**Apache Spark**

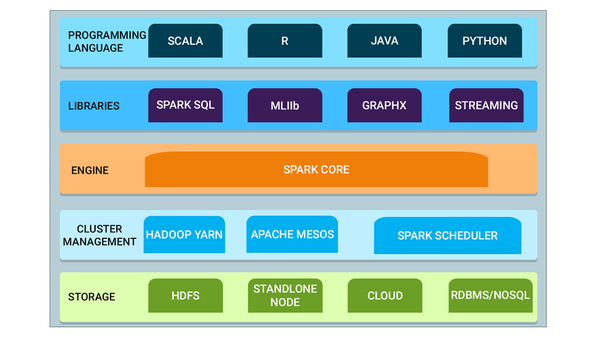
**The programming used is pyspark, java, R or scala all are api**

Spark is open source, parallel processing, scalable in memory execution environment for running analytics application. Spark distribute the work across clusters and do parallel processing of data. In **Apache Spark**, In-memory computation is done instead of storing data in some slow disk drives(in Hadoop) the data is kept in random access memory(RAM).

Features of spark

**Speed, Powerful caching, Real - Time, Deployment, Polyglot.**

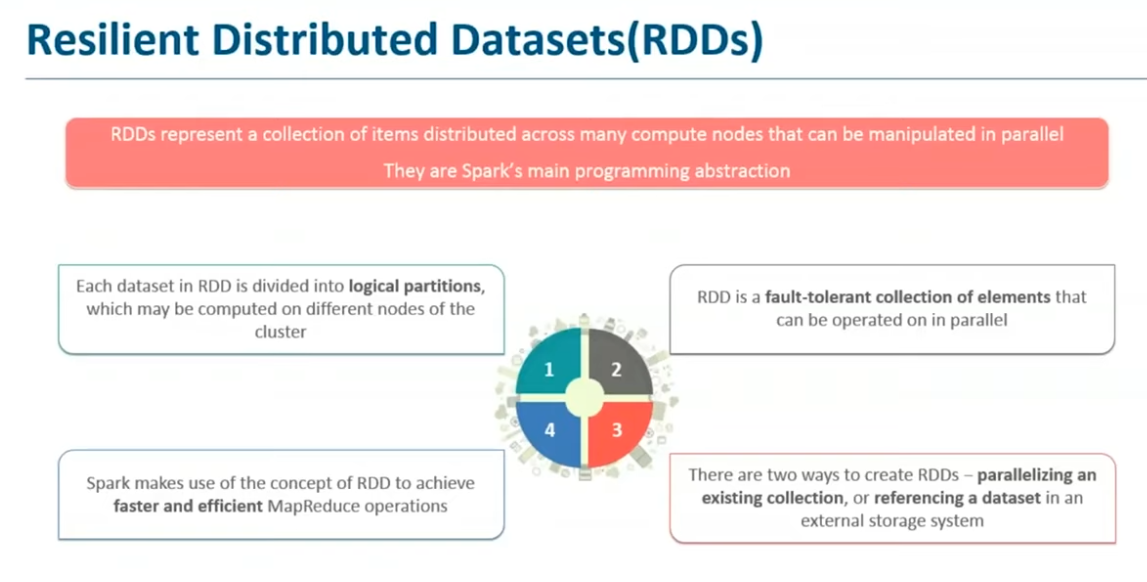
**Components/ecosystem of SPARK**

****

**RDD (Resilient Distributed Dateset)**

RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. RDD are immutable. They are spark's main programming abstraction.

**RDD is an immutable distributed collection of objects. Objects can be anything like strings, rows etc.**

****

Features of RDD

In-memory computation.

Lazy Evaluation,

Fault Tolerant,

Immutability, Partitioning,

Persistence.

Crete an RDD, there are totally 3 ways to create.

1. Parallelized Collections
2. From RDDs
3. External Data.

**Parallelized Collections**

scala> sc.parallelize(1 to 100, 5).collect() -- this creat ean array of elements from 1, 2,3, 4, ..100. Which is referred as RDD.

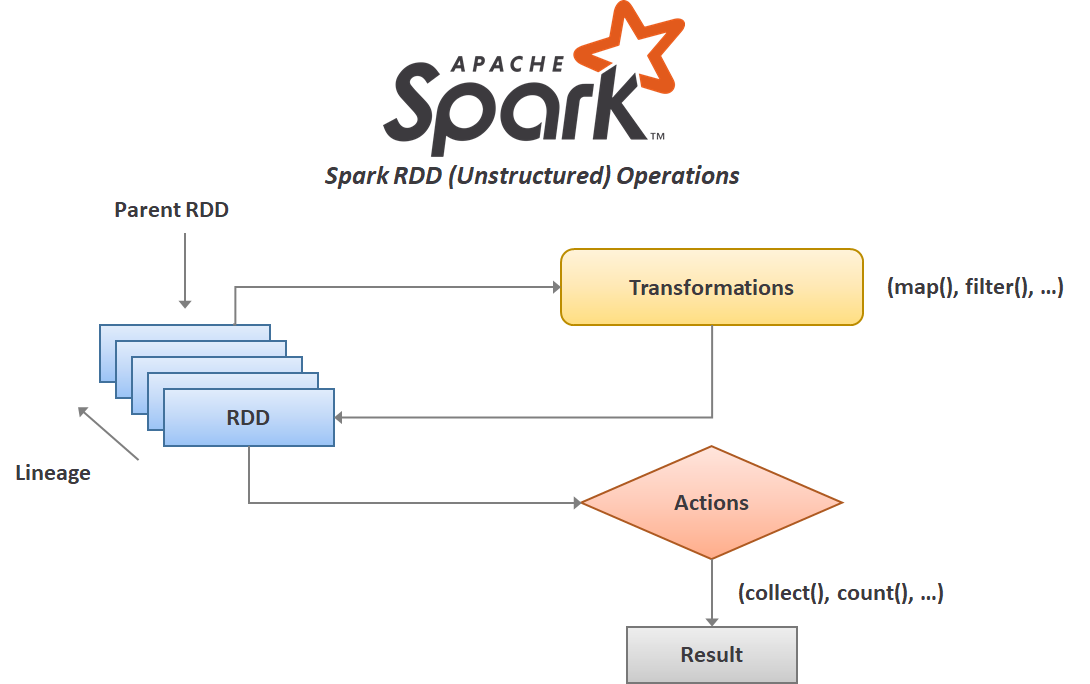
**From RDDs**

scala>Var a1 = Array(1, 2, 3, 4,5, 6,7)

scala>Var r1 = sc.parallelize(a1) --the result of this is an rdd

scala>Val newrdd = a1.map(data → (data\*2))

**RDD Operations**

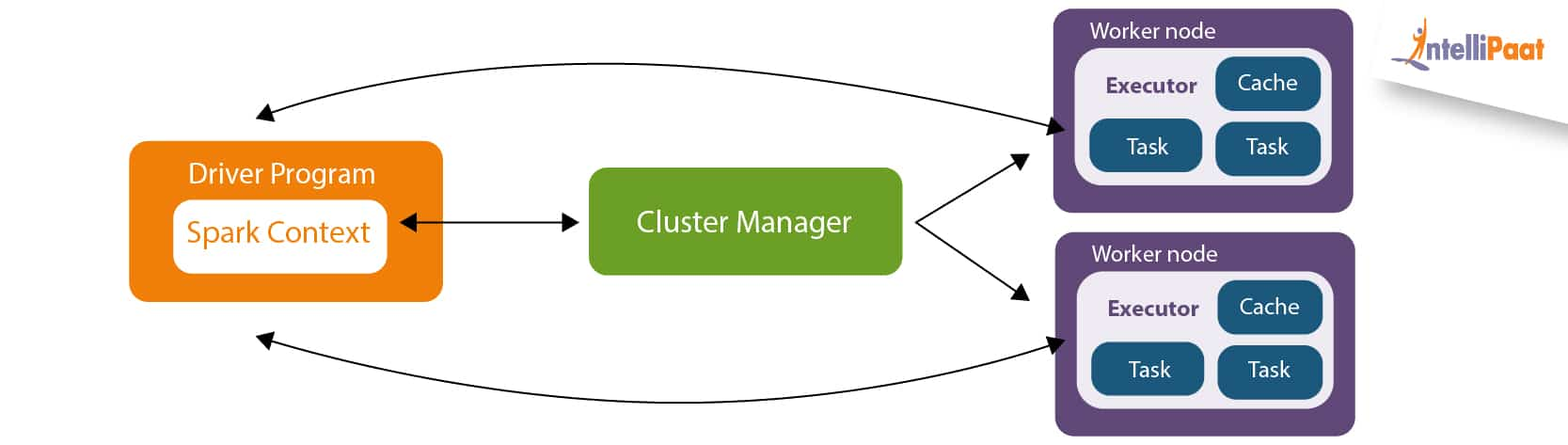


1. Transformations

2. Actions

RDDs support two types of operations: **transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset**.

**Apache Spark Architecture**

****

Driver Program - Drives the application, Inside the driver you create a spark context. Spark context is the gateway to all the spark functionality.

Spark Context - Spark context communicate with cluster manager(YARN) to manage various jobs. The driver program and spark context take care the job execution across cluster. A job is split into task, and distributed across worker nodes.

Worker node - These nodes's job is to execute the tasks and then update back to spark context.

Caching - You can cache the jobs for faster execution. By caching you create a checkpoint in your spark application and if further down the execution of the application any of the tasks fail your application will be able to recompute the lost RDD partition from the cache.



Example, to count the number of words in text file

Always run command sudo jps - to check all the demons are running are not.

>sudo jps

>spark-shell

scala>> var test = sc.textFile(“Path)

scala>> test.collect()

Scala>> var map =sc.textFile(apth).flatMap(line →line.split(“ “).map(word → (word,1));

Apply the action now, before that spark don’t start execution.

scala>> var counts = map.reduceByKey( \_ + \_);

scala>> counts.saveAsTextFile(“Location path”)

**SPARK RDD[Resilient Distributed Dateset]**

RDD is a distributed collection of memory where data needed is always stored and kept available at RAM.

RDD are fundamental data structure of spark, They are fault tolerant, multiple executable nodes, Automatic data Rollback, In-memory computation(they don’t need and hard disk they just need RAM). RDD can process any kind of data(structured, semi-structured, unstructured)

Sources of RDD - You can load the data from hdfs, Amazon s3, Hive, Hbase.

**FEATURES OF RDD**

1. **In-Memory Computation.**
2. **Lazy Evaluation.**
3. **Fault Tolerance.**
4. **Immutability.**
5. **Partitioning.**
6. **Persistence.**
7. **Coarse grained operation.**

**Creation of RDD**

Parallelized collections →

val PRDD = spark.sparkContext.parallelize(Array(“Monday”,”Tuseday”),2)

From External storage

Val sparkfile = spark.read.textFile(“location/path”)

From RDD

**Operations Performed on RDD**

1. **Transformations**

Transformations are applied on RDDs to Access, Modify and filter the current RDD to generate a new RDD. The new RDD returns a pointer to the previous RDD to ensure the dependency between them.

1. **Narrow Transformation -** Narrow transformations are applied on to a single partition of the parent RDD to generate the new RDD as the data required to process the RDD is available on the single partition of the parent RDD.

**map(), filter(),flatmap(),partition(),mapPartition()**

1. **Wide transformation -** Wide transformations are applied on to a multiple partitions of RDD as the data required to generate a new RDD is present among the multiple partitions of RDD.

**union(),reduceBy()**

**2. Actions**

Actions are called as the instructions given to apache spark to apply the computations and generate the results and display them to the end user.

**collect(), count(), take(), First()**

****

**SPARK DataFrame**

A data frame is a distributed collection of data, which is organized in rows under named columns.

**Data frame supports multiple programming language, Multiple data source, Process structured and unstructured dataset, Slicing and Dicing data.**

Features of Data Frame

1. Immutability.
2. Lazy Evaluation, [Spark will not throw an output on to screen unless an action operation is provoked.]
3. Fault Tolerance
4. Distributed Storage.

**Creation of Dataframe**

**Employee data**

Val employee = Seq(Row(“Mike”, “Robert”,”[abc@gmail.com](mailto:abc@gmail.com)”,10000), Row(“vike”, “kobert”,”xyz[c@gmail.com](mailto:abc@gmail.com)”,15000))

Employee schema

Val EmployeeSchema = List(StructField(“FirstName”, StringType, true), structField(“LastName”, String, true))

Create Data Frame

Val EmpDf = spark.CreatedateFrame(spark.sparkcontext.parallelize(Employee), structType(Employeeschema))

EmpDF.show

**SPARK SQL**

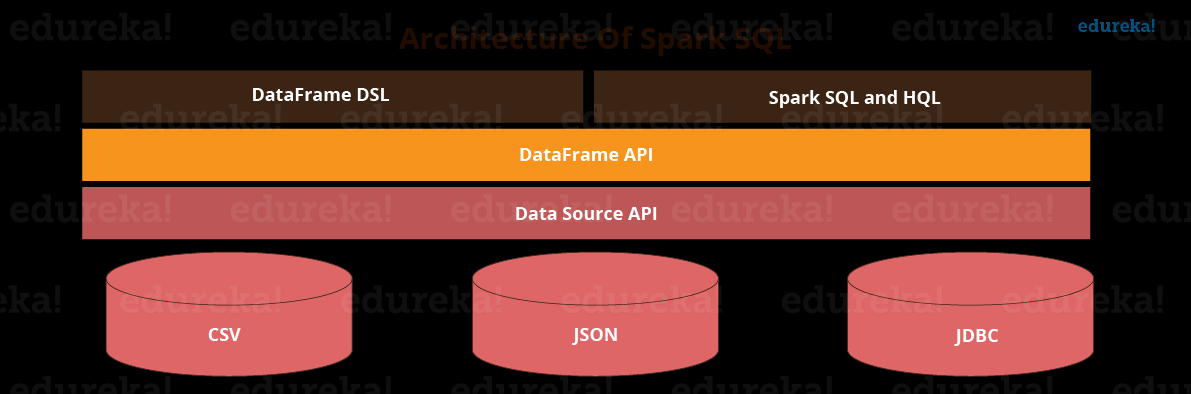
**Spark SQL can’t deal with unstructured data.**

Spark sql was built to overcome the limitations of apache hive. We can do real time querying compare to hive which is batch wise.

You can convert hive sql to spark sql.

Spark SQL uses the **metastore services** of hive to query the data stored and managed by hive.

**Spark Sql Architecture**



**Spark SQL Libraries**

**Data Source API -** Data source api is used to read and store structured and semi-structured data into spark sql.

**Data Frame API -** Data Frame API converts the data that is read through data source api into tabular columns to help perform sql operations.

**SQL Interpreter and Optimizer -** SQL interpreter and optimizer handles the functional programming part of spark sql. It transforms the dataframes RDDs to get the required results in the required format.

**Spark Streaming**

Data streaming is a technique for transferring data so that it can be processes as a steady and continuous stream.

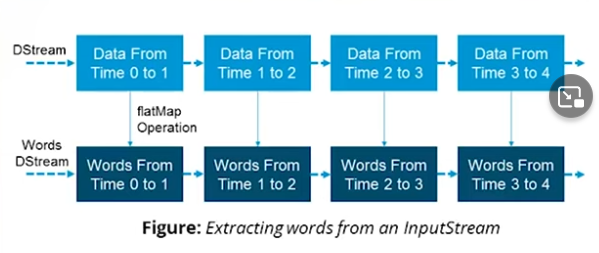
**Spark Streaming features**

1. Scaling,
2. Speed
3. Fault Tolerance,
4. Integration,
5. Business Anlysis.

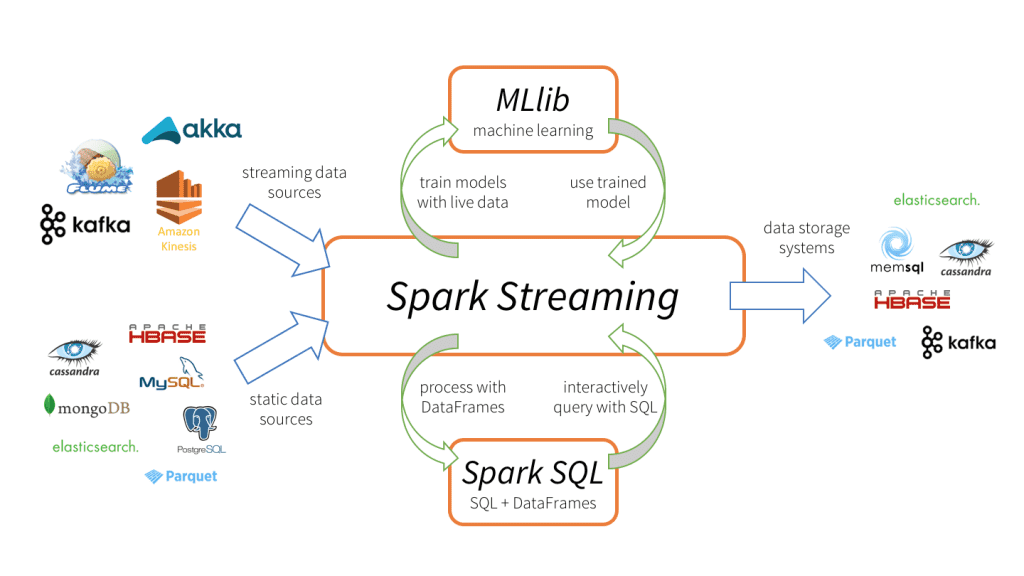
****

**The fundamental stream unit is DStream which is basically a series of RDDs to processes the real time data.**

****

****

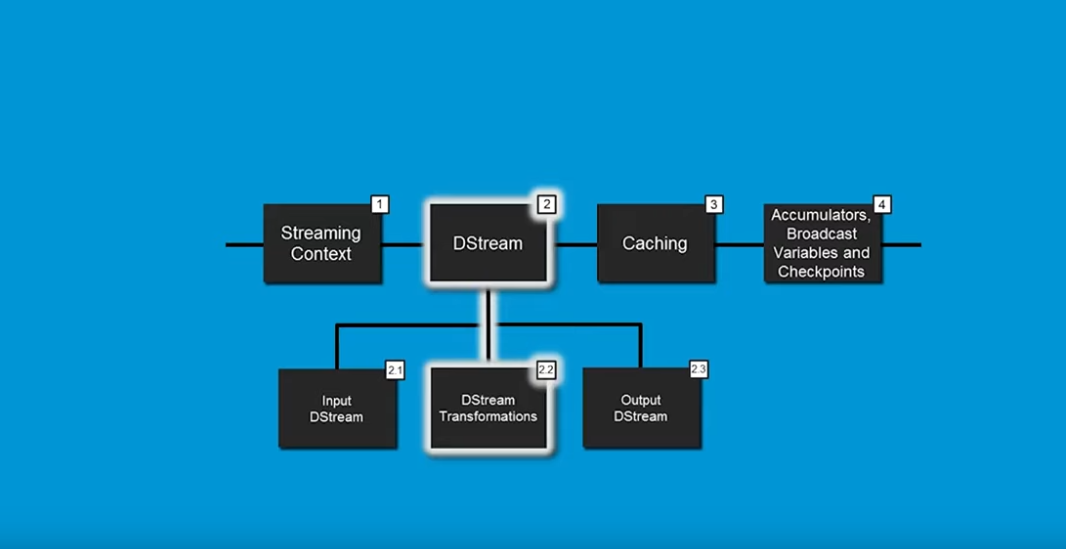
**Spark streaming workflow**

****

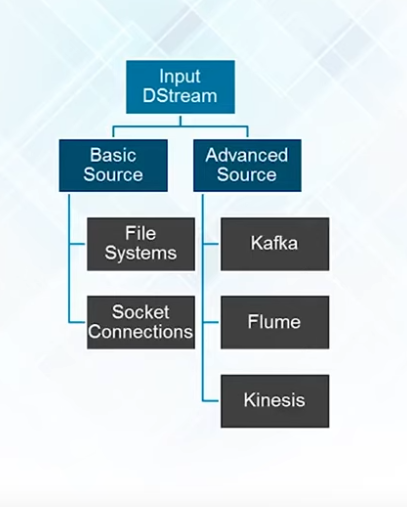
**DStream(Discretized Stream)**

**It is the basic abstraction provided by spark streaming. It is a continuous stream of data. DStream is represented by a continuous series of RDDs.**

****

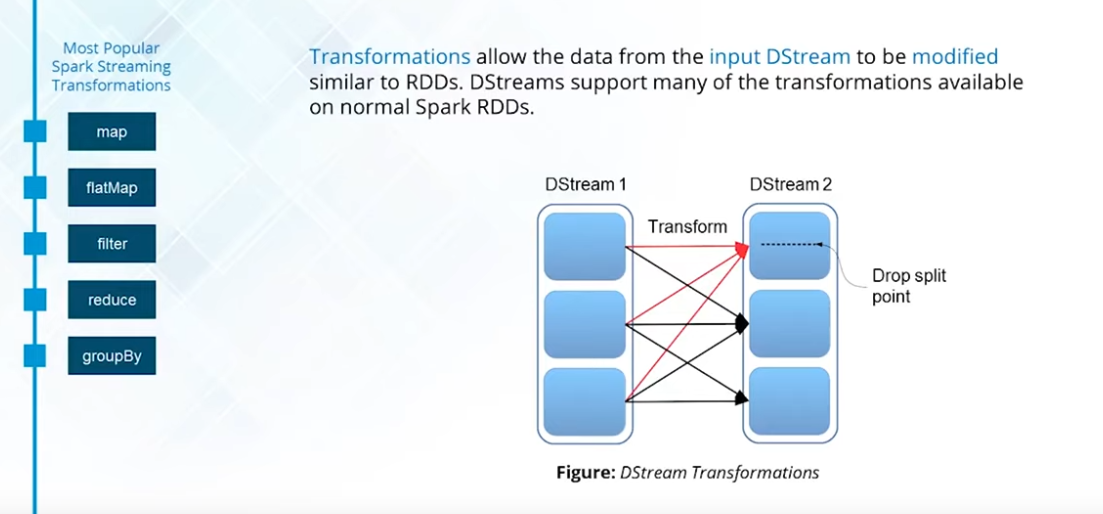
****

**Input Dstream represent the stream of inputs data received from streaming source. The streaming source are Basic Source, and Advance source.**

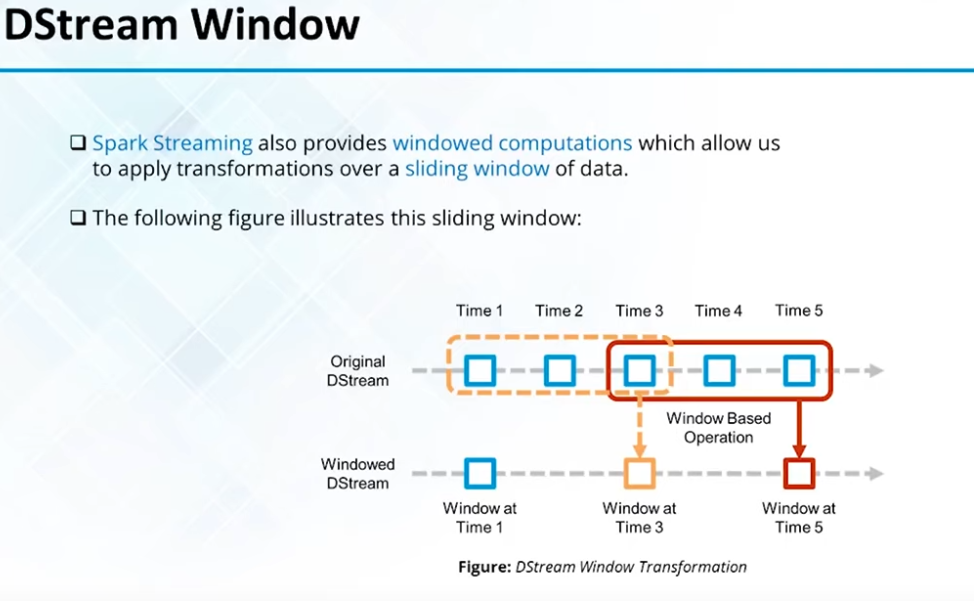
****

DStream Transformation

The data is modified according to business logic.



**DStream Window**

****

**Spark MLlib**

**Mllib Techniques**

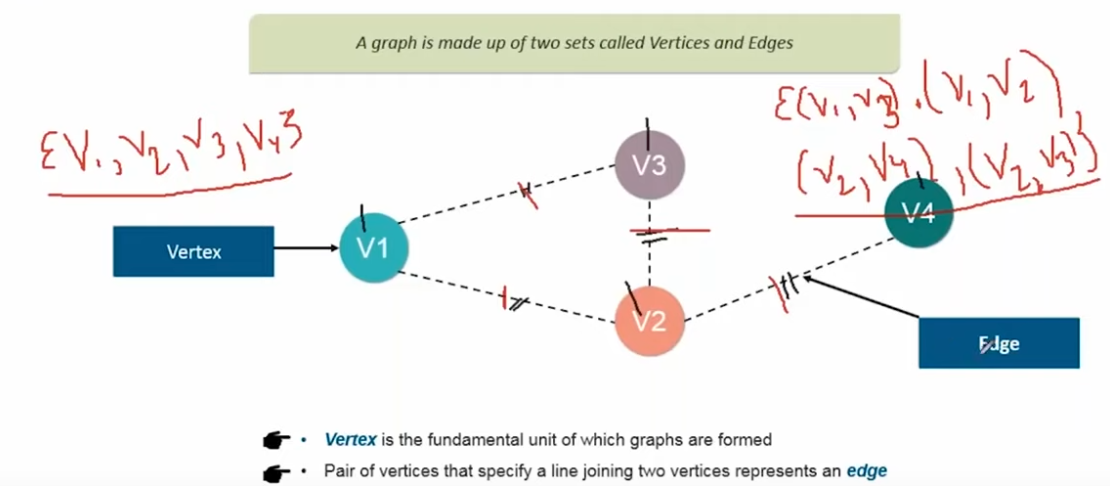
1. Classification
2. Clustering
3. Collaborative Filtering
4. Regression

**Spark MLlib tools:**

1. ML Algorithms
2. Featurization
3. Pipelines
4. Persistence
5. Utilities

**SPARK GRAPH X**

A graph is made up of two sets called vertices and edges.



**Types of Graphs**

1. Undirected Graphs: The order of vertices doesn’t matter.
2. Directed Graphs: The order of the vertices in pairs in edge set matter.
3. Vertex labeled Graphs: Each vertex is labelled with some data in addition to the data that identifies the vertex.
4. Cyclic Graph
5. Edge Labelled Graphs
6. Weighted graphs
7. Directed Acyclic Graph
8. Disconnected Graphs

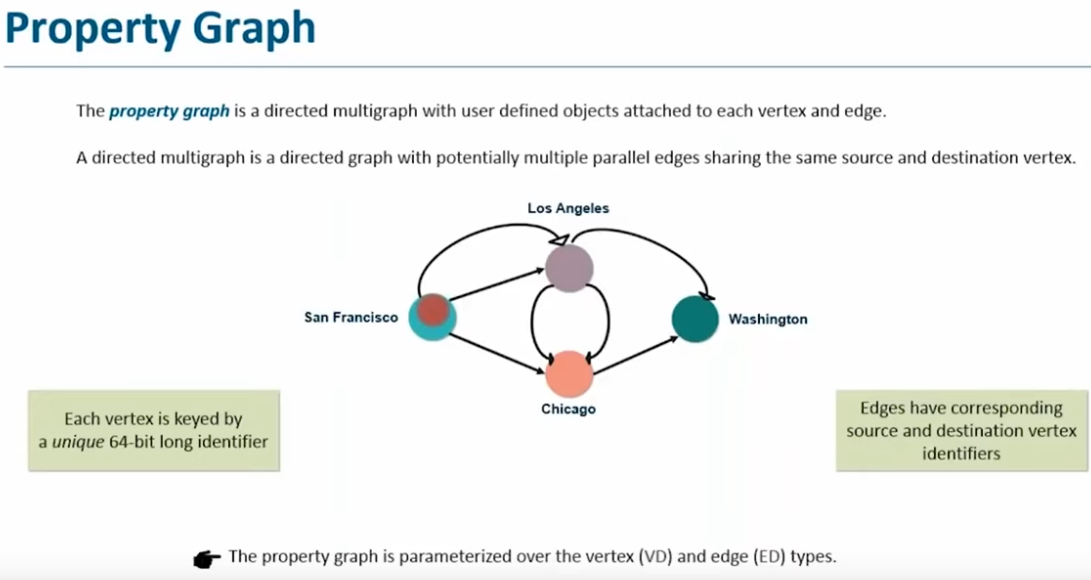
**Spark GraphX**

GraphX is a new component in spark for graphs and parallel computation. At high level graphX extends the spark RDD.

**Property Graph**

The property graph is directed multigraph with user defined objects attached to each vertex and edges.

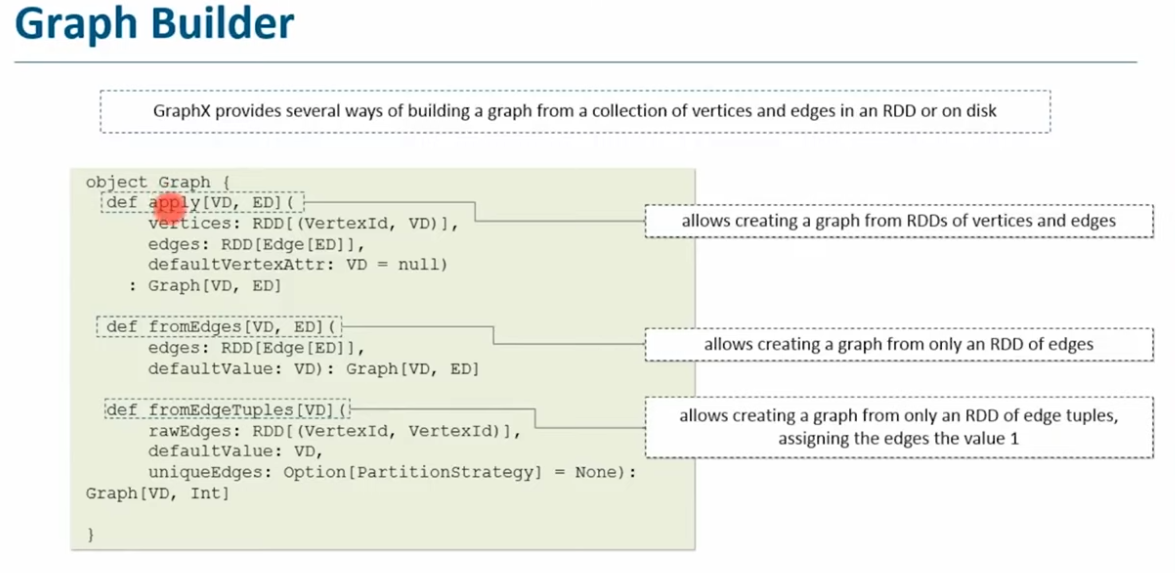
A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex.



**Each vertex has 64 identifiers.**

**GRAPH Builder**

GraphX provides several ways of building a graph from a collection of vertices and edges in an RDD or on Disk.



**VERTEX RDD**

The VertexRDD[A] extends RDD[(VertexId, VD)] and adds the additional constraint that each vertexId occurs only once.



**EDGE RDD**

The EdgeRDD[ED], which extends RDD[Edge[ED]] organizes the edges in blocks partitioned using one of the various partitioning strategies.

****

**Graph Operators**

1. Property Operator
2. Structural Operator
3. Join Operator
4. Neighborhood Operator

**Kafka with Spark Streaming**

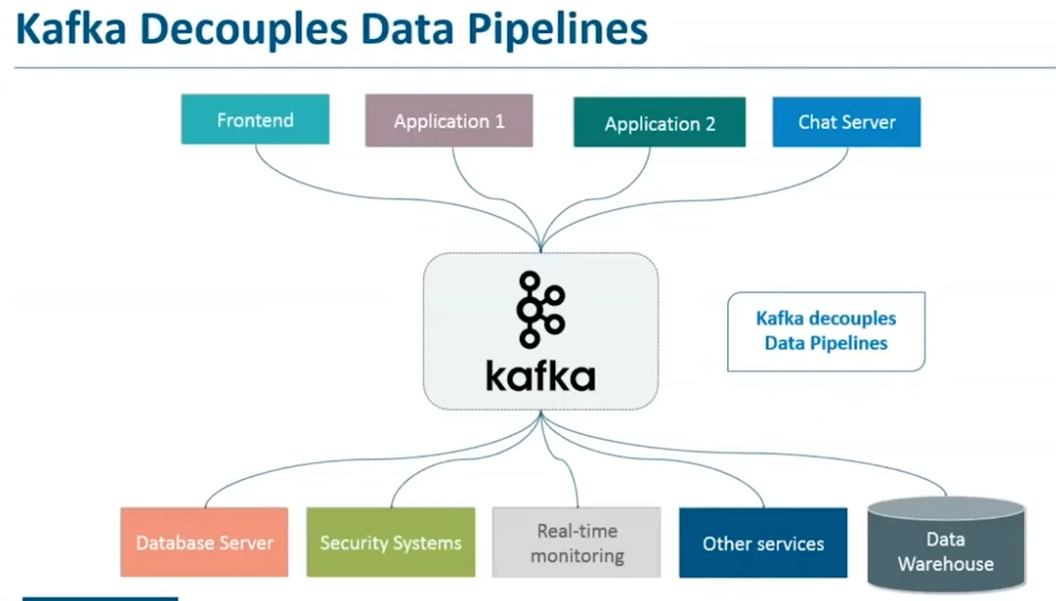
**Apache kafka is a distributed publish-subscribe messaging syste**m.

In messaging system, there are two models **Queuing and Publish-subscribe**

**Queuing -** A pool of consumer read from server.

**Publish-subscribe - A record is broadcasted across all customers so multiple customer can consume.**

Kafka decouples data pipelines.



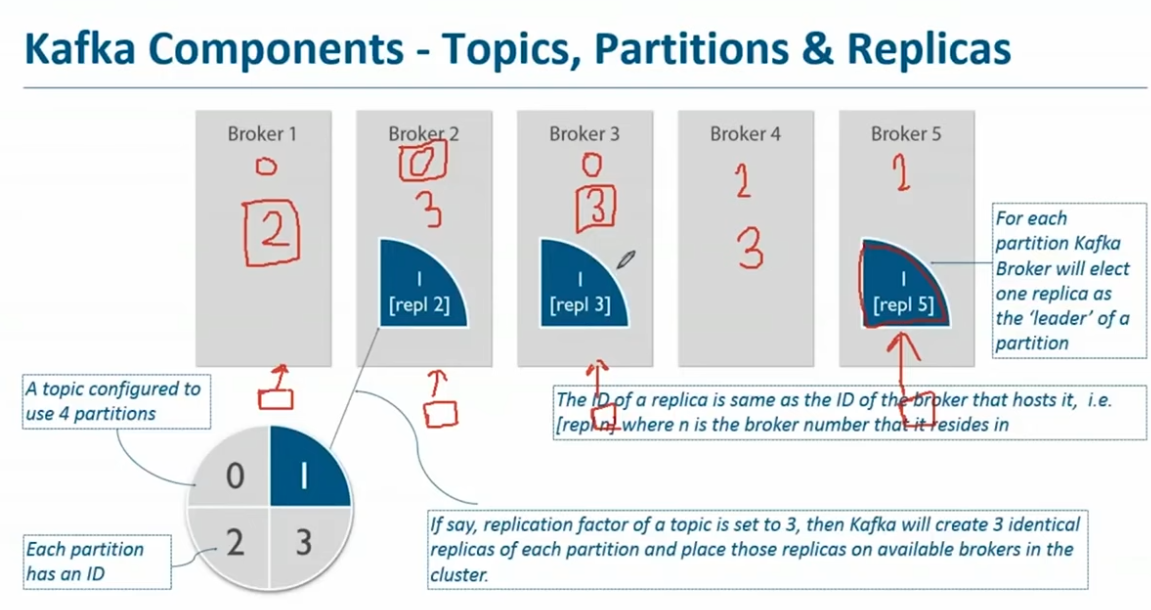
**Kafka Terminologies**

1. **Producer -** A producer can be any application who can publish messages to a topic.
2. **Consumer -** A consumer can be any application that subscribe to a topic and consume the messages.
3. **Broker -** Kafka cluster is a set of servers, each of which is called a broker.
4. **Topic -** A topic is a category or feed name to which records are published.
5. **Partition -** Topics are broken up into ordered commit logs called partitions.
6. **Zookeeper -** Zookeeper is used for managing and coordinating Kafka broker.

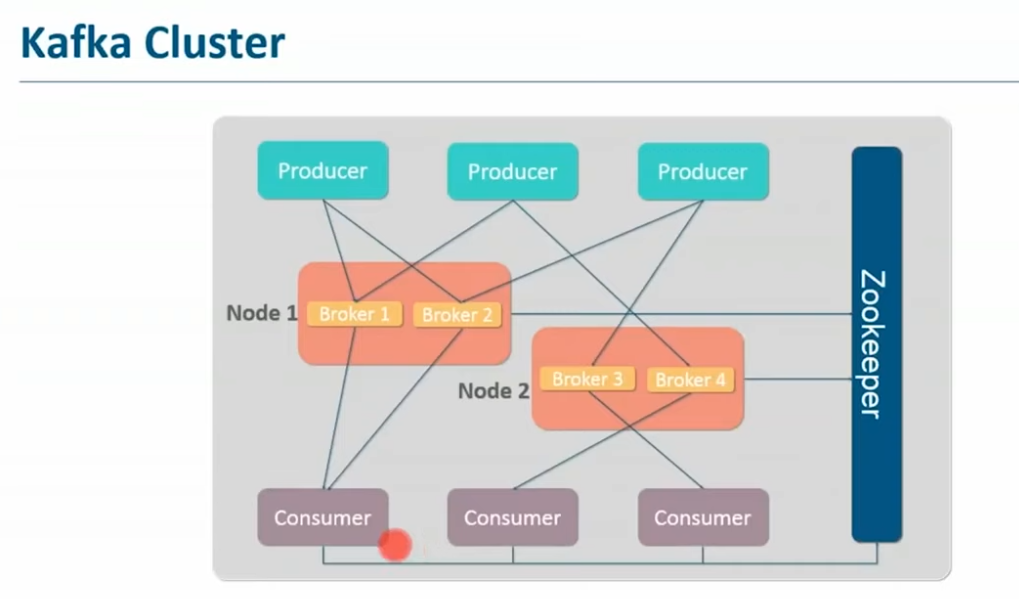
**Kafka Components** - Topics, Partitions and Replicas.

In the below image, the circular shape is a Topic. A topic is category name (like sales) which is further broken into four partitions.

Brokers are server, and replication is 3 here, so there will be 3 replica of partition 1, 2, 3, 4. Out of which one is leader. Any consumer which come for consuming partition 1 will be served by this replica. The other two are for fault tolerance.

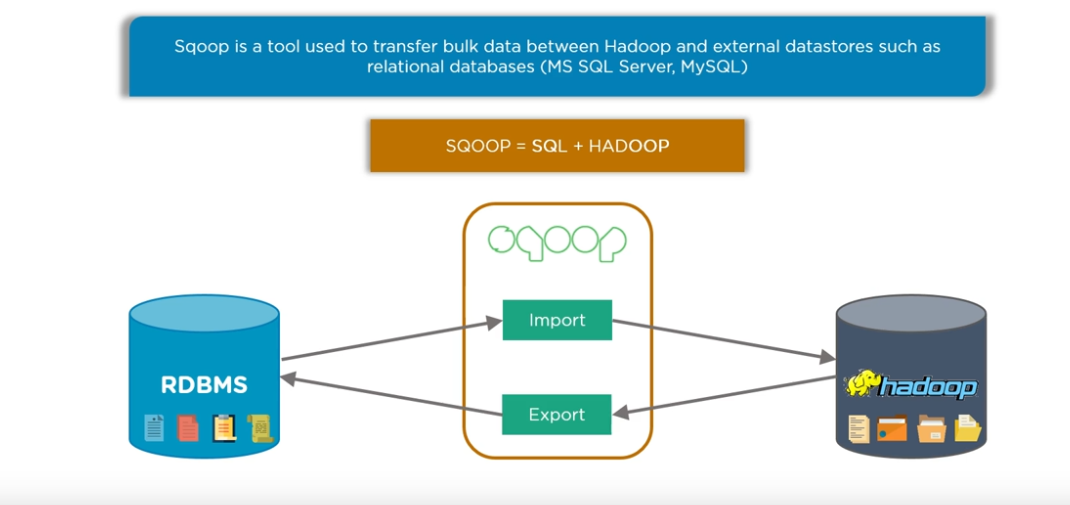


**Kafka cluster**



**Apache Sqoop**

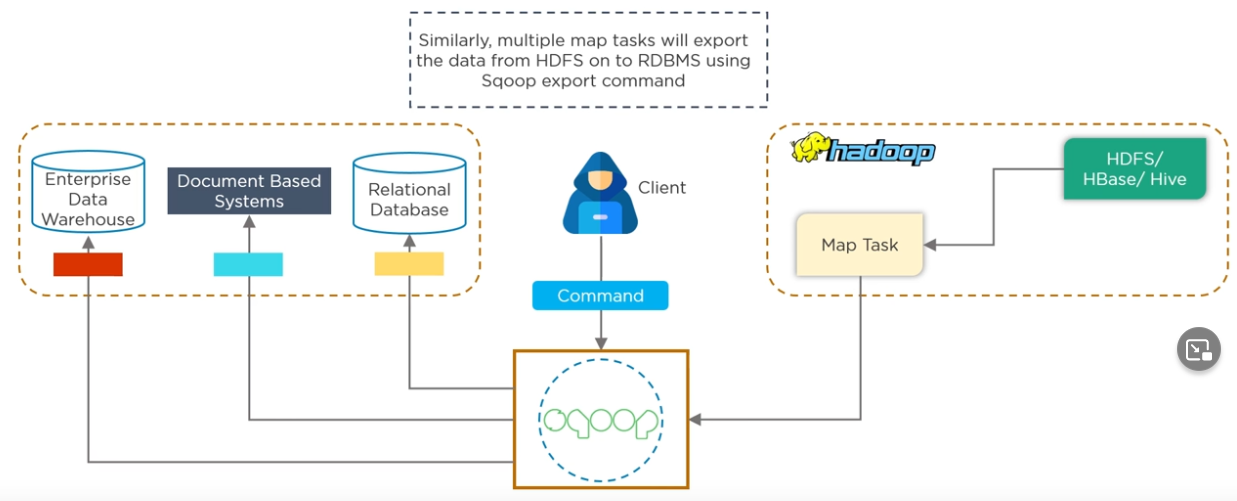
Sqoop is a tool used to transfer bulk data between Hadoop and external data stores such as rational database(MS SQL, My SQL



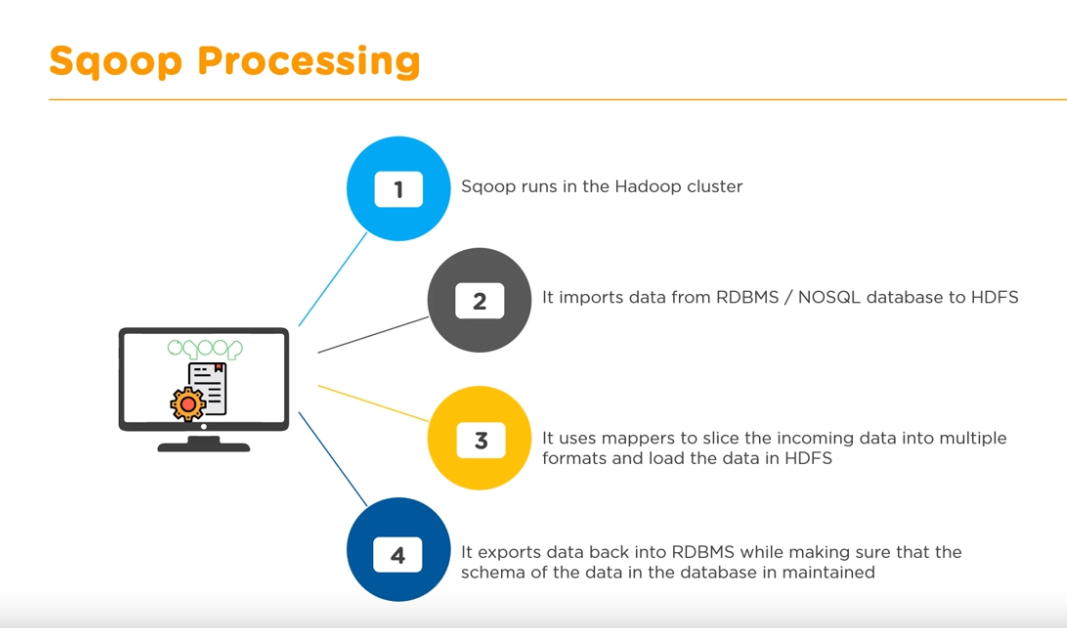
**Sqoop Features:**

1. Parallel import/export
2. Import results of sql query
3. Connectors for all major RDBMS database
4. Kerberos' security Integration
5. Provides full and incremental load.

**Sqoop Architecture**



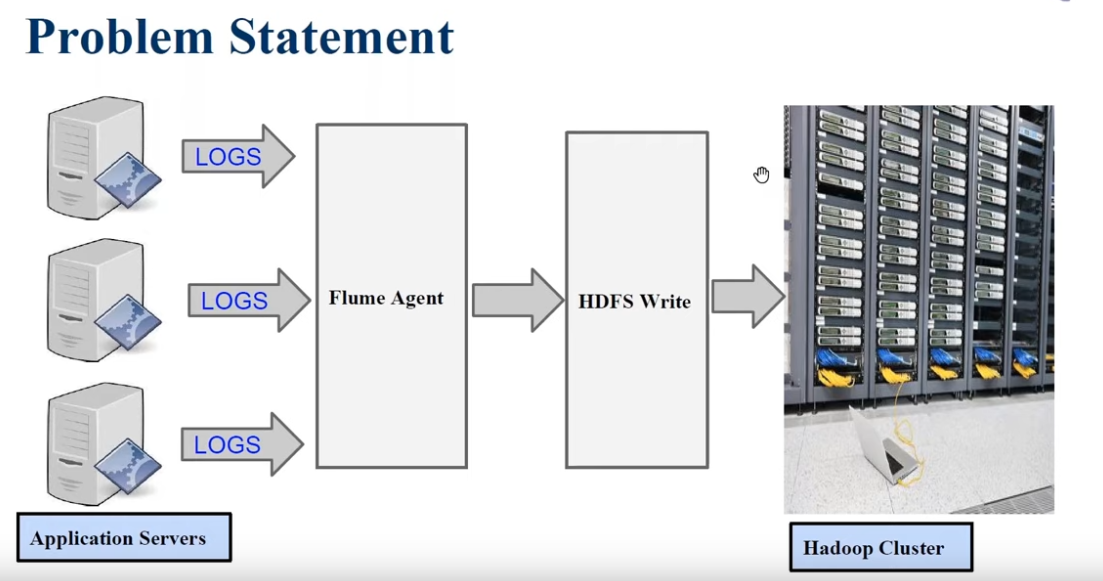
**Sqoop Processing**



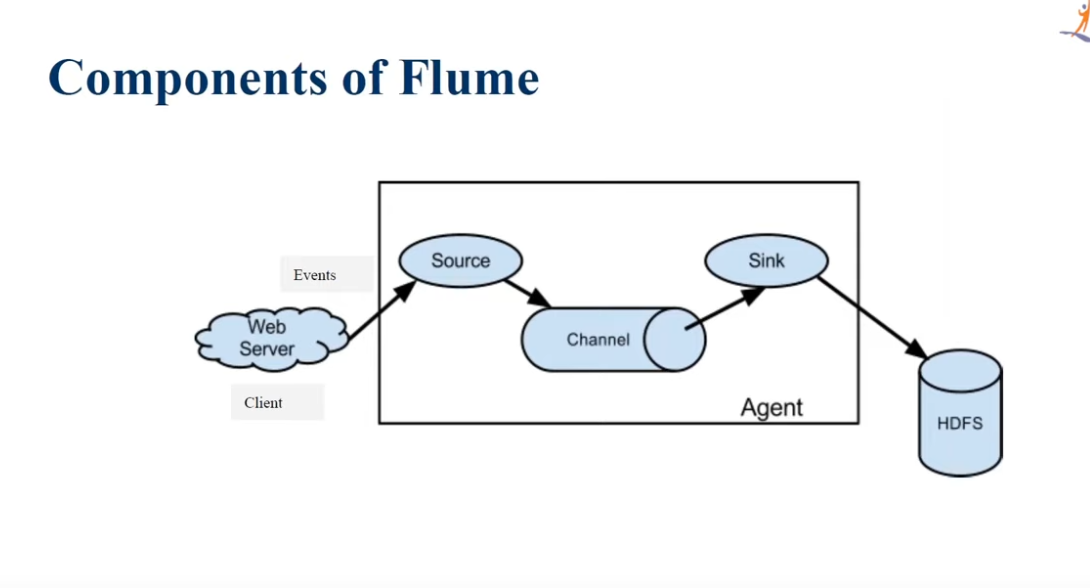
**Apache Flume**

When an application(e\_comerse) is running will have log files, which is located in Us and our Hadoop server is located in Russia. We need an efficient platform to send the logs data to server. And we can’t use SQOOP(**Sqoop is good only with RDBMS or No Sql)** as this is unstructured data. Also for **streaming data we use flume**.

Flume was created to bring log files to hadoop cluster.



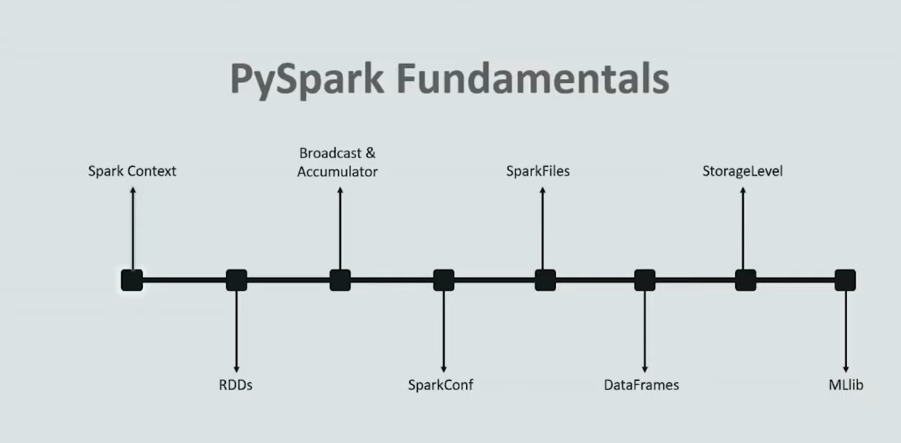
**Flume Architecture/components**

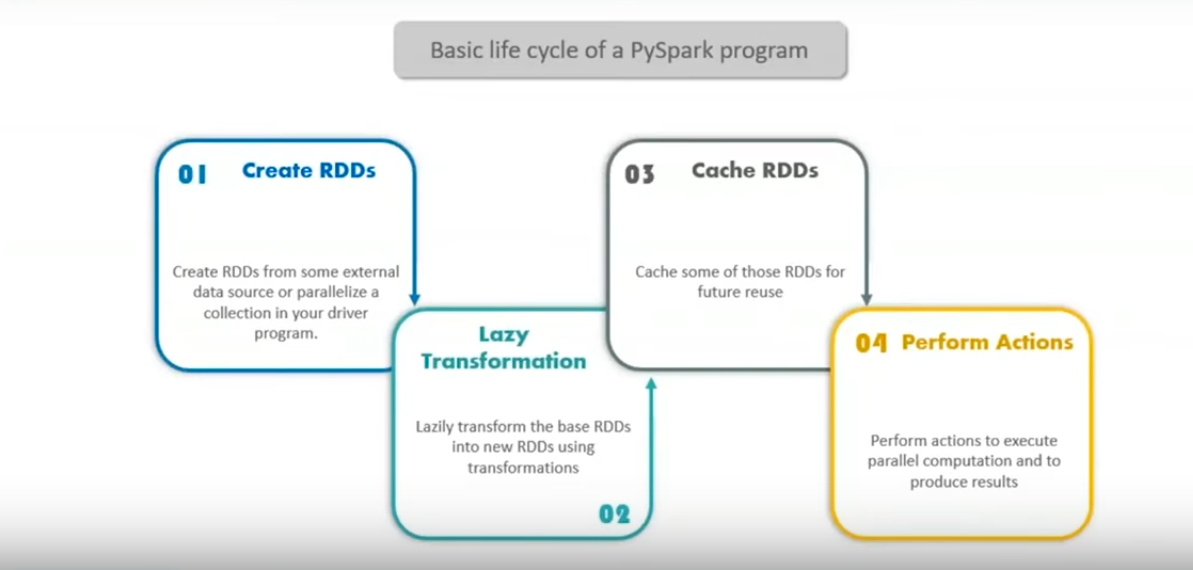


**PySpark**

PySpark is an python API for spark majorly used for data analysis, With PySpark you can work with spark RDDs in python.

PySpark Fundaments





**AWS - EC2- Cloud Service**

**S3 -Storage**

**SNS- Simple notification service.**

**========================================================================**

**different methods to manage partitions**

**Repartition() vs Coalesce()**

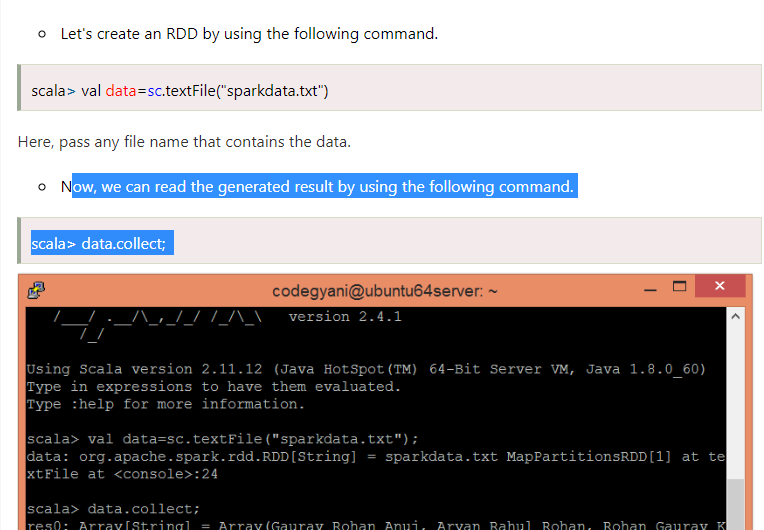
**Spark repartition() vs coalesce() – repartition() is used to increase or decrease the RDD, DataFrame, Dataset partitions whereas the coalesce() is used to only decrease the number of partitions in an efficient way.**

**======================================================**

**Word count program with Spark in scala**

* **Let's create an RDD by using the following command.**

1. **scala> val data=sc.textFile("sparkdata.txt")**
2. **ow, we can read the generated result by using the following command.**
3. **scala> data.collect;**

****

* **Here, we split the existing data in the form of individual words by using the following command.**

1. **scala> val splitdata = data.flatMap(line => line.split(" "));**

* **Now, we can read the generated result by using the following command.**

1. **scala> splitdata.collect;**

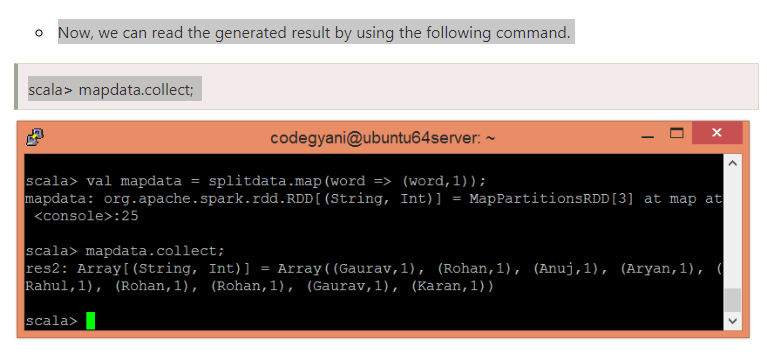
****

1. **Now, perform the map operation.**
2. **scala> val mapdata = splitdata.map(word => (word,1));**

**Here, we are assigning a value 1 to each word.**

* **Now, we can read the generated result by using the following command.**

1. **scala> mapdata.collect;**

****

* **Now, perform the reduce operation**

1. **scala> val reducedata = mapdata.reduceByKey(\_+\_);**

**Here, we are summarizing the generated data.**

* **Now, we can read the generated result by using the following command.**

1. **scala> reducedata.collect;**

**Or this will do it**

**val text = sc.textFile("mytextfile.txt")**

**Val counts = text.flatMap(line => line.split(" ")).**

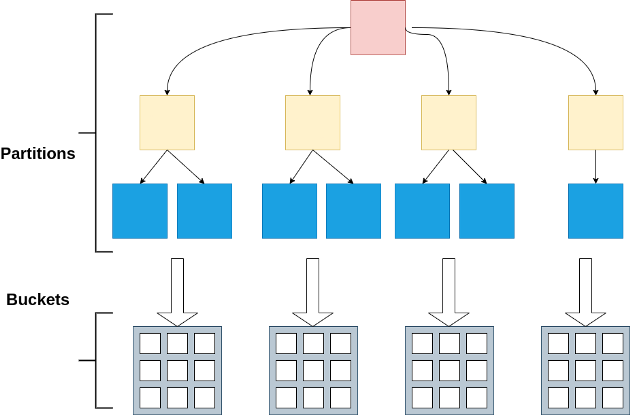
**map(word => (word,1)).reduceByKey(\_+\_) counts.collect**

**====================================================================**

**Bucketing vs Partitioning**

**What is difference between bucketing and partitioning in Spark?**

**Bucketing is similar to partitioning, but partitioning creates a directory for each partition, whereas bucketing distributes data across a fixed number of buckets by a hash on the bucket value**

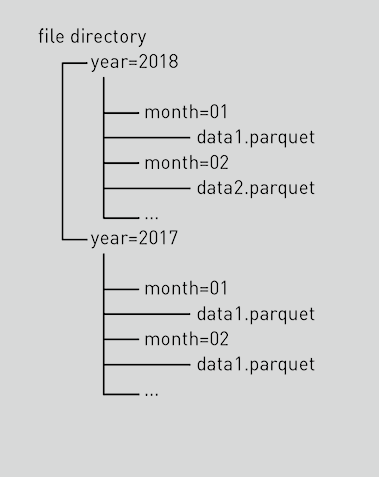
****

# **Partitioning in Spark**

# **The main idea of partitioning is to optimize job performance. Job performance can be obtained by ensuring each workload is distributed equally to each Spark executor.**

# **Partitioning in Spark**

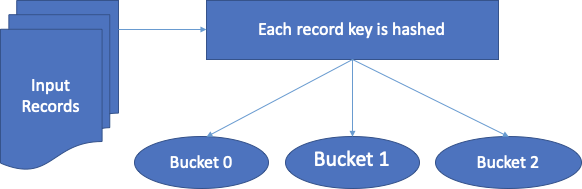
**The main idea of partitioning is to optimize job performance. Job performance can be obtained by ensuring each workload is distributed equally to each Spark executor.**

****

# **Bucketing in Spark**

**Bucketing is a technique in both Spark used to optimize the performance of the task. In bucketing buckets (clustering columns) determine data partitioning and prevent data shuffle. Based on the value of one or more bucketing columns, the data is allocated to a predefined number of buckets.**

**Bucketing is an optimization technique in Spark SQL that uses buckets and bucketing columns to determine data partitioning.**

****

**Bucketing has two key benefits:**

1. **Improved query performance:**
2. **Improved sampling:**

**==============================================================**