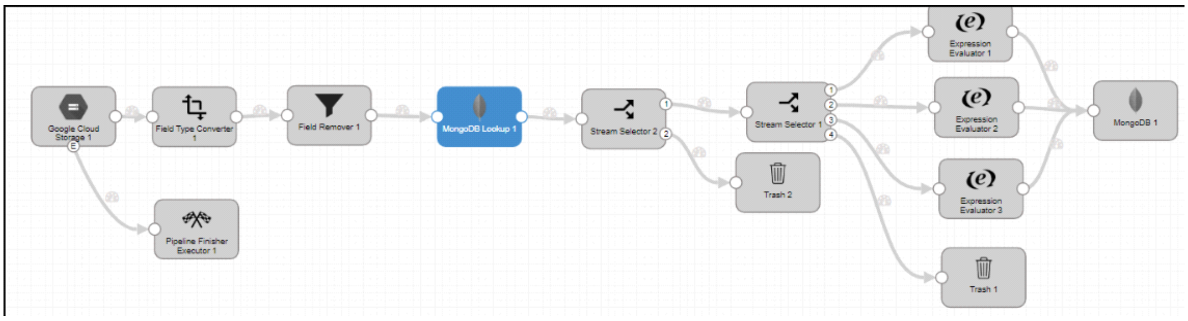


Output Data

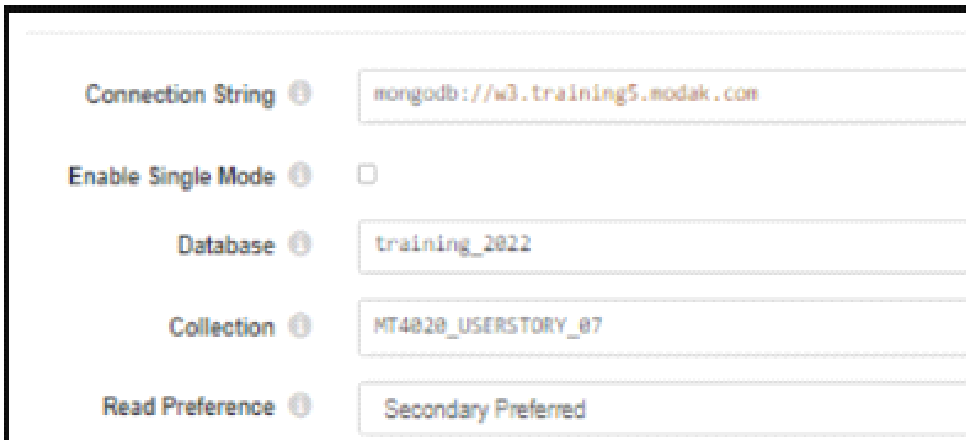
```
▼ Record1-Output Record1 : {MAP}
   cdc_flag : {STRING}  "C"
   stac : {INTEGER}  15001
   stacpb : {INTEGER}  15001010
   ecino : {STRING}  "HJC0110635"
   name : {STRING}  "Lalrinsangi"
   guardiantype : {STRING}  "H"
   houseno_english : {INTEGER}  36
   age : {INTEGER}  38
   gender : {STRING}  "Female"
   pincode : {INTEGER}  1500015
   dob : {DATETIME}  Apr 12, 1984 12:00:00 AM
   year : {INTEGER}  1984
```

Records will be in a file path, need to look for a new file & process when received :

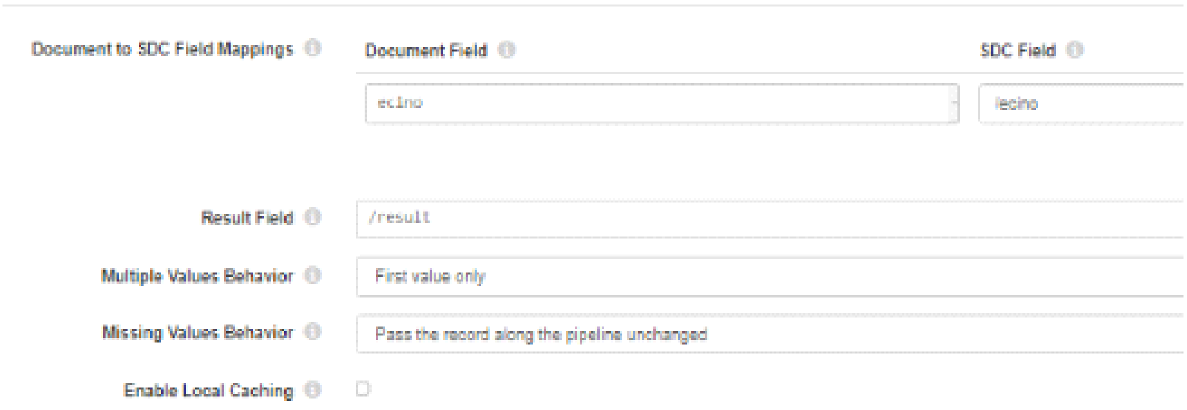
- Used MongoDB Lookup to see whether it is a new record or not.



- MongoDB will look new record in destination. If it present in destination it will send data to Result field or else Result field will be empty.
- Provided the Connection String and other fields.



Connection String	mongodb://w3.training5.modak.com
Enable Single Mode	☐
Database	training_2022
Collection	MT4020_USERSTORY_07
Read Preference	Secondary Preferred

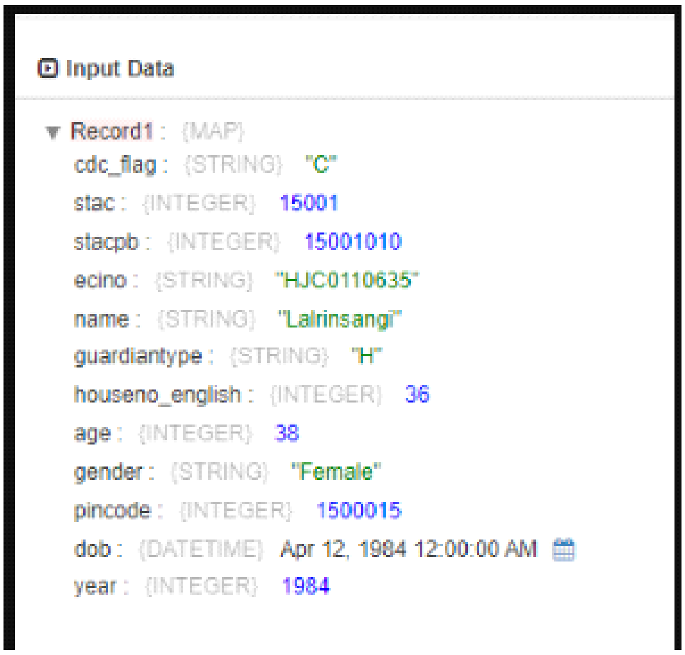


Document to SDC Field Mappings

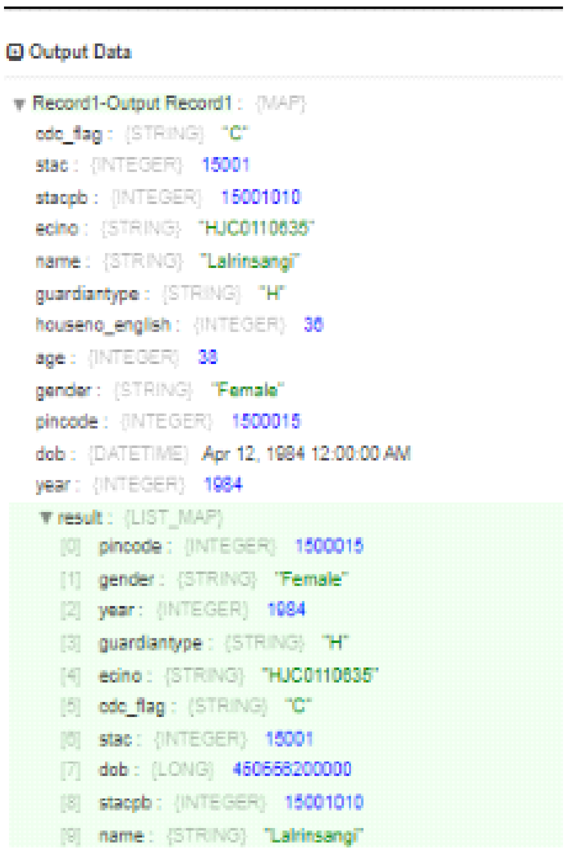
Document Field	SDC Field
ecino	/ecino

Result Field	/result
Multiple Values Behavior	First value only
Missing Values Behavior	Pass the record along the pipeline unchanged
Enable Local Caching	☐

- **Input and output:**



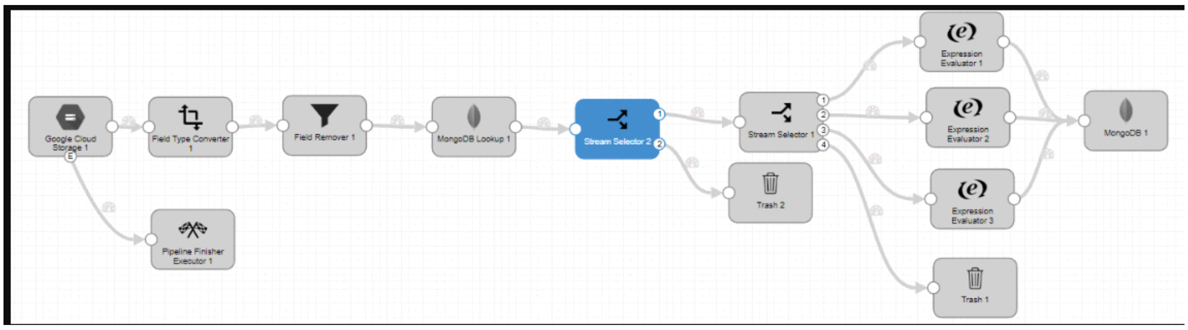
```
Input Data

▼ Record1 :  {MAP}
    cdc_flag :  {STRING}   "C"
    stac :  {INTEGER}   15001
    stacpb :  {INTEGER}   15001010
    ecino :  {STRING}   "HJC0110635"
    name :  {STRING}   "Lalrinsangi"
    guardiantype :  {STRING}   "H"
    houseno_english :  {INTEGER}   36
    age :  {INTEGER}   38
    gender :  {STRING}   "Female"
    pincode :  {INTEGER}   1500015
    dob :  {DATETIME}   Apr 12, 1984 12:00:00 AM
    year :  {INTEGER}   1984
```



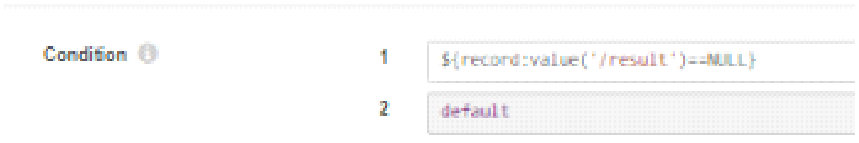
```
Output Data

▼ Record1-Output Record1 :  {MAP}
    cdc_flag :  {STRING}   "C"
    stac :  {INTEGER}   15001
    stacpb :  {INTEGER}   15001010
    ecino :  {STRING}   "HJC0110635"
    name :  {STRING}   "Lalrinsangi"
    guardiantype :  {STRING}   "H"
    houseno_english :  {INTEGER}   36
    age :  {INTEGER}   38
    gender :  {STRING}   "Female"
    pincode :  {INTEGER}   1500015
    dob :  {DATETIME}   Apr 12, 1984 12:00:00 AM
    year :  {INTEGER}   1984
  ▼ result :  {LIST_MAP}
      [0]   pincode :  {INTEGER}   1500015
      [1]   gender :  {STRING}   "Female"
      [2]   year :  {INTEGER}   1984
      [3]   guardiantype :  {STRING}   "H"
      [4]   ecino :  {STRING}   "HJC0110635"
      [5]   cdc_flag :  {STRING}   "C"
      [6]   stac :  {INTEGER}   15001
      [7]   dob :  {LONG}   450668200000
      [8]   stacpb :  {INTEGER}   15001010
      [9]   name :  {STRING}   "Lalrinsangi"
```

Used Stream Selector to send only new records:

- Used Stream Selector to send only new records to next stage.

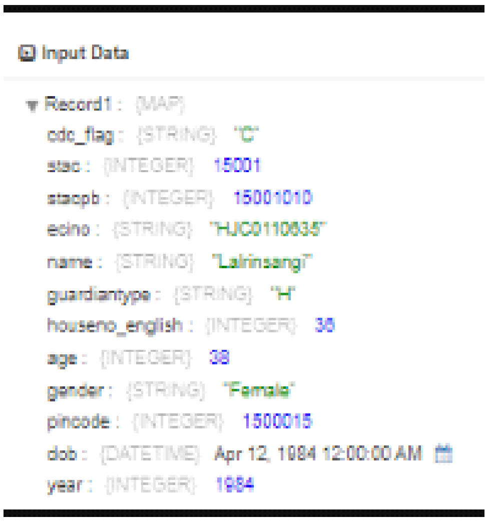


- If new value comes it will not present in the destination. So result field will be null.
- If it is null then it goes to Stream Selector else it will go Trash.



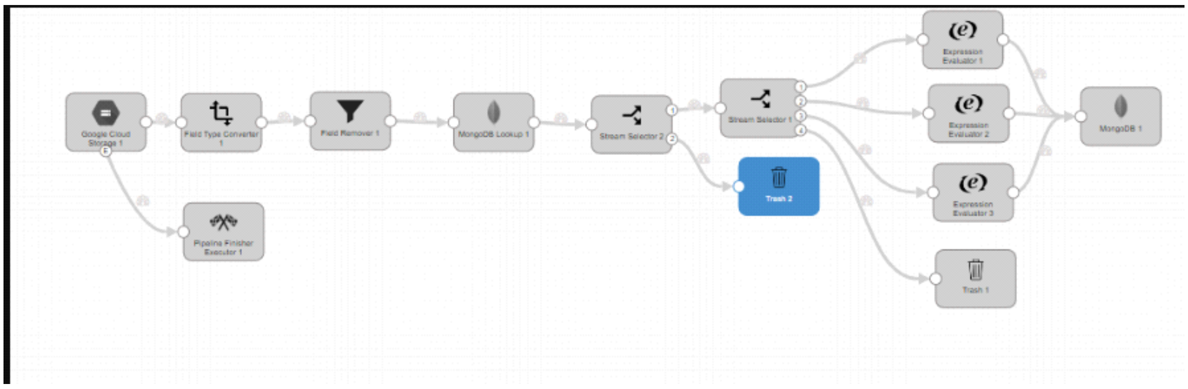
Condition	1	`${record:value('/result')==NULL}`
	2	`default`

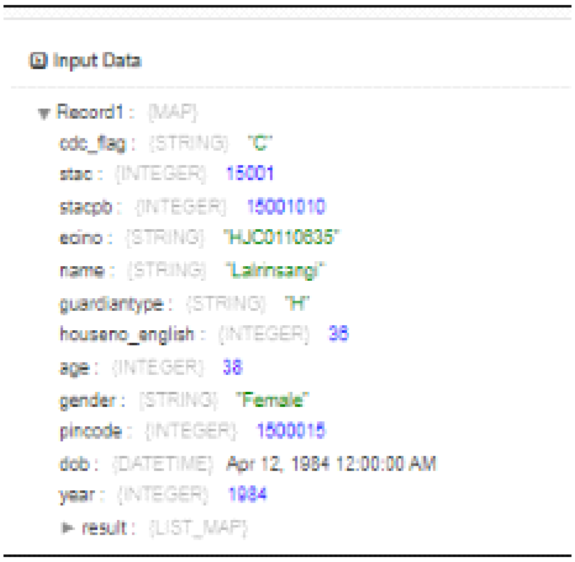
- First time the records went to Stream Selector.



```
Input Data
▾ Record1 :  {MAP}
    cdc_flag :  {STRING}  "C"
    stac :  {INTEGER}  15001
    stacpb :  {INTEGER}  15001010
    ecino :  {STRING}  "HJC0110635"
    name :  {STRING}  "Lalrinsangi"
    guardiantype :  {STRING}  "H"
    houseno_english :  {INTEGER}  36
    age :  {INTEGER}  38
    gender :  {STRING}  "Female"
    pincode :  {INTEGER}  1500015
    dob :  {DATETIME}  Apr 12, 1984 12:00:00 AM
    year :  {INTEGER}  1984
```

- Second time the records went to trash.

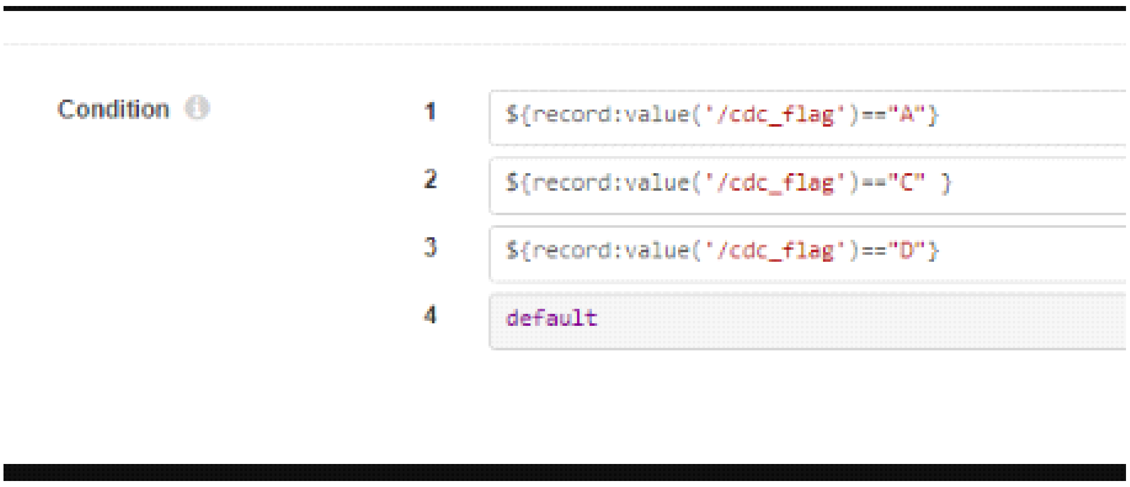




```
Input Data

▾ Record1 : {MAP}
    cdc_flag : {STRING}  "C"
    stac : {INTEGER}  15001
    stacpb : {INTEGER}  15001010
    ecino : {STRING}  "HJC0110635"
    name : {STRING}  "Lalrinsangi"
    guardiantype : {STRING}  "H"
    houseno_english : {INTEGER}  38
    age : {INTEGER}  38
    gender : {STRING}  "Female"
    pincode : {INTEGER}  1500015
    dob : {DATETIME}  Apr 12, 1984 12:00:00 AM
    year : {INTEGER}  1984
  ▸ result : {LIST_MAP}
```

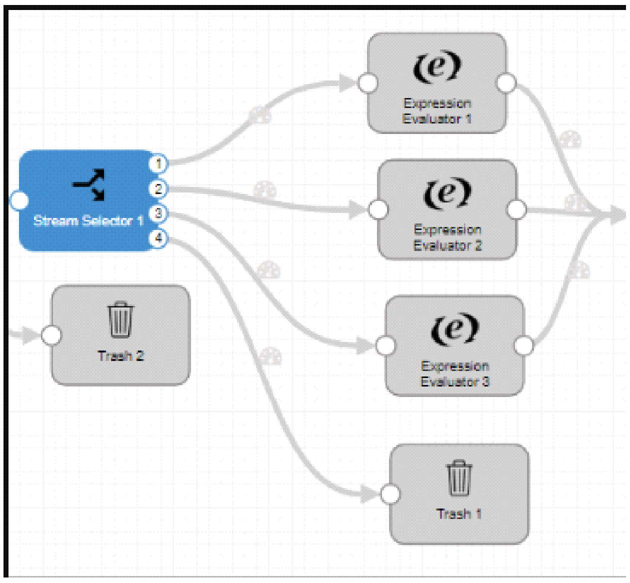
Performed operations based on values in the flag column:

- Used Stream selector for performing SDC operation.
- Based on "rollno_flag" it went to specific stream.
    - If "A" it performed insert operation.
    - If "C" it performed update operation
    - If "D" it performed delete operation



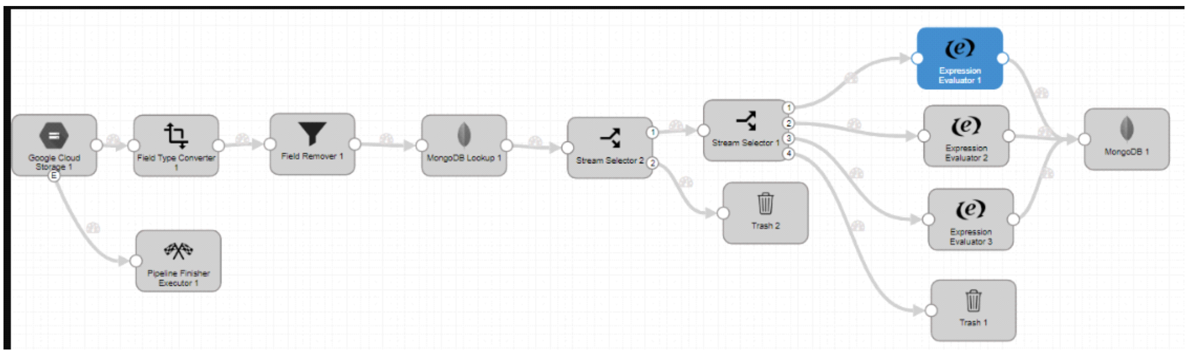
Condition

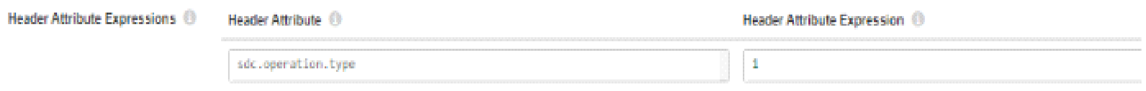
1	`${record:value('/cdc_flag')=="A"}`
2	`${record:value('/cdc_flag')=="C" }`
3	`${record:value('/cdc_flag')=="D"}`
4	`default`



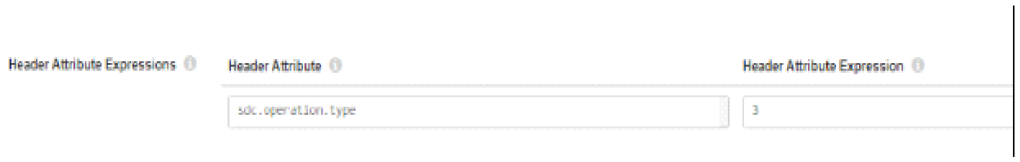
Processor Expression Evaluator:

- Used Expression Evaluator to perform specific SDC operation.
- If sdc.operation.type is 1 then it perform insert.
- If sdc.operation.type is 2 then it perform delete.
- If sdc.operation.type is 3 then it perform update.

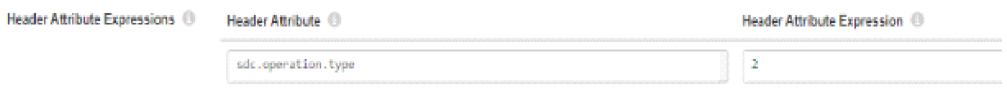




Header Attribute Expressions	Header Attribute	Header Attribute Expression
	sdc.operation.type	1



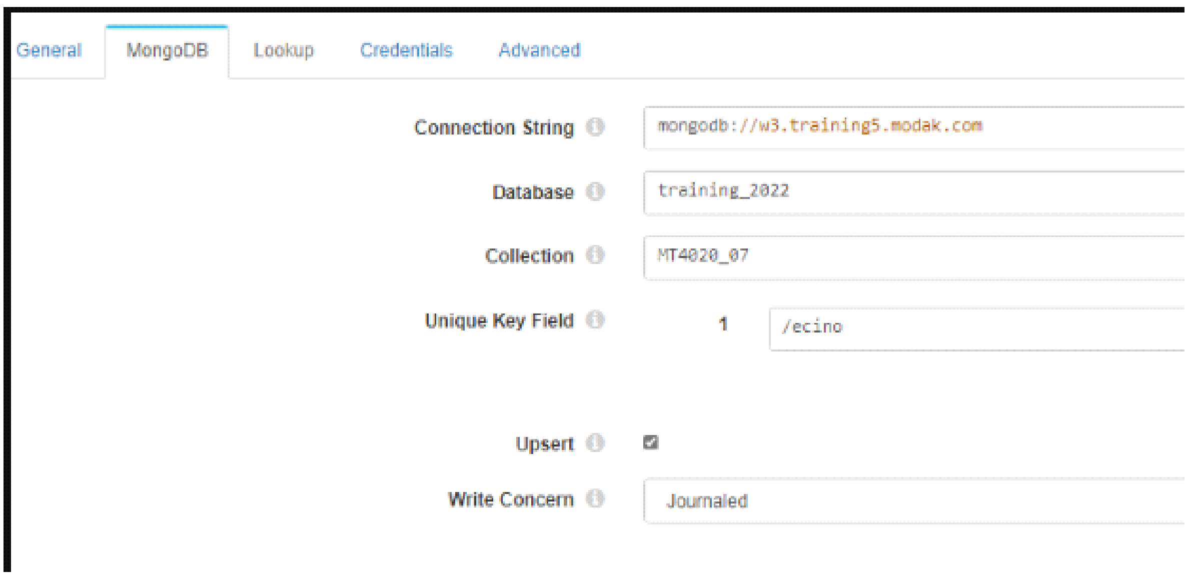
Header Attribute Expressions	Header Attribute	Header Attribute Expression
	sdc.operation.type	3



Header Attribute Expressions	Header Attribute	Header Attribute Expression
	sdc.operation.type	2

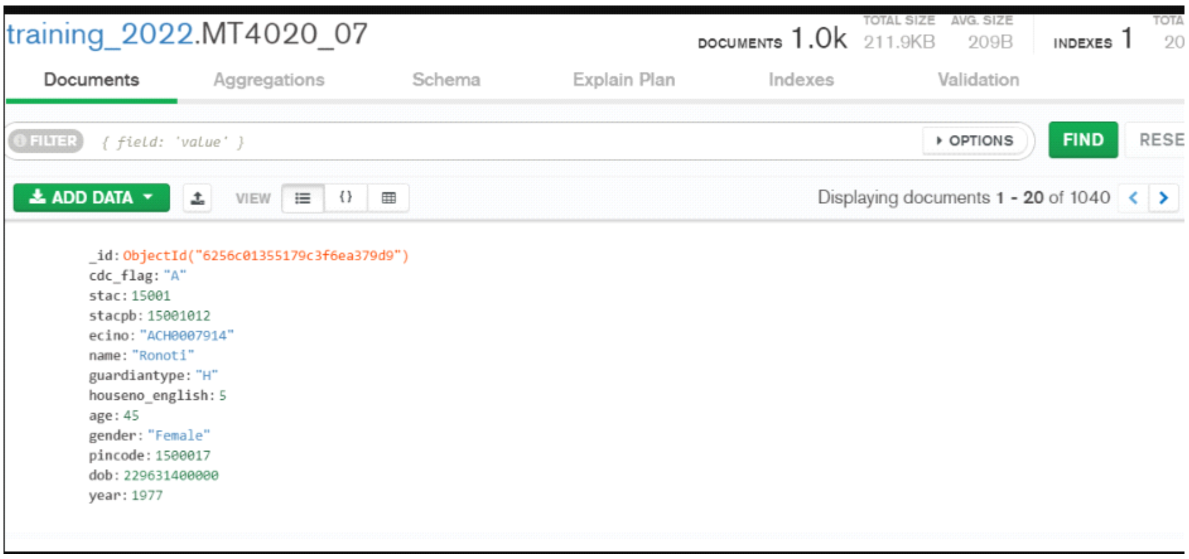
Destination:

- Used MongoDB as destination.
- Created new collection USERSTORY-7 in MongoDB.
- Provided Credentials and other information to connect to MongoDB.



Connection String	mongodb://w3.training5.modak.com
Database	training_2022
Collection	MT4020_07
Unique Key Field	1 /ecino
Upsert	✓
Write Concern	Journaled

- Data MongoDB



```
_id: ObjectId("6256c01355179c3f6ea379d9")
cdc_flag: "A"
stac: 15001
stacpb: 15001012
ecino: "ACH0007914"
name: "Ronoti"
guardiantype: "H"
houseno_english: 5
age: 45
gender: "Female"
pincode: 1500017
dob: 229631400000
year: 1977
```

USERSTORY-9

- **HTTP Client - Get data over HTTP and produce to a topic**

As a developer, process a GET/POST request and publish the response on a topic.

- Response will be in List of Maps.
- Produce each map as separate message.
- Topic-name to Publish - customer-status
- Rest service created at #10 should be used here as input.

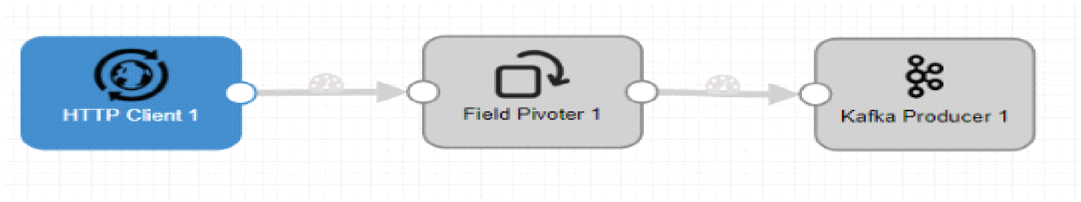
Acceptance Criteria : Data validated at the destination

- Userstory reviewed by the Technical Lead
- Azure board to be updated
- Approach and Implementation of user story should be documented

EXPLANATION:

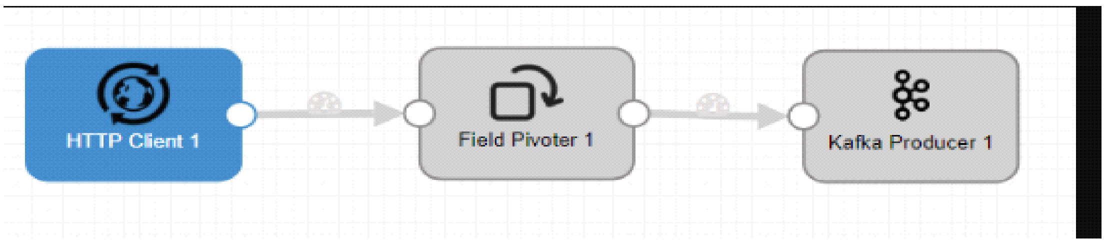
Pipeline:

- Here Origin is HTTP client
- Response will be in List of Maps. Used Field Pivoter to produce each map as separate message.
- Destination is Kafka Producer.

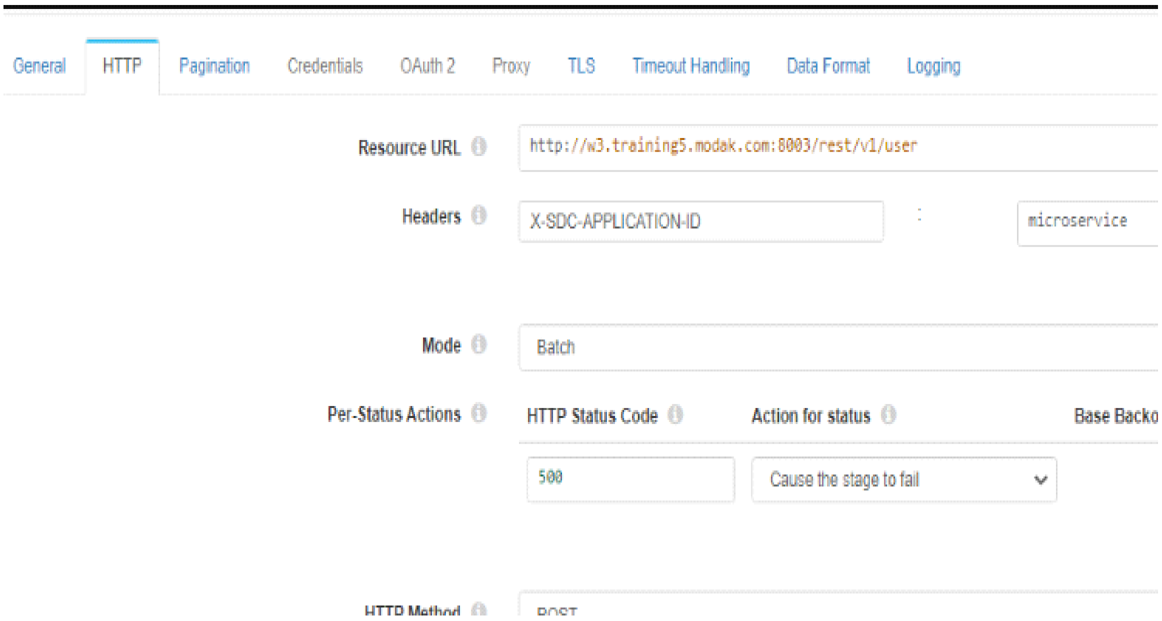


Origin HTTP Client :

- HTTP client origin reads data from an HTTP resource URL.



- Provided resource URL ,Header other details.



General | HTTP | Pagination | Credentials | OAuth 2 | Proxy | TLS | Timeout Handling | Data Format | Logging

Resource URL: http://w3.training5.modak.com:8003/rest/v1/user

Headers: X-SDC-APPLICATION-ID : microservice

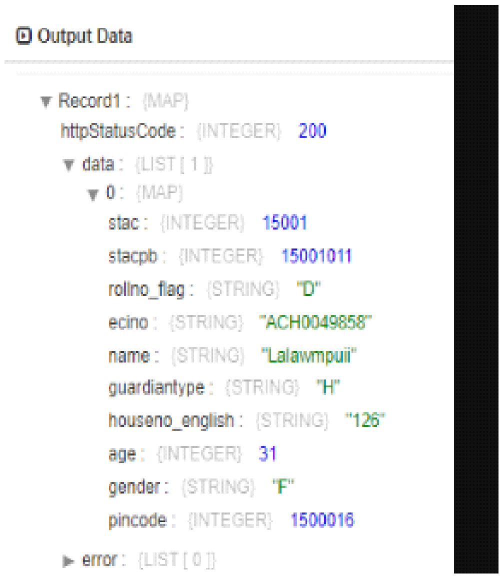
Mode: Batch

Per-Status Actions: HTTP Status Code | Action for status | Base Backo

500 | Cause the stage to fail

HTTP Method: POST

- **Resource URL :** URL where data resides.
- **Mode :** Used Batch as mode.
    - HTTP Client processes all available data and then stops the pipeline. Use to process data as needed.
- **HTTP Method :** HTTP method to use to request data from the server.
- **Output:**

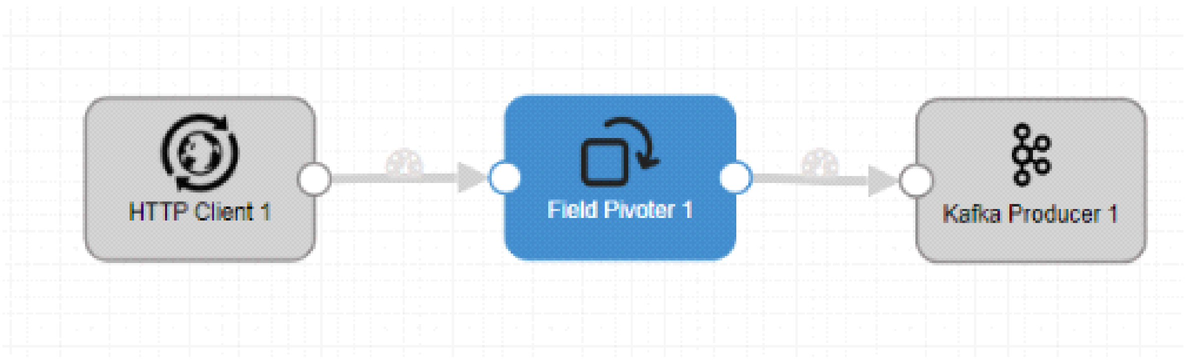


Output Data

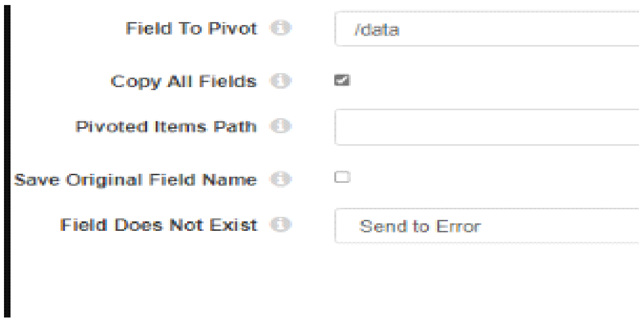
```
▼ Record1 : {MAP}
  httpStatusCode : {INTEGER} 200
  ▼ data : {LIST [ 1 ]}
    ▼ 0 : {MAP}
        stac : {INTEGER} 15001
        stacpb : {INTEGER} 15001011
        rollno_flag : {STRING} "D"
        ecino : {STRING} "ACH0049858"
        name : {STRING} "Lalawmpuii"
        guardiantype : {STRING} "H"
        houseno_english : {STRING} "126"
        age : {INTEGER} 31
        gender : {STRING} "F"
        pincode : {INTEGER} 1500016
  ► error : {LIST [ 0 ]}
```

Processor:

- Response will be in List of Maps. Used Field Pivoter to produce each map as separate message.

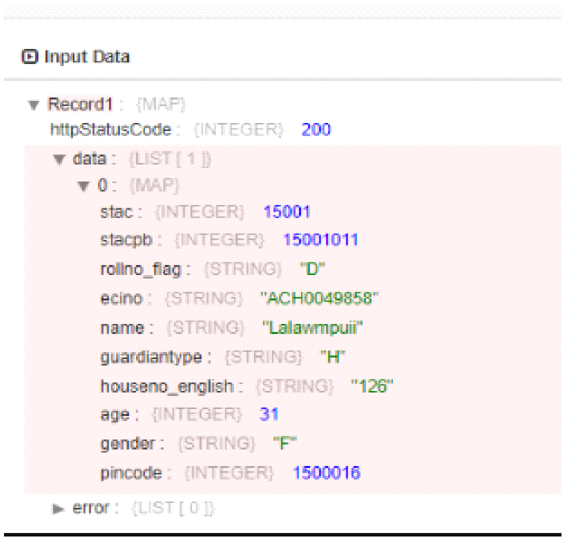


- Provided required fields.
- The Field Pivoter pivots data in a list, map or list-map fields and Creates a records for each item in the field.

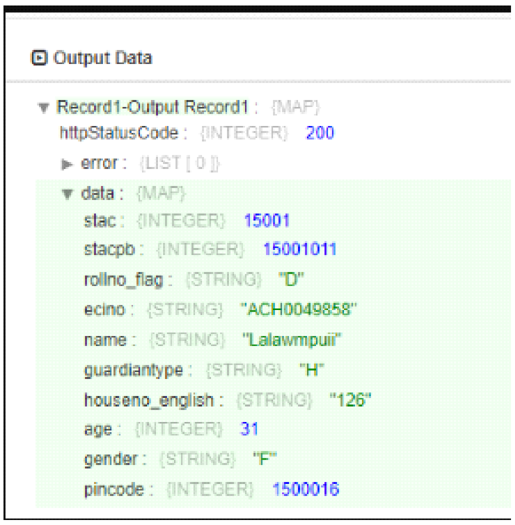


Field To Pivot	/data
Copy All Fields	✓
Pivoted Items Path	
Save Original Field Name	☐
Field Does Not Exist	Send to Error

- **Input and output:** Here it mapped list-map to map.



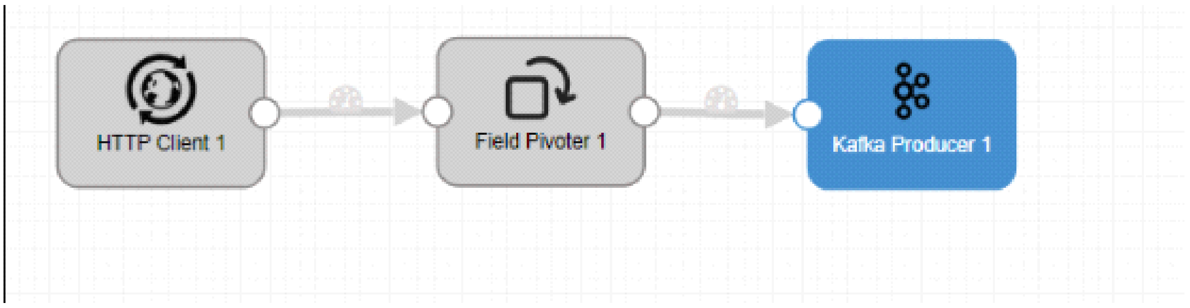
```
Input Data
▼ Record1 : {MAP}
  httpStatusCode : {INTEGER}  200
  ▼ data : {LIST [ 1 ]}
    ▼ 0 : {MAP}
       stac : {INTEGER}  15001
       stacpb : {INTEGER}  15001011
       rollno_flag : {STRING}  "D"
       ecino : {STRING}  "ACH0049858"
       name : {STRING}  "Lalawmpuii"
       guardiantype : {STRING}  "H"
       houseno_english : {STRING}  "126"
       age : {INTEGER}  31
       gender : {STRING}  "F"
       pincode : {INTEGER}  1500016
  ► error : {LIST [ 0 ]}
```



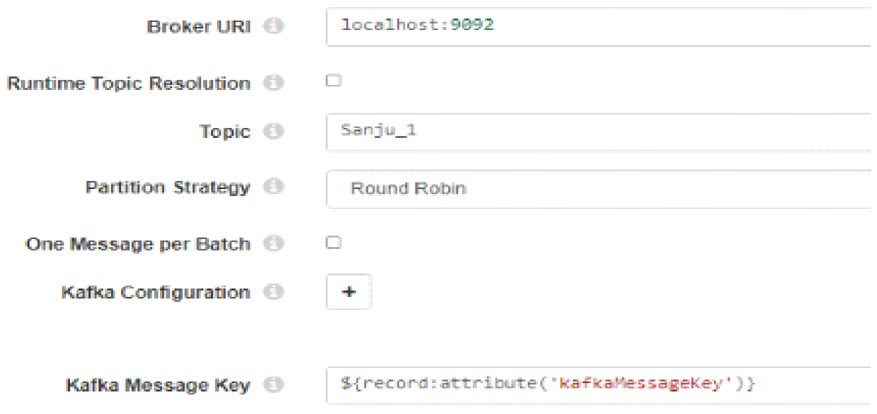
```
Output Data

▼ Record1-Output Record1 : {MAP}
  httpStatusCode : {INTEGER} 200
  ▶ error : {LIST [ 0 ]}
  ▼ data : {MAP}
    stac : {INTEGER} 15001
    stacpb : {INTEGER} 15001011
    rollno_flag : {STRING} "D"
    ecino : {STRING} "ACH0049858"
    name : {STRING} "Lalawmpuii"
    guardiantype : {STRING} "H"
    houseno_english : {STRING} "126"
    age : {INTEGER} 31
    gender : {STRING} "F"
    pincode : {INTEGER} 1500016
```

Destination:

- Used kafka Producer as destination.

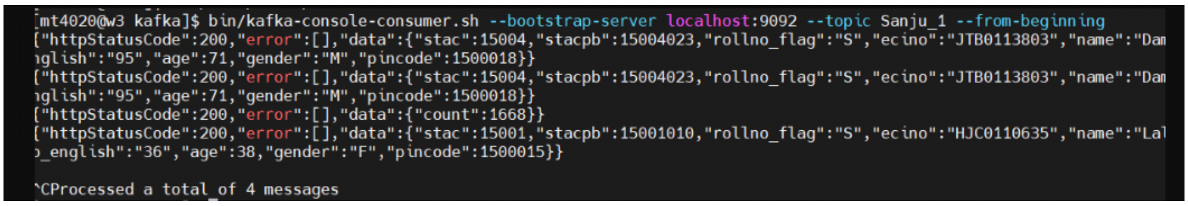


- Provided all the fields required.



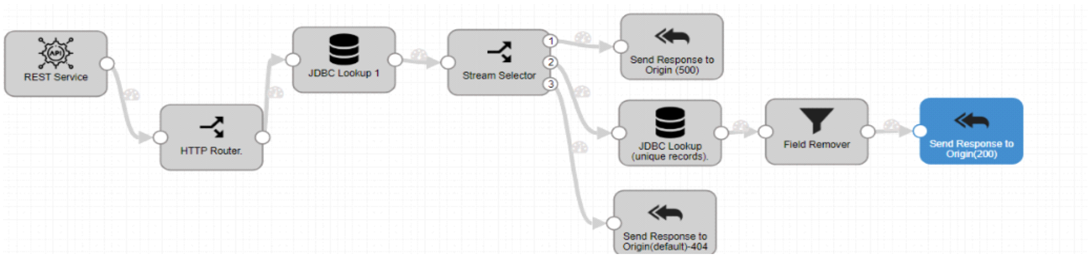
Broker URI	localhost:9092
Runtime Topic Resolution	☐
Topic	Sanju_1
Partition Strategy	Round Robin
One Message per Batch	☐
Kafka Configuration	+
Kafka Message Key	${record:attribute('kafkaMessageKey')}

- Checked records in kafka :



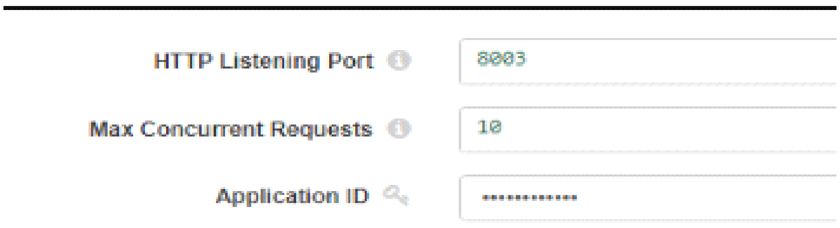
```
[mt4020@w3 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic Sanju_1 --from-beginning
{"httpStatusCode":200,"error":[],"data":{"stac":15004,"stacpb":15004023,"rollno_flag":"S","ecino":"JTB0113803","name":"Dam
nglish":"95","age":71,"gender":"M","pincode":1500018}}
{"httpStatusCode":200,"error":[],"data":{"stac":15004,"stacpb":15004023,"rollno_flag":"S","ecino":"JTB0113803","name":"Dam
nglish":"95","age":71,"gender":"M","pincode":1500018}}
{"httpStatusCode":200,"error":[],"data":{"count":1668}}
{"httpStatusCode":200,"error":[],"data":{"stac":15001,"stacpb":15001010,"rollno_flag":"S","ecino":"HJC0110635","name":"Lal
o_english":"36","age":38,"gender":"F","pincode":1500015}}

^CProcessed a total of 4 messages
```

The Dependency pipeline:


REST Service
HTTP Router.
JDBC Lookup 1
Stream Selector
Send Response to Origin (500)
JDBC Lookup (unique records).
Field Remover
Send Response to Origin(200)
Send Response to Origin(default)-404

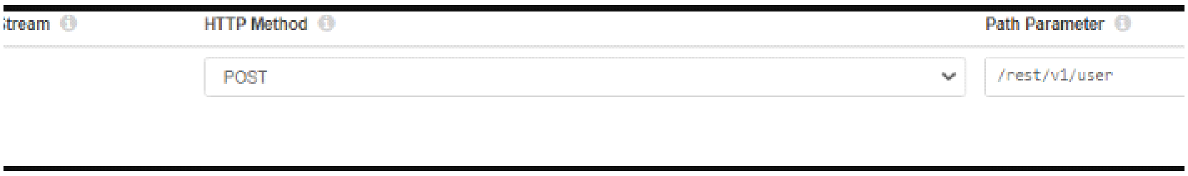
Explanation:

- **Rest Service** : The REST Service origin listens at an HTTP endpoint for requests.
- We have to give same listening port same as HTTP client.



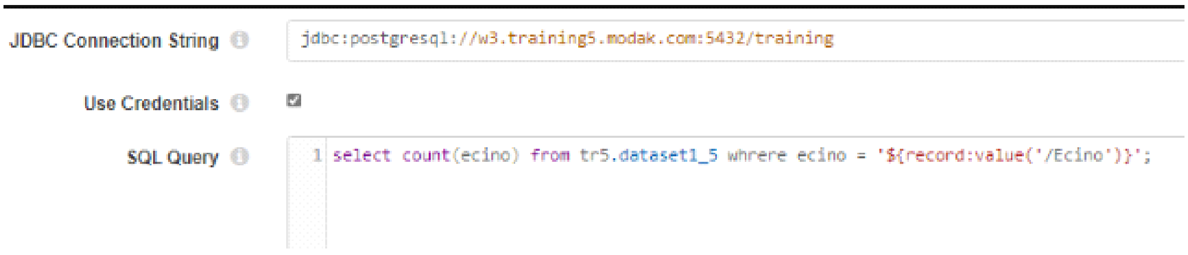
HTTP Listening Port	8003
Max Concurrent Requests	10
Application ID	•••••••••••••

- **HTTP Router :** The HTTP Router processor passes records to data streams based on the HTTP method and URL path in the record header attributes.



;tream	HTTP Method	Path Parameter
	POST	/rest/v1/user

- **JDBC Lookup :** The JDBC Lookup processor uses a JDBC connection to perform loookups in a database table and pass the lookup values to fields.



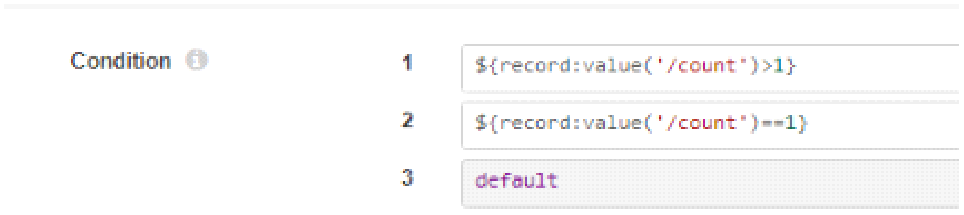
JDBC Connection String: jdbc:postgresql://w3.training5.modak.com:5432/training

Use Credentials: ✓

SQL Query:

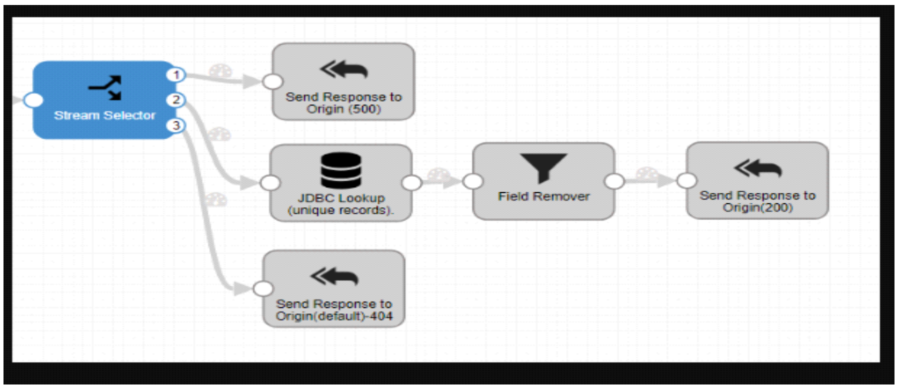
```
select count(ecino) from tr5.dataset1_5 whrere ecino = '${record:value('/Ecino')}';
```

- **Stream Selector :** Used to check whether it is a unique record or not.



Condition

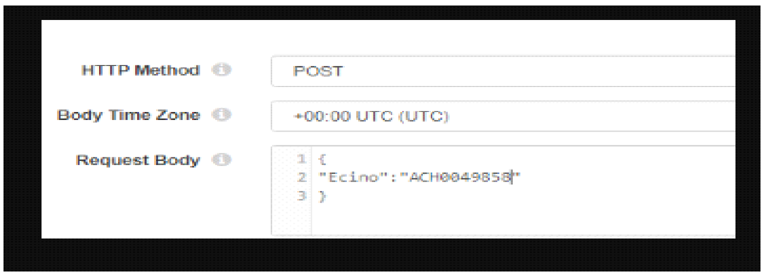
1	`${record:value('/count')>1}`
2	`${record:value('/count')==1}`
3	`default`


Stream Selector
1
2
3
Send Response to Origin (500)
JDBC Lookup (unique records)
Field Remover
Send Response to Origin(200)
Send Response to Origin(default)-404

- If it is a unique record it will send response to origin with status code 200.
- If it is not unique it will send Response to origin with status code 500.
- If it doesn't exists it will send Response to origin with status code 404.

Response when given unique record 'ecino' as Request:

- Requested with ecino which as count=1.



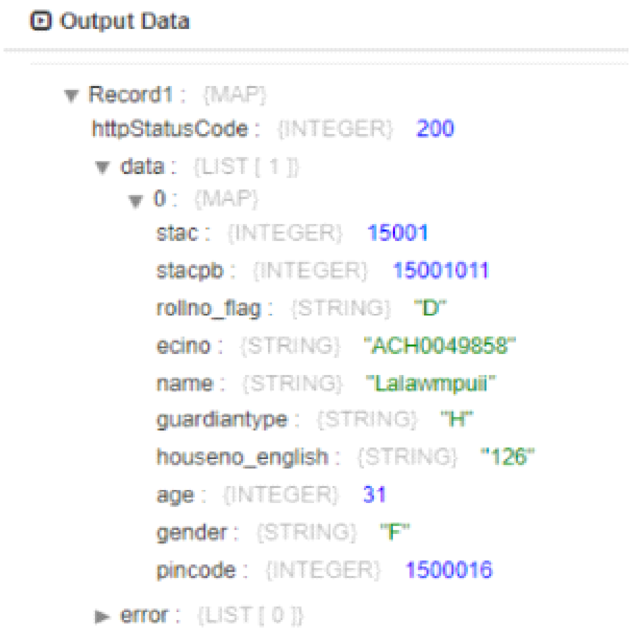
HTTP Method: POST

Body Time Zone: +00:00 UTC (UTC)

Request Body:

```
{
"Ecino":"ACH0049858"
}
```

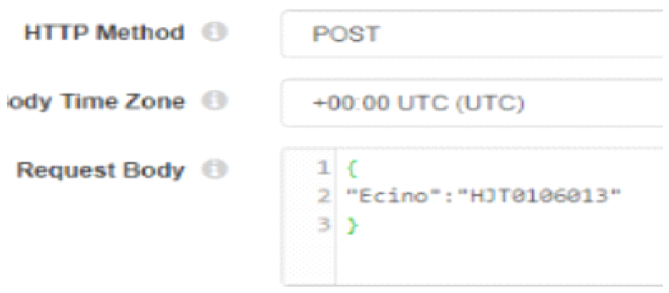
- Got data with httpStatus Code 200.



```
Output Data

▼ Record1 : {MAP}
  httpStatusCode : {INTEGER} 200
  ▼ data : {LIST [ 1 ]}
    ▼ 0 : {MAP}
        stac : {INTEGER} 15001
        stacpb : {INTEGER} 15001011
        rollno_flag : {STRING} "D"
        ecino : {STRING} "ACH0049858"
        name : {STRING} "Lalawmpuii"
        guardiantype : {STRING} "H"
        houseno_english : {STRING} "126"
        age : {INTEGER} 31
        gender : {STRING} "F"
        pincode : {INTEGER} 1500016
  ▶ error : {LIST [ 0 ]}
```

Response when not given unique record 'ecino' as Request :

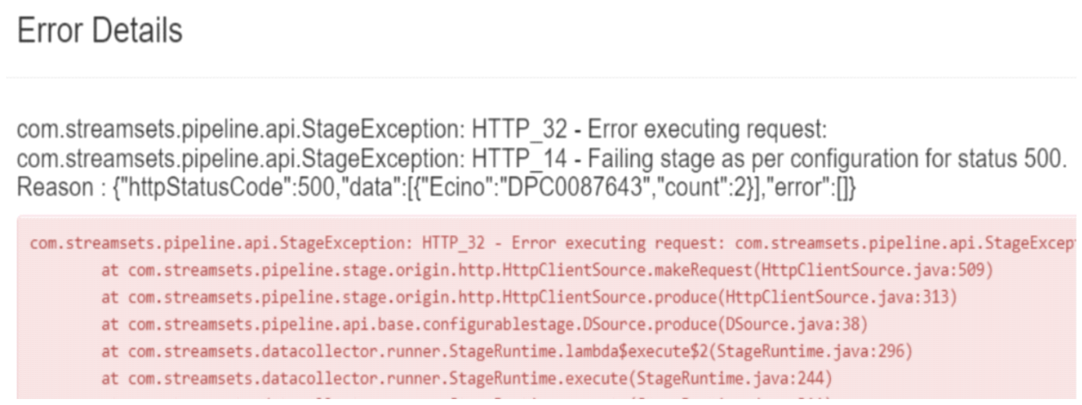
- Requested with ecino which as count>1.



HTTP Method	POST
ody Time Zone	+00:00 UTC (UTC)
Request Body	{ "Ecino":"HJT0106013" }

```
{
"Ecino":"HJT0106013"
}
```

- Got error message with status code 500.



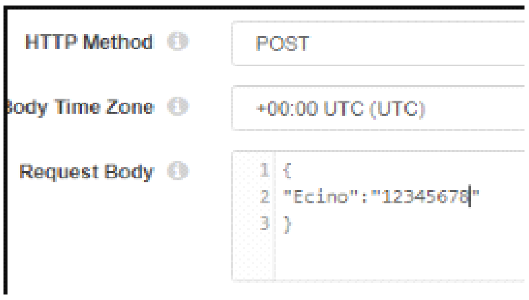
Error Details

com.streamsets.pipeline.api.StageException: HTTP_32 - Error executing request: com.streamsets.pipeline.api.StageException: HTTP_14 - Failing stage as per configuration for status 500. Reason : {"httpStatusCode":500,"data":[{"Ecino":"DPC0087643","count":2}],"error":[]}

```
com.streamsets.pipeline.api.StageException: HTTP_32 - Error executing request: com.streamsets.pipeline.api.StageExcep
	at com.streamsets.pipeline.stage.origin.http.HttpClientSource.makeRequest(HttpClientSource.java:509)
	at com.streamsets.pipeline.stage.origin.http.HttpClientSource.produce(HttpClientSource.java:313)
	at com.streamsets.pipeline.api.base.configurablestage.DSource.produce(DSource.java:38)
	at com.streamsets.datacollector.runner.StageRuntime.lambda$execute$2(StageRuntime.java:296)
	at com.streamsets.datacollector.runner.StageRuntime.execute(StageRuntime.java:244)
```

Response when given ecino which is not in table :

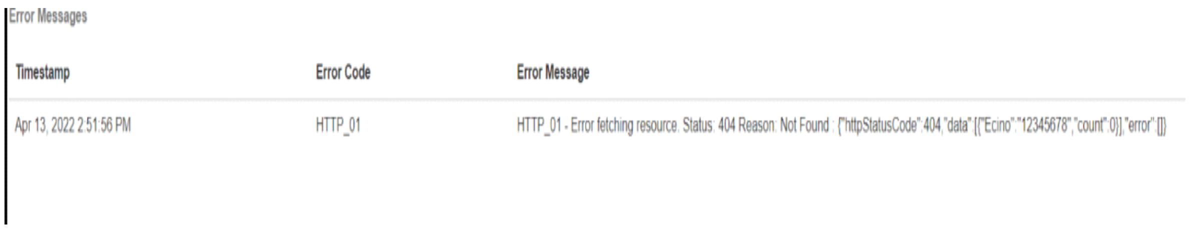
- Requested with ecino which is not in table.



HTTP Method	POST
Body Time Zone	+00:00 UTC (UTC)
Request Body	`{ "Ecino":"12345678" }`

```
{
"Ecino":"12345678"
}
```

- Got message with status code 404.



Error Messages

Timestamp	Error Code	Error Message
Apr 13, 2022 2:51:56 PM	HTTP_01	HTTP_01 - Error fetching resource. Status: 404 Reason: Not Found : {"httpStatusCode":404,"data":[{"Ecino":"12345678","count":0}],"error":[]}

SPIKE USER STORIES

Spike Story 1:

Explore Confluent Schema Registry basic concepts and document the observations.

Explanation:

Benefits of Schema Registry

Apache Kafka producers write data to Kafka topics and Kafka consumers read data from Kafkatopics. There is an implicit “contract” that producers write data with a schema that can be read byconsumers, even as producers and consumers evolve their schemas. Schema Registry helps ensure that this contract is met with compatibility checks.

It is useful to think about schemas as APIs. Applications depend on APIs and expect any changesmade to APIs are still compatible and applications can still run. Similarly, streaming applications depend on schemas and expect any changes made to schemas are still compatible and they can still run. Schema evolution requires compatibility checks to ensure that the producer-consumer contract is not broken. This is where Schema Registry helps: it provides centralized schema management and compatibility checks as schemas evolve.

Target Audience

The target audience is a developer writing Kafka streaming applications who wants to build a robust application leveraging Avro data and Schema Registry. The principles in this tutorial apply to any Kafka client that interacts with Schema Registry.

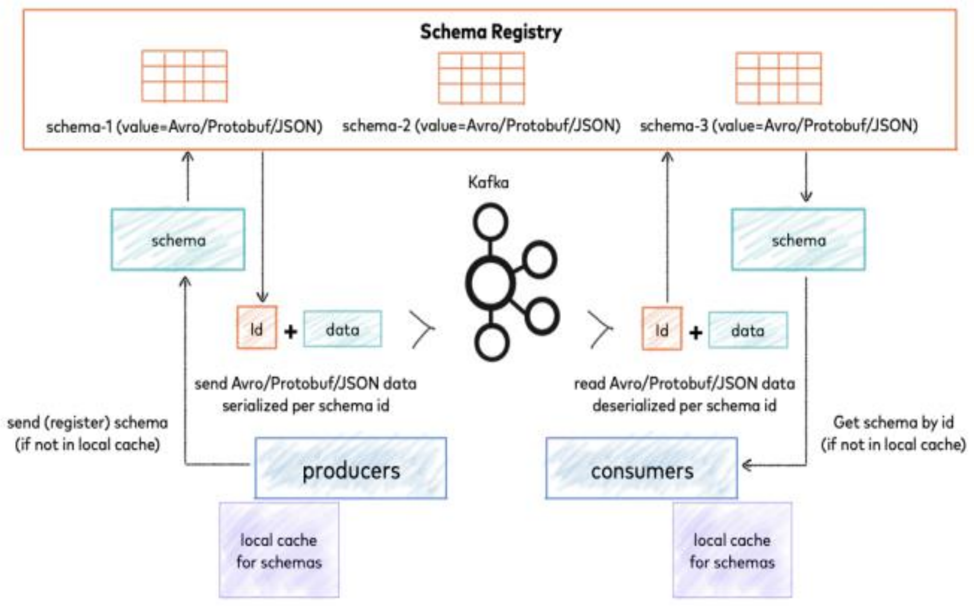
Confluent Schema Registry

It provides a serving layer for your metadata. It provides a RESTful interface for storing and retrieving your Avro®, JSON Schema, and Protobuf schemas. It stores a versioned history of all schemas based on a specified subject name strategy, provides multiple compatibility settings and allows evolution of schemas according to the configured compatibility settings and expanded support for these schema types. It provides serializers that plug into Apache Kafka® clients that handle schema storage and retrieval for Kafka messages that are sent in any of the supported formats.

Schema Registry lives outside of and separately from your Kafka brokers. Your producers and consumers still talk to Kafka to publish and read data (messages) to topics. Concurrently, they can also talk to Schema Registry to send and retrieve schemas that describe the data models for the messages.

Schema Registry is a distributed storage layer for schemas which uses Kafka as its underlying storage mechanism. Some key design decisions:

- Assigns globally unique ID to each registered schema. Allocated IDs are guaranteed to be monotonically increasing and unique, but not necessarily consecutive.
- Kafka provides the durable backend, and functions as a write-ahead changelog for the state of Schema Registry and the schemas it contains.
- Schema Registry is designed to be distributed, with single-primary architecture, and ZooKeeper/Kafka coordinates primary election (based on the configuration).


Schema Registry
schema-1 (value=Avro/Protobuf/JSON)
schema-2 (value=Avro/Protobuf/JSON)
schema-3 (value=Avro/Protobuf/JSON)
Kafka
schema
schema
Id
data
Id
data
send Avro/Protobuf/JSON data serialized per schema id
read Avro/Protobuf/JSON data deserialized per schema id
send (register) schema (if not in local cache)
Get schema by id (if not in local cache)
producers
consumers
local cache for schemas
local cache for schemas

Confluent Schema Registry for storing and retrieving schemas

Spike Story 2:

Explore Confluent Rest Proxy basic concepts and document the observations.

Explanation:

The Confluent REST Proxy provides a RESTful interface to a Apache Kafka® cluster, making it easy to produce and consume messages, view the state of the cluster, and perform administrative actions without using the native Kafka protocol or clients

The Admin REST APIs allow you to create and manage topics, manage MDS, and produce and consume to topics. The Admin REST APIs are available in these forms:

- You can deploy Confluent Server, which exposes Admin REST APIs directly on the brokers by default. Confluent Server is shipped with Confluent Enterprise.
- You can deploy standalone REST Proxy node(s), which in addition to Produce and Consume APIs, also offer Admin REST APIs as of API v3.
- Admin REST APIs are being incrementally added to Confluent Cloud, as documented at Confluent Cloud.

FEATURES:

The following functionality is currently exposed and available through Confluent REST APIs.

Metadata - Most metadata about the cluster – brokers, topics, partitions, and configs – can be read using GET requests for the corresponding URLs.

Producers - Instead of exposing producer objects, the API accepts produce requests targeted at specific topics or partitions and routes them all through a small pool of producers.

Producer configuration - Producer instances are shared, so configs cannot be set on a per-request basis. However, you can adjust settings globally by passing new producer settings in the REST Proxy configuration. For example, you might pass in the compression. Type option to enable site-wide compression to reduce storage and network overhead.

Consumers - Consumers are stateful and therefore tied to specific REST Proxy instances. Offset commit can be either automatic or explicitly requested by the user. Currently limited to one thread per consumer; use multiple consumers for higher throughput. The REST Proxy uses either the high level consumer (v1 api) or the new 0.9 consumer (v2 api) to implement consumer-groups that can read from topics. Note: the v1 API has been marked for deprecation.

Consumer configuration - Although consumer instances are not shared, they do share the underlying server resources. Therefore, limited configuration options are exposed via the API. However, you can adjust settings globally by-passing consumer settings in the REST Proxy configuration.

Data Formats - The REST Proxy can read and write data using JSON, raw bytes encoded with base64 or using JSON-encoded Avro, Protobuf, or JSON Schema. With Avro, Protobuf, or JSON Schema, schemas are registered and validated against Schema Registry.

REST Proxy Clusters and Load Balancing - The REST Proxy is designed to support multiple instances running together to spread load and can safely be run behind various load balancing mechanisms (e.g. round robin DNS, discovery services, load balancers) as long as instances are configured correctly.

Simple Consumer - The high-level consumer should generally be preferred. However, it is occasionally useful to use low-level read operations, for example to retrieve messages at specific offsets.

Admin operations - With the API v3, you can create or delete topics, and update or reset topic configurations. For hands-on examples, see the Confluent Admin REST

APIs demo. (To start the demo, clone the Confluent demo-scene repository from GitHub then follow the guide for the Confluent Admin REST APIs demo.)

Just as important, here’s a list of features that aren’t yet supported:

Multi-topic Produce Requests - Currently each produce request may only address a single topic or topic-partition. Most use cases do not require multi-topic produce requests, they introduce additional complexity into the API, and clients can easily split data across multiple requests if necessary

Most Producer/Consumer Overrides in Requests - Only a few key overrides are exposed in the API (but global overrides can be set by the administrator). The reason is two-fold. First, proxies are multi-tenant and therefore most user-requested overrides need additional restrictions to ensure they do not impact other users. Second, tying the API too much to the implementation restricts future API improvements; this is especially important with the new upcoming consumer implementation.