# Harvard University Extension School
# "Principles of Big Data Processing"
# CSCI E-88,
# Final Project Document.
# by Sayanti Bag

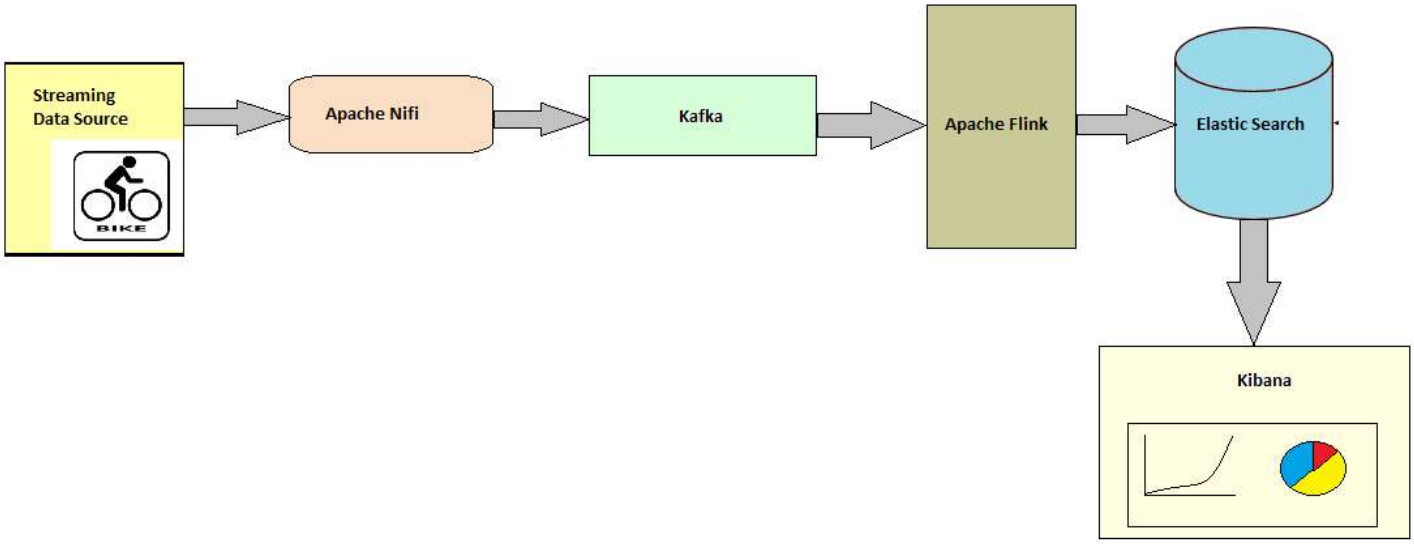
### Project Goal and Problem Statement

This project's goal is to demonstrate how one can build a system that collects Bike sharing Data by and indexes them into Elasticsearch for further analytics.

### Big Data Source

International Bike sharing data. This is a public data source from multiple bike sharing stations round the world. It contains data from ~50 cities such as: Portland, OR; Ottawa and Toronto CA; Abu Dhabi, and Prague - Brno, CZ. Data is published in open General Bikeshare Feed Specification (GBFS).

### Processing Pipeline




### Pipeline Overview and Technologies used

- Collection tier:  Apache Nifi with International Bike Sharing Channel data via Satori.
- Messaging Tier: Kafka
    - Apache Nifi will act as a Kafka Producer, delivering the data to a Kafka Sink.

● Stream Processing Tier: Apache Flink Streaming Framework.

- ♦ Apache Flink executes stream analysis jobs that processes a data stream, apply transformations to analyze, transform, and model the data in motion, and write their results to an Elasticsearch sink .

● Visualization Tier: Kibana will be used with Elasticsearch to visualize received data and discover and compare locations where bike sharing is most popular across the world.

## Summary of the Results

Using Kibana UI we can query the bike sharing data based on required criteria.

As per the data US has the highest number of bike rentals.

Capitalsharebike.com has the highest bike rentals.

## URL of the Youtube video

https://www.youtube.com/watch?v=IVmP-spNty0

## Solution and Implementation details

### Apache Nifi

### Download and Installation

Link - https://nifi.apache.org/download.html

Commands for installation and unpacking

```
wget http://mirrors.ibiblio.org/apache/nifi/1.4.0/nifi-1.4.0-bin.tar.gz
```

```
tar xvf nifi-1.4.0-bin.tar.gz
```

Satori is a cloud-based live platform that provides a publish-subscribe messaging service called RTM, and also makes available a set of free real-time data feeds as part of their Open Data Channels initiative:

https://www.satori.com/opendata/channels/US-Bike-Sharing-Channel

To consume from Satori's Open Data Channels in NiFi, I have used a custom NiFi processor.

### Installing the custom processor

To create the required nar file, simply clone and build the following repo with maven:

```
git clone https://github.com/laurencedaluz/nifi-satori-bundle.git
```

```
cd nifi-satori-bundle
```

```
mvn clean install
```

This created the below .nar file in the targer folder

```
[centos@ip-172-31-90-14 target]$ ls -lart
total 2700
drwxrwxr-x. 3 centos centos      22 Dec 10 22:03 classes
drwxrwxr-x. 3 centos centos      22 Dec 10 22:03 maven-shared-archive-resources
drwxrwxr-x. 3 centos centos      35 Dec 10 22:03 ..
-rw-rw-r--. 1 centos centos      30 Dec 10 22:03 .plxarc
drwxrwxr-x. 3 centos centos      22 Dec 10 22:03 test-classes
drwxrwxr-x. 2 centos centos      28 Dec 10 22:03 maven-archiver
drwxrwxr-x. 6 centos centos     163 Dec 10 22:03 .
-rw-rw-r--. 1 centos centos 2756635 Dec 10 22:03 nifi-satori-bundle-nar-1.0-SNAPSHOT.nar
[centos@ip-172-31-90-14 target]$
```

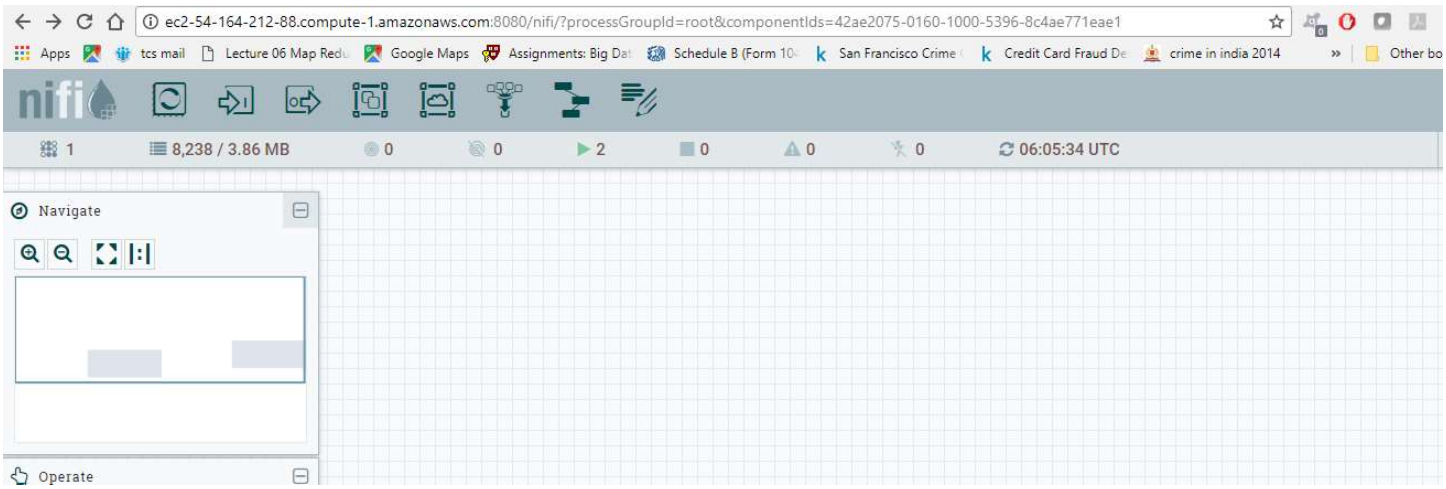
Copy this file into the lib directory of your NiFi instance

```
   cp /home/centos/nifi-satori-bundle/nifi-satori-bundle-nar/target/nifi-satori-bundle-nar-1.0-SNAPSHOT.nar /home/centos/nifi-1.4.0/lib
```

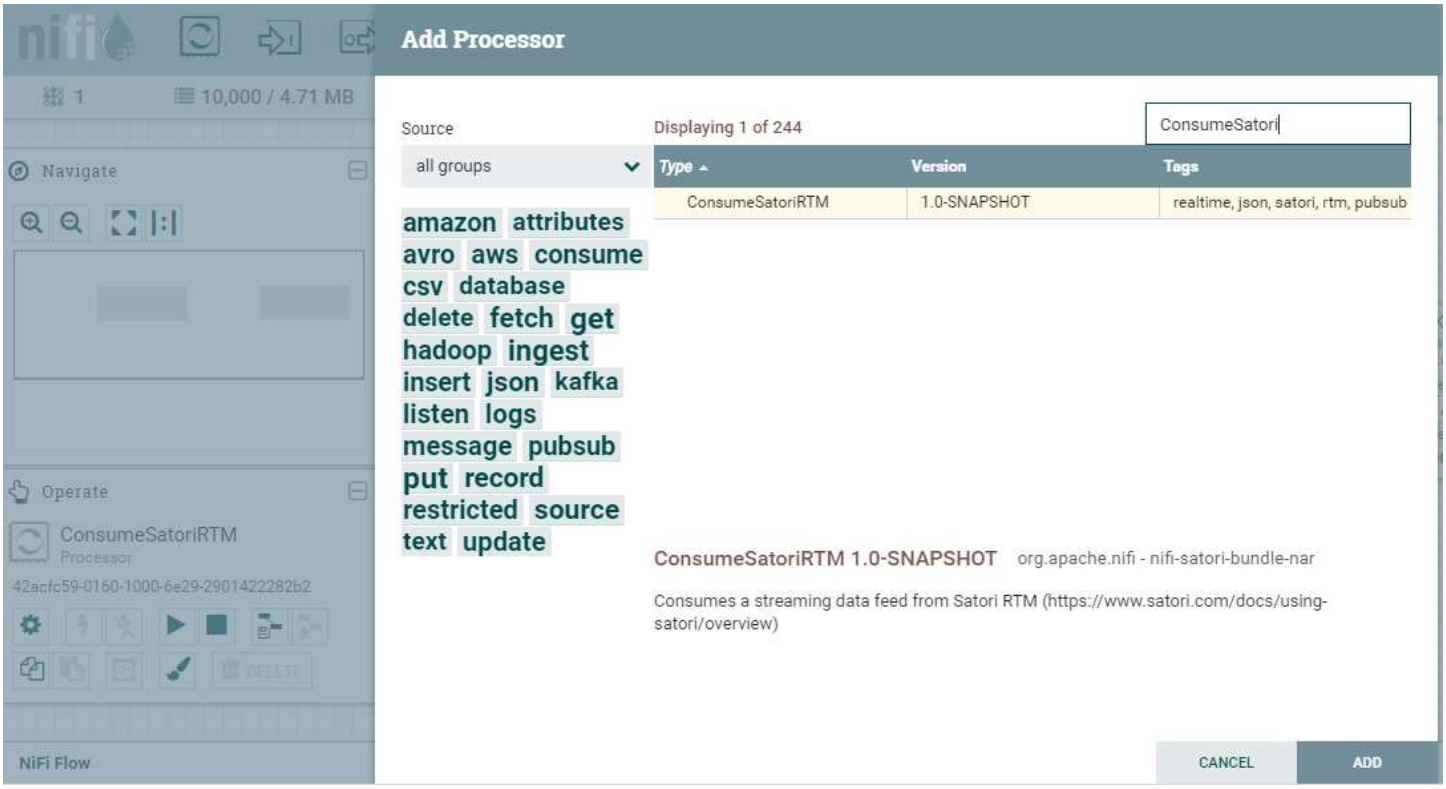
Start nifi now,

```
[centos@ip-172-31-90-14 nifi-1.4.0]$ ./bin/nifi.sh start

Java home: /usr/jdk1.8.0_131
NiFi home: /home/centos/nifi-1.4.0

Bootstrap Config File: /home/centos/nifi-1.4.0/conf/bootstrap.conf


[centos@ip-172-31-90-14 nifi-1.4.0]$ ./bin/nifi.sh status

Java home: /usr/jdk1.8.0_131
NiFi home: /home/centos/nifi-1.4.0

Bootstrap Config File: /home/centos/nifi-1.4.0/conf/bootstrap.conf

2017-12-09 17:51:33,227 INFO [main] org.apache.nifi.bootstrap.Command Apache NiFi is currently running, listening to Bootstrap on port 34428, PID=1899
```

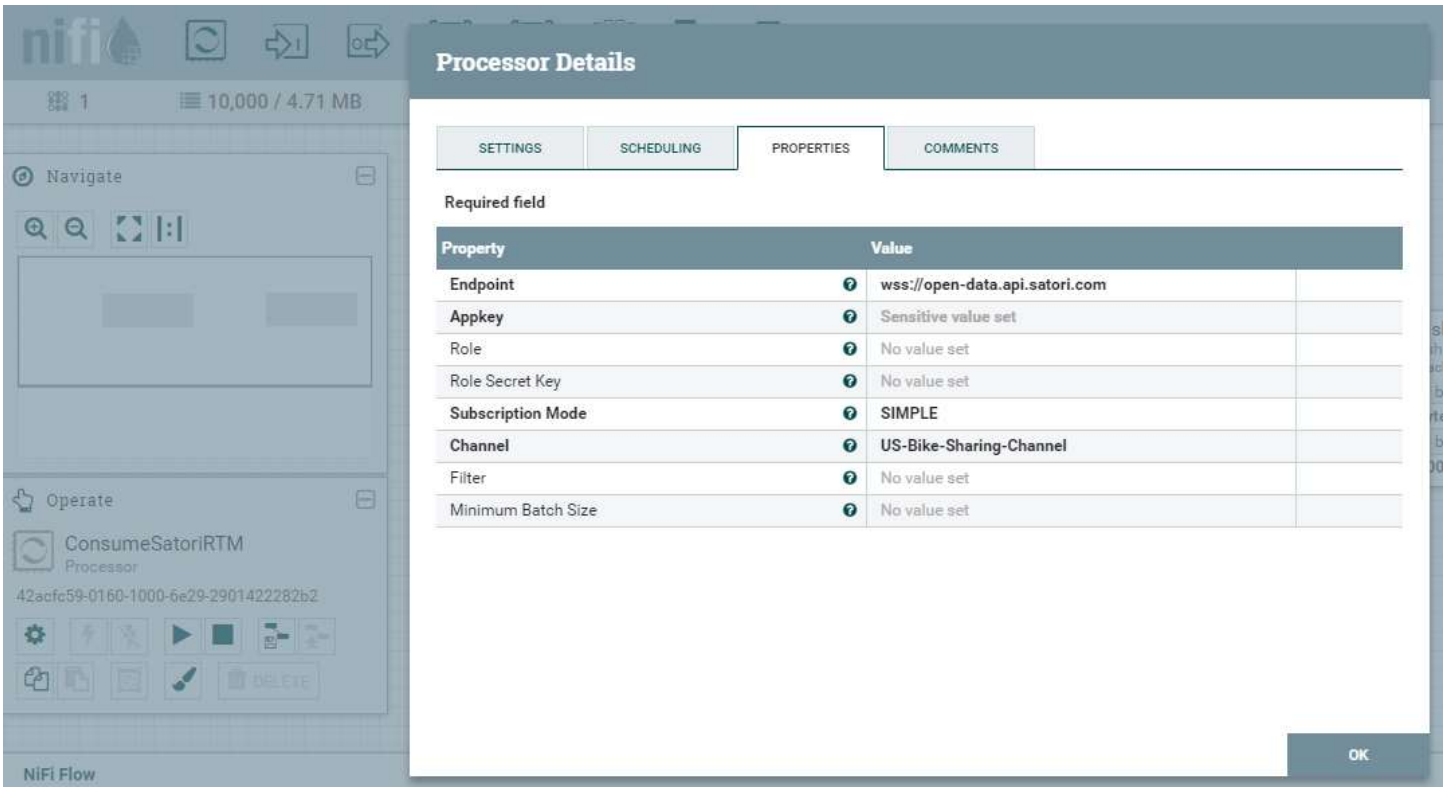
The UI for Apache Nifi loads at port 80 –



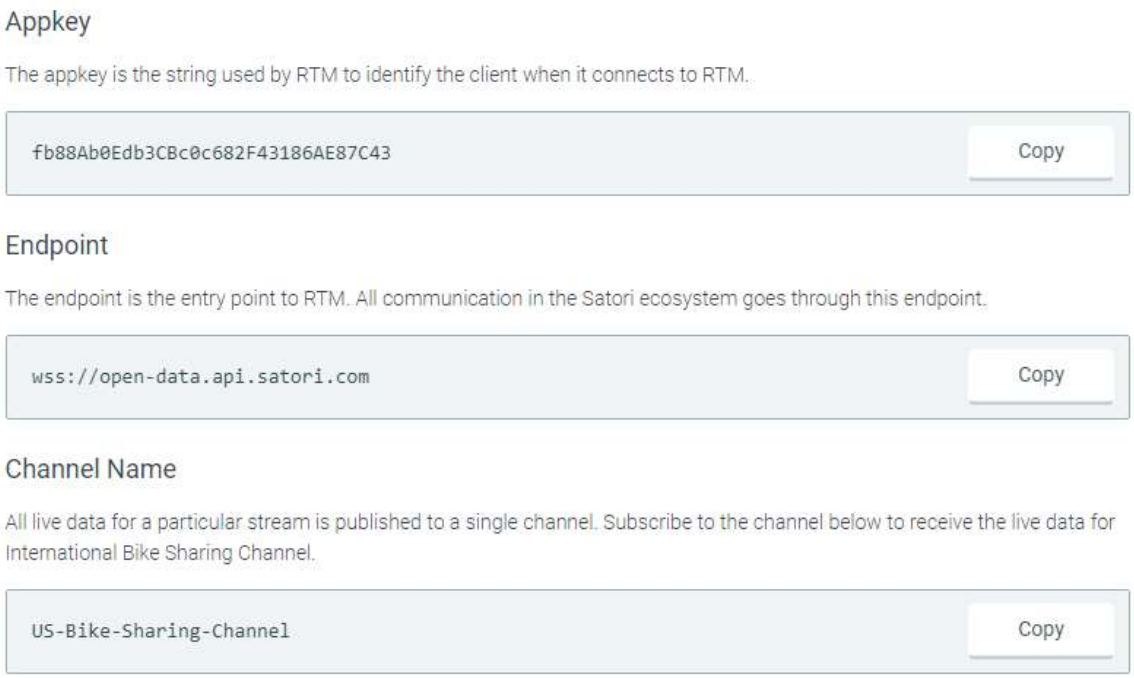
Add Processor – Selected ConsumeSatoriRTM




The endpoint, Appkey and Channel name is required for configuring the processor.



The below information is provided in the satori website.




**Apache Kafka**

Download Link

https://kafka.apache.org/downloads

Commands to download or unpack

```
wget http://mirrors.ocf.berkeley.edu/apache/kafka/1.0.0/kafka_2.11-1.0.0.tgz


tar xvf kafka_2.11-1.0.0.tgz
```

Kafka uses ZooKeeper so we need to first start a ZooKeeper server

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

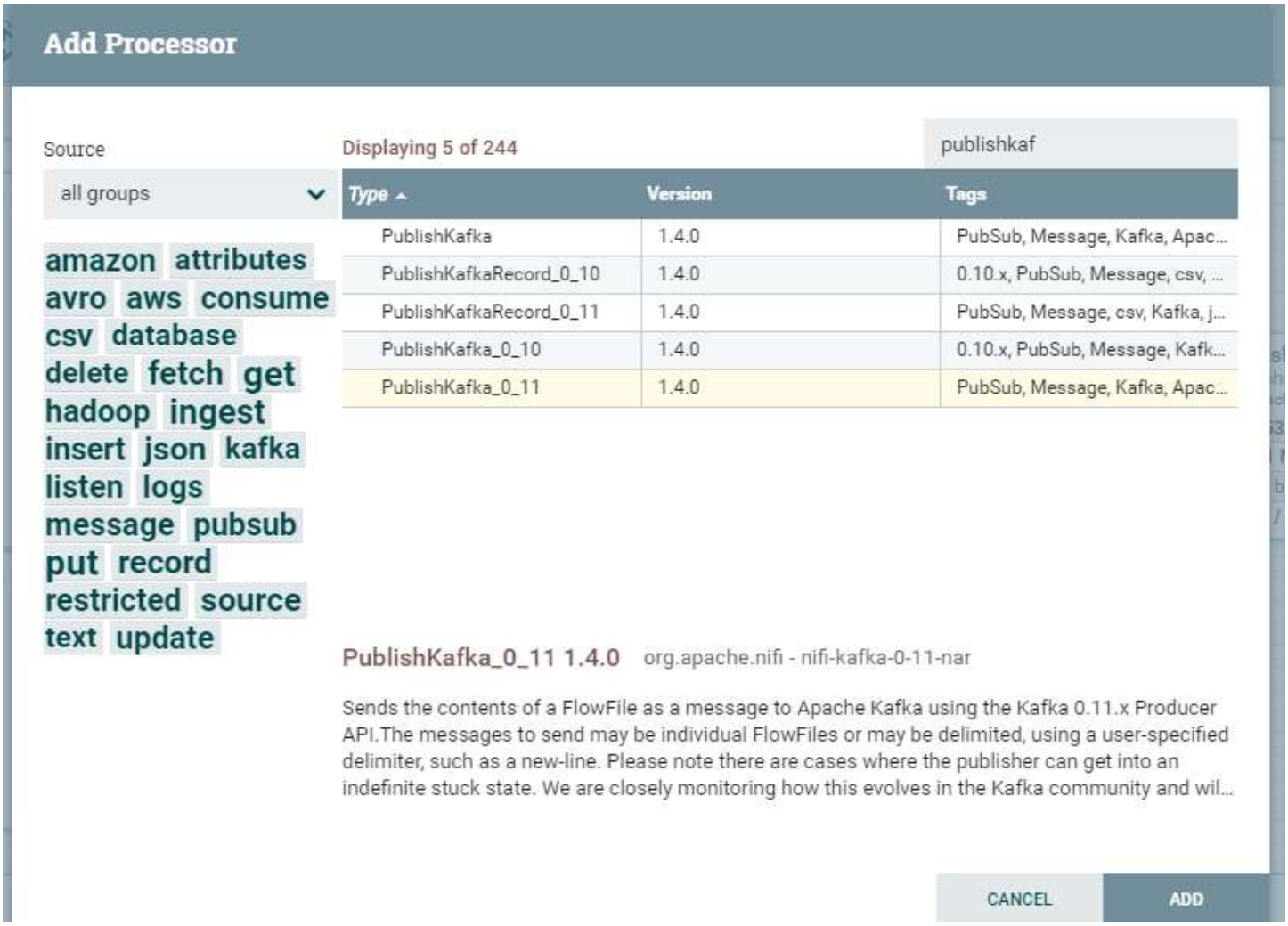
Start the Kafka server:

```
./bin/kafka-server-start.sh config/server.properties
```

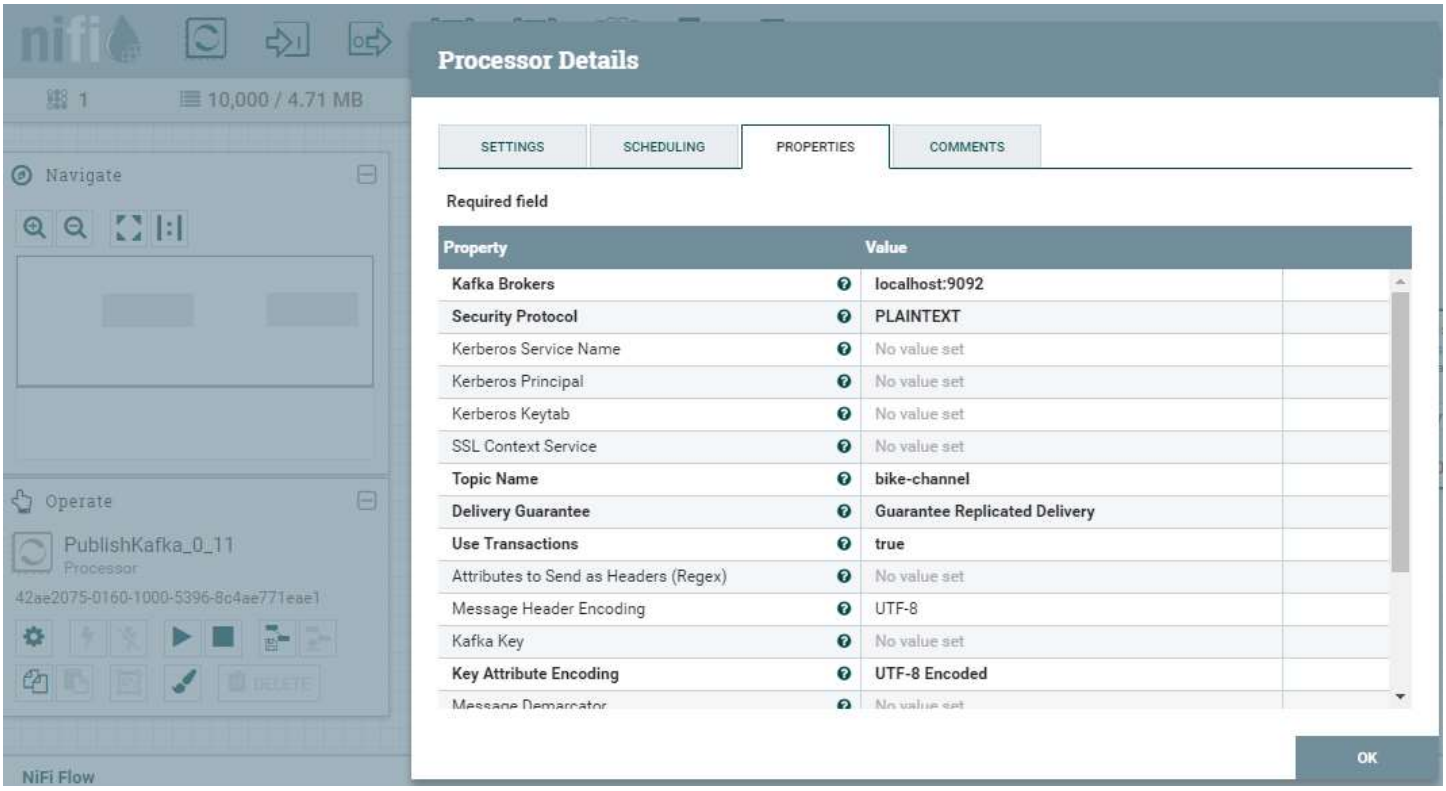
Creating a topic

```
[centos@ip-172-31-19-178 kafka_2.11-0.11.0.2]$ bin/kafka-topics.sh --create --
zookeeper localhost:2181 --replication-factor 1 --partitions 4 --topic bike-channel
```

```
Created topic "bike-channel".
```

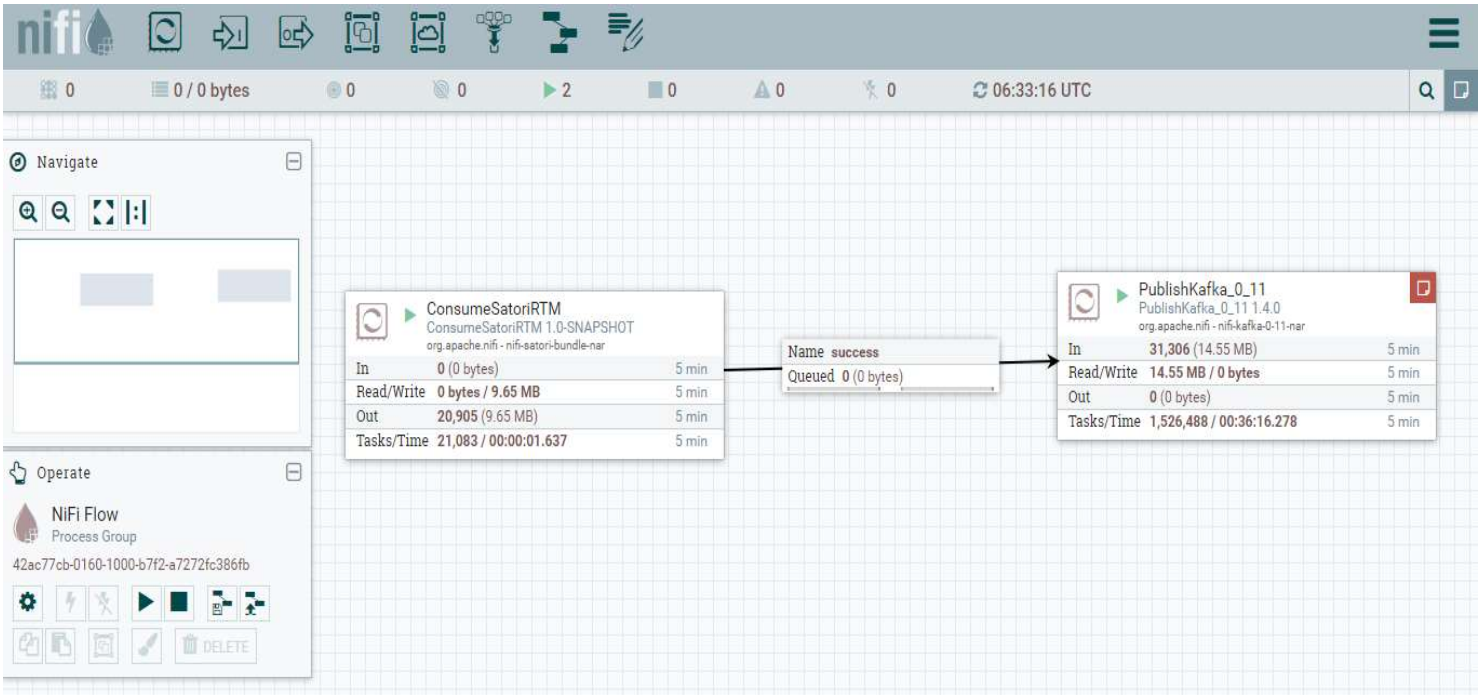



Sends the contents of a FlowFile as a message to Apache Kafka using the Kafka 0.11.x Producer AP



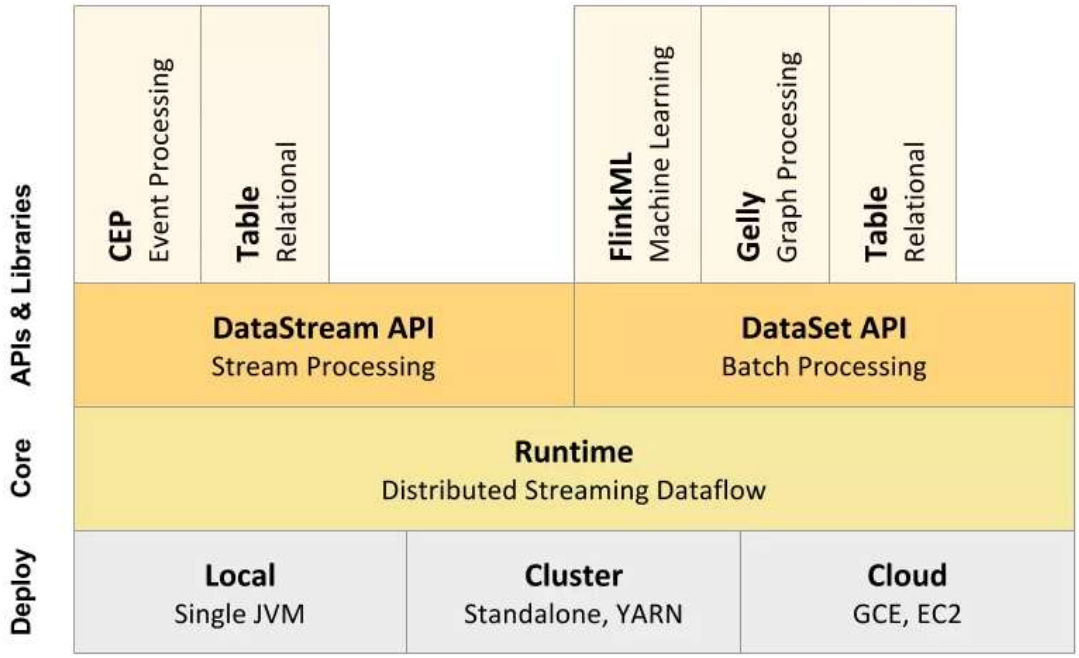

NiFi was built to automate the flow of data between systems. While the term *dataflow* is used in a variety of contexts, we use it here to mean the automated and managed flow of information between systems.

We add a Nifi dataflow from the ConsumeSatoriRTM to the Kafka




**Apache Flink**

Apache Flink is a big data processing tool and it is known to process big data quickly with low data latency and high fault tolerance on distributed systems on a large scale. Its defining feature is its ability to process streaming data in real time.



**Download and Installation**

Below is the download link for flink. We have used v1.3.2 here.

https://flink.apache.org/downloads.html

This link is very useful in downloading and installing Flink –

https://ci.apache.org/projects/flink/flink-docs-release-1.3/quickstart/setup_quickstart.html

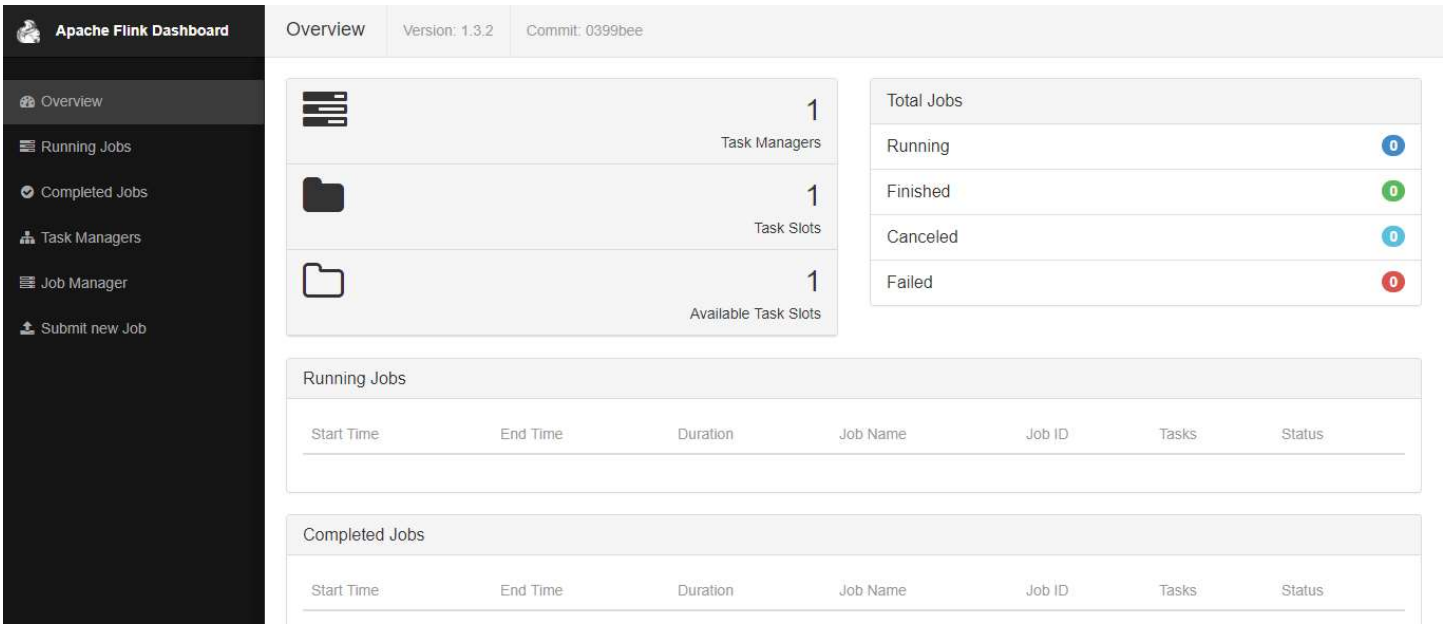
Unix Commands –

```
wget http://www-eu.apache.org/dist/flink/flink-1.3.2/flink-1.3.2-bin-hadoop26-scala_2.11.tgz

tar xvf flink-1.3.2-bin-hadoop26-scala.2.11.tgz
```

Starting Flink -

```
[centos@ip-172-31-90-14 flink-1.3.2]$ ./bin/start-cluster.sh local
Starting cluster.
Starting jobmanager daemon on host ip-172-31-90-14.ec2.internal.
Starting taskmanager daemon on host ip-172-31-90-14.ec2.internal.
```

Check the **JobManager's web frontend** at http://localhost:8081




## Elasticsearch and Kibana

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time.

Kibana is an open source data visualization plugin for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.

**Elasticsearch 2.3.5 and Kibana 4.5.4 has been used here.**

**Download and installation**

**Links for download –**

Elasticsearch - https://www.elastic.co/downloads/past-releases/elasticsearch-2-3-5

`wget` https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/tar/elasticsearch/2.3.5/elasticsearch-2.3.5.tar.gz

Edit elasticsearch.yml in config folder and add below changes

Mention data and logs path

```
# ---------------------------------- Paths ------------------------------------
#
# Path to directory where to store the data (separate multiple locations by comma):
#
path.data: /home/centos/elastic/data
#
# Path to log files:
#
path.logs: /home/centos/elastic/logs
#
```

Since, I am using a EC2 instance I have provided the private IP address in the network.host field

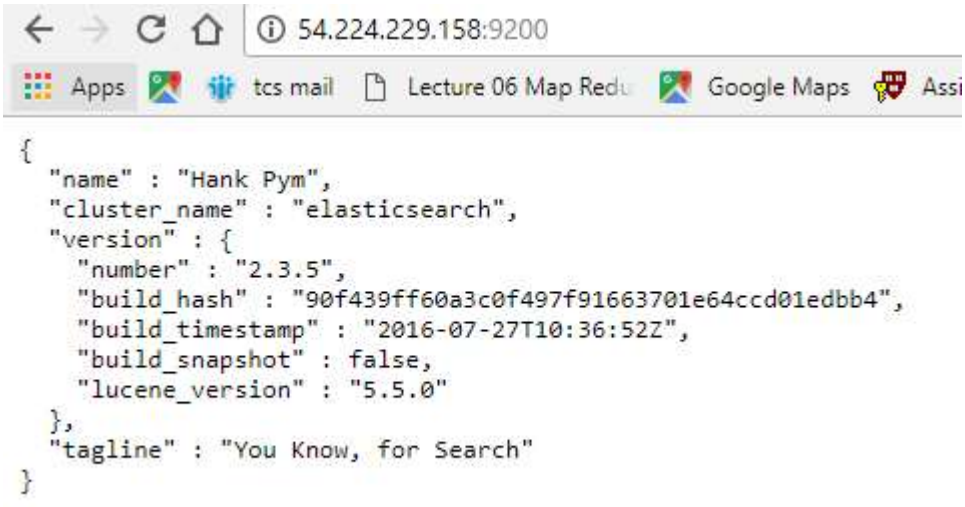
```
# ---------------------------------- Network -----------------------------------
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: 172.31.90.14
#
# Set a custom port for HTTP:
#
# http.port: 9200
#
```

Start elasticsearch

```
[centos@ip-172-31-90-14 elasticsearch-2.3.5]$ ./bin/elasticsearch
[2017-12-12 04:03:37,054][INFO ][node                     ] [Hank Pym] version[2.3.5], pid[2264],
build[90f439f/2016-07-27T10:36:52Z]
[2017-12-12 04:03:37,055][INFO ][node                     ] [Hank Pym] initializing ...
[2017-12-12 04:03:37,512][INFO ][plugins                  ] [Hank Pym] modules [reindex, lang-
expression, lang-groovy], plugins [], sites []
[2017-12-12 04:03:37,530][INFO ][env                      ] [Hank Pym] using [1] data paths, mounts
[[/ (rootfs)]], net usable_space [2gb], net total_space [7.9gb], spins? [unknown], types [rootfs]
[2017-12-12 04:03:37,531][INFO ][env                      ] [Hank Pym] heap size [990.7mb],
compressed ordinary object pointers [true]
[2017-12-12 04:03:38,919][INFO ][node                     ] [Hank Pym] initialized
[2017-12-12 04:03:38,919][INFO ][node                     ] [Hank Pym] starting ...
[2017-12-12 04:03:38,968][INFO ][transport                ] [Hank Pym] publish_address
{172.31.90.14:9300}, bound_addresses {172.31.90.14:9300}
```

```
[2017-12-12 04:03:38,972][INFO ][discovery                ] [Hank Pym]
elasticsearch/TUWercrVSCWtR5k56a42qA
[2017-12-12 04:03:42,004][INFO ][cluster.service          ] [Hank Pym] new_master {Hank
Pym}{TUWercrVSCWtR5k56a42qA}{172.31.90.14}{172.31.90.14:9300}, reason: zen-disco-
join(elected_as_master, [0] joins received)
[2017-12-12 04:03:42,017][INFO ][http                     ] [Hank Pym] publish_address
{172.31.90.14:9200}, bound_addresses {172.31.90.14:9200}
[2017-12-12 04:03:42,017][INFO ][node                     ] [Hank Pym] started
```

To ensure Elasticsearch is up and running, goto localhost:9200. Since, I am using an EC2 instance, I have used the public IP address:9200.




### Kibana

Download - https://www.elastic.co/downloads/past-releases/kibana-4-5-4

`wget https://download.elastic.co/kibana/kibana/kibana-4.5.4-linux-x64.tar.gz`

`tar xvf kibana-4.5.4-linux-x64.tar.gz`

Edit kibana.yml in config folder and do the below changes.

Add the private IP address to server.host field and elasticsearch url

```
# The host to bind the server to.
server.host: 172.31.90.14
```

```
# The Elasticsearch instance to use for all your queries.
elasticsearch.url: "http://172.31.90.14:9200"
```

### Starting Kibana

```
[centos@ip-172-31-90-14 kibana-4.5.4-linux-x64]$ ./bin/kibana
  log   [05:34:50.382] [info][status][plugin:kibana] Status changed from uninitialized to green -
Ready
  log   [05:34:50.414] [info][status][plugin:elasticsearch] Status changed from uninitialized to
yellow - Waiting for Elasticsearch
```

```
  log   [05:34:50.428] [info][status][plugin:kbn_vislib_vis_types] Status changed from
uninitialized to green - Ready
  log   [05:34:50.437] [info][status][plugin:markdown_vis] Status changed from uninitialized to
green - Ready
  log   [05:34:50.442] [info][status][plugin:metric_vis] Status changed from uninitialized to green
- Ready
  log   [05:34:50.445] [info][status][plugin:spyModes] Status changed from uninitialized to green -
Ready
  log   [05:34:50.448] [info][status][plugin:statusPage] Status changed from uninitialized to green
- Ready
  log   [05:34:50.458] [info][status][plugin:table_vis] Status changed from uninitialized to green
- Ready
  log   [05:34:50.466] [info][listening] Server running at http://172.31.90.14:5601
  log   [05:34:50.470] [info][status][plugin:elasticsearch] Status changed from yellow to green -
Kibana index ready
```

Create an index named “bike-channel”

Mention the type and name of two fields -location ( geo-point) and last_reported (date)

```
[centos@ip-172-31-90-14 nifi-1.4.0]$ curl -XPUT 'localhost:9200/bike-channel' -H 'Content-Type:
application/json' -d'
> {
>   "mappings": {
>     "bike": {
>       "properties": {
>         "location": {
>           "type": "geo_point"
>         },
>        "last_reported": {
>           "type": "date",
>           "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd"
>         }
>
>       }
>     }
>   }
> }'
{"acknowledged":true}
```

**Scala configurations for build**

The programming language for the code to read streaming data from kafka source, clean it / format it and write it to the elasticsearch sink is Scala.

Sbt is used for build here. Below is the download link for sbt.

http://www.scala-sbt.org/download.html

Below changes need to be done to build.sbt file in the sbt project.

```
/home/centos/bike-channel/bike-streaming/build.sbt
```

Since, the scala version used is 2.11.8. This needs to be added.

```
scalaVersion := "2.11.8"
```

```
Add name to the application
```

```
name := "bike-streaming"
organization := "ch.epfl.scala"
version := "1.0
```

Adding following dependencies for scala compiler, kafka, flink-scala, flink streaming , flink kafka connector and flink elasticsearch2 connector. ( Elasticsearch version is 2.3.5, so we have used elasticsearch2 connector)

```
libraryDependencies += "org.scala-lang" % "scala-compiler" % "2.11.8"
```

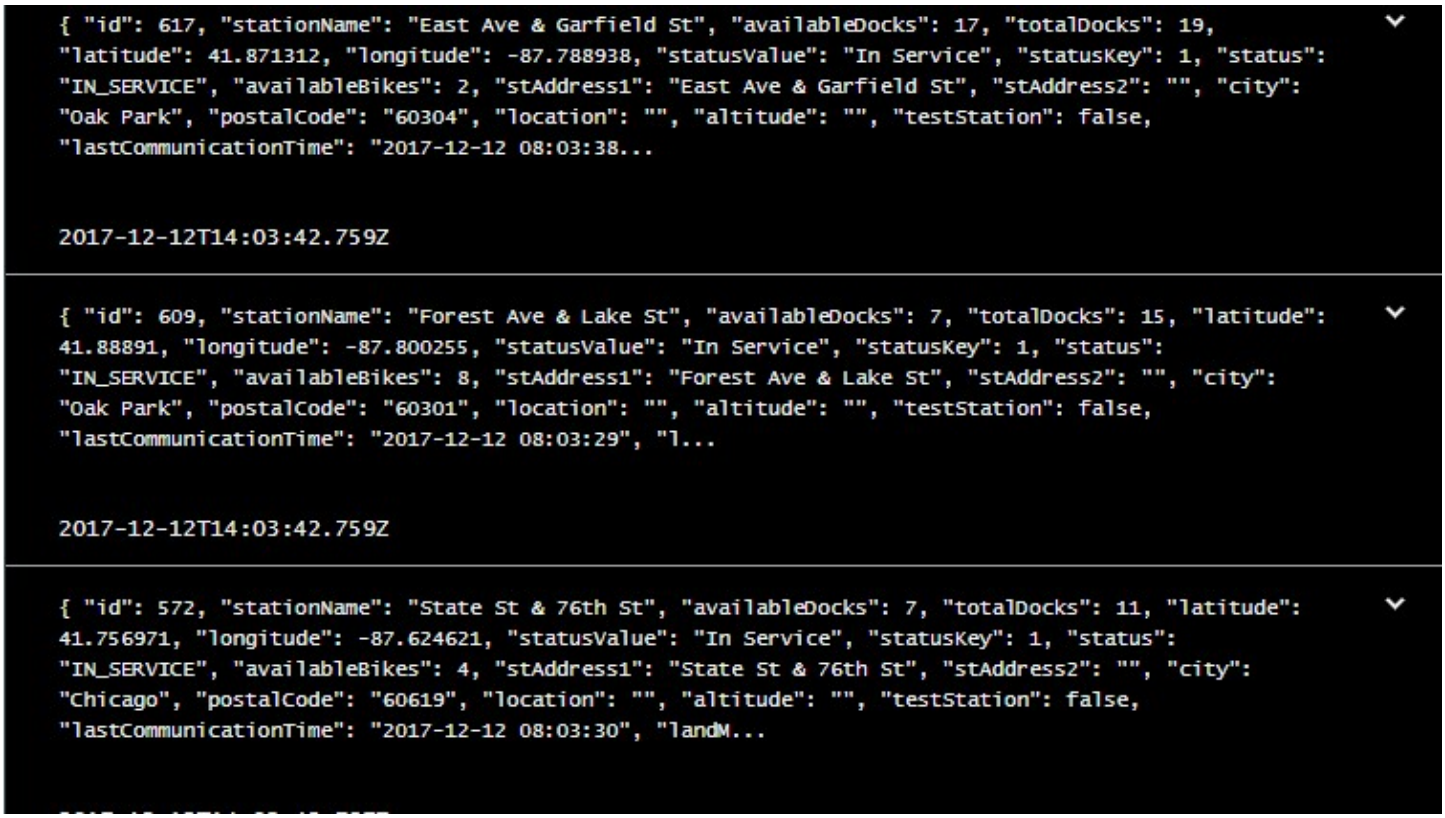
```
// https://mvnrepository.com/artifact/org.apache.kafka/kafka
libraryDependencies += "org.apache.kafka" %% "kafka" % "1.0.0"
// https://mvnrepository.com/artifact/org.apache.flink/flink-scala
libraryDependencies += "org.apache.flink" %% "flink-scala" % "1.3.2"

// https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-scala
libraryDependencies += "org.apache.flink" %% "flink-streaming-scala" % "1.3.2"


// https://mvnrepository.com/artifact/org.apache.flink/flink-connector-kafka-0.10
libraryDependencies += "org.apache.flink" %% "flink-connector-kafka-0.10" % "1.3.2"

// https://mvnrepository.com/artifact/net.liftweb/lift-json
libraryDependencies += "net.liftweb" %% "lift-json" % "2.6-M4

// https://mvnrepository.com/artifact/org.apache.flink/flink-connector-elasticsearch2
libraryDependencies += "org.apache.flink" %% "flink-connector-elasticsearch2" % "1.3.2"
```

Now we have everything setup and running, we will execute the sbt-scala project.

```
[centos@ip-172-31-90-14 bike-streaming]$ sbt
[info] Loading project definition from /home/centos/bike-channel/bike-streaming/project
[info] Loading settings from build.sbt ...
[info] Set current project to bike-streaming (in build file:/home/centos/bike-channel/bike-streaming/)
[info] sbt server started at 127.0.0.1:4054
sbt:bike-streaming> run
[info] Compiling 1 Scala source to /home/centos/bike-channel/bike-streaming/target/scala-2.11/classes ...
[warn] there were three deprecation warnings; re-run with -deprecation for details
[warn] one warning found
[info] Done compiling.
[info] Packaging /home/centos/bike-channel/bike-streaming/target/scala-2.11/bike-streaming_2.11-1.0.jar ...
[info] Done packaging.
[info] Running Main
[debug] Waiting for threads to exit or System.exit to be called.
[debug]   Classpath:
[debug]         /tmp/sbt_8ea0112d/job-1/target/dbdf62bd/bike-streaming_2.11-1.0.jar
[debug]         /tmp/sbt_8ea0112d/target/fe1285c9/scala-compiler-2.11.8.jar
[debug]         /tmp/sbt_8ea0112d/target/ddd5a8bc/scala-library-2.11.8.jar
[debug]         /tmp/sbt_8ea0112d/target/b74530de/scala-reflect-2.11.8.jar
[debug]         /tmp/sbt_8ea0112d/target/b67c76af/scala-xml_2.11-1.0.4.jar
[debug]         /tmp/sbt_8ea0112d/target/7369d653/scala-parser-combinators_2.11-1.0.4.jar
[debug]         /tmp/sbt_8ea0112d/target/3b337637/cats_2.11-0.9.0.jar
[debug]         /tmp/sbt_8ea0112d/target/be02ae0c/cats-macros_2.11-0.9.0.jar
[debug]         /tmp/sbt_8ea0112d/target/59bdbd0d/simulacrum_2.11-0.10.0.jar
[debug]         /tmp/sbt_8ea0112d/target/0cb87cb7/macro-compat_2.11-1.1.1.jar
[debug]         /tmp/sbt_8ea0112d/target/239a5628/machinist_2.11-0.6.1.jar
[debug]         /tmp/sbt_8ea0112d/target/6c8cf24b/cats-kernel_2.11-0.9.0.jar
[debug]         /tmp/sbt_8ea0112d/target/2aaed786/cats-kernel-laws_2.11-0.9.0.jar
[debug]         /tmp/sbt_8ea0112d/target/78458166/scalacheck_2.11-1.13.4.jar
[debug]         /tmp/sbt_8ea0112d/target/0a3f14d0/test-interface-1.0.jar
[debug]         /tmp/sbt_8ea0112d/target/b0859633/discipline_2.11-0.7.2.jar
```

**Sample of Input Data.**

Input Data is the form of json.




{"lon": -94.57731, "lat": 39.14975, "address": "2898 Swift St. ", "name": "29th & Swift", "station_id": "bcycle_kc_3673", "num_bikes_available": 5, "num_docks_available": 6, "is_installed": 1, "is_renting": 1, "is_returning": 1, "last_reported": 1513070212, "country_code": "US", "channel": "US-Bike-Sharing-Channel", "publisher": "kc.bcycle.com"}

**Details of the code –**

1. Reading data from Kafka using, Flink Kafka connector. Data is read in form of String ( SimpleStringSchema)

```
//Creating a context of StreamExecutionEnvironment
    val env = StreamExecutionEnvironment.getExecutionEnvironment

    val properties = new Properties()
    // comma separated list of Kafka brokers
    properties.setProperty("bootstrap.servers", "localhost:9092")
    // id of the consumer group
    properties.setProperty("group.id", "test")
    //Adding a data source which is a kafka topic here
    properties.setProperty("group.id", "test")
    //Adding a data source which is a kafka topic here
    val stream = env
      // bike-channel is our Kafka topic
```

```
      .addSource(new FlinkKafkaConsumer010[String]("bike-channel", new
SimpleStringSchema(), properties))
```

2. Now we read the variable stream, to fetch each message in the Datastream format them and insert into the elasticseach index “bike-channel” which we had created earlier.

I have used the default cluster name here and also mentioned the private Ip address of the EC2 instance.

```
val config = new util.HashMap[String, String]
    config.put("bulk.flush.max.actions", "1")
    //specifying the cluster name for elasticsearch sink
    config.put("cluster.name", "elasticsearch") //default cluster name:
elasticsearch

    val transports = new util.ArrayList[InetSocketAddress]
    transports.add(new InetSocketAddress(InetAddress.getByName("172.31.90.14"),
9300))
```

3. Build data and insert into elasticsearch sink using flink elasticsearch connector

```
//defining the addSink function to send data to an elasticsearch sink
    stream.addSink(new ElasticsearchSink(config, transports, new
ElasticsearchSinkFunction[String] {
          def createIndexRequest(element: String): IndexRequest = {
              val json = new util.HashMap[String, AnyRef]
              //Parsing the string element to json
              val jValue = parse(element)
              val bikeStream = jValue.extract[BikeStream]
              //formatting the location field, merging the latitude and longitude
value and creatinf a tuple
              var i= 0
              var j=0
              val last = bikeStream.last_reported
              var temp1 = ""
              var temp2 = ""
              if(bikeStream.lon == None) {//if lon field is not present use
longitude field
                   temp1 = bikeStream.latitude.get
                   temp2 = bikeStream.longitude.get
                     }
              else
                   {
                   temp1 = bikeStream.lat.get
                   temp2 = bikeStream.lon.get
                   }
              //select the substring length based on the sign of the latitude or
longtitude
              if(temp1.charAt(0) == '-') { i= 6}
                     else { i = 5}
```

```
              if(temp2.charAt(0) == '-') { j= 6}
                    else { j = 5}

              val str1 = temp1.substring(0,i).concat(",")
              var location= str1.concat(temp2.substring(0,j))

              //adding location field
              json.put("location", location)
              //adding location field
              json.put("location", location)
              //pick up location or address field whichever is present
              if(bikeStream.address == None){
                        json.put("address",
bikeStream.location.getOrElse("NotFound"))}
              else{
                        json.put("address",
bikeStream.address.getOrElse("NotFound"))
                  }

              //adding other fields
              json.put("num_bikes_available", bikeStream.num_bikes_available)
              json.put("is_installed", bikeStream.is_installed)
              json.put("is_renting", bikeStream.is_renting)
              //There may be last_reported field or lastCommunicationTime field,
picking up either based on how they are present in datastream
              if(bikeStream.last_reported == None){ //if last_reported filed not
present use lastComm field
                   val date =
bikeStream.lastCommunicationTime.get.substring(0,19)
                   json.put("last_reported", date)
                }
              else{
                  //formatting the time to YYYY:mm:dd hh:mm:ss
                   val instant = Instant.ofEpochMilli((last.get*1000))
                   val parser1 = instant.toString
                   val parser2 = parser1.substring(0,10)+"
"+parser1.substring(11,19)

                   json.put("last_reported", parser2)
                   }
               //pick up the country code
               json.put("country_code",
bikeStream.country_code.getOrElse("NotFound"))
              //pick up the name field
              json.put("name", bikeStream.name.getOrElse("NotFound"))
              //pick up the channel name
              json.put("channel", bikeStream.channel.getOrElse("NotFound"))
              //pick up the publisher name
              json.put("publisher", bikeStream.publisher.getOrElse("NotFound"))
              //pick up the station id
              json.put("station_id",
bikeStream.station_id.getOrElse("NotFound"))
```

```
                Requests.indexRequest.index("bike-
channel").`type`("bike").source(json)
      }

     override def process(element: String, ctx: RuntimeContext, indexer:
RequestIndexer) {
     indexer.add(createIndexRequest(element))
                              }
      }))
```

4. Start the execution of the job .

```
env.execute("Bike Streaming")
```

## Results

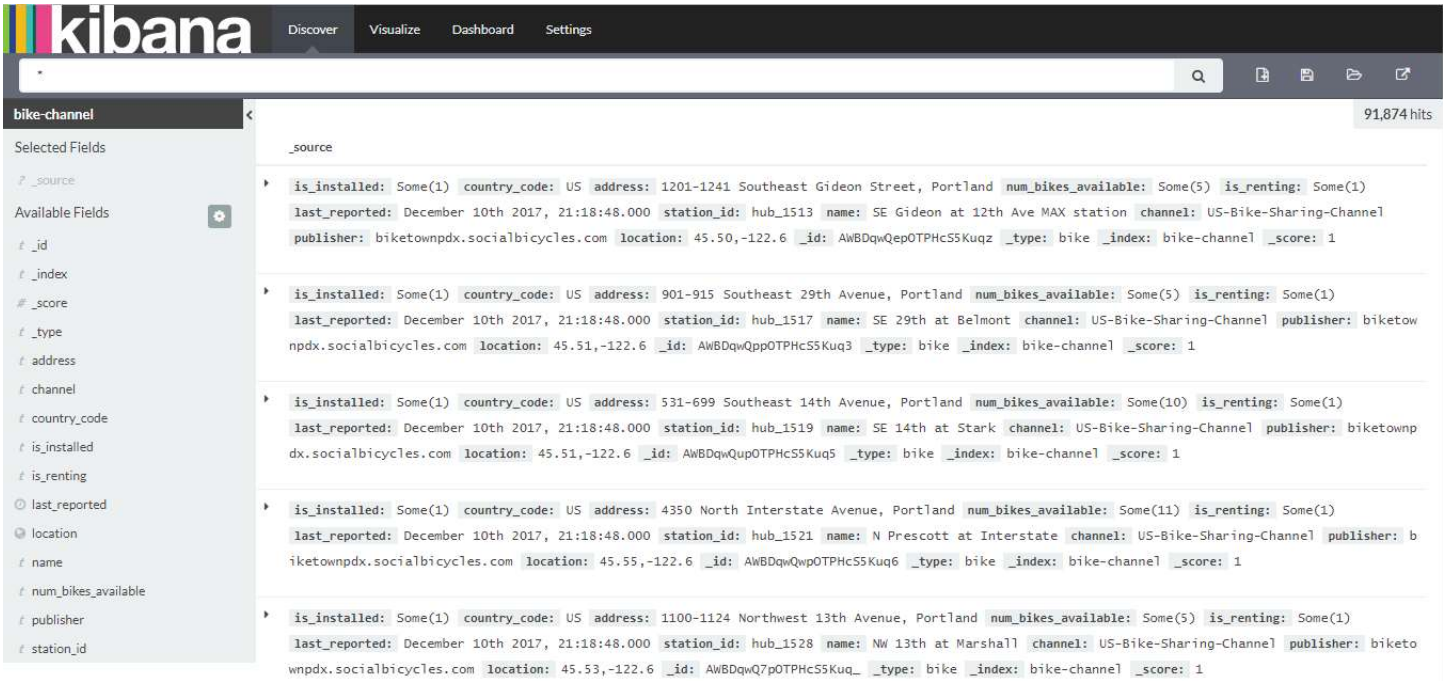
Kibana UI

We create an Index pattern first

Go to Settings > Indices> Configure an index pattern

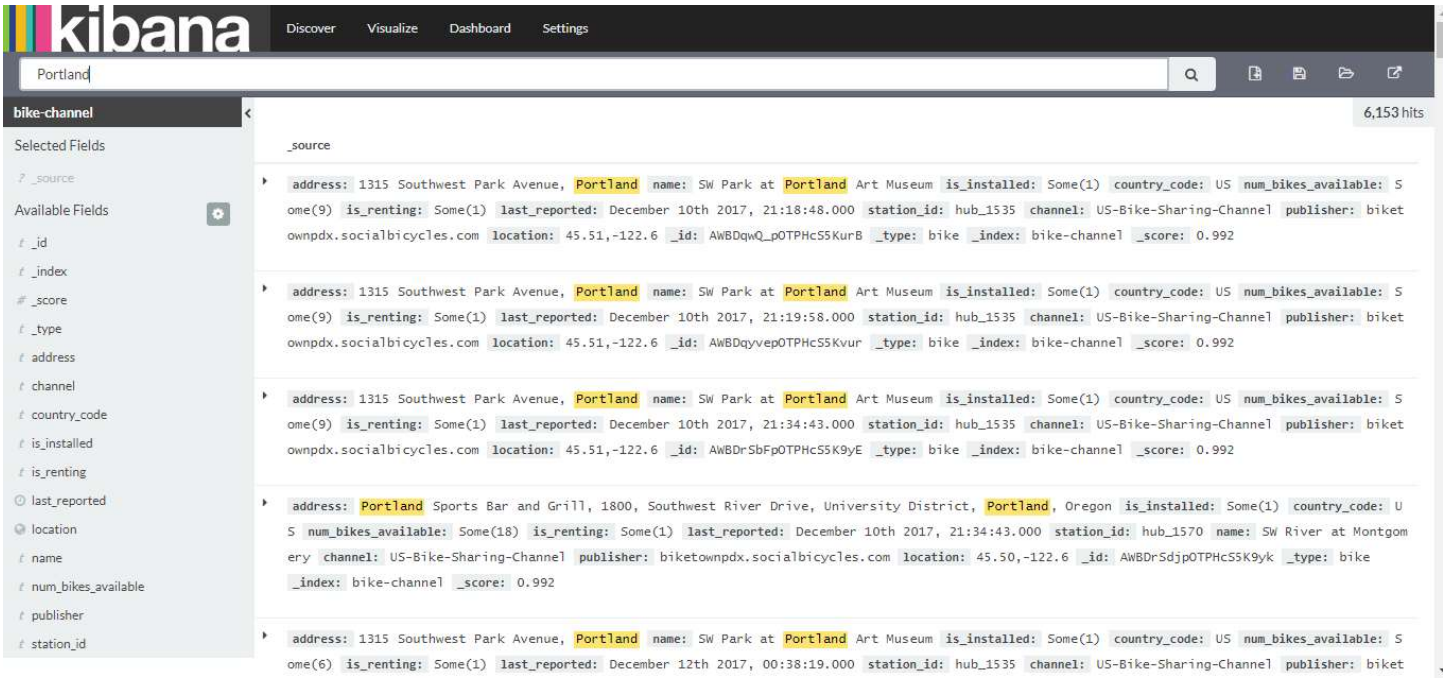
The index created here is bike-channel

The discover tab shows the data and the available fields

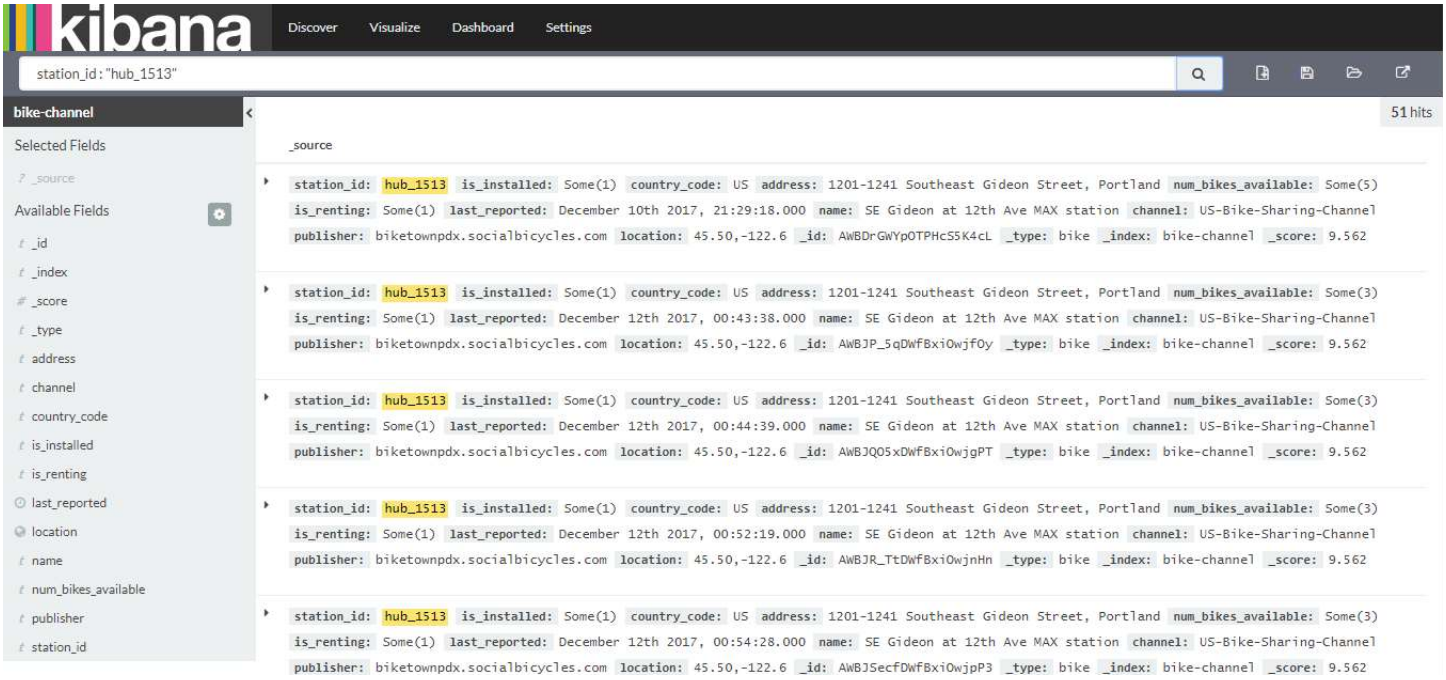


**Perform below queries.**

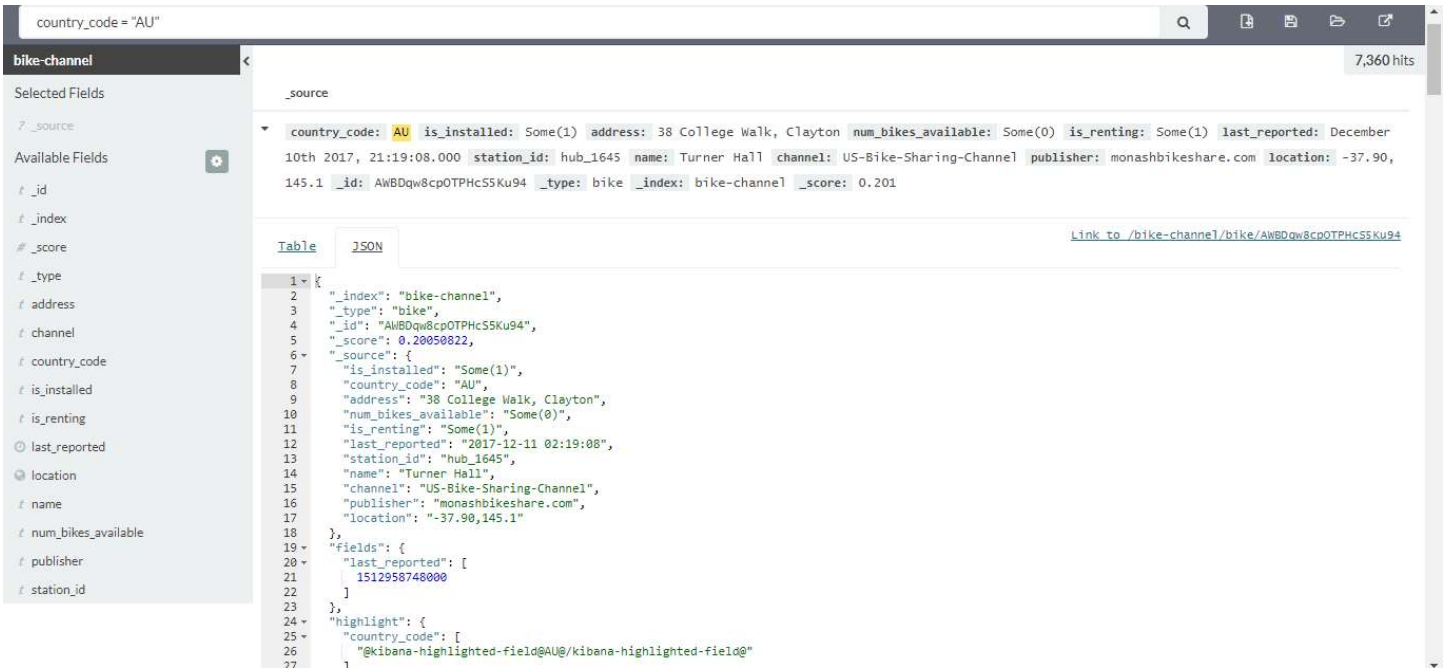
Find the bike rentals in Portland



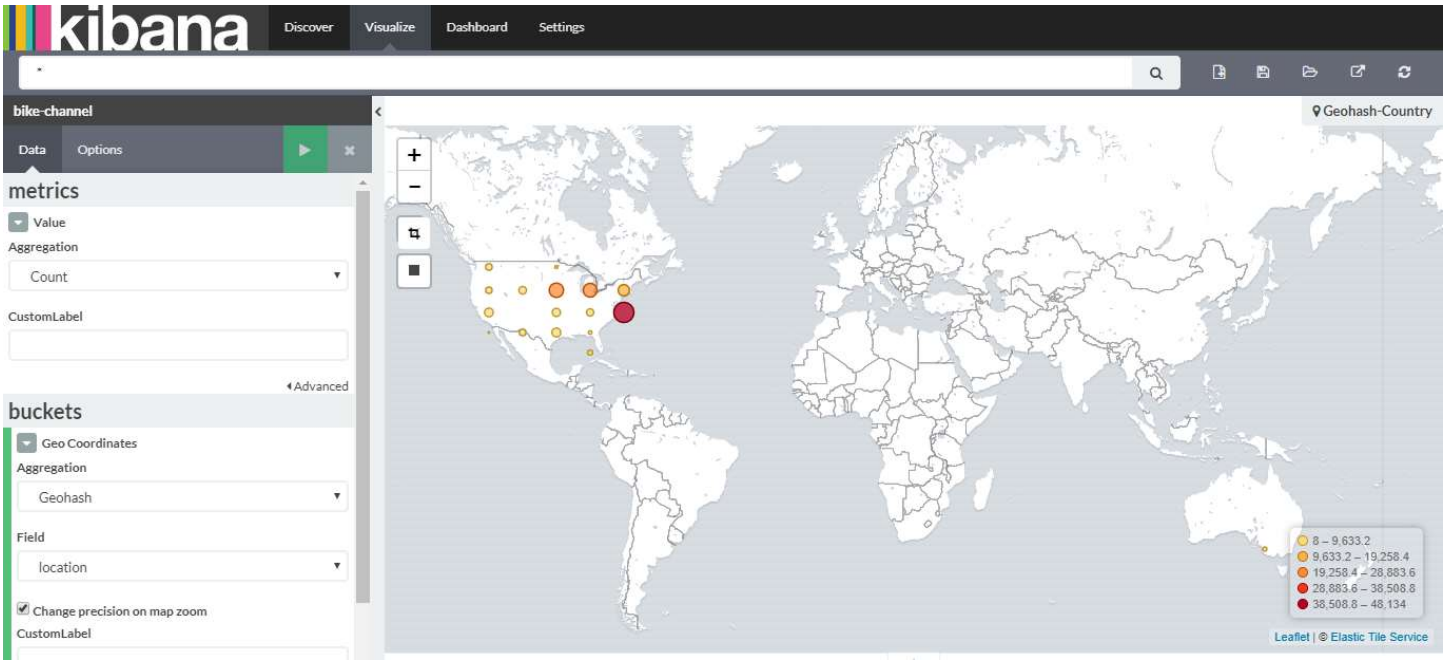

Find the bike rentals from station_id = “hub_1513”



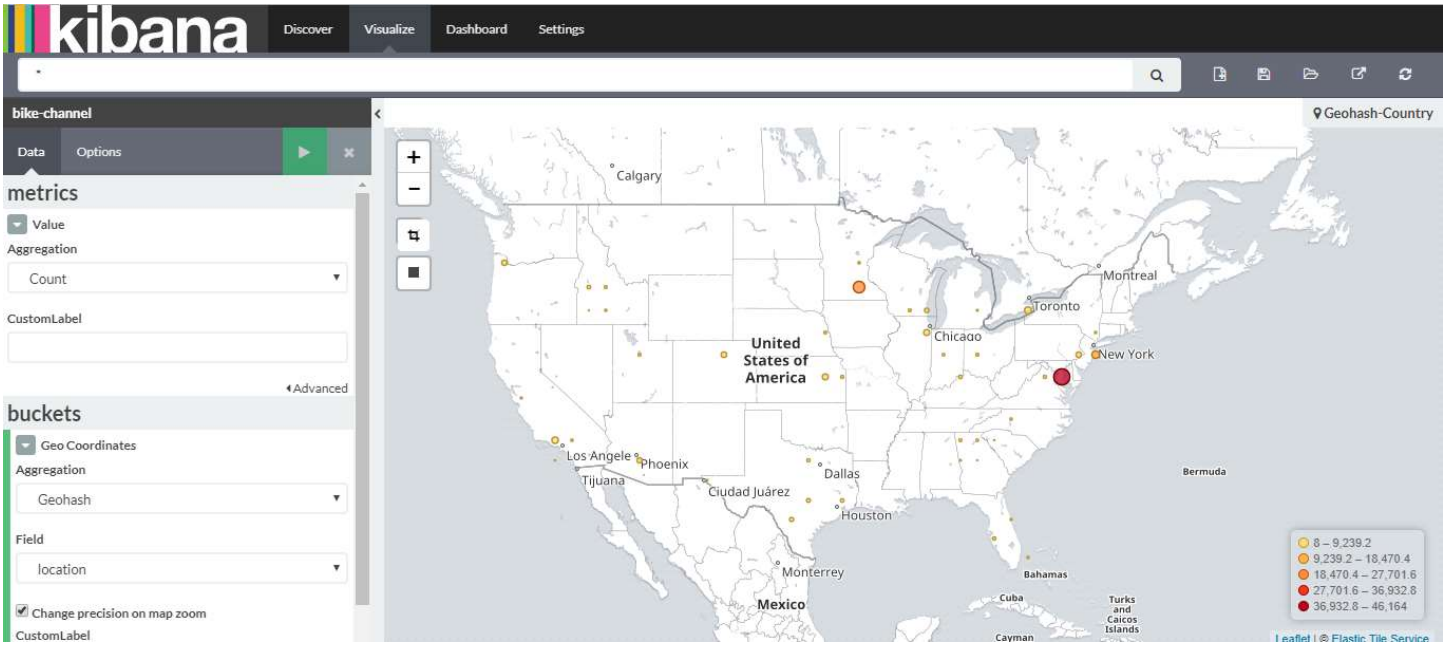

Selecting one of the events -



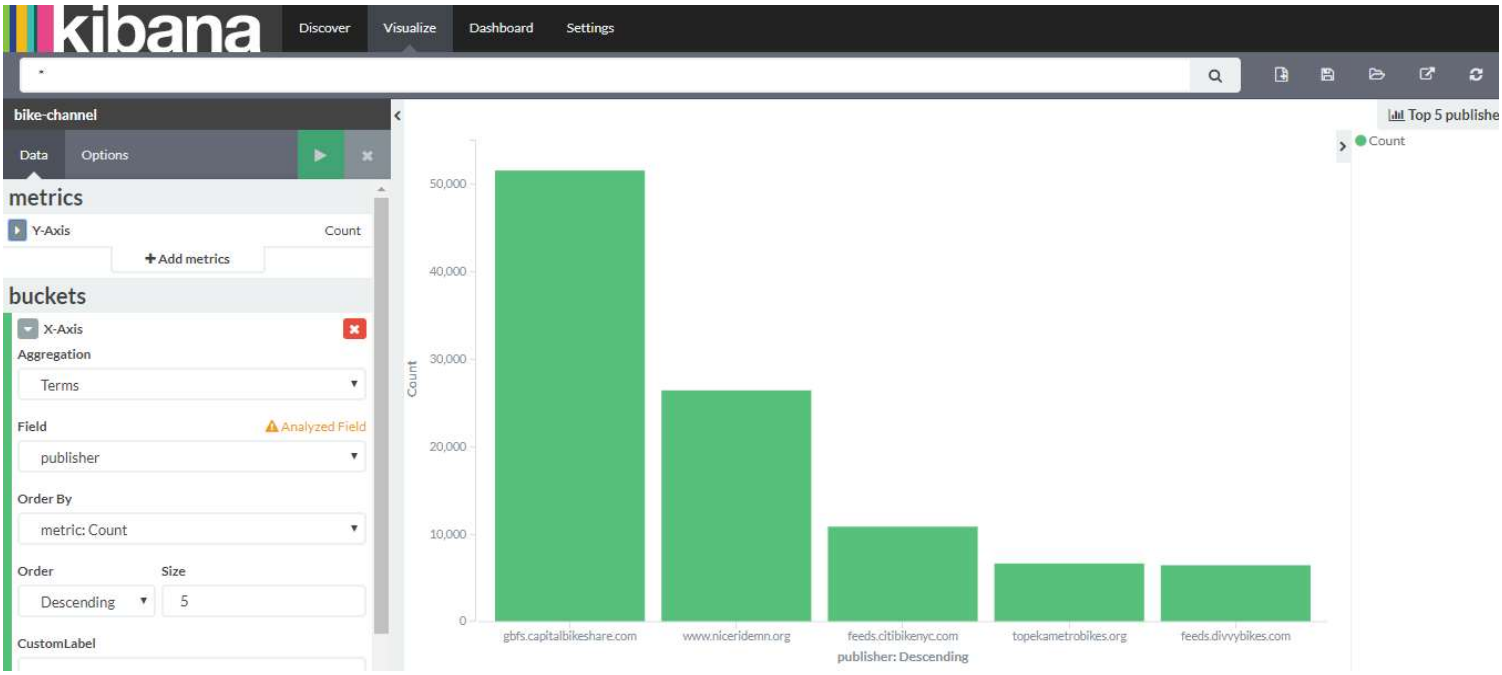

Geo  Visualization for Bike renting data



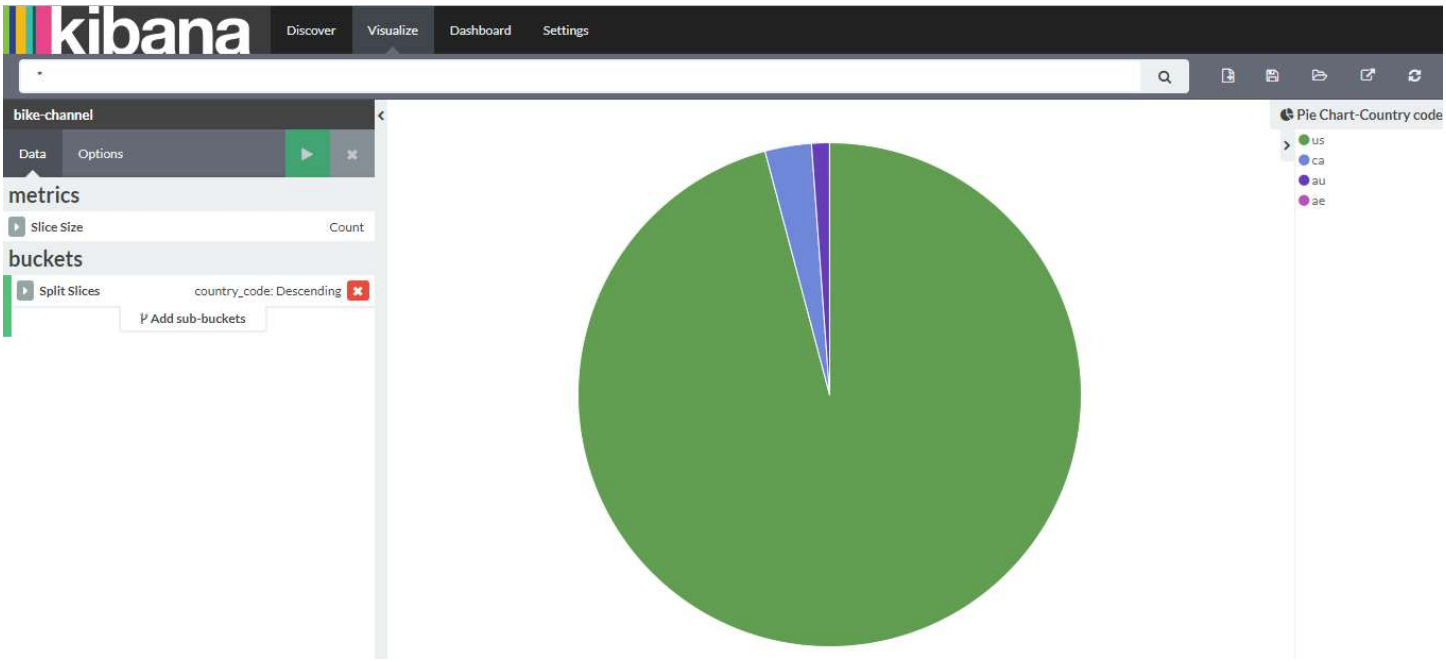

Locations in USA



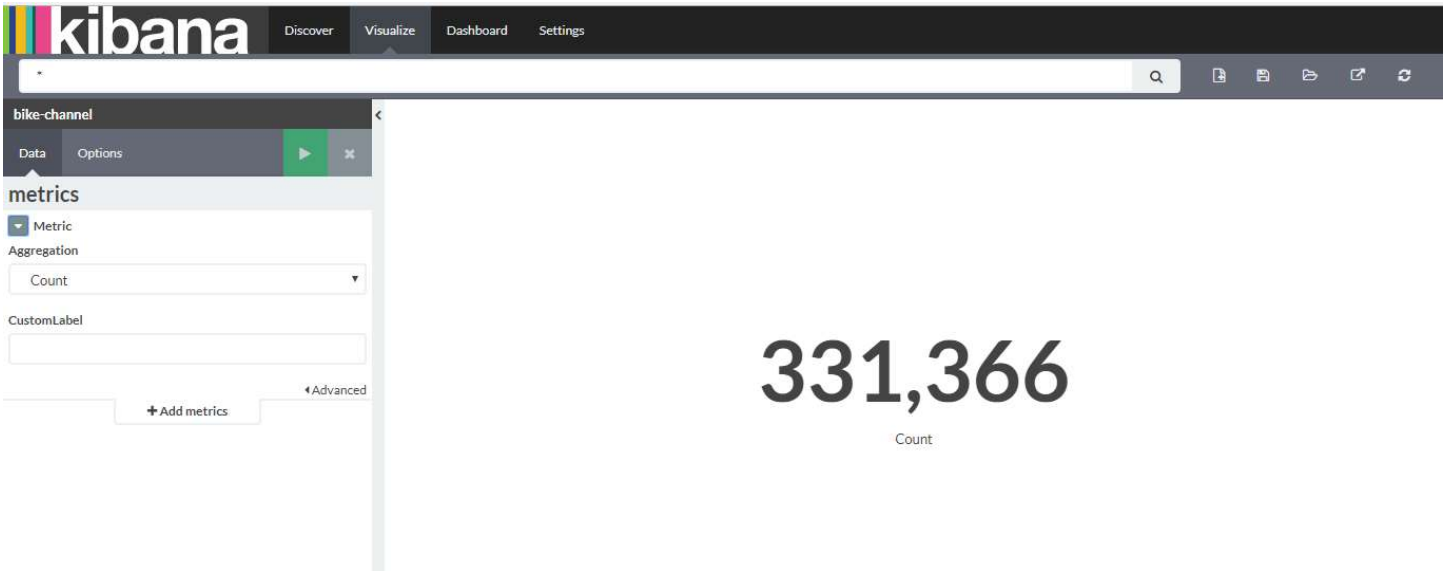

Creating a bike chart visualization for Top 5 “publisher” names.



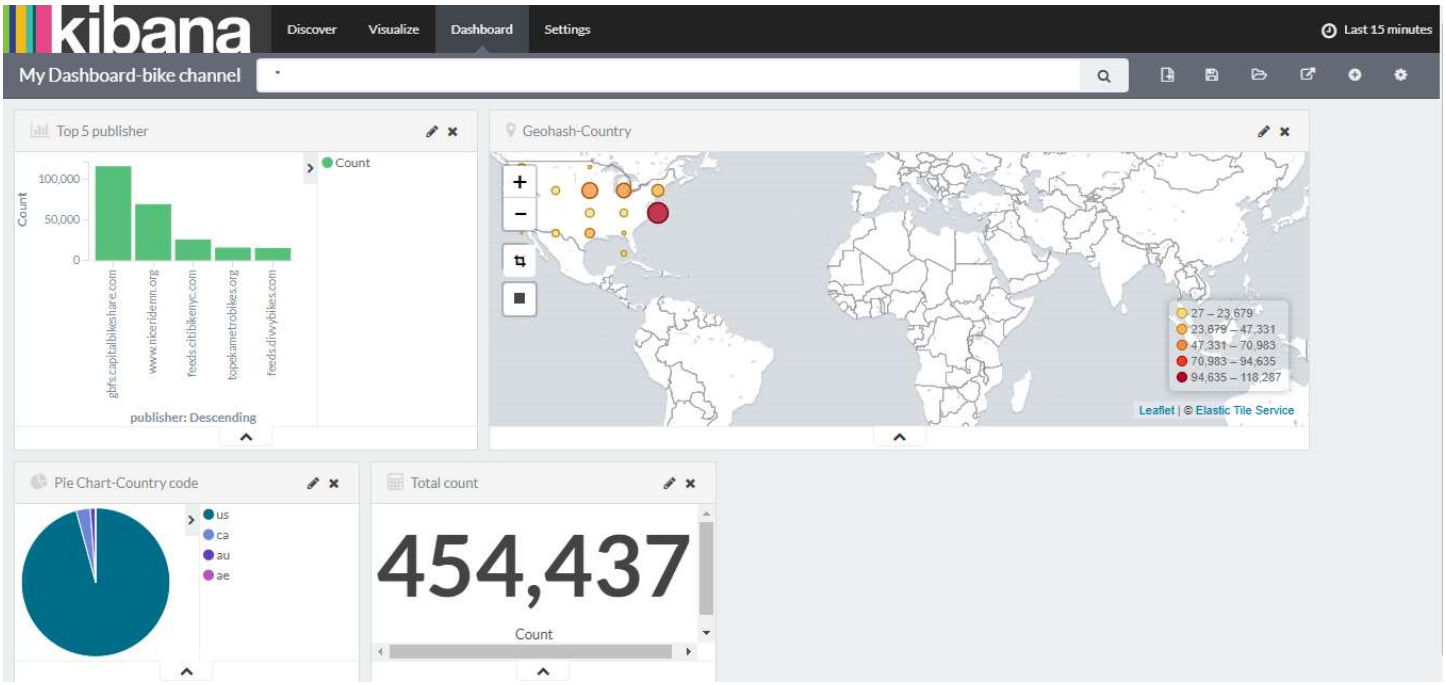
Pie chart for number of bike rentals per country




Visualization for total count



Creating Dashboard




## Conclusion and Lessons Learnt

Thus, this project implements a framework which reads International Bike sharing Streaming data and then inserts it into Elasticsearch indexes for further various analytics via Kibana UI.

We can extend this to further experimentation with more fields and then study the trends throughout the world.

Lessons Learnt – I first tried using flink elasticsearch5 connectior with elasticsearch 5.6. But, there was a lot of issues with making it work. It requires a log4j dependency to be included which I could not resolve.

So, I switched back to a lower version of elasticsearch – 2.3.5. elasticsearch2 worked fine. Also the reference materials for elasticsearch2 are readily available.

Useful links –

https://nifi.apache.org/docs/nifi-docs/html/overview.html

https://ci.apache.org/projects/flink/flink-docs-release-1.3/

https://mvnrepository.com/

https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/connectors/elasticsearch.html

Documents included with this project-

1. Configuration files for elasticsearch and kibana.
2. Build.sbt file for sbt build.
3. Scala code for Flink-Kafka Connector and Flink-elasticsearch connector.

Rest all other configuration files have been used as is ( as downloaded from internet)  without any changes.