Here's everything you need to know about Hadoop and HDFS

What is Hadoop?

Apache Hadoop is an open-source framework that allows for the distributed storage and processing of large data sets using clusters of computers. It follows a master-slave architecture and is designed to scale from single servers to thousands of machines.

Core Components of Hadoop:

1. HDFS (Hadoop Distributed File System) – Storage layer
2. MapReduce – Processing layer
3. YARN (Yet Another Resource Negotiator) – Resource management layer
4. Common Utilities – Shared libraries and utilities

What is HDFS?

HDFS is a distributed file system designed to run on commodity hardware. It stores data across multiple machines and ensures fault tolerance by replicating the data.

Key Features of HDFS:

- Master-slave architecture: NameNode (master) and DataNodes (slaves)
- Default replication factor: 3
- Handles large files efficiently
- Optimized for batch processing, not for real-time access

Hadoop Architecture Overview

- NameNode: Manages the filesystem namespace and metadata
- DataNode: Stores actual data blocks
- Secondary NameNode: Periodically merges edits with fsimage to help recover NameNode
- ResourceManager (YARN): Allocates system resources
- NodeManager: Manages resources on a single node

Installation of Hadoop

Prerequisites:

- OS: Linux (Ubuntu/CentOS preferred)
- Java JDK (Java 8 recommended)
- SSH enabled (for communication between nodes)

Basic HDFS Commands

- Create directory:

```
hdfs dfs -mkdir /user
```

- Upload a file:

```
hdfs dfs -put localfile.txt /user
```

- List files:

hdfs dfs -ls /

- Read file:

hdfs dfs -cat /user/localfile.txt

- Delete file:

hdfs dfs -rm /user/localfile.txt

Web Interfaces

- NameNode UI: http://localhost:9870
- ResourceManager UI: http://localhost:8088

How Hadoop Works (Functioning)

1. Storage (HDFS)
   - Files are split into blocks (default 128MB) and stored across DataNodes
   - Each block is replicated to ensure fault tolerance

2. Processing (MapReduce)
   - Map: Processes input into intermediate key-value pairs
   - Shuffle and Sort: Groups data
   - Reduce: Aggregates final output

3. Resource Management (YARN)

   - ResourceManager coordinates resources globally

   - NodeManager handles resources at the node level

Modes of Operation

1. Local (Standalone) – for testing
2. Pseudo-distributed – all daemons run on one machine
3. Fully-distributed – real cluster with multiple nodes

Summary

Component	Function
HDFS	Distributed storage system
MapReduce	Batch data processing
YARN	Resource allocation and job scheduling
NameNode	Metadata and namespace manager
DataNode	Data storage

PySpark

Here's everything you need to know about PySpark in simple text format without any bold or emojis:

What is PySpark?

PySpark is the Python API for Apache Spark. It allows you to write Spark applications using Python instead of Scala or Java. Spark is a fast and general-purpose cluster computing system for big data processing.

What is Apache Spark?

Apache Spark is an open-source distributed computing system that provides high-level APIs for Java, Scala, Python, and R. It is known for its speed and ease of use in processing large-scale data, both in batch and real-time.

Key Components of Spark:

1. Spark Core – Base engine for scheduling and memory management
2. Spark SQL – For structured data processing
3. Spark Streaming – For real-time stream processing
4. MLlib – Machine Learning library
5. GraphX – For graph processing

Why Use PySpark?

- Easy to use with Python syntax
- Supports interactive data analysis
- Integrates well with Hadoop and HDFS
- Compatible with major data sources like CSV, JSON, Parquet, JDBC

Installing PySpark

Prerequisites:

- Python 3.x
- Java 8 or Java 11
- pip installed
- Optional: Hadoop (if using HDFS or running on YARN)

Running PySpark

1. Start PySpark shell

```bash
pyspark
```

2. Start Jupyter Notebook with PySpark

```bash
pip install notebook findspark
```

Then in a Python file or notebook:

```python
import findspark
findspark.init()

from pyspark.sql import SparkSession
```

spark = SparkSession.builder.appName("MyApp").getOrCreate()
```

---

Basic PySpark Operations

Creating a Spark session:

```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Example").getOrCreate()
```

Creating a DataFrame:

```python
data = [("Alice", 25), ("Bob", 30)]
df = spark.createDataFrame(data, ["name", "age"])
df.show()
```

Reading CSV file:

```python
df = spark.read.csv("data.csv", header=True, inferSchema=True)
df.printSchema()
df.show()
```

DataFrame Operations:

```python
df.select("name").show()
df.filter(df.age > 25).show()

df.groupBy("age").count().show()
```

Writing Data:
```python
df.write.csv("output.csv")
```

RDD vs DataFrame

- RDD (Resilient Distributed Dataset): Low-level abstraction for distributed data
- DataFrame: High-level abstraction built on RDDs with schema support

Example using RDD:
```python
rdd = spark.sparkContext.parallelize([1, 2, 3, 4])
rdd.map(lambda x: x * x).collect()
```

Using PySpark with HDFS

If Hadoop is installed and HDFS is running, you can read/write files from HDFS like:
```python
df = spark.read.text("hdfs://localhost:9000/path/to/file.txt")
df.write.text("hdfs://localhost:9000/output/")
```
```

Using PySpark with SQL

```python
df.createOrReplaceTempView("people")
spark.sql("SELECT * FROM people WHERE age > 20").show()
```

Spark Modes

1. Local mode – runs locally on one machine
2. Standalone cluster – runs on multiple machines with Spark's built-in cluster manager
3. YARN – runs on Hadoop YARN
4. Mesos – runs on Apache Mesos

To run on YARN (with Hadoop setup):

```bash
spark-submit --master yarn my_script.py
```

Machine Learning with MLlib

Example: Linear Regression

```python
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
```

```
data = [(1, 2.0), (2, 2.5), (3, 3.5)]
df = spark.createDataFrame(data, ["x", "y"])

vec = VectorAssembler(inputCols=["x"], outputCol="features")
df = vec.transform(df)

lr = LinearRegression(featuresCol="features", labelCol="y")
model = lr.fit(df)
model.coefficients
model.intercept
```

Summary

Component	Use
Spark Core	Basic job scheduling and memory mgmt
Spark SQL	Working with structured data
Spark Streaming	Real-time data processing
MLlib	Machine learning pipelines
GraphX	Graph analytics
DataFrame	High-level tabular data abstraction
RDD	Low-level distributed data structure
PySpark Shell	Python interactive Spark shell

DIFFERENCE BETWEEN HADOOP AND PYSPARK

Feature	Hadoop	PySpark
Definition	Big data framework using HDFS + MapReduce for distributed storage and processing	Python API for Apache Spark, a fast in-memory distributed computing engine
Language	Mostly Java	Python
Processing Model	Disk-based (MapReduce)	In-memory (Spark engine)
Speed	Slower due to disk I/O	Faster due to in-memory processing
Ease of Use	Complex to code and debug	Simple syntax, user-friendly
Data Processing	Batch processing only	Batch, real-time, and interactive
Machine Learning	External tools like Mahout	Built-in MLlib for machine learning
Storage System	HDFS	HDFS, S3, Cassandra, HBase, etc.
Fault Tolerance	HDFS replication	DAG lineage-based recovery
Ecosystem	Hive, Pig, HBase, Oozie	Hive, Spark SQL, GraphX, Streaming
Resource Manager	YARN	YARN, Mesos, Kubernetes, Standalone
Use Case	Traditional batch processing	Modern data pipelines, streaming, ML

s