LTIMindtree Data Engineer Interview Guide – Experienced 3+

Round 1 - Technical

1. **Project Explanation and Technologies Used**
    - Describe the projects emphasizing Spark, Hadoop, or Azure for large-scale data processing.
    - Discuss tools for ETL, orchestration (Airflow, ADF), and real-time pipelines (Kafka, Spark Streaming).

2. **Performance Tuning Techniques**
    - Optimize shuffles: Use appropriate partitioning.
    - Use persist()/cache() for reused datasets.
    - Leverage broadcast joins for smaller datasets.
    - Optimize query plans using explain().

3. **Accumulator and Broadcast Variables**
    - Accumulator: Write-only, for aggregating data across tasks (e.g., counters).
    - Broadcast Variables: Read-only, for sharing smaller datasets among executors.

4. **Difference Between SparkSession and SparkContext**
    - SparkSession: Unified entry point for DataFrame, SQL, and streaming (introduced in Spark 2.0).
    - SparkContext: Core entry point for RDD-based operations.

5. **Difference Between Dataset and DataFrame**
    - Dataset: Provides compile-time type safety (Scala/Java).
    - DataFrame: Schema-based API; simpler for SQL-like operations.

6. **Spark Session Command**

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

7. **Command to Read JSON Data and Options**

```
df = spark.read.option("multiline", "true").json("path_to_json")
```

8. **CSV Without Column Names/Schema**

```
df = spark.read.option("header", "false").csv("path_to_csv")
df = df.toDF("col1", "col2", "col3")  # Assign custom column names
```

9. **Find 3rd Highest Salary**

```
from pyspark.sql.functions import col, dense_rank
from pyspark.sql.window import Window

window_spec = Window.orderBy(col("salary").desc())
df = df.withColumn("rank", dense_rank().over(window_spec))
third_highest = df.filter(col("rank") == 3)
```

10. **Filter Rows Where Employee Salary > Manager Salary**

```
emp_df = df.alias("emp")
mgr_df = df.alias("mgr")

result = emp_df.join(mgr_df, emp_df["mngName"] == mgr_df["Empname"]) \
            .filter(emp_df["salary"] > mgr_df["salary"]) \
            .
```

11. **Palindrome**

```
def is_palindrome(s):
    return s == s[::-1]
print(is_palindrome("madam"))  # Example
```

12. **Spark Submit**

spark-submit --class <MainClass> --master <ClusterURL> <AppJar>

13. **Memory Tuning**

- Adjust **executor memory** (--executor-memory).
- Use **storage levels** (MEMORY_AND_DISK).
- Tune **GC settings** for large heaps.

14. **Created JARs**

Describe building custom JARs for Spark jobs using Maven/SBT.

15. **Worked with UDFs**

Share examples of custom UDFs using Python or Scala.

16. **Dynamic Resource Allocation**

Automatically adjusts resources based on workload. Enable with:

--conf spark.dynamicAllocation.enabled=true

17. **Daily Data Volume**

Quantify daily data (e.g., 1TB/day) and its source (e.g., logs, transactions).

18. **Production Experience**

Discuss deploying Spark jobs and monitoring them using tools like Airflow or YARN.

Round 2 – Technical

1. **Difference Between DataFrame and Dataset**

   DataFrame: Untyped, optimized for SQL operations.

   Dataset: Typed, provides compile-time safety.

2. **Load CSV from HDFS**

```
df = spark.read.option("header", "true").csv("hdfs://path_to_csv")
```

3. **Syntax for CSV Loading**

```
df = spark.read.option("header", "true").csv("hdfs://path_to_csv")
```

4. **What is Multiline?**

   Multiline option handles JSON files with nested records across multiple lines.

5. **No Column Names in CSV**

```
df = spark.read.option("header", "false").csv("path_to_csv")
df = df.toDF("col1", "col2", "col3")  # Assign column names
```

6. **Case Class and StructType Syntax**

```
# Scala Case Class
case class Employee(name: String, salary: Double, dept: String)

# Python StructType
from pyspark.sql.types import StructType, StructField, StringType, DoubleType
schema = StructType([
    StructField("name", StringType(), True),
    StructField("salary", DoubleType(), True),
    StructField("dept", StringType(), True)
])
```

7. **Partitioning vs. Bucketing**

   Partitioning: Divides data into directories based on keys.

   Bucketing: Splits data into fixed-size buckets based on hash functions.

8. **Closure Function**

   A function that references variables from its enclosing scope.

9. **Count of Alphabets in String**

```python
from collections import Counter
s = "I am a data engineer"
counts = Counter(c for c in s.lower() if c.isalpha())
print(counts)
```

10. **Difference Between List and Tuple**

    List: Mutable, slower.

    Tuple: Immutable, faster.

11. **List Comprehension**

```python
squares = [x**2 for x in range(10)]
```

Glassdoor LTI Mindtree Review –

https://www.glassdoor.co.in/Reviews/LTIMindtree-work-life-balance-Reviews-EI_IE8441464.0,11_KH12,29.htm

LTI Mindtree Careers –

https://www.ltimindtree.com/careers/

Subscribe to my YouTube Channel for Free Data Engineering Content –

https://www.youtube.com/@shubhamwadekar27

Connect with me here –

https://bento.me/shubhamwadekar

Checkout more Interview Preparation Material on –

https://topmate.io/shubham_wadekar