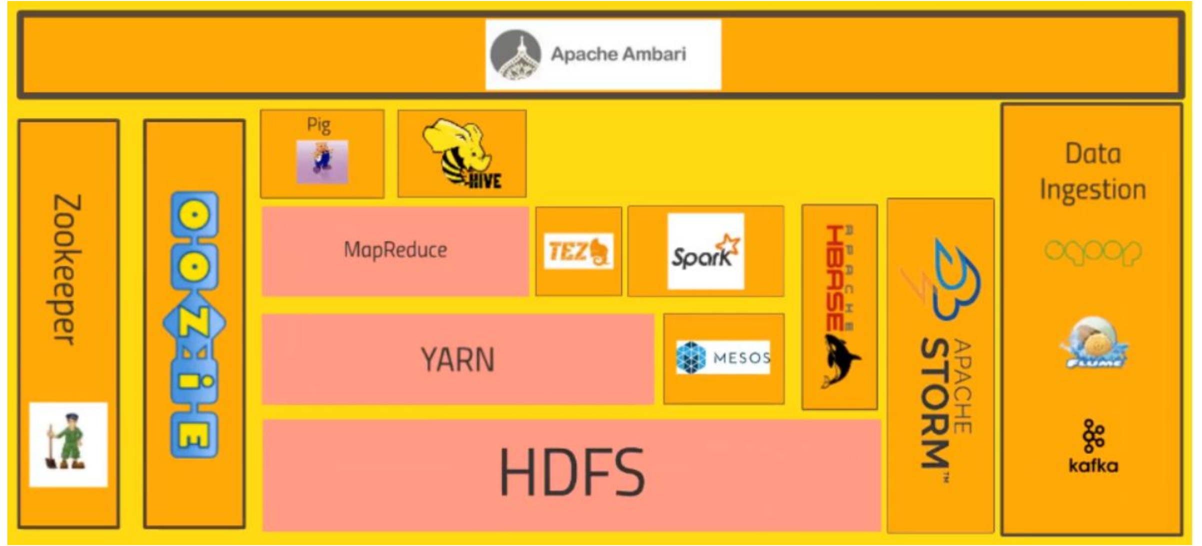
Hadoop & Distributed System Design

V2020.11.9

(Dr Yan Xu)

## Hadoop Ecosystem




source: https://www.udemy.com/course/the-ultimate-hands-on-hadoop-tame-your-big-data/

**1: Hadoop Core**

- HDFS: a distributed file system
- YARN: a system to schedule applications
- MapReduce: a distributed algorithm framework

**2: MapReduce Enhancement**

- Pig: Scripts (pig latin) -> MapReduce
- Hive: SQL (hiveql) -> MapReduce
- Tez: a faster alternative of MapReduce

**3: Spark**

- In-memory processing
- Spark SQL + Spark Streaming

**4: Databases**

- HBase, Cassandra, MongoDB

**5: Data Feeding**

- Kafka, Flume, Sqoop

**6: Analysing Data Streams**

- Spark Streaming, Storm, Flink

**7: Cluster Management**

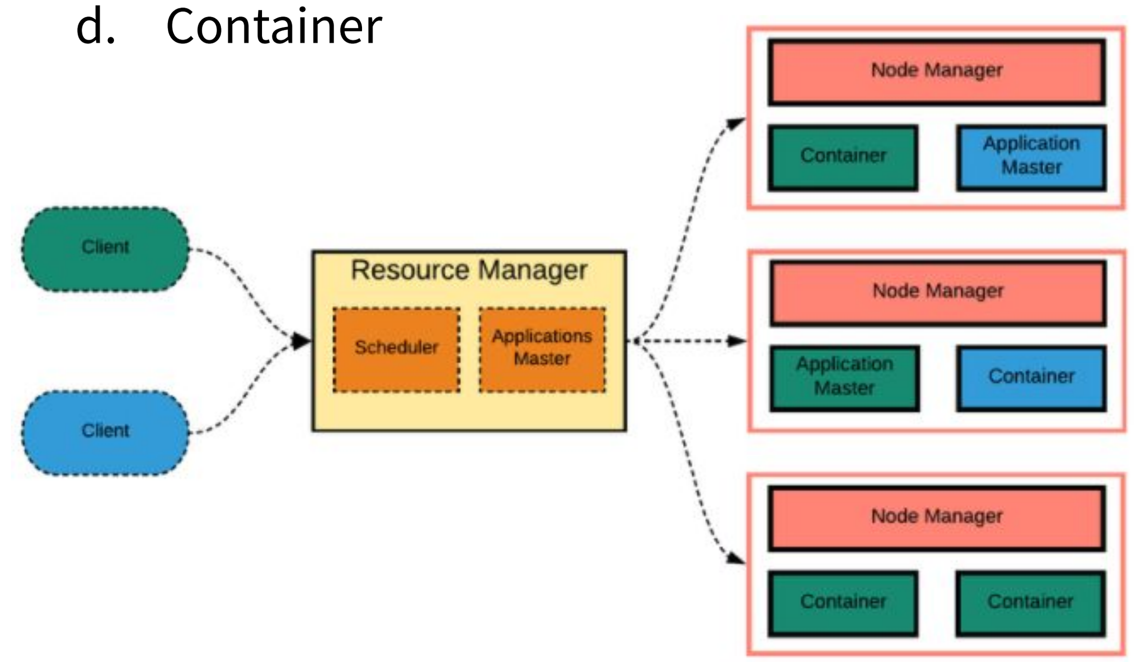
- ZooKeeper, Mesos, Oozie
- Zeppelin/Hue, Ambari

## HDFS

- → A master/slave architecture
  **NameNode** (master): manages the file system namespace and regulates access to files by clients
  **DataNodes** (slaves): manage storage attached to the nodes
- → HDFS is built using the Java language
- → Data Storage:
  - Block size
  - Block replication factor: e.g. 3
  - Rack-aware replica placement policy
- → Transaction log (Metadata Persistence)
- → Redundant NameNodes
- → FS Shell
  - bin/hadoop dfs -mkdir /foodir
  - bin/hadoop dfs -cat /foodir/myfile.txt

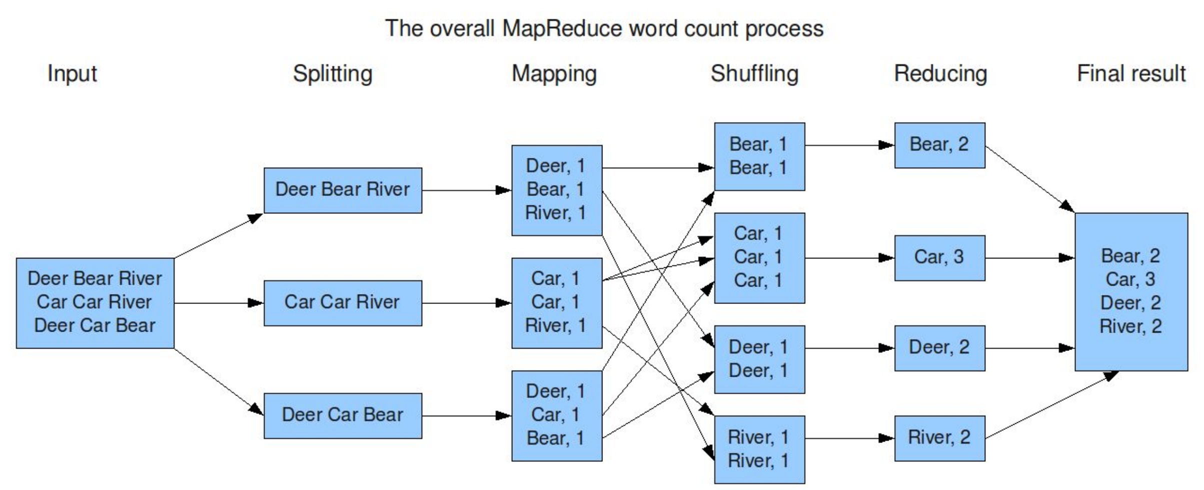
## YARN

- → Separates resource management layer from the processing layer, so to allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing on HDFS cluster.
- → Main components:
  a. Resource Manager
  b. Node Manager
  c. Application Master
  d. Container




## MapReduce

- → A programming model to process big data on a cluster.
- → Wordcount example:




- → A MapReduce framework:
  - map()
  - shuffle(): redistribute data based on the output keys by the map func
  - reduce()
- → MapReduce is designed to recover from the loss of whole nodes during the computation, so it writes interim results to distributed storage, which may not be a concern for your app and thus becomes not effective.

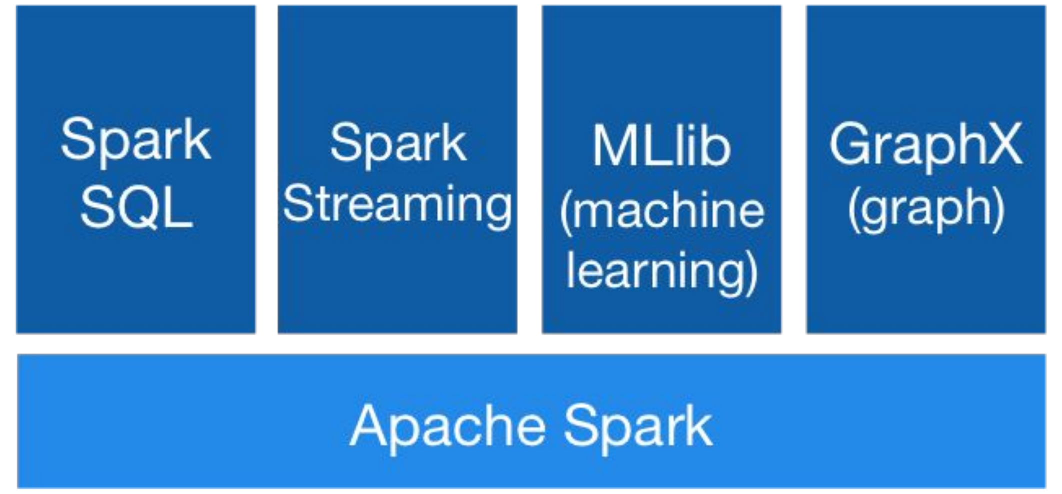
## Hive

- → **HiveQL**: gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.
- → In Hive, HiveQL is translated to a directed acyclic graph of MapReduce, Tez, or Spark jobs, which are submitted to Hadoop for execution.
- → example:

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS word FROM docs) temp
GROUP BY word
ORDER BY word;
```

## Spark

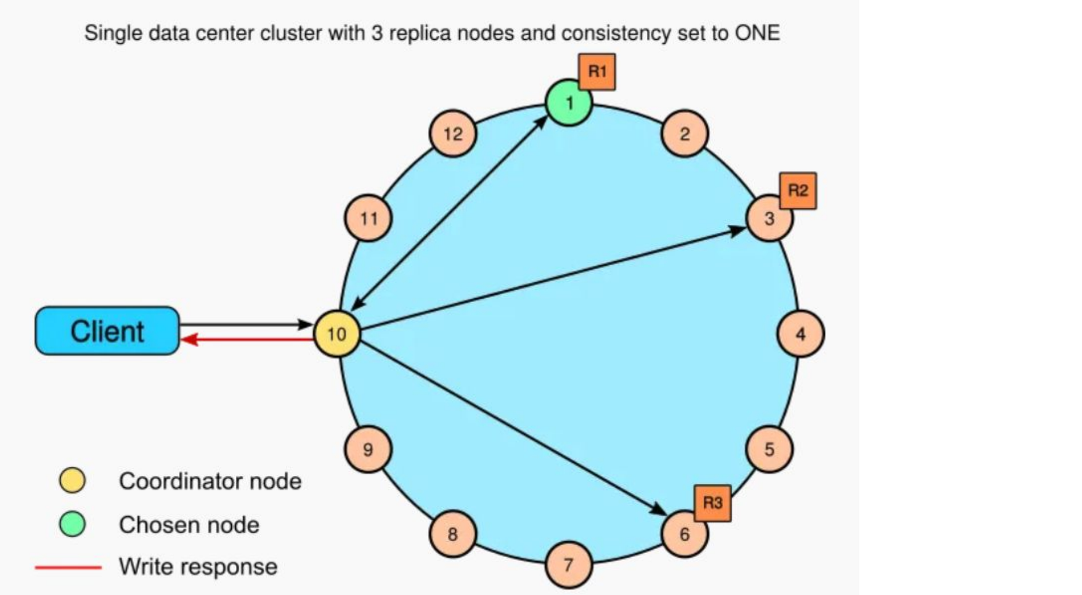
- → An open-source general-purpose distributed cluster-computing framework.




Details: https://github.com/xuyannus/CheatSheet/blob/main/PySpark%20Basic%20Cheat%20Sheet.pdf

## Cassandra

- → **Horizontal Scaling**: read and write throughput increase linearly as new machines are added
- → **Masterless** (no single point of failure)
- → Key Concepts:
  1. Gossip Protocol
  2. Consistent Hashing
  3. Wide-column Storage (or two-dimensional key–value store.)
  4. Cassandra Query Language (CQL)
  5. Shell: CQLSH

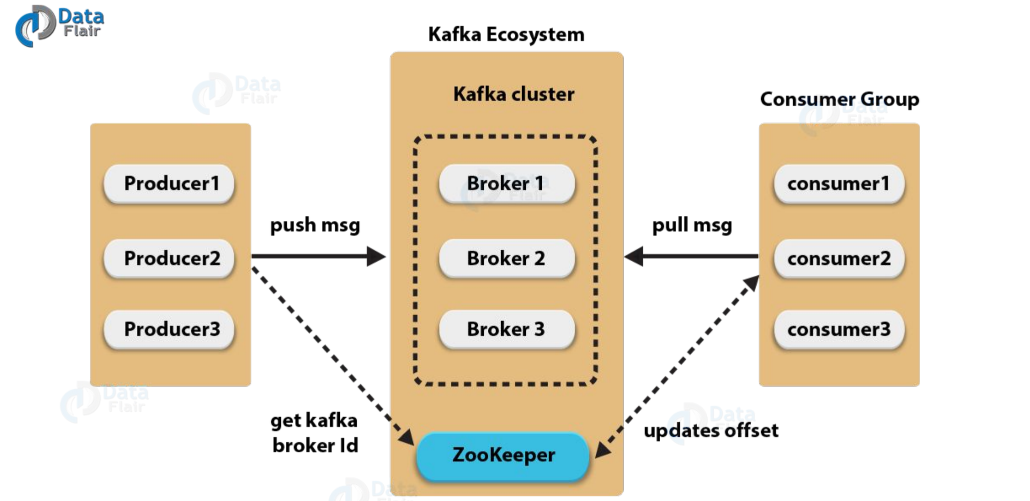



- → Example CQL:

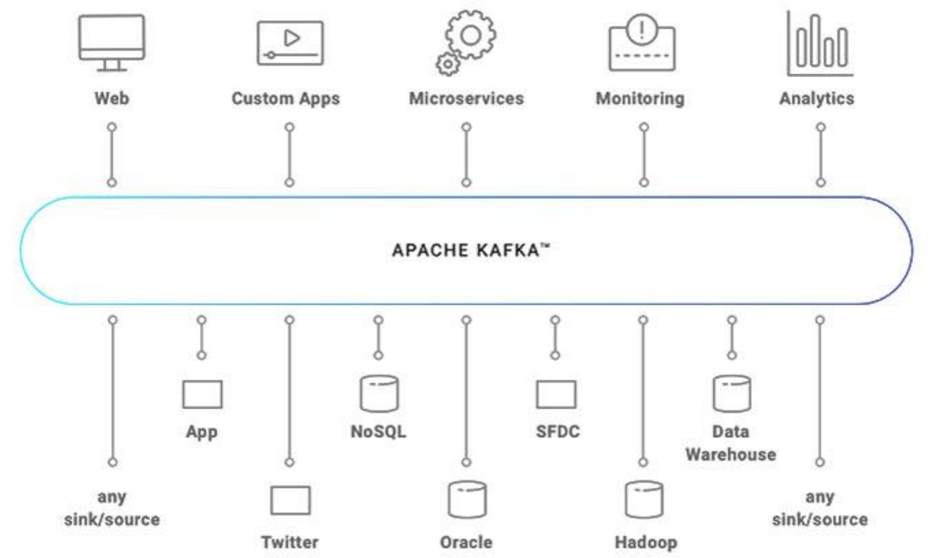
```
CREATE KEYSPACE MyKeySpace
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

USE MyKeySpace;

CREATE COLUMNFAMILY MyColumns (id text, Last text, First text, PRIMARY KEY(id));

INSERT INTO MyColumns (id, Last, First) VALUES ('1', 'Doe', 'John');

SELECT * FROM MyColumns;
```

## Kafka

- → to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.
- → Architecture




- → Key KPIS:
  - Producer API
  - Consumer API
  - Streams API (input topics -> process -> output topics)
  - Connector API (connect to existing applications or data systems)
- → Topic Partition
- → Topic Replication Factor (e.g. 2)
- → Application Scalability




## Sqoop

- → a command-line interface application for transferring data between RDBMS and Hadoop.
- → Incremental load & Parallel load
- → Commands:
  - sqoop import --connect jdbc:xx --table xx --target xx
  - sqoop export --connect jdbc:xx --table xx --export-dir xx
  - sqoop list
- → Interactive shell: sqoop2-shell

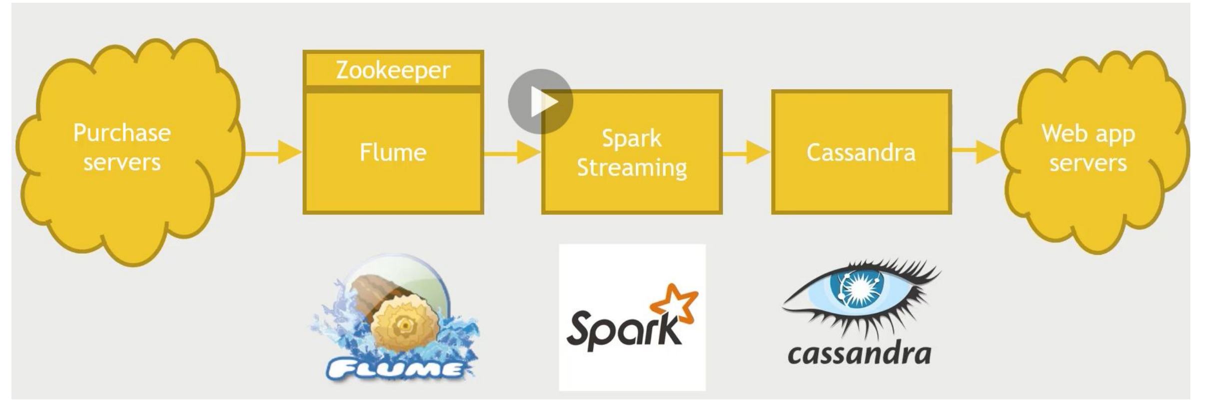
## Spark Streaming

- → A streaming processing system that natively supports both batch and streaming workloads.
- → **DStream**: RDDs with small batches
- → **Microbatches** (e.g. 1 sec) & windowing (e.g. 30 mins)
- → Example code:

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

sc = SparkContext(appName="PythonSparkStreamingKafka")
ssc = StreamingContext(sc, 5)   # batch duration 5 sec
kafkaStream = KafkaUtils.createStream(ssc, kafka-stream-xxx)    # kafkaStream is RDD
kafkaStream.map(xxx).countByValueAndWindow(60, 5)   # window calculation

ssc.start()
ssc.awaitTermination(timeout=180)
```

## System Design 1: top 10 best-selling books

Keywords: millions users, hourly update, low consistency

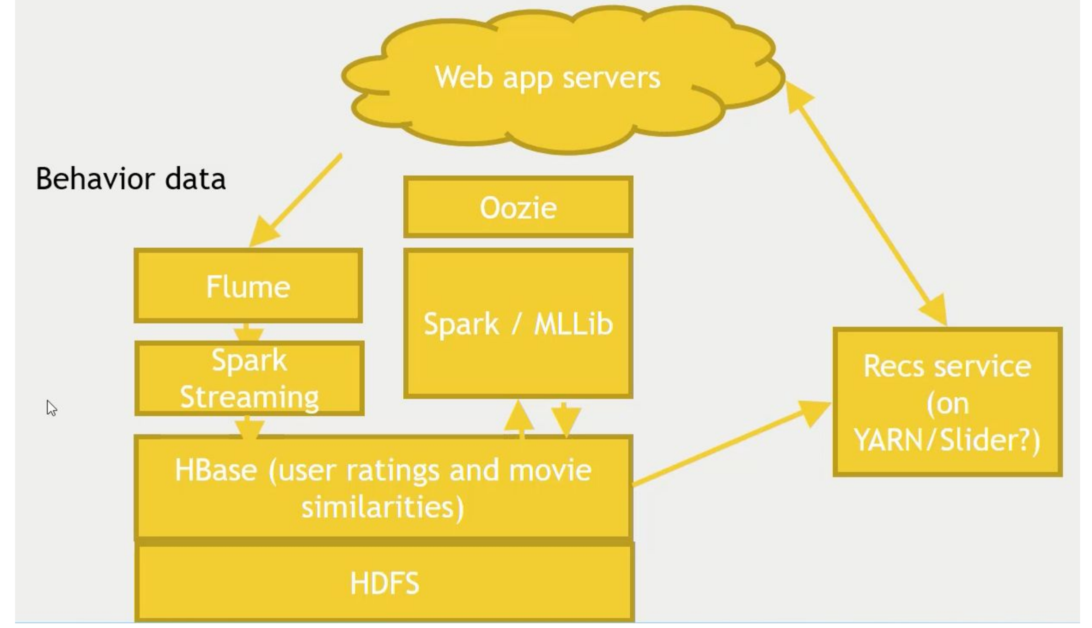



source: https://www.udemy.com/course/the-ultimate-hands-on-hadoop-tame-your-big-data/

1. Each Service itself (e.g. Spark, Cassandra) is running on a cluster

## System Design 2: movie recommendation

Keywords: millions users, fast response to user behavior, low consistency




source: https://www.udemy.com/course/the-ultimate-hands-on-hadoop-tame-your-big-data/

1. ML model captures the more stable movie-movie similarity.
2. After a user action, Recs Service needs:
   a. the user's recent actions (likes & dislikes)
   b. movie-movie similarity
   to rank candidates and pick the top K movies