What is Dynamic Resource Allocation?

Dynamic Resource Allocation (DRA) is a Spark feature that **automatically scales the number of executors up or down** during the lifecycle of a Spark application. It does this based on the **current workload**, such as pending tasks and idle resources.

⚙️ How It Works

Executor Addition:

When there are **pending tasks** and not enough executors, Spark **adds executors** to speed up processing.

Executor allocation depends on available cluster capacity and configured limits (minExecutors, maxExecutors).

Executor Removal:

Executors that remain **idle for a certain duration** (executor Idle Timeout) are automatically removed to free resources.

Spark tracks task queues and executor usage continuously.

Tracking:

Spark tracks task queues, active stages, and executor usage via **TaskScheduler**.

When task demand increases or decreases, it sends a request to the cluster manager to scale resources accordingly.

Prerequisites

External Shuffle Service must be enabled:

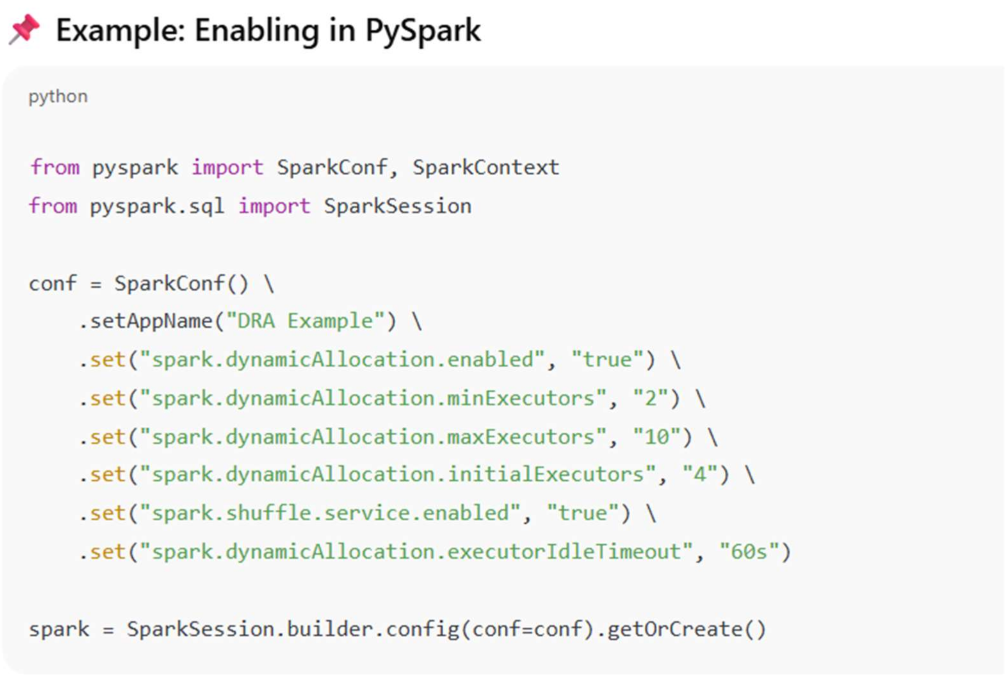
This allows shuffle data to be preserved even after an executor is removed.

Required setting:

```
spark.shuffle.service.enabled=true
```

Works with cluster managers like **YARN**, **Kubernetes**, or **Standalone** (with shuffle service).

Parameter	Description	Example
spark.dynamicAllocation.enabled	Enables or disables dynamic allocation.	true
spark.dynamicAllocation.minExecutors	Minimum executors Spark should allocate.	2
spark.dynamicAllocation.maxExecutors	Maximum executors Spark can allocate.	50
spark.dynamicAllocation.initialExecutors	Executors to start with.	5
spark.dynamicAllocation.executorIdleTimeout	Time to wait before killing idle executors.	60s
spark.shuffle.service.enabled	Required for DRA to keep shuffle files when executors are removed.	true



📌 Example: Enabling in PySpark

python

```python
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

conf = SparkConf() \
    .setAppName("DRA Example") \
    .set("spark.dynamicAllocation.enabled", "true") \
    .set("spark.dynamicAllocation.minExecutors", "2") \
    .set("spark.dynamicAllocation.maxExecutors", "10") \
    .set("spark.dynamicAllocation.initialExecutors", "4") \
    .set("spark.shuffle.service.enabled", "true") \
    .set("spark.dynamicAllocation.executorIdleTimeout", "60s")

spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

Benefit	Description
Cost Efficiency	Frees unused executors → saves cost in cloud environments.
Adaptability	Adjusts to varying loads: e.g., wide joins need more resources, narrow transformations less.
Better Utilization	Prevents waste by scaling down idle executors.
Elastic Scaling	Useful in shared clusters with unpredictable workloads.