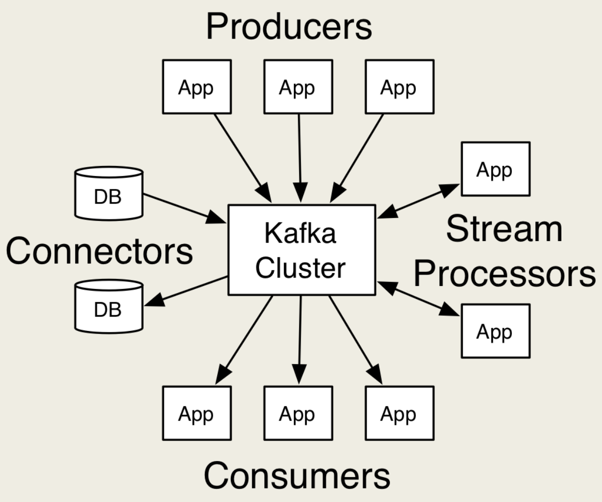
INDEXING AND SEARCHING FOR HOT TOPICS

Rajat Narang [rnarang]
Shreya More [ssmore]
Udit Misra [umisra]

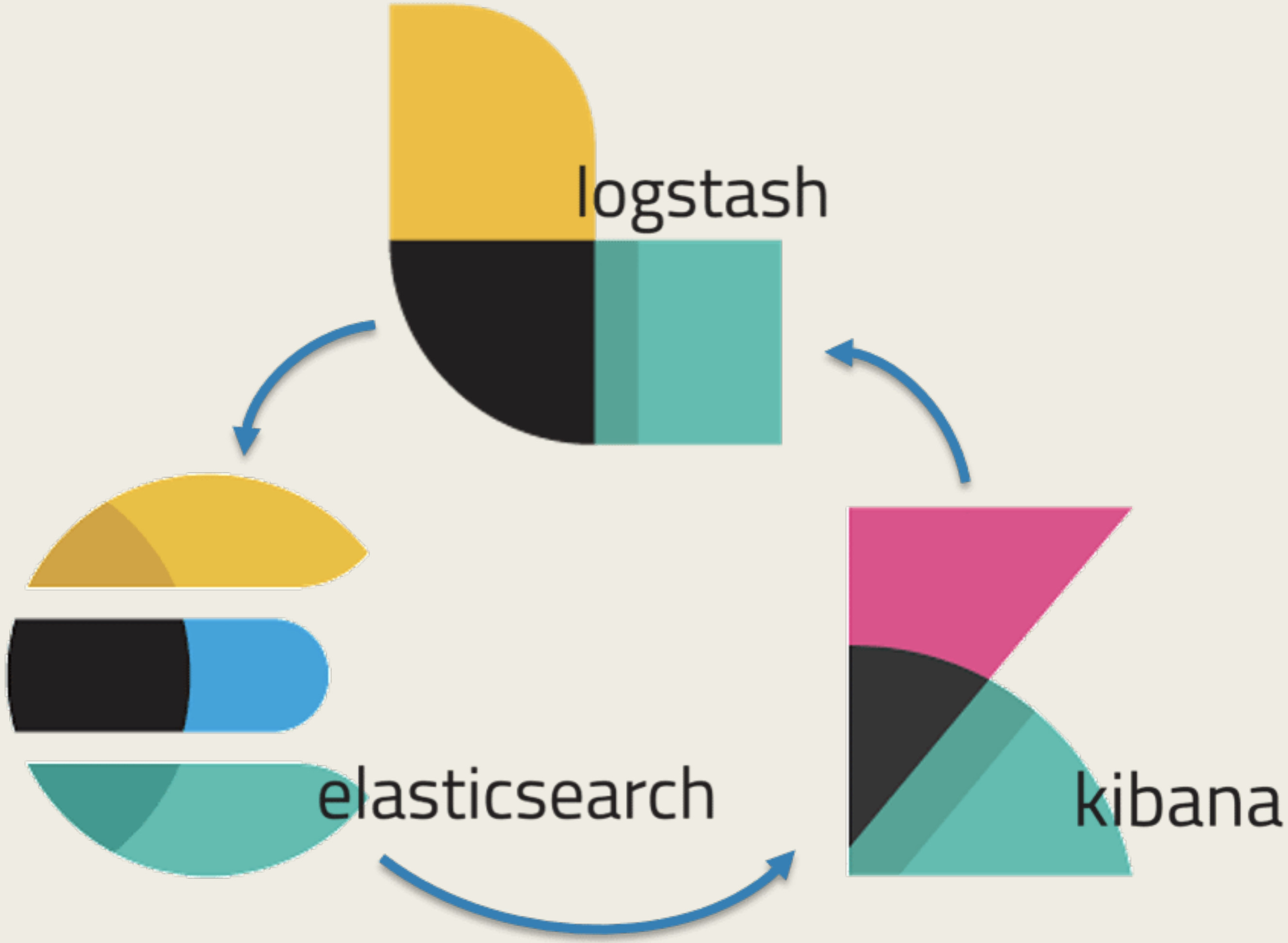
Kafka And Tweepy

- Tweepy:
  An easy-to-use Python library for accessing the twitter API

- Kakfa:
  Distributed queuing service to buffer the tweets before processing



ELK Stack

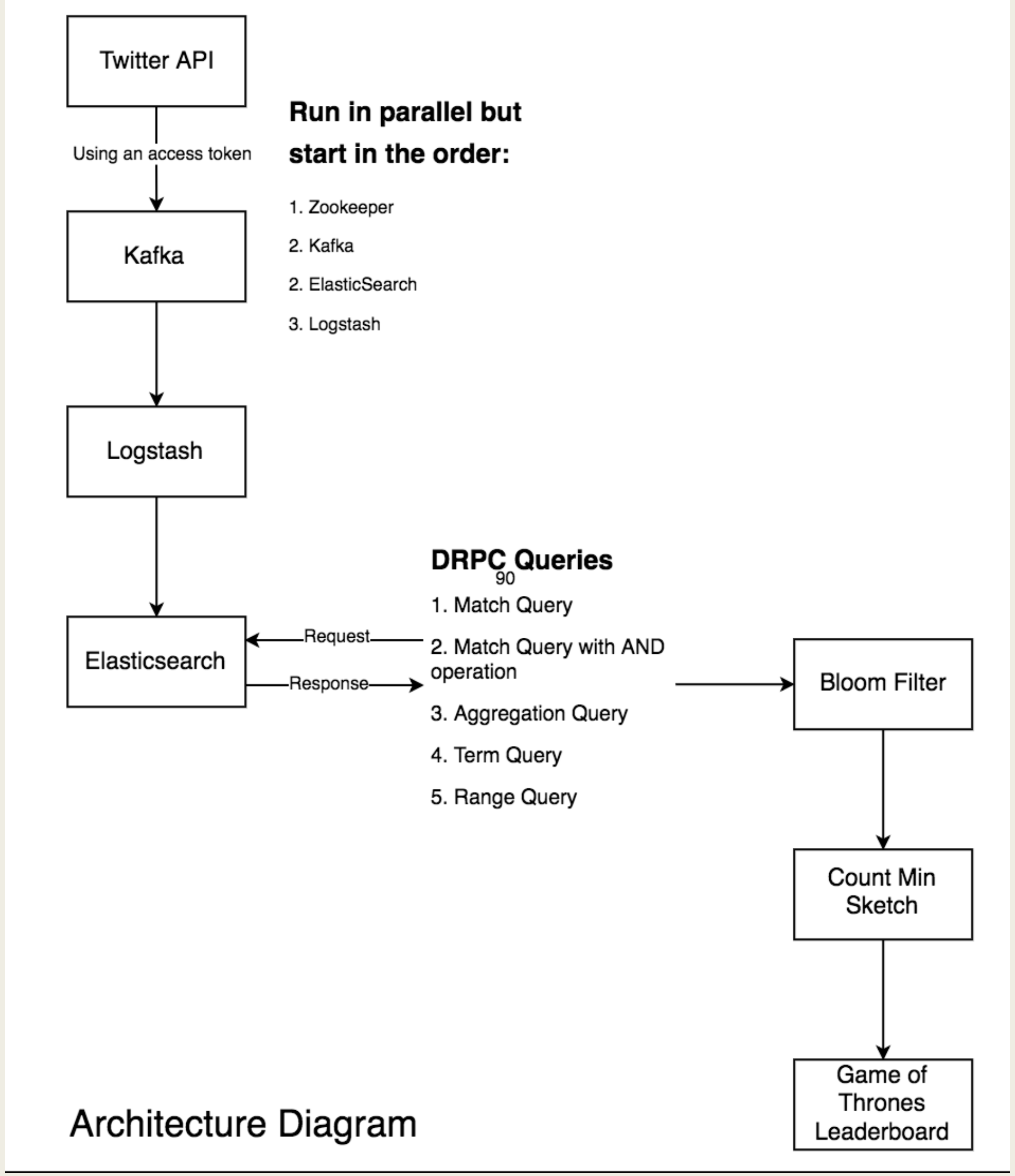


<u>ElasticSearch</u>:
Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases

<u>Logstash</u>:
Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite “stash.”

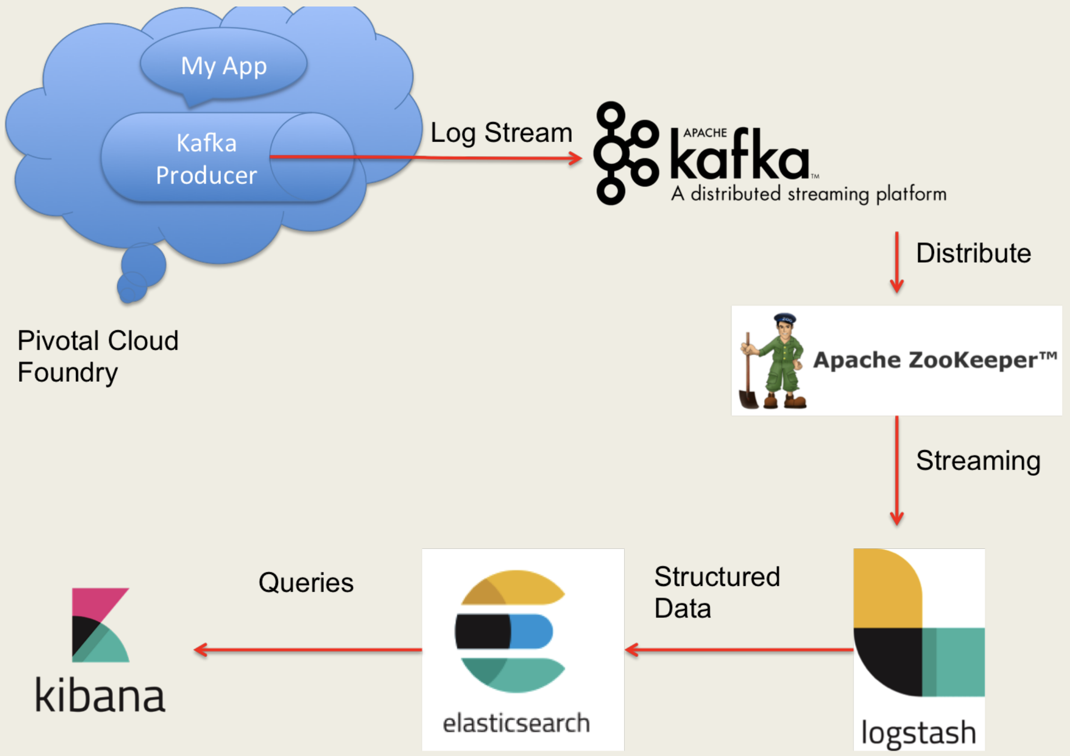
<u>Kibana</u>:
Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack

Project Architecture



Connection between Kafka & ES

- Pipeline:
  - *Extract data using tweepy and push to kafka*
  - *Using Logstash transfer data from kafka to elasticsearch*
  - *Using Kibana run DRPC queries and visualize data*

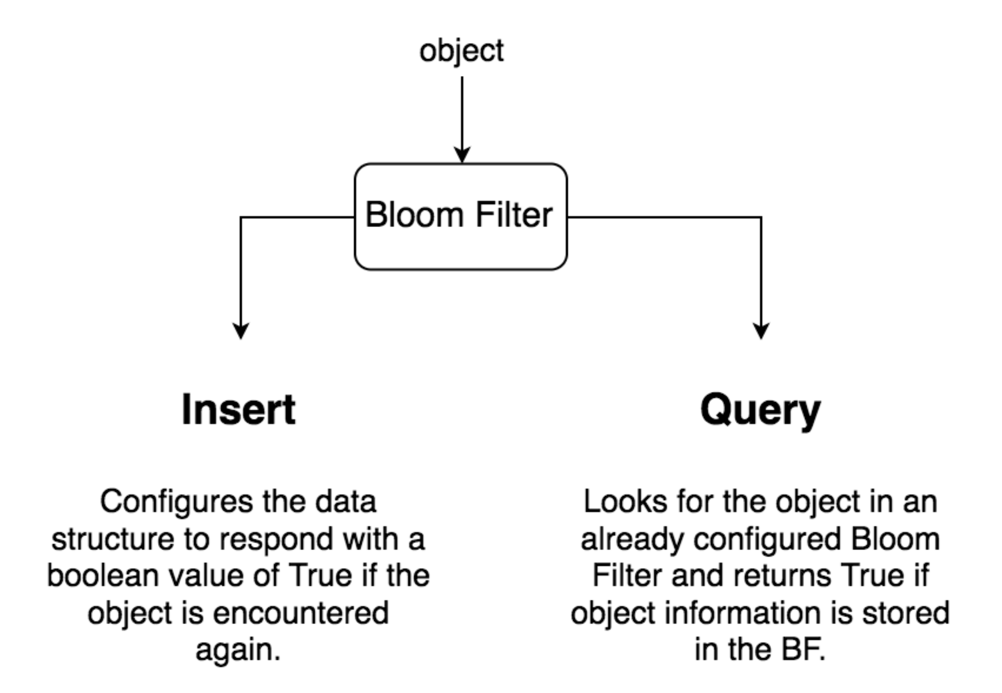


Count Min Sketches & Bloom Filters

- Sub linear space data structures
- Summarize data streams
- Probabilistic nature
- CMS are a refinement of BF

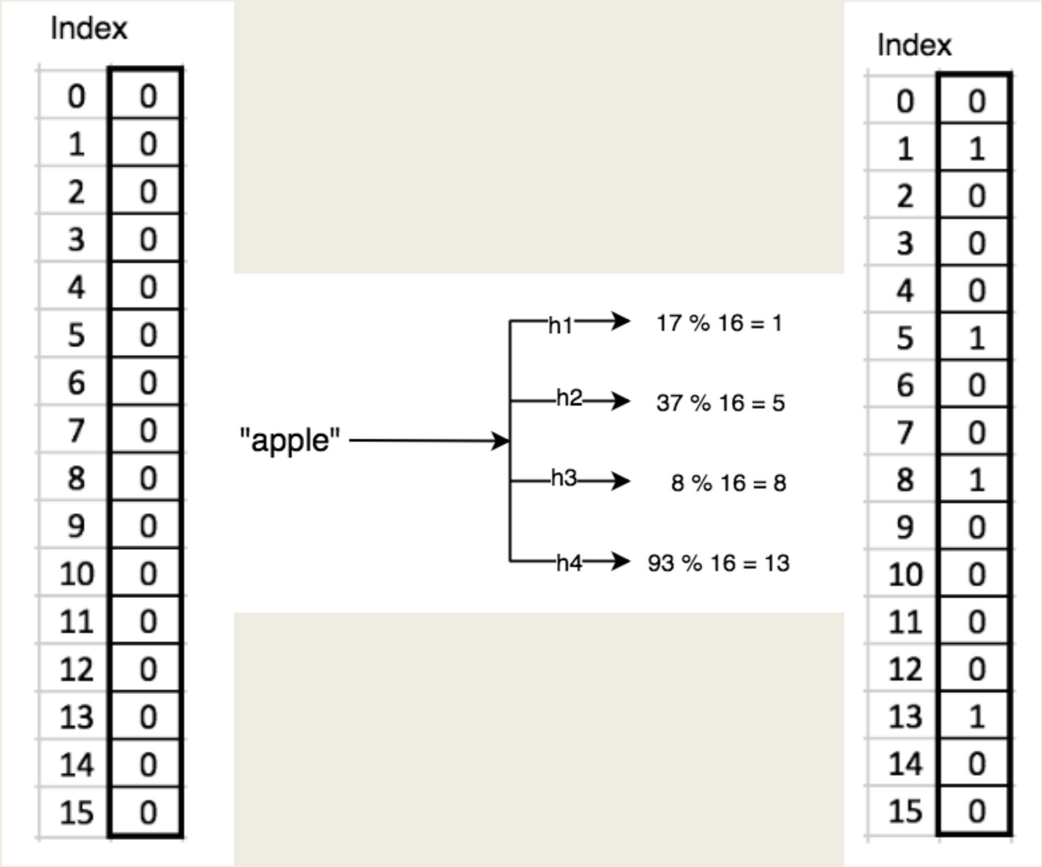
Bloom Filters Design

- Allows False Positives but not False Negatives
- The output a Bloom Filter gives is whether an input object has already been registered in it.
- BF supports insert and query operations
- It works by using k independent hash functions <h1, h2, …, hk> that map to indices in the range of 1 to m.



Bloom Filters Insert

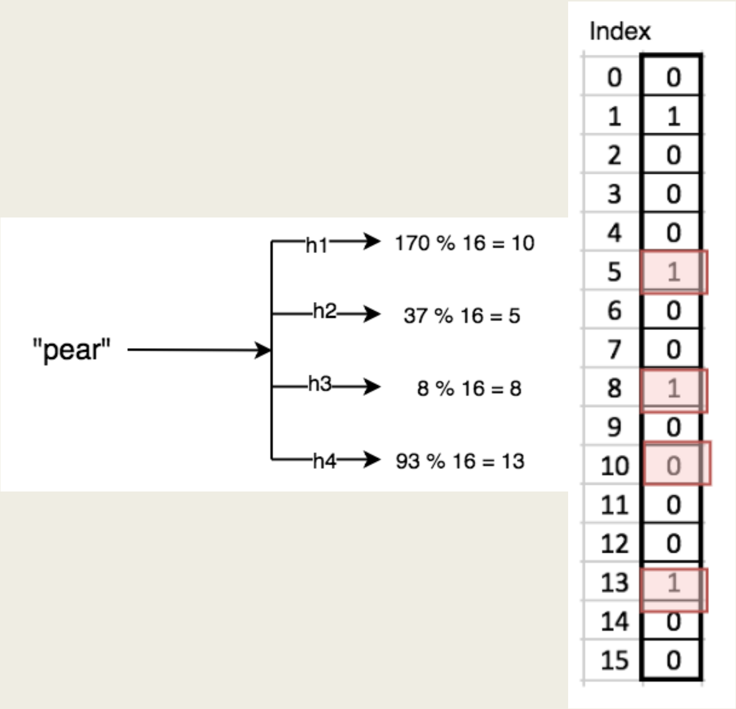
- We initialise an array of n bits to zero and use k independent hash functions.
- Assume we are working with a bloom filter of size 16 with 4 hash functions.
- O(k) complexity



Bloom Filters Query

- Perform AND operation on all indices found through hash functions.
- "pear" was not contained in the Bloom Filter in which we had inserted "apple".

  1 and 1 and 0 and 1 = 0
- O(k) complexity



Index	
0	0
1	1
2	0
3	0
4	0
5	1
6	0
7	0
8	1
9	0
10	0
11	0
12	0
13	1
14	0
15	0

Bloom Filter Equations

$$m = -\frac{n \ln p}{(\ln 2)^2}$$

$$k = \frac{m}{n} \ln 2.$$

Where,

- n = number of items in set
- m = number of bits in filter
- k = number of hash functions
- p = probability of a false positive

We generally fix values of n and p to determine the values of m and k

Count Min Sketches

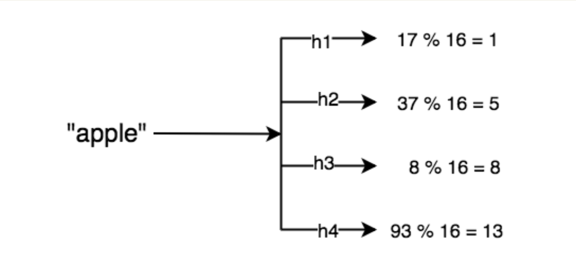
- An extension to Bloom Filters
- Acts as a frequency table of objects in a data stream
- Instead of an array of length m, CMS maintains a 2-dimensional data structure (mxk)
- The hash functions can deliver any value less than m. So every hash function and a value under m, map to a cell in the CMS

k hash functions

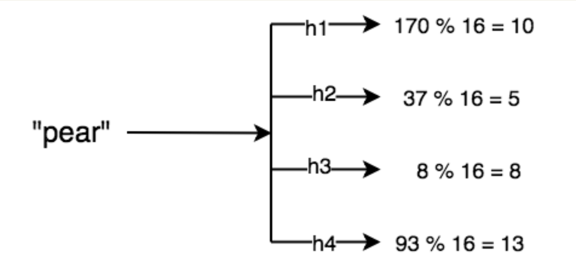
m rows

	h1	h2	h3	h4
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0

CMS Example



	h1	h2	h3	h4
0	0	0	0	0
1	**1**	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	**1**	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	**1**	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	**1**
14	0	0	0	0
15	0	0	0	0



	h1	h2	h3	h4
0	0	0	0	0
1	1	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	**2**	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	**2**	0
9	0	0	0	0
10	**1**	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	**2**
14	0	0	0	0
15	0	0	0	0

min(2,2,1,2) = 1

Top K Heavy Hitters

- Heavy hitters problem:
  - *Given a stream of length T and a parameter k, in a single pass over the stream we want to find any elements that appear at least T/k times.*
- Top k:
  - *Find the top k frequently occurring items in the data stream*

Game of Thrones Leaderboard

- Built a Bloom Filter based on 31 popular characters from Game of Thrones.
- We use k=7 hash functions and m=300 bit vectors for n=31 characters and an error rate of 1%
- Tweets checked for these names
- If present in the Bloom Filter, the name is added to a CMS that backs up the leaderboard

Leaderboard: [(27, 'samwell'), (41, 'bran'), (60, 'got\xe2\x80\xa6'), (86, 'cersei'), (176, 'jon')]

Conclusion

- We successfully built a pipeline to stream tweets from Twitter's API through Kafka, Logstash and index it in Elasticsearch.
- Results were obtained using CMS and BF by performing top k Heavy Hitters on their result.
- All top characters stayed at relatively same positions, even though their frequencies changed over time. The change was not found to be very drastic.
- We did not expect the leaderboard to change constantly. However, fluctuations were noticed after the release of a new episode.