Disclaimer: **These slides are copyrighted and strictly for personal use only**

- This document is reserved for people enrolled into the [Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir]

- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

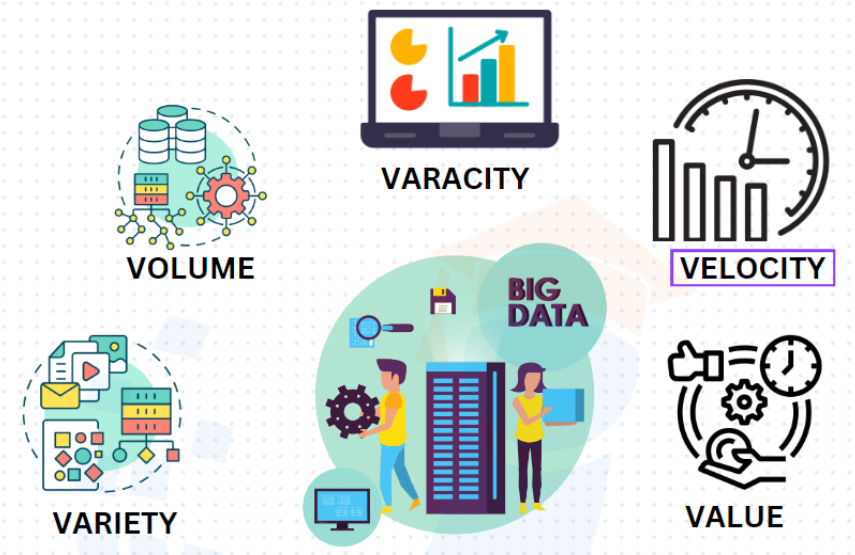
- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

- All the Best and Happy Learning!

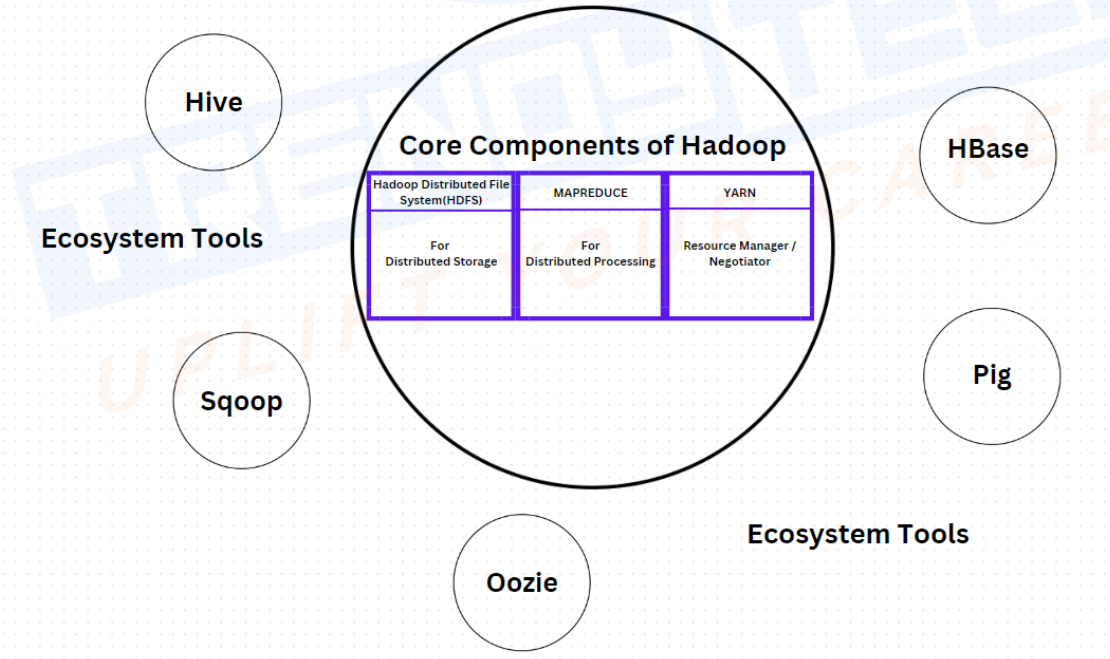
# Dataframes

## Recap of Concepts Learnt

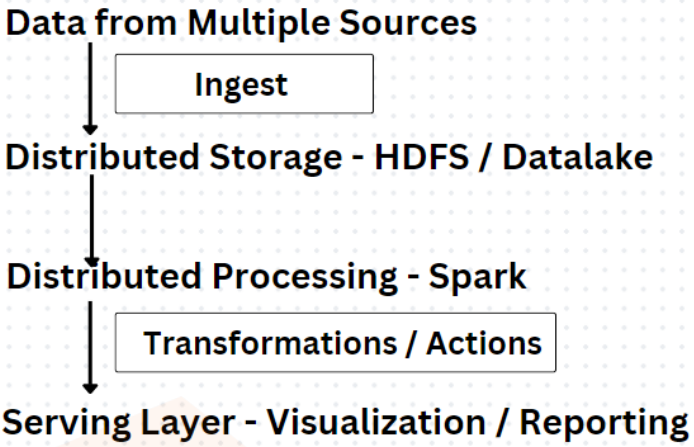
- **What is Big Data - 4vs**




- **Introduction to Hadoop Core and its Ecosystem**



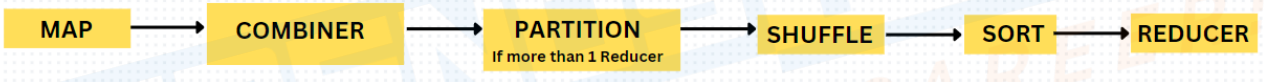

- **Distributed Framework**




**Distributed Storage** - How data is distributed across a cluster of machines that even if a node fails, the data is not lost. Ex - HDFS, Datalake,.

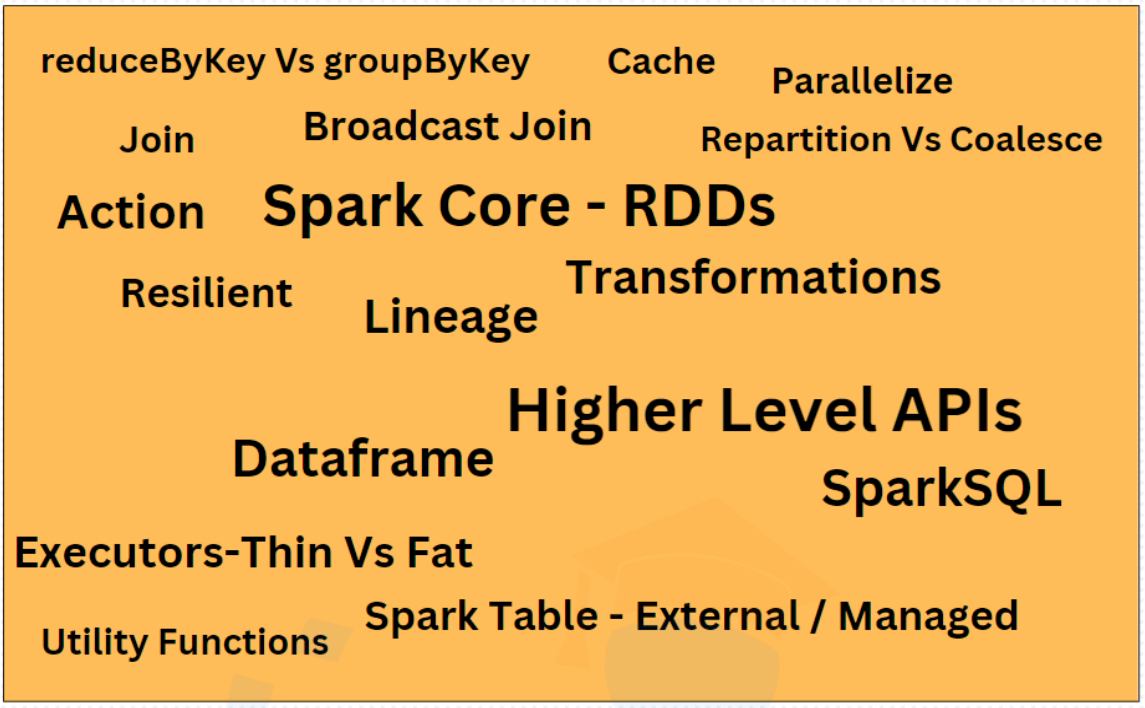
**HDFS** Architecture and Commands along with Linux commands

**Distributed Processing** -




- How distributed processing works under the hood : **MapReduce**
- How to execute Mapreduce code and its challenges.

**To overcome the challenges of MapReduce, a more performant Distributed Framework - Spark was introduced**




**Schema Inference Challenges** Inferring schema is not the best choice in spite of its code level advantages because -

1. It could lead to incorrect schema inference

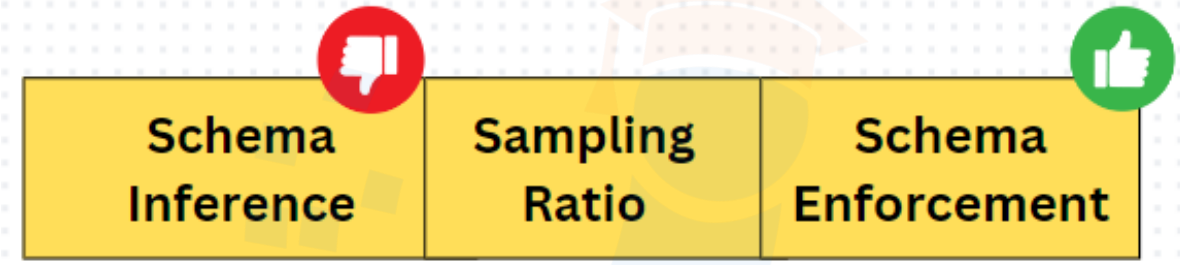
2. Spark has to scan the data to infer the schema which is time consuming and burdens the system, thereby affecting the performance.

```
df = spark.read \
.format("csv") \
.option("header","true") \
.option("inferSchema","true") \
.load("/public/yelp-dataset/yelp_user.csv")
```

**Note: Schema should be enforced and not inferred.**

**Sampling Ratio** Instead of scanning the entire dataset, inferring the schema based on the ratio provided.

```
df = spark.read \
.format("csv") \
.option("header","true") \
.option("inferSchema","true") \
.option("samplingRatio",.1) \
.load("/public/yelp-dataset/yelp_user.csv")
```




## Two Ways to Enforce Schema

### 1. Schema option - Schema DDL

```
orders_schema = 'order_id long, order_date date, cust_id long, order_status string'
```

```
df = spark.read \
.format("csv") \
.schema(orders_schema) \
.load("/public/trendytech/datasets/orders_sample1.csv")
```

```
df.printSchema()
```

```
root
 |-- order_id: long (nullable = true)
 |-- order_date: date (nullable = true)
 |-- cust_id: long (nullable = true)
 |-- order_status: string (nullable = true)
```

## 2. StructType

Importing the system defined function StructType →

```
from pyspark.sql.types import *
```

```
orders_schema_struct = StructType([
StructField("orderid",LongType()),
StructField("orderdate",DateType()),
StructField("customerid",IntegerType()),
StructField("orderstatus",StringType()),
])
```

```
df = spark.read \
.format("csv") \
.schema(orders_schema_struct) \
.load("/public/trendytech/datasets/orders_sample1.csv")
```

```
df.printSchema()
```

```
root
 |-- orderid: long (nullable = true)
 |-- orderdate: date (nullable = true)
 |-- customerid: integer (nullable = true)
 |-- orderstatus: string (nullable = true)
```

## How to handle Date Type

Default format of date type in Spark is  yyyy-mm-dd

If the date format is different from the one mentioned above, then it will lead to Parse Error

**Different ways to handle different formats of date -**

1. Use **option** while creating dataframe to explicitly specify the date format.

```
df = spark.read \
.format("csv") \
.schema(orders_schema) \
.option("dateFormat","MM-dd-yyyy")
.load("/public/trendytech/datasets/orders_sample1.csv")
```

2. Load Date as a string and then apply transformation to convert to the date format.

```
orders_schema = 'order_id long, order_date string, cust_id long, order_status string'
```

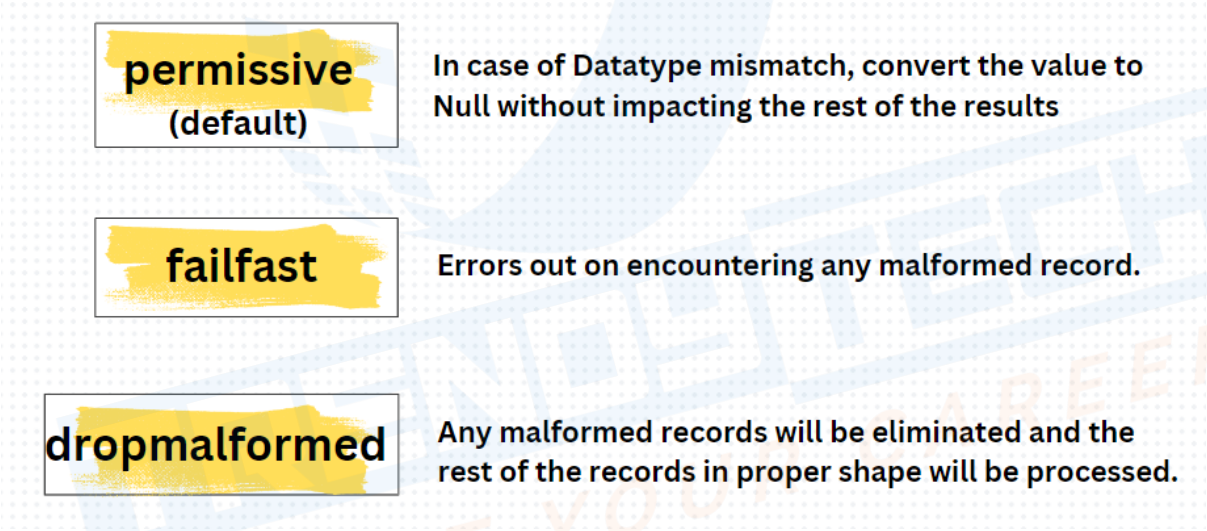
```
df = spark.read \
.format("csv") \
.schema(orders_schema) \
.load("/public/trendytech/datasets/orders_sample1.csv")
```

```
from pyspark.sql.functions import to_date
```

```
new_df = df.withColumn("order_date_new",to_date("order_date","MM-dd-yyyy"))
```

**Note:** In case of Parse Issues, the complete date column shows up as null.

## Dataframe Read Modes




**Note:** You can choose the respective read modes based on the business requirement.

# Different ways of creating a Dataframe

Using **spark.read**

Ex:

**df = spark.read.format("csv").option("header","true").load(filePath)**

Using **spark.sql**

Ex:

**df = spark.sql(“select * from <table-name>”)**

**[ Note : Results of a Spark SQL query is a Dataframe ]**

Using **spark.table**

Ex:

**df = spark.table(“<table-name>”)**

Using **spark.range**

Range gives a one column dataframe

Ex:

**-> df = spark.range(<range-size>)**

**-> df = spark.range(<start-range>, <end-range>)**

**-> df = spark.range(<start-range>, <end-range>, <increment>)**

Creating Dataframe from **Local List**

Ex:

**-> df = spark.createDataFrame( list )**

**Two Step Process of creating a Dataframe -** If you want to explicitly specify the column names and not go with the default values

**-> df = spark.createDataFrame( list ).toDF(<column-name>)**

**One Step Process of creating a Dataframe -** To enforce the schema explicitly

**//Define the schema first**

**//Approach 1 -** fixing only the column names

**schema = [ "<column-name-1>", "<column-name-2>",..]**

**//Approach 2 -** fixing the column names and Datatypes

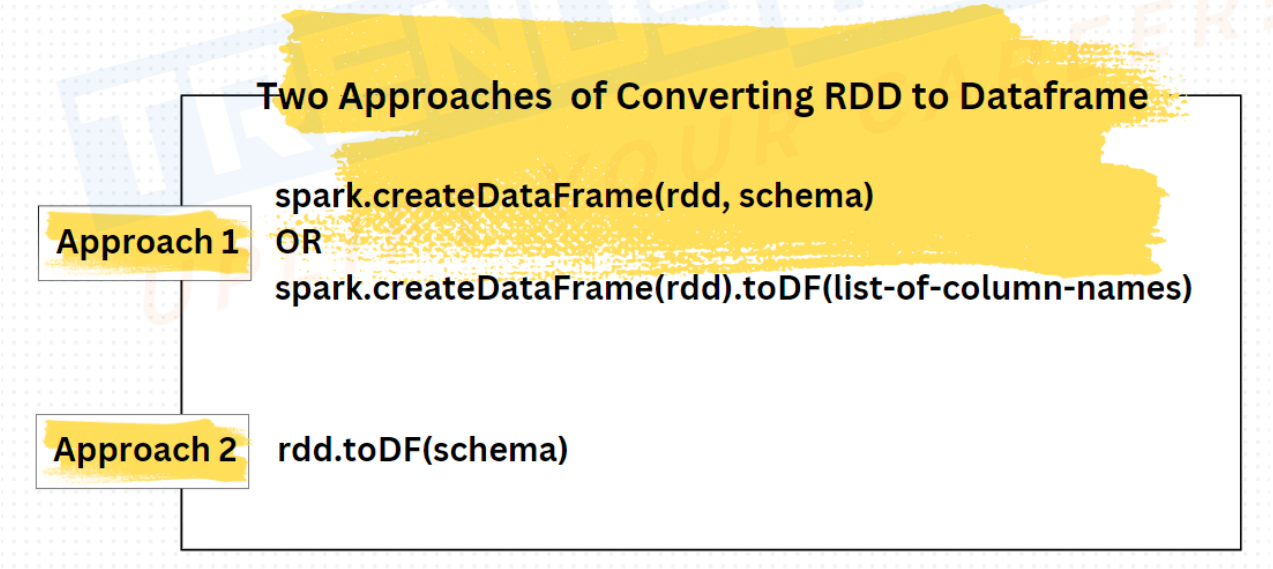
**schema = ' <column-name-1> <datatype-1>, <column-name-2><datatype-2>,... '**

**//Create DF using the defined schema**

**-> df = spark.createDataFrame( list, schema )**

Creating Dataframe from **RDD**

Dataframe - Is an RDD with a structure associated with it.



## Different ways of handling Nested Schema

### Approach 1 :

Example -

```
ddlSchema = "customer_id long,fullname
struct<firstname:string,lastname:string>,city string"

df=spark.read.format("json").schema(ddlSchema).load("/public/trendytech/datasets/customer_nested/*"
```

### Approach 2 :

Example -

```
customer_schema = StructType([

StructField("customer_id",LongType()),

StructField("fullname",StructType([StructField("firstname",StringType()),StructField("lastname",StringType())])),

StructField("city",StringType())

])

df=spark.read.format("json").schema(customer_schema).load("/public/trendytech/datasets/customer_nested/*")
```

(OR)

```
customer_list = [

(1,("sumit","mittal"),"bangalore"),

(2,("ram","kumar"),"hyderabad"),

(3,("vijay","kumar"),"pune")]
```

```
ddlSchema = "customer_id long,fullname
struct<firstname:string,lastname:string>,city string"

df = spark.createDataFrame(customer_list,ddlSchema)
```

(OR)

```
customer_schema = StructType([

StructField("customer_id",LongType()),

StructField("fullname",StructType([StructField("firstname",StringType()),Struct
Field("lastname",StringType())])),

StructField("city",StringType())

])

df=spark.createDataFrame(customer_list,customer_schema)
```

## Dataframe Transformations

| Transformations | Description | Syntax |
|---|---|---|
| 1. withColumn | To add a new Column or change existing column | df2 = df1.select("<list-of-column-names>",expr("<expression>")) or df2 = df1.selectExpr("<list-of-column-names-and-expressions>") |
| 2. withColumnRenamed | To rename an existing Column | df2 = df1.withColumnRenamed("<existing-column-names>", "<new-column-name>") |
| 3. drop | To drop a Column | df2 = df1.drop("<list-of-column-names>") |

**Note:**

- **In case of "select" we will have to explicitly segregate the column names and expressions and mention the expressions used within an expr.**
- **In case of "selectExpr", it automatically identifies whether the value passed is a column name or an expression and accordingly actions it.**

### Removal of duplicates from Dataframe

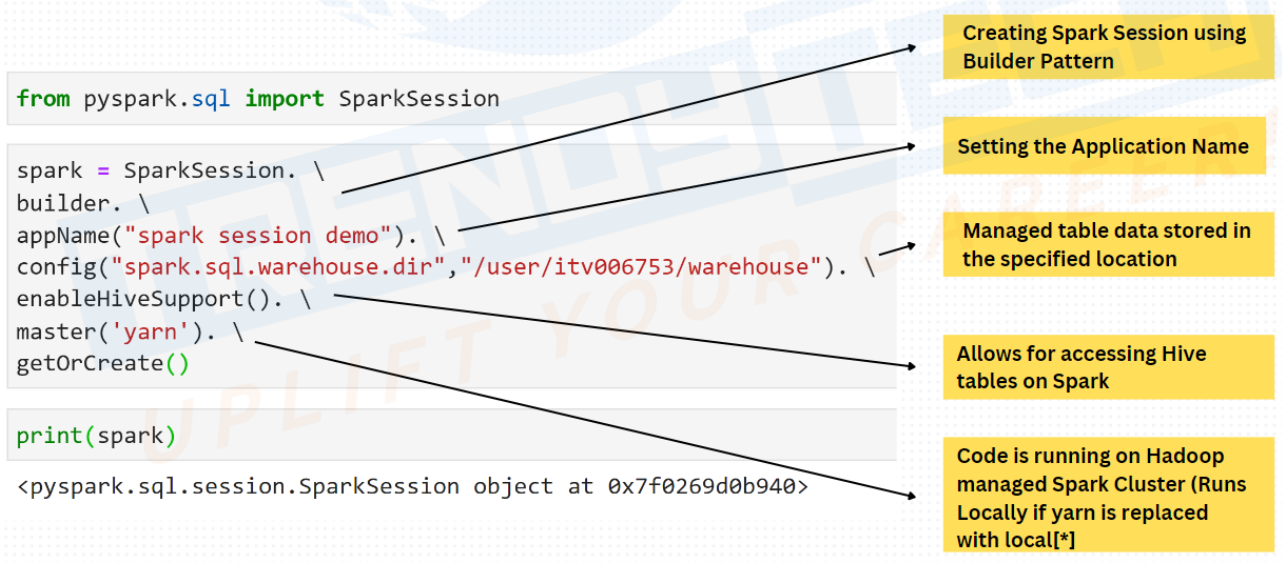
Transformations to handle duplicate

1. df2 = df1.distinct() [removes duplicates when all the columns are considered]
2. df2 = df1.dropDuplicates() [removes duplicates when a subset of columns are considered]

# Creation of Spark Session

- Spark Session acts as an entry point to the Spark Cluster. To run the code on Spark Cluster, a Spark Session has to be created.
- In order to work with Higher Level APIs like Dataframes and Spark SQL, Spark Session has to be created to run the code across the cluster.
- To work at RDD level, Spark Context is required.
- Spark Session acts as an umbrella that encapsulates and unifies the different contexts like Spark Context, Hive Context, SQL Context…

**Using Builder Pattern -**




**Note: Only one Spark Session Object is created per application.**

**What is the need for Spark Session when we already have Spark Context?**

1. Spark Session encapsulates the different contexts like Spark,Hive,SQL and allows these contexts to be accessed from a single session.
2. When there is a need to have more than one Spark Session for a single application with their respective isolated environments.

**Creating Multiple Spark Sessions -**

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession. \
builder. \
appName("spark session demo"). \
config("spark.sql.warehouse.dir","/user/itv006753/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
print(spark)
```

```
<pyspark.sql.session.SparkSession object at 0x7f0269d0b940>
```

First Spark Session Object

```
sparksession2 = spark.newSession()
```

```
print(sparksession2)
```

```
<pyspark.sql.session.SparkSession object at 0x7f0269b809b0>
```

Second Spark Session Object

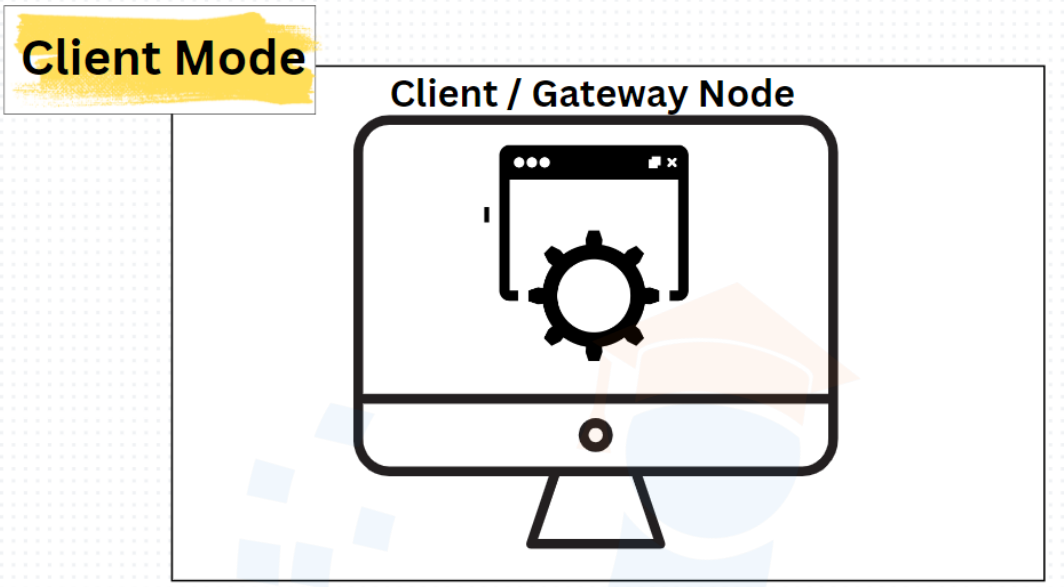
- Same Spark Context will be shared across multiple spark sessions created.

- Every Spark Application has a driver (Master) and multiple Executors (Workers).

## Spark Application Deployment Modes-

1. **Client Mode (Interactive Mode)**

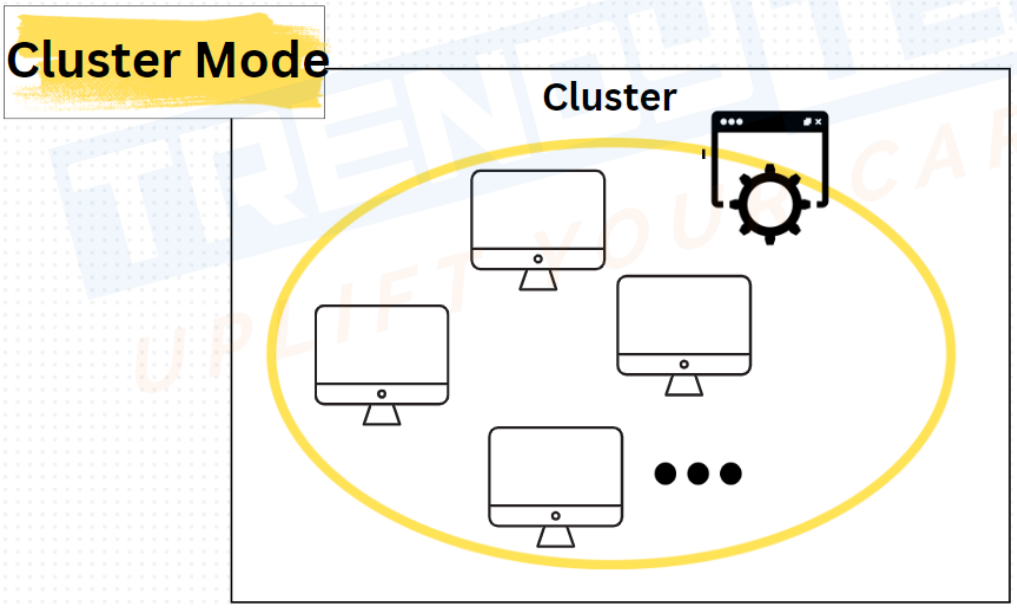
   Driver runs on the Client Machine / Gateway node.




The application is dependent on the Client / Gateway Node . If the node is turned off or goes down, then the application also goes down

2. **Cluster Mode (Non-interactive Mode)**

   Driver runs on the Cluster




Driver runs on the cluster. Deployment mode should always be a Cluster Mode for Production Environments. The code is packaged and deployed on the cluster