



DAY 2 | Apache Spark Fundamentals



This image shows a slide from the Databricks AI Challenge, specifically Day 02. The slide has a dark background with various data-related icons like gears, charts, and databases on the sides. At the top left is the Indian Data Club (IDC) logo, and at the top right is the ODE BASICS logo. In the center, there's a Databricks logo and the text "14 DAYS AI CHALLENGE". Below that, it says "DAY 02". A large, rounded rectangular box contains the "Topic" and "Challenge" sections. The topic is "Apache Spark Fundamentals" and the challenge involves uploading an e-commerce CSV, reading it into a DataFrame, performing basic operations, and exporting results. The bottom of the slide features the hashtag "#DatabricksWithIDC".

Topic:
Apache Spark Fundamentals

Challenge:

1. Upload sample e-commerce CSV
2. Read data into DataFrame
3. Basic operations: select, filter, groupBy, orderBy
4. Export results

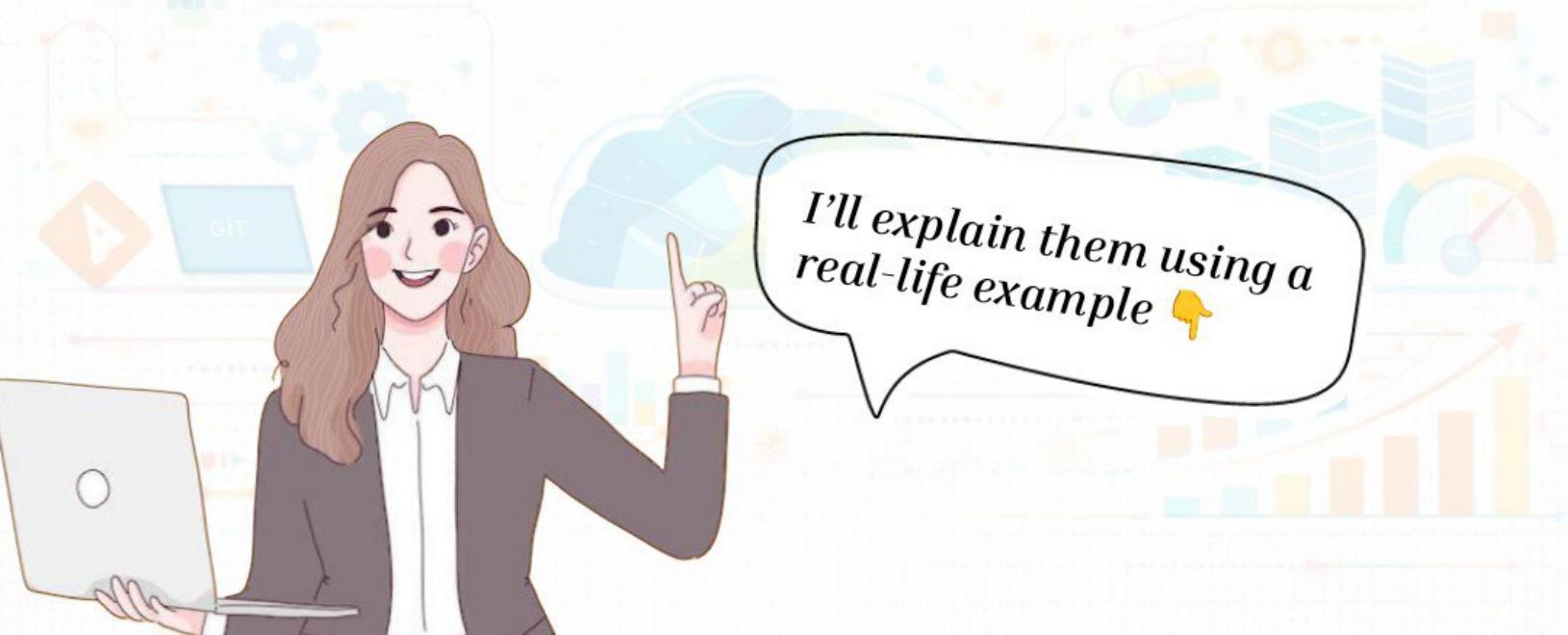
#DatabricksWithIDC



Apache Spark Architecture (Simple Explanation)

Spark architecture mainly has 4 core parts:

- 1. Driver*
- 2. Cluster Manager*
- 3. Executors*
- 4. DAG (Directed Acyclic Graph)*





1. Driver (The Brain)

Driver is the main controller of your Spark application

What the Driver does:

- *Runs your notebook code*
- *Creates SparkSession*
- *Understands your SQL / PySpark commands*
- *Builds the execution plan (DAG)*
- *Sends work to executors*
- *Collects results*



📌 In Databricks:

- Your notebook is running on the driver
- When you click Run, driver starts working

Example :

```
df = spark.read.csv("file.csv")
df.count()
```

👉 Driver understands what needs to be done





2. Cluster Manager (The Allocator)

It decides:

- How many executors?*
- How much memory?*
- How many cores?*

Examples:

- Databricks (managed)*
- YARN*
- Kubernetes*
- Standalone Spark*

 *In Databricks:
You don't manage it
manually
Databricks handles
this for you*





3. Executors (The Workers)

Executors do the actual data processing

What Executors do:

- Run tasks in parallel
- Process chunks of data
- Store data in memory/disk
- Send results back to driver



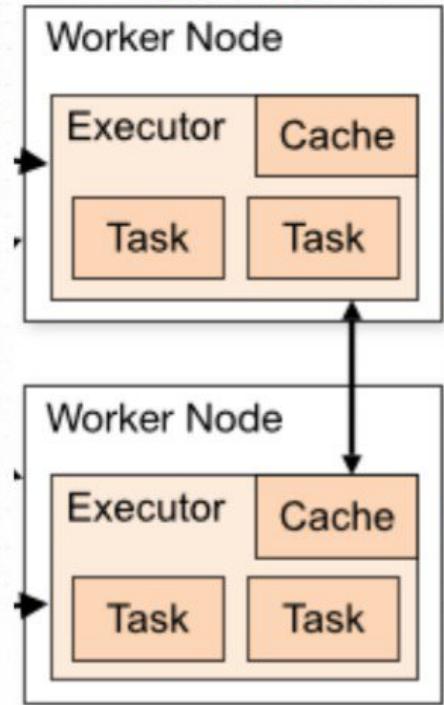
Each executor:

- Runs on a different node
- Handles partitions of data



Example: If your CSV has 10 million rows

- > *Spark splits it into partitions*
- > *Each executor processes some rows*



👉 This is why
Spark is fast





4. DAG (Directed Acyclic Graph)

DAG is Spark's execution plan

What DAG means:

- *Directed → execution flows in one direction*
- *Acyclic → no loops*
- *Graph → steps connected logically*

Spark creates DAG based on:

- *select*
- *filter*
- *groupBy*
- *orderBy*

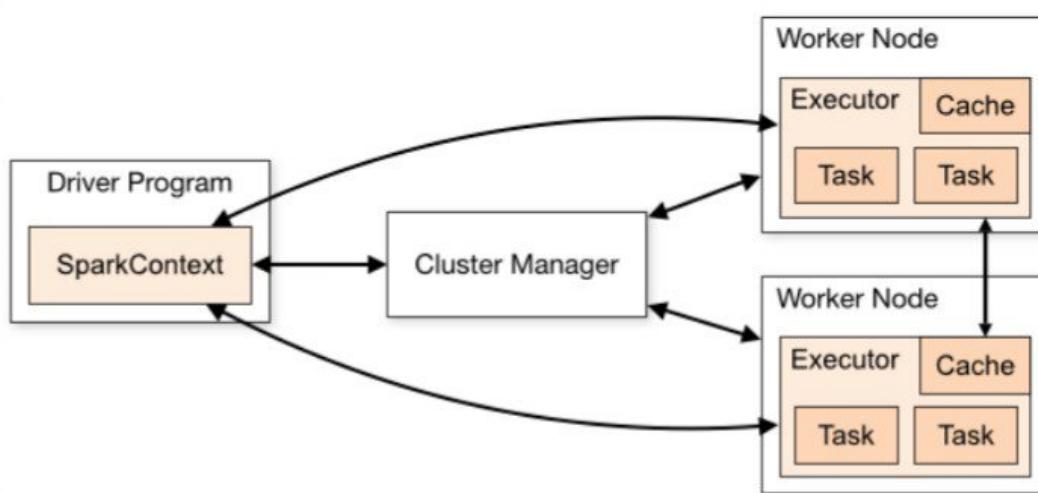
You never write DAG manually
Spark creates it automatically



How Everything Works Together (Flow)

Let's say you run this:

```
df = spark.read.csv("data.csv")
df.filter(df.price >
100).groupBy("category").count().show()
```





How Everything Works Together (Flow)

Step-by-step:

1. Driver

- Reads your code
- Builds logical plan (DAG)

2. DAG

- filter → groupBy → count

3. Cluster Manager

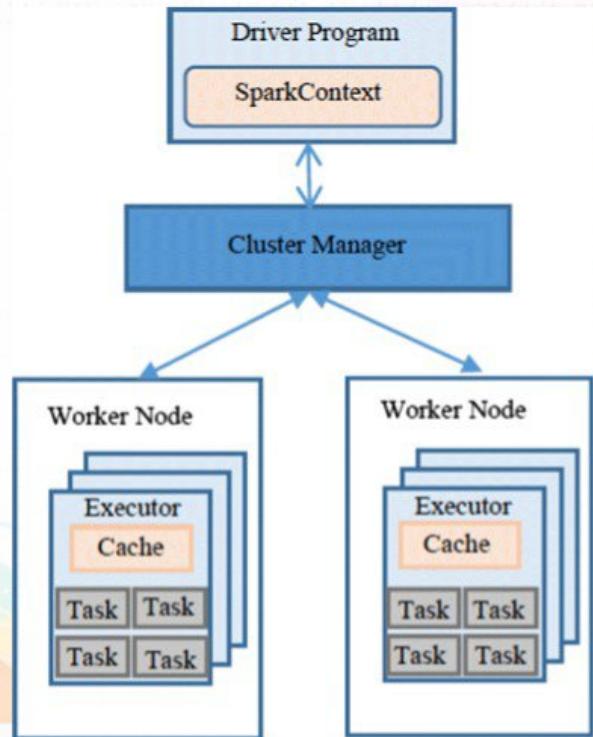
- Assigns executors

4. Executors

- Execute tasks in parallel
- Process data partitions

5. Driver

- Collects final result
- Displays output





💤 Important Concept: Lazy Evaluation

Spark does nothing until an action is called.

Transformations (Lazy):

- *select*
- *filter*
- *groupBy*

Actions (Trigger execution):

- *count()*
- *show()*
- *collect()*
- *write()*

📌 *DAG is executed only when action is called*





Remember This

Driver = Brain

Executors = Workers

DAG = Plan

Cluster = Infrastructure

Parallel processing = Spark's power





DataFrame vs RDD (Apache Spark)

What is an RDD?

RDD (Resilient Distributed Dataset) is:

- *The lowest-level data structure in Spark*
- *A distributed collection of objects*
- *Immutable and fault-tolerant*



📌 You manually control:
*Data processing
Transformations
Logic*

Example :

```
rdd =  
spark.sparkContext.textFile("file.txt")
```



What is a DataFrame?

DataFrame is:

- *A higher-level abstraction built on RDD*
- *Structured data (rows + columns)*
- *Similar to a SQL table*



- 📌 *Spark handles:*
- Optimization
- Execution planning
- Memory management

Example:

```
df = spark.read.csv("file.csv",  
header=True)
```



⚡ Why DataFrames are Faster?

1 Catalyst

- Optimizer
- Reorders operations
- Removes unnecessary steps
- Chooses best execution plan

👉 RDDs don't get these benefits

2 Tungsten Engine

- Efficient memory usage
- CPU optimization
- Less JVM overhead





Lazy Evaluation (Connection)

Both:

- *Use lazy evaluation*
- *Execute only on actions*

But:

- *DataFrame builds optimized DAG*
- *RDD builds basic DAG*



>Notebook Magic Commands

Magic commands tell Databricks which language or tool to use for a notebook cell.

They always start with % and are written at the top of a cell.

Databricks notebooks are multi-language, and magic commands make that possible.





Why Magic Commands are Needed

Without magic commands:

- Notebook runs in one default language (usually Python)

With magic commands:

- You can run SQL, Python, shell, file system commands in the same notebook

👉 Very powerful for data engineers & analysts.





Most Important Magic Commands

1 %python (Default)

-> Used to write PySpark / Python code

- ✓ Used for transformations
- ✓ DataFrame operations
- ✓ Most common in Databricks

```
%python
df = spark.read.csv(
    "/Volumes/workspace/e-commerce/
e-commerce_data/2019-Oct.csv",
    header=True,
    inferSchema=True
)
df.show(5)
```



Most Important Magic Commands

2 %sql

-> Used to run SQL queries directly

- ✓ Best for analysts
- ✓ Clean & readable
- ✓ Works on tables & temp views

```
%sql
SELECT event_type, COUNT(*)
FROM oct_events
GROUP BY event_type
ORDER BY COUNT(*) DESC
```



Most Important Magic Commands

3 %fs

->*Used for file system operations*

Common uses:

- *List files*
- *Check paths*
- *Debug missing files*

```
%fs  
ls /Volumes/workspace/ecommerce/  
ecommerce_data/
```



Most Important Magic Commands

4 %sh

->*Runs Linux shell commands*

✓ Used for:

- *kaggle download*
- *unzip*
- *rm, mv, cp*

```
%sh  
cd /Volumes/workspace/ecommerce/  
ecommerce_data  
ls -lh
```



Remember This

- *%python → Data processing*
- *%sql → Analysis & queries*
- *%fs → Check files*
- *%sh → System-level tasks*



Day 2 - Tasks Completed

- ✓ Uploaded dataset using Volumes
- ✓ Read CSV into Spark DataFrames
- ✓ Verified schema & row counts
- ✓ Created temporary views
- ✓ Performed:
 - select
 - filter
 - groupBy
 - orderBy



Key Learning Takeaways 🧠

- Spark uses *lazy evaluation*
- Execution happens only on *actions*
- SQL cannot read • *DataFrames*
- directly
- Temp views bridge • *PySpark & SQL*
- Volumes are the modern way to store data

 Big clarity on how Spark actually works



Biggest “Aha” Moment



Spark doesn't execute code line by line – it builds a plan first, then executes efficiently.

This changed how I look at data processing 🚀





Day 2 built a strong Spark foundation

Ready to go deeper into real transformations & analytics 🔥



On to Day 3!

