## 1. What is Spark?

Apache Spark is a unified, distributed data processing engine for big data and ML.

Fastest, most expressive, supports batch, interactive, streaming, ML, graph, and SQL — all in one system.

## 2. History of Spark

| Year | Milestone |
|---|---|
| 2009 | Started at UC Berkeley AMPLab |
| 2013 | Apache Top-Level Project |
| 2014 | Spark 1.0 — Databricks founded |
| 2016 | Spark 2.0 — Dataset/DataFrame API + Structured Streaming |
| 2020 | Spark 3.0 — ANSI SQL, Adaptive Query Execution (AQE) |
| 2023–2025 | Spark 3.5+ — Project Lightspeed (faster networking), native Delta Lake 3.0, better Python/Scala parity |

## 3. Spark Architecture (High Level)

text

```
Driver Program
  ├── SparkContext / SparkSession
  ├── Cluster Manager (Standalone, YARN, Kubernetes, Mesos)
  └── Executors (JVM processes on worker nodes)
      ├── Slots/Threads (parallel tasks)
      └── Cache (RDD/DataFrame caching)
```

## 4. Spark Shell (Scala)

Bash

```
spark-shell --master local[*] \
        --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1
```

You now have a live sc: SparkContext and spark: SparkSession

## 5. PySpark vs Scala — Why Scala Here?

- Native language of Spark
- Best performance & type safety
- Full access to all low-level APIs (RDDs, Accumulators, etc.)

## 6. Spark Architecture – Deep Dive (Scala Code)

Scala

```scala
// spark-submit or spark-shell
val spark = SparkSession.builder()
  .appName("ArchitectureDemo")
  .master("local[4]")          // 4 cores locally
  .config("spark.sql.adaptive.enabled", "true")
  .config("spark.sql.adaptive.coalescePartitions.enabled", "true")
  .getOrCreate()

import spark.implicits._

println(s"Spark Version: ${spark.version}")
println(s"Default parallelism: ${spark.sparkContext.defaultParallelism}")
// Output example: 8 (on a typical laptop)
```

## 7. SparkContext vs SparkSession

Scala

```scala
val sc = spark.sparkContext        // Old way (still works)
val spark2 = SparkSession.builder().getOrCreate()  // New way (recommended)
```

## 8. Word Count – The Classic Example (Scala)

Scala

```scala
val textFile = spark.read.textFile("hdfs://path/to/book.txt")   // or local file

val counts = textFile
  .flatMap(_.split(" "))
  .filter(_.nonEmpty)
  .map(word => (word.toLowerCase.replaceAll("[^a-z]", ""), 1))
  .reduceByKey(_ + _)
  .sortBy(-_._2)

counts.take(20).foreach(println)
// (the,12345), (and,9876), ...
counts.write.mode("overwrite").parquet("output/wordcount")
```

## 9. RDD Basics – Creating, Actions, Transformations

Scala

```scala
// 1. parallelize
val data = 1 to 10000
val rdd = sc.parallelize(data, 8)   // 8 partitions

// 2. Actions
rdd.collect()        // Brings ALL data to driver → dangerous!
rdd.take(10)         // Safe
```

```
rdd.first()
rdd.count()

// 3. Partitions
println(rdd.getNumPartitions)  // → 8
val repartitioned = rdd.repartition(20)
val coalesced = repartitioned.coalesce(4)  // Reduce without shuffle when possible

// 4. Save
rdd.map(_.toString).saveAsTextFile("output/numbers")
```

## 10. RDD from External Data

Scala

```
val rddFromFile = sc.textFile("data/people.json")
val sequenceFileRdd = sc.sequenceFile[String, Int]("hdfs://path/seq")
```

## 11. Core Transformations (Scala)

Scala

```
val lines = sc.textFile("data/log.txt")

lines.flatMap(_.split(" "))
    .map(_.toLowerCase)
    .filter(_.startsWith("error"))
    .map((_, 1))
    .reduceByKey(_ + _)
    .collect()
    .foreach(println)
```

## 12. map vs flatMap

Scala

```
val sentences = sc.parallelize(Seq("hello world", "spark is fast"))
sentences.map(_.split(" "))   // Array[Array[String]]
sentences.flatMap(_.split(" ")) // Array[String]
```

## 13. reduceByKey vs groupByKey (CRITICAL!)

Scala

```
// Preferred (shuffle less data)
rdd.map(x => (x._1, x._2)).reduceByKey(_ + _)

// Avoid (brings all values to one executor)
rdd.groupByKey().mapValues(_.sum)
```

## 14. Caching & Persistence

Scala

```scala
import org.apache.spark.storage.StorageLevel

val importantRdd = sc.textFile("bigfile")
  .cache()                                  // MEMORY_ONLY
  .persist(StorageLevel.MEMORY_AND_DISK_SER)    // Serialized + spill

importantRdd.count()  // Triggers caching
```

## 15. Accumulators & Broadcast Variables

Scala

```scala
val errorCounter = sc.longAccumulator("Error Counter")
val broadcastDict = sc.broadcast(Set("spam", "viagra"))

lines.foreach { line =>
  if (line.contains("ERROR")) errorCounter.add(1)
  if (broadcastDict.value.contains(line)) println("Blocked!")
}

println(s"Total errors: ${errorCounter.value}")
```

## 16. Hello DataFrames (Scala)

Scala

```scala
case class Person(name: String, age: Int)

val df = Seq(
  Person("Alice", 34),
  Person("Bob", 45),
  Person("Cathy", null)
).toDF()

df.show()
df.printSchema()
df.filter($"age" > 30).show()
```

## 17. Loading CSV, JSON, Parquet

Scala

```scala
val csvDF = spark.read
  .option("header", "true")
  .option("inferSchema", "true")
  .csv("data/sales.csv")

val jsonDF = spark.read.json("data/people.json")
val parquetDF = spark.read.parquet("data/events/")
```

## 18. Spark SQL & Temporary Views

Scala
```
df.createOrReplaceTempView("people")

spark.sql("""
  SELECT name, age FROM people WHERE age > 30 ORDER BY age DESC
""").show()
```

## 19. Aggregations & Joins

Scala
```
import org.apache.spark.sql.functions._

val sales = spark.read.parquet("sales/")
val products = spark.read.parquet("products/")

sales.join(products, Seq("product_id"), "left")
    .groupBy($"category")
    .agg(sum("revenue").as("total_revenue"))
    .orderBy(desc("total_revenue"))
    .show()
```

## 20. UDFs in Scala

Scala
```
val cleanUdf = udf((s: String) => s.replaceAll("[^a-zA-Z0-9]", "").toLowerCase)

df.withColumn("clean_name", cleanUdf($"name")).show()
```

## 21. Spark Structured Streaming (Scala)

Scala
```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words = lines.as[String].flatMap(_.split(" "))
val wordCounts = words.groupBy("value").count()

val query = wordCounts.writeStream
  .outputMode("complete")
  .format("console")
  .start()

query.awaitTermination()
```

## 22. Kafka + Structured Streaming (Scala)

Scala
```
val df = spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "broker:9092")
  .option("subscribe", "tweets")
  .load()

df.selectExpr("CAST(value AS STRING) as json")
  .writeStream
  .format("console")
  .start()
```

## 23. Performance Tuning Best Practices (Scala)

Scala
```
spark.conf.set("spark.sql.adaptive.enabled", "true")
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", "true")
spark.conf.set("spark.sql.shuffle.partitions", "200")  // default = 200
spark.conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
spark.conf.set("spark.sql.parquet.compression.codec", "zstd")
```

## 24. Deploying a Spark App (Scala + sbt)

**build.sbt**

Scala
```
name := "MySparkApp"
version := "1.0"
scalaVersion := "2.13.14"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.5.1",
  "org.apache.spark" %% "spark-sql" % "3.5.1"
)
```

**src/main/scala/WordCountApp.scala**

Scala
```
object WordCountApp {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("WordCountApp")
      .getOrCreate()
    import spark.implicits._

    val counts = spark.read.textFile(args(0))
```

```
      .flatMap(_.split(" "))
      .map((_, 1))
      .reduceByKey(_ + _)

    counts.write.mode("overwrite").parquet(args(1))
    spark.stop()
  }
}
```

Submit:

Bash

```
spark-submit --class WordCountApp --master yarn target/scala-2.13/mysparkapp_2.13-1.0.jar \
  hdfs:///books/hamlet.txt hdfs:///output/hamlet-wc
```