

## **Kalyan Big Data Projects – Project 2**

### **How To Stream JSON Data Into Phoenix**

### **Using Apache Kafka**

#### **Pre-Requisites of Flume Project:**

hadoop-2.6.0  
kafka-0.9.0  
hbase-1.1.2  
phoenix-4.7.0  
java-1.7

**NOTE:** Make sure that install all the above components

#### **Flume Project Download Links:**

`hadoop-2.6.0.tar.gz` ==> [link](#)

(<https://archive.apache.org/dist/hadoop/core/hadoop-2.6.0/hadoop-2.6.0.tar.gz>)

`kafka\_2.11-0.9.0.0.tgz` ==> [link](#)

([https://archive.apache.org/dist/kafka/0.9.0.0/kafka\\_2.11-0.9.0.0.tgz](https://archive.apache.org/dist/kafka/0.9.0.0/kafka_2.11-0.9.0.0.tgz))

`hbase-1.1.2-bin.tar.gz` ==> [link](#)

(<https://archive.apache.org/dist/hbase/1.1.2/hbase-1.1.2-bin.tar.gz>)

`phoenix-4.7.0-HBase-1.1-bin.tar.gz` ==> [link](#)

(<https://archive.apache.org/dist/phoenix/phoenix-4.7.0-HBase-1.1/bin/phoenix-4.7.0-HBase-1.1-bin.tar.gz>)

`kalyan-kafka-consumer-json.properties` ==> [link](#)

(<https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/kafka/project2-phoenix-csv/kalyan-kafka-consumer-json.properties>)

`kalyan-bigdata-examples.jar` ==> [link](#)

(<https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/kalyan/kalyan-bigdata-examples.jar>)

`kalyan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar` ==> [link](#)

(<https://github.com/kalyanhadooptraining/kalyan-bigdata-realtime-projects/blob/master/kalyan/kalyan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar>)

`json-path-2.2.0.jar` ==> [link](#)

(<http://central.maven.org/maven2/com/jayway/jsonpath/json-path/2.2.0/json-path-2.2.0.jar>)

`commons-io-2.4.jar` ==> [link](#)

(<http://central.maven.org/maven2/commons-io/commons-io/2.4/commons-io-2.4.jar>)

---

**Learnings of this Project:**

---

- We will learn Kafka Configurations and Commands
  - Kafka Information
    1. Kalyan Util (JSON data generator)
    2. Kafka Producer (Listen on JSON data)
    3. Kafka Consumer (Receives the data from Kafka Producer)
    4. Phoenix Consumer (Write the data into Phoenix Table)
  - Major project in Real Time `Product Log Analysis`
    1. We are extracting the data from server logs
    2. This data will be useful to do analysis on product views
    3. JSON is the output format
  - We can use phoenix to analyze this data
- 

1. create "**kalyan-kafka-consumer-json.properties**" file with below content in **"/home/orienit"** folder

```
serializer=json
serializer.columnMapping = {"userid":"userid", "username":"username", "password":"password",
"email":"email", "country":"country", "state":"state", "city":"city", "dt":"dt"}
serializer.partialSchema = true
serializer.columns = userid,username,password,email,country,state,city,dt
```

```
jdbcUrl=jdbc:phoenix:localhost
table=users5
ddl=CREATE TABLE IF NOT EXISTS users5 (userid BIGINT NOT NULL, username
VARCHAR, password VARCHAR, email VARCHAR, country VARCHAR, state VARCHAR, city
VARCHAR, dt VARCHAR NOT NULL CONSTRAINT PK PRIMARY KEY (userid, dt))
```

```
bootstrap.servers=localhost:9092
topics=json-topic1,json-topic2
poll.timeout.ms=100
```

2. Copy "**kalyan-kafka-consumer-json.properties**" file into hdfs location is **"/kalyan/kafka"**

- Create **`/kalyan/kafka`** folder in hdfs using below command
  - ◆ `hadoop fs -mkdir -p /kalyan/kafka`
- Put the properties file into hdfs using below command
  - ◆ `hadoop fs -put /home/orienit/kalyan-kafka-consumer-json.properties /kalyan/kafka/kalyan-kafka-consumer-json.properties`

3. Copy "**kalyan-phoenix-kafka-4.7.0-Hbase-1.1-minimal.jar**" file into **"\$PHONIEX\_HOME"** folder

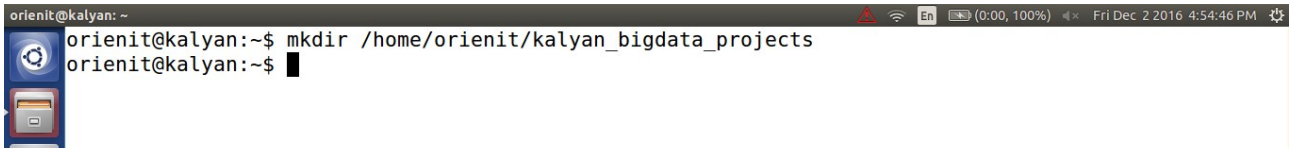
4. Generate Large Amount of Sample JSON data follow this [article](#).

(<http://kalyanbigdatatraining.blogspot.com/2016/12/how-to-generate-large-amount-of-sample.html>)

5. Follow below steps...

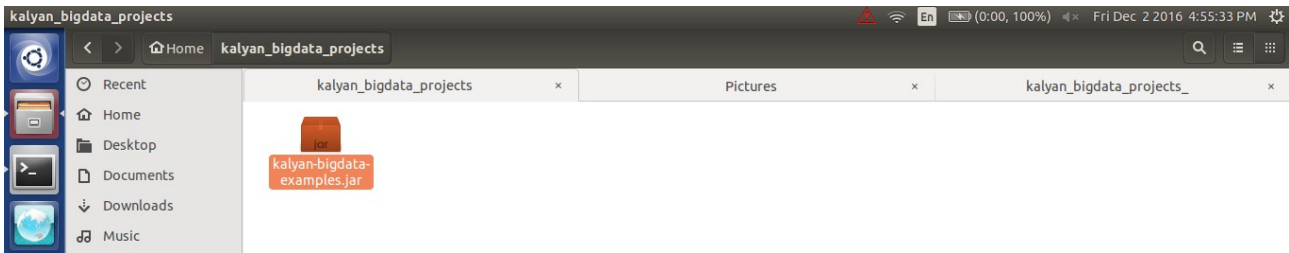
i) Create '**kalyan\_bigdata\_projects**' folder in **user home** (i.e **/home/orienit**)

**Command:** `mkdir /home/orienit/kalyan_bigdata_projects`



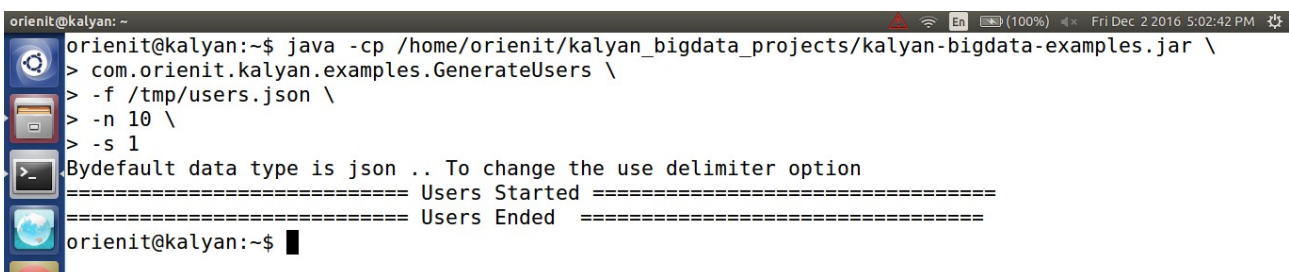
```
orienit@kalyan: ~  
orienit@kalyan:~$ mkdir /home/orienit/kalyan_bigdata_projects  
orienit@kalyan:~$
```

ii) Copy '**kalyan-bigdata-examples.jar**' jar file into '**/home/orienit/kalyan\_bigdata\_projects**' folder



iii) Execute below command to Generate Sample JSON data with 10 lines. Increase this number to get more data ...

```
java -cp /home/orienit/kalyan_bigdata_projects/kalyan-bigdata-examples.jar \  
com.orienit.kalyan.examples.GenerateUsers \  
-f /tmp/users.json \  
-n 10 \  
-s 1
```



```
orienit@kalyan: ~  
orienit@kalyan:~$ java -cp /home/orienit/kalyan_bigdata_projects/kalyan-bigdata-examples.jar \  
> com.orienit.kalyan.examples.GenerateUsers \  
> -f /tmp/users.json \  
> -n 10 \  
> -s 1  
By default data type is json .. To change the use delimiter option  
===== Users Started =====  
===== Users Ended =====  
orienit@kalyan:~$
```

6. Verify the Sample JSON data in Console, using below command

`cat /tmp/users.json`

```
orienit@kalyan: ~$ cat /tmp/users.json
{"userid":1,"username":"user1","password":"user1","email":"user1@gmail.com","country":"India","state":"Andhra Pradesh","city":"Guntur","dt":"2016-02-02 05:02:39"}
{"userid":2,"username":"user2","password":"user2","email":"user2@gmail.com","country":"US","state":"Washington","city":"Renton","dt":"2016-02-02 05:02:39"}
{"userid":3,"username":"user3","password":"user3","email":"user3@gmail.com","country":"US","state":"Hawaii","city":"Hanapepe","dt":"2016-02-02 05:02:39"}
{"userid":4,"username":"user4","password":"user4","email":"user4@gmail.com","country":"US","state":"Washington","city":"Bellingham","dt":"2016-02-02 05:02:39"}
{"userid":5,"username":"user5","password":"user5","email":"user5@gmail.com","country":"India","state":"Andhra Pradesh","city":"Kakinada","dt":"2016-02-02 05:02:39"}
{"userid":6,"username":"user6","password":"user6","email":"user6@gmail.com","country":"India","state":"Telangana","city":"Karimnagar","dt":"2016-02-02 05:02:39"}
{"userid":7,"username":"user7","password":"user7","email":"user7@gmail.com","country":"US","state":"New York","city":"Albany","dt":"2016-02-02 05:02:40"}
{"userid":8,"username":"user8","password":"user8","email":"user8@gmail.com","country":"India","state":"Karnataka","city":"Bidar","dt":"2016-02-02 05:02:40"}
{"userid":9,"username":"user9","password":"user9","email":"user9@gmail.com","country":"India","state":"Chennai","city":"Virugambakkam","dt":"2016-02-02 05:02:40"}
{"userid":10,"username":"user10","password":"user10","email":"user10@gmail.com","country":"US","state":"Hawaii","city":"Honolulu","dt":"2016-02-02 05:02:40"}
orienit@kalyan:~$
```

7. To work with **Kafka + Phoenix Integration**, Follow the below steps

i. Start the hbase using below '**start-hbase.sh**' command.

```
orienit@kalyan: ~$ start-hbase.sh
localhost: starting zookeeper, logging to /home/orienit/work/hbase-0.98.4-hadoop2/bin/./logs/hbase-orientit-zookeeper-kalyan.out
starting master, logging to /home/orienit/work/hbase-0.98.4-hadoop2/logs/hbase-orientit-master-kalyan.out
localhost: starting regionserver, logging to /home/orienit/work/hbase-0.98.4-hadoop2/bin/./logs/hbase-orientit-regionserver-kalyan.out
orienit@kalyan:~$
```

ii. verify the hbase is running or not with "**jps**" command

```
orienit@kalyan: ~$ jps
13904 DataNode
24529 HQuorumPeer
24835 HRegionServer
14259 ResourceManager
24596 HMaster
13749 NameNode
20725 Application
14392 NodeManager
14104 SecondaryNameNode
25486 Jps
7183 org.eclipse.equinox.launcher_1.3.200.v20160318-1642.jar
orienit@kalyan:~$
```

iii. Start the phoenix using below '**sqlline.py localhost**' command.

```
orienit@kalyan: ~$ sqlline.py localhost
Setting property: [incremental, false]
Setting property: [isolation, TRANSACTION_READ_COMMITTED]
issuing: !connect jdbc:phoenix:localhost none none org.apache.phoenix.jdbc.PhoenixDriver
Connecting to jdbc:phoenix:localhost
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/orienit/work/phoenix-4.7.0-HBase-1.1-bin/phoenix-4.7.0-HBase-1.1-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
16/10/06 17:25:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Connected to: Phoenix (version 4.7)
Driver: PhoenixEmbeddedDriver (version 4.7)
Autocommit status: true
Transaction isolation: TRANSACTION_READ_COMMITTED
Building list of tables and columns for tab-completion (set fastconnect to true to skip)...
83/83 (100%) Done
Done
sqlline version 1.1.8
0: jdbc:phoenix:localhost>
```

iv. List out all the tables in phoenix using '!tables' command

```
0: jdbc:phoenix:localhost> !tables
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENC
	SYSTEM	CATALOG	SYSTEM TABLE			
	SYSTEM	FUNCTION	SYSTEM TABLE			
	SYSTEM	SEQUENCE	SYSTEM TABLE			
	SYSTEM	STATS	SYSTEM TABLE			

```
0: jdbc:phoenix:localhost>
```

v. Start the `kafka server` using below command (New Terminal)  
\$KAFKA\_HOME/bin/kafka-server-start.sh \$KAFKA\_HOME/config/server.properties

```
orienit@kalyan: ~$ $KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server.properties
[2017-01-03 16:43:14,430] INFO KafkaConfig values:
  advertised.host.name = null
  metric.reporters = []
  quota.producer.default = 9223372036854775807
  offsets.topic.num.partitions = 50
  log.flush.interval.messages = 9223372036854775807
  auto.create.topics.enable = true
  controller.socket.timeout.ms = 30000
  log.flush.interval.ms = null
  principal.builder.class = class org.apache.kafka.common.security.auth.DefaultPrincipalBuild
er
  replica.socket.receive.buffer.bytes = 65536
  min.insync.replicas = 1
  replica.fetch.wait.max.ms = 500
  num.recovery.threads.per.data.dir = 1
  ssl.keystore.type = JKS
  default.replication.factor = 1
  ssl.truststore.password = null
  log.preallocate = false
```



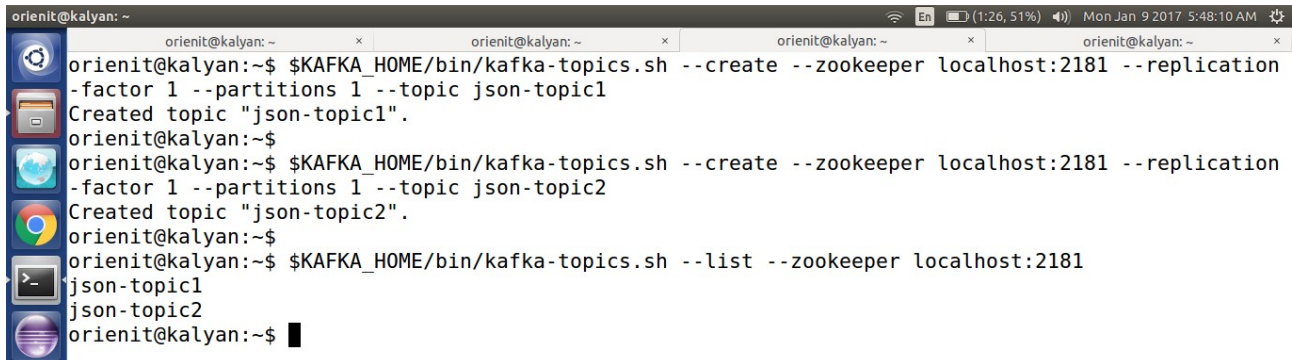
- vi. Create a `json-topic1` & `json-topic2` topics using below command (New Terminal)

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic json-topic1
```

```
$KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic json-topic2
```

- vii. List out all the topics

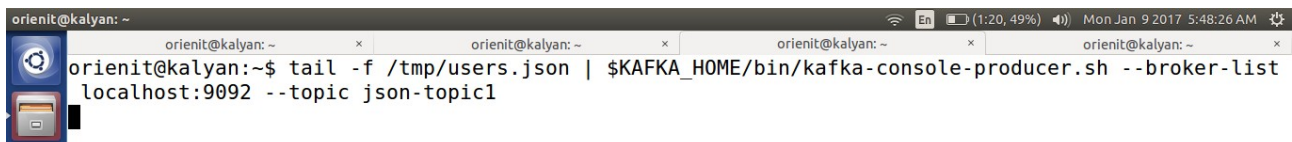
```
$KAFKA_HOME/bin/kafka-topics.sh --list --zookeeper localhost:2181
```



```
orienit@kalyan: ~  
orienit@kalyan:~$ $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic json-topic1  
Created topic "json-topic1".  
orienit@kalyan:~$  
orienit@kalyan:~$ $KAFKA_HOME/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic json-topic2  
Created topic "json-topic2".  
orienit@kalyan:~$  
orienit@kalyan:~$ $KAFKA_HOME/bin/kafka-topics.sh --list --zookeeper localhost:2181  
json-topic1  
json-topic2  
orienit@kalyan:~$
```

- viii. Start the `kafka producer` using below command (New Terminal)

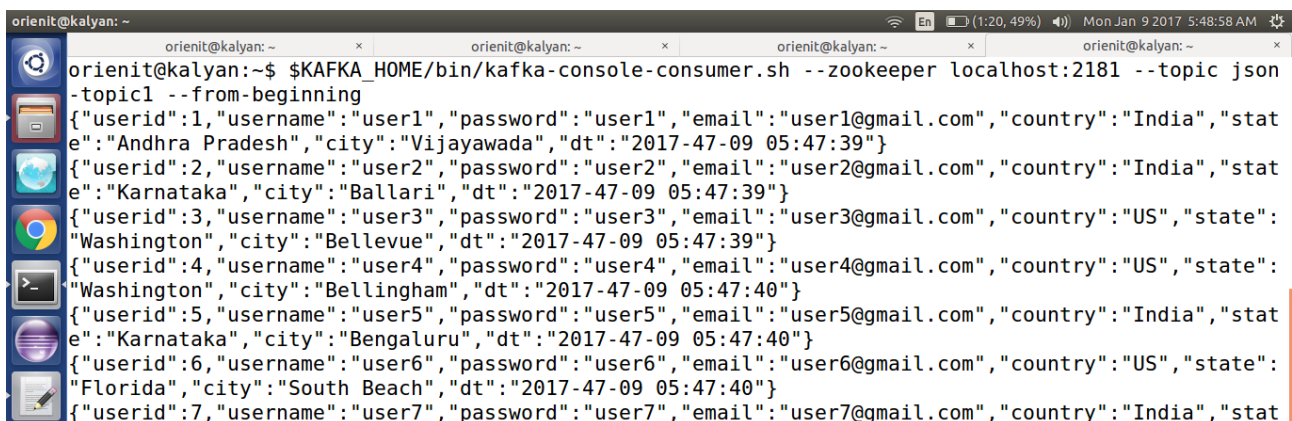
```
tail -f /tmp/users.json | $KAFKA_HOME/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic json-topic1
```



```
orienit@kalyan: ~  
orienit@kalyan:~$ tail -f /tmp/users.json | $KAFKA_HOME/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic json-topic1
```

- ix. Start the `kafka consumer` using below command (New Terminal)

```
$KAFKA_HOME/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic json-topic1 --from-beginning
```

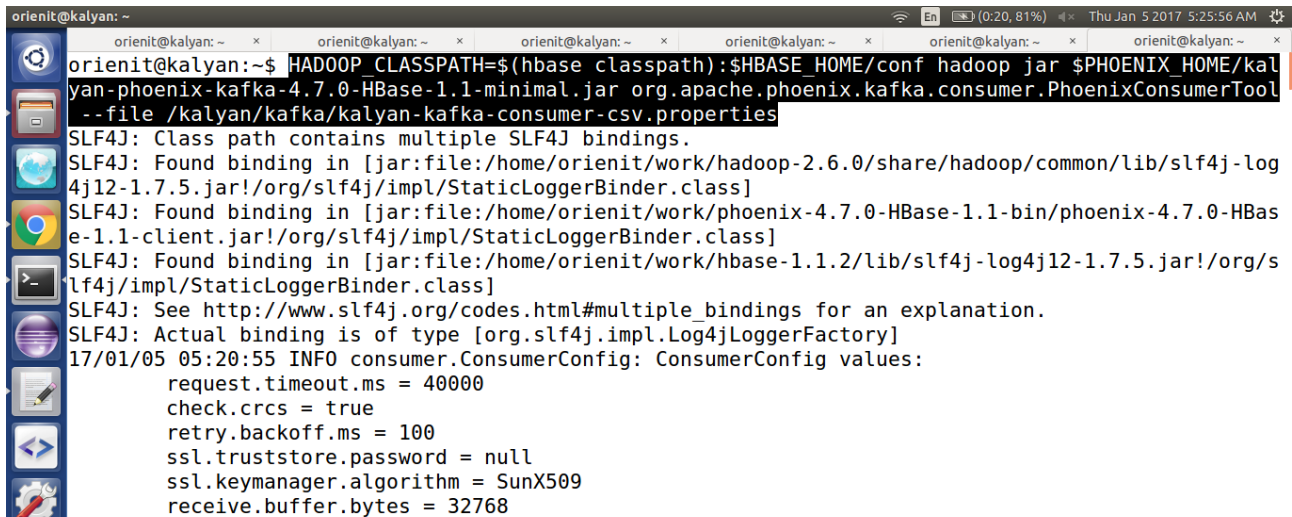


```
orienit@kalyan: ~  
orienit@kalyan:~$ $KAFKA_HOME/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic json-topic1 --from-beginning  
{ "userid":1, "username":"user1", "password":"user1", "email":"user1@gmail.com", "country":"India", "state":"Andhra Pradesh", "city":"Vijayawada", "dt":"2017-47-09 05:47:39" }  
{ "userid":2, "username":"user2", "password":"user2", "email":"user2@gmail.com", "country":"India", "state":"Karnataka", "city":"Ballari", "dt":"2017-47-09 05:47:39" }  
{ "userid":3, "username":"user3", "password":"user3", "email":"user3@gmail.com", "country":"US", "state":"Washington", "city":"Bellevue", "dt":"2017-47-09 05:47:39" }  
{ "userid":4, "username":"user4", "password":"user4", "email":"user4@gmail.com", "country":"US", "state":"Washington", "city":"Bellingham", "dt":"2017-47-09 05:47:40" }  
{ "userid":5, "username":"user5", "password":"user5", "email":"user5@gmail.com", "country":"India", "state":"Karnataka", "city":"Bengaluru", "dt":"2017-47-09 05:47:40" }  
{ "userid":6, "username":"user6", "password":"user6", "email":"user6@gmail.com", "country":"US", "state":"Florida", "city":"South Beach", "dt":"2017-47-09 05:47:40" }  
{ "userid":7, "username":"user7", "password":"user7", "email":"user7@gmail.com", "country":"India", "state":"Andhra Pradesh", "city":"Vijayawada", "dt":"2017-47-09 05:47:40" }
```

8. Execute the below command to `Extract data from JSON data into Phoenix using Kafka`

### Properties file from HDFS:

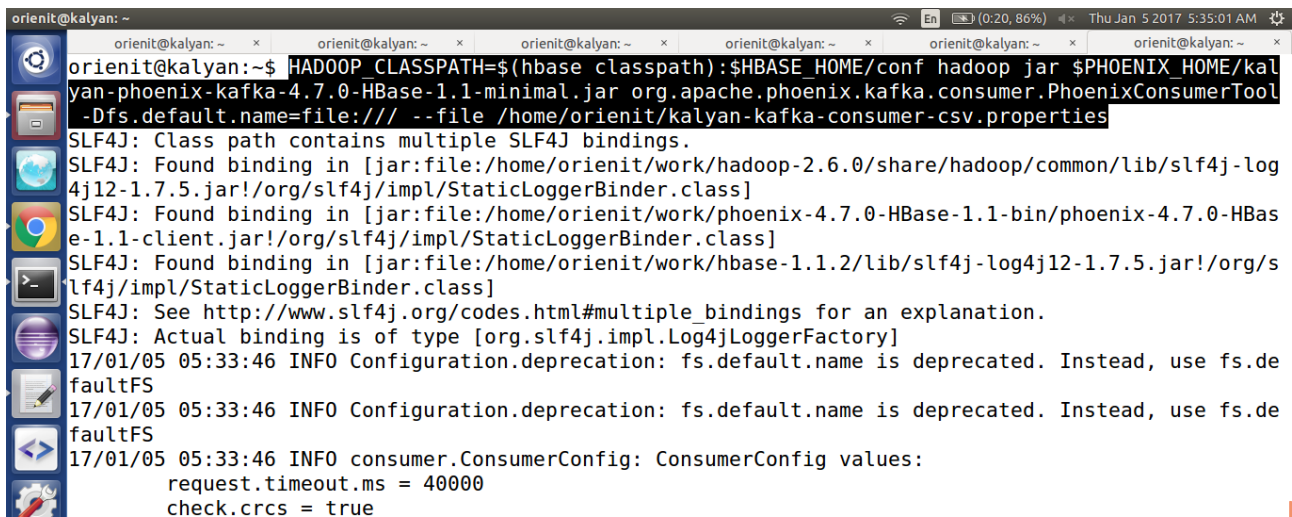
```
HADOOP_CLASSPATH=$(hbase classpath):$HBASE_HOME/conf hadoop jar  
$PHOENIX_HOME/kalyan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar  
org.apache.phoenix.kafka.consumer.PhoenixConsumerTool --file /kalyan/kafka/kalyan-kafka-  
consumer-json.properties
```



```
orienit@kalyan: ~$ HADOOP_CLASSPATH=$(hbase classpath):$HBASE_HOME/conf hadoop jar $PHOENIX_HOME/kal  
yan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar org.apache.phoenix.kafka.consumer.PhoenixConsumerTool  
--file /kalyan/kafka/kalyan-kafka-consumer-csv.properties  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log  
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/home/orienit/work/phoenix-4.7.0-HBase-1.1-bin/phoenix-4.7.0-HBas  
e-1.1-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/home/orienit/work/hbase-1.1.2/lib/slf4j-log4j12-1.7.5.jar!/org/s  
lf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
17/01/05 05:20:55 INFO consumer.ConsumerConfig: ConsumerConfig values:  
    request.timeout.ms = 40000  
    check.crcs = true  
    retry.backoff.ms = 100  
    ssl.truststore.password = null  
    ssl.keymanager.algorithm = SunX509  
    receive.buffer.bytes = 32768
```

### Properties file from Local Fs:

```
HADOOP_CLASSPATH=$(hbase classpath):$HBASE_HOME/conf hadoop jar  
$PHOENIX_HOME/kalyan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar  
org.apache.phoenix.kafka.consumer.PhoenixConsumerTool -Dfs.default.name=file:/// --file  
/home/orienit/kalyan-kafka-consumer-json.properties
```



```
orienit@kalyan: ~$ HADOOP_CLASSPATH=$(hbase classpath):$HBASE_HOME/conf hadoop jar $PHOENIX_HOME/kal  
yan-phoenix-kafka-4.7.0-HBase-1.1-minimal.jar org.apache.phoenix.kafka.consumer.PhoenixConsumerTool  
-Dfs.default.name=file:/// --file /home/orienit/kalyan-kafka-consumer-csv.properties  
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/home/orienit/work/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log  
4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/home/orienit/work/phoenix-4.7.0-HBase-1.1-bin/phoenix-4.7.0-HBas  
e-1.1-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]  
SLF4J: Found binding in [jar:file:/home/orienit/work/hbase-1.1.2/lib/slf4j-log4j12-1.7.5.jar!/org/s  
lf4j/impl/StaticLoggerBinder.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  
17/01/05 05:33:46 INFO Configuration.deprecation: fs.default.name is deprecated. Instead, use fs.de  
faultFS  
17/01/05 05:33:46 INFO Configuration.deprecation: fs.default.name is deprecated. Instead, use fs.de  
faultFS  
17/01/05 05:33:46 INFO consumer.ConsumerConfig: ConsumerConfig values:  
    request.timeout.ms = 40000  
    check.crcs = true
```

## 9. Verify the data in console

```
17/01/05 05:20:56 INFO metrics.Metrics: Initializing metrics system: phoenix
17/01/05 05:20:56 INFO impl.MetricsConfig: loaded properties from hadoop-metrics2.properties
17/01/05 05:20:56 INFO impl.MetricsSystemImpl: Scheduled snapshot period at 10 second(s).
17/01/05 05:20:56 INFO impl.MetricsSystemImpl: phoenix metrics system started
17/01/05 05:21:00 INFO client.HBaseAdmin: Created USERS4
17/01/05 05:21:00 INFO serializer.BaseEventSerializer: the upsert statement is UPSERT INTO users4
("USERID", "USERNAME", "PASSWORD", "EMAIL", "COUNTRY", "STATE", "CITY", "DT") VALUES (?, ?, ?, ?,
?, ?, ?, ?)
Got 10 records after 886 timeouts
```

## 10. Verify the data in Phoenix , `users5` table is created or not using below command

!tables

```
orientit@kalyan: ~
0: jdbc:phoenix:localhost> !tables
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENC
	SYSTEM	CATALOG	SYSTEM TABLE			
	SYSTEM	FUNCTION	SYSTEM TABLE			
	SYSTEM	SEQUENCE	SYSTEM TABLE			
	SYSTEM	STATS	SYSTEM TABLE			
		USERS5	TABLE			

```
0: jdbc:phoenix:localhost>
```

## 11. Execute below command to get the data from phoenix table 'users5'

select count(\*) from users5;

select \* from users5;

```
orientit@kalyan: ~
0: jdbc:phoenix:localhost> select count(*) from users5;
```

COUNT(1)
10

```
1 row selected (0.021 seconds)
0: jdbc:phoenix:localhost> select * from users5;
```

USERID	USERNAME	PASSWORD	EMAIL	COUNTRY	STATE	CITY
1	user1	user1	user1@gmail.com	India	Andhra Pradesh	Kakinada
2	user2	user2	user2@gmail.com	US	Florida	South Beach
3	user3	user3	user3@gmail.com	India	Telangana	Hyderabad
4	user4	user4	user4@gmail.com	India	Andhra Pradesh	Rajahmundry
5	user5	user5	user5@gmail.com	India	Telangana	Nalgonda
6	user6	user6	user6@gmail.com	India	Chennai	Tambaram
7	user7	user7	user7@gmail.com	India	Telangana	Karimnagar
8	user8	user8	user8@gmail.com	India	Karnataka	Mangaluru
9	user9	user9	user9@gmail.com	US	Hawaii	Pupukea
10	user10	user10	user10@gmail.com	US	Washington	Bellingham

```
10 rows selected (0.099 seconds)
0: jdbc:phoenix:localhost>
```