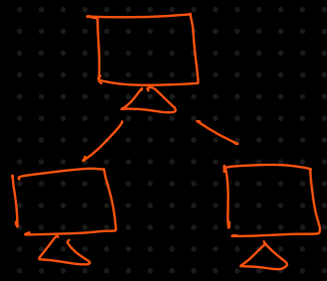
Spark Architecture and components

Apache spark is designed to process large datasets quickly and efficiently by distributing the workload across multiple machines in a cluster.

Its Architecture ensures parallel execution and in-memory computation, making it significantly faster than Traditional Frameworks like Hadoop MapReduce

★ processing framework



How Spark can run?

1. Standalone mode

   Spark has a built in cluster manager

   Suitable for small scale setups or testing purpose

2. YARN

   Spark can get integrated with YARN

   Use for large-scale production environments

   Google dataproc cluster ⇓ Spark — YARN

3. Other resource manager

   Mesos

   Kubernetes

Distributed Nature of Spark

Spark splits data into smaller chunks ⇓ partition

partition are processed in parallel across clusters.

1. Massive Datasets
2. Parallel Processing

100,000 ⟶ 100 people ↙ 1000 papers

In-memory Computation

Unlike MR, which write intermediate results to disk, Spark perform computation memory.

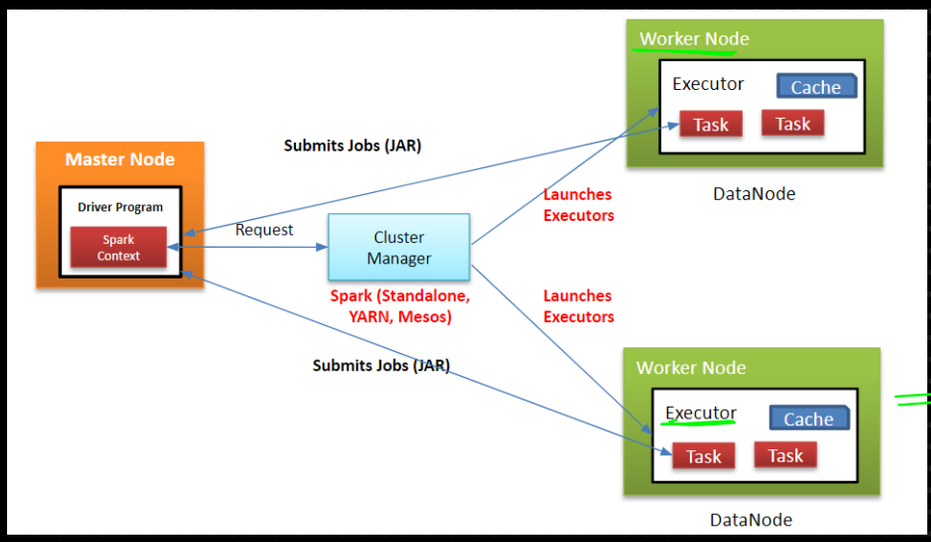
2^8

$2^1=2$ $2^2=4$ $2^3=8$ 2^4

2 4 8 16 ... 256

256

Component of Spark architecture



all components work together to ensure distributed data processing

⇒ don't have any storage

1. Driver Program → entry point of our Spark Application main()

Roles
- Creating DAG from user defined code
- Scheduling tasks and managing the execution plan
- Collect the result from executors.

2. Spark Context: entry point for the spark functionality.

3. Cluster manager: manages and allocate resources for spark application

Standalone YARN mesos Kubernetes

4. Executors worker node in the spark cluster ≈ Container

3 cores, 8gb ram
1 core → 1 task

- executing tasks assigned by the driver
- Perform data transformation and action
- Stores the intermediate results in the memory.

4. Task Smallest unit of work in Spark ≈ partition

Created by splitting data

DAG → direct acyclic graph Scheduler

convert high level transformation (map, filter) → series of stages

stage can have task to execute

Typical Spark Job execution

1. Job submission: driver program submits spark application to cluster manager
2. DAG creation: driver constructs a DAG representing series of transf.
3. Task division: DAG scheduler breaks down DAG → stages
   each with parallelizable tasks
4. Resource Allocation: Cluster manager allocates resources based on resource requirements.
5. Task execution: executors who process task in parallel
6. Intermediate storage: results in between can be stored in memory/disk
7. Result Collection: executor sends back the result to driver.

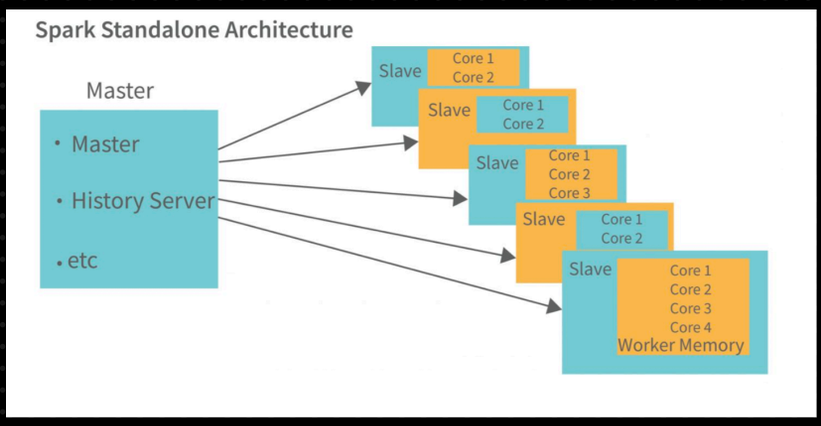
How each component interacts in the process

- **Driver** communicates with the **cluster manager** to request resources.
- **Cluster manager** assigns resources (executors) to the driver.
- **Executors** perform the actual computation on partitions of data.
- **Tasks** are executed by executors, ensuring parallel processing.
- Intermediate results are stored in the **storage layer** if memory is insufficient.

Worker :- recieves task from the master and execute them.
each worker has multiple executor

Spark Standalone architecture

★ Interview

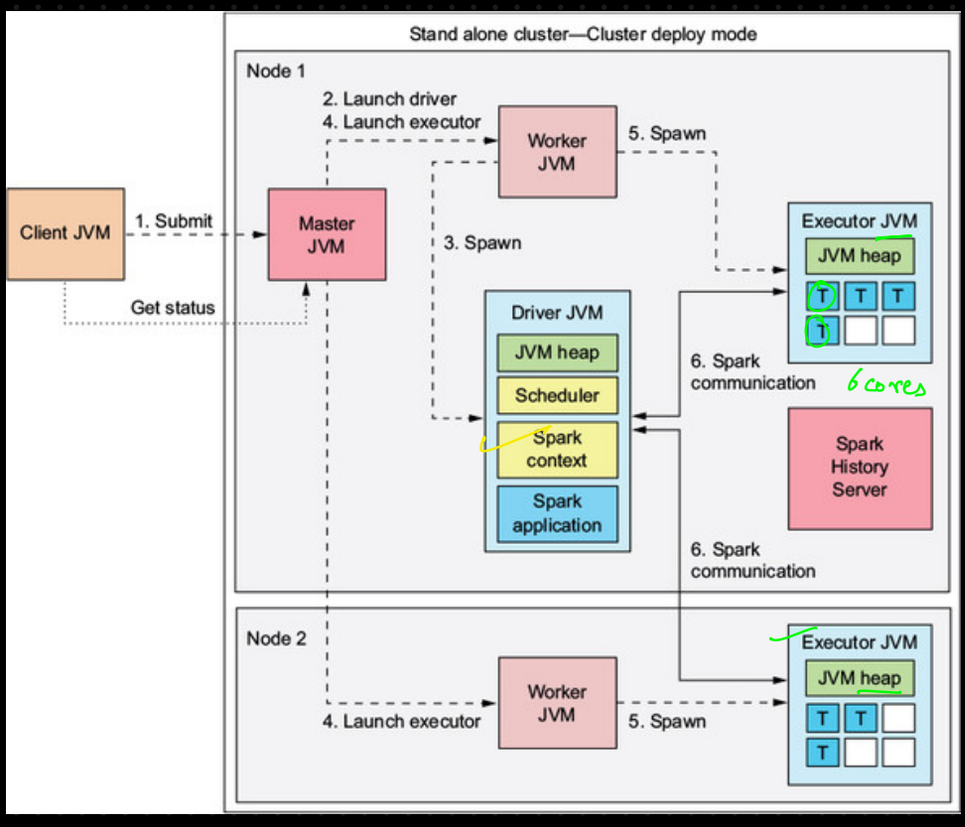


1 core ≃ 1 task

Practically ⇒ standalone

1. draw
2. write the role of each comp & process.

Master: Base of spark standalone cluster
→ central point & entry to the spark cluster.



Master
central point of coordination
→ Register application
→ allocate resources
→ schedule task to run on worker
→ Monitoring the job execution

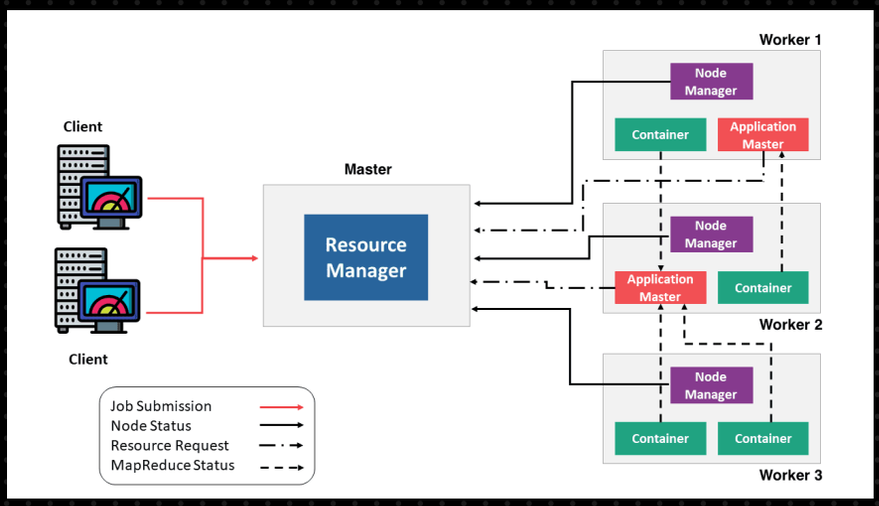
Driver -
- Contains the spark application main logic
-

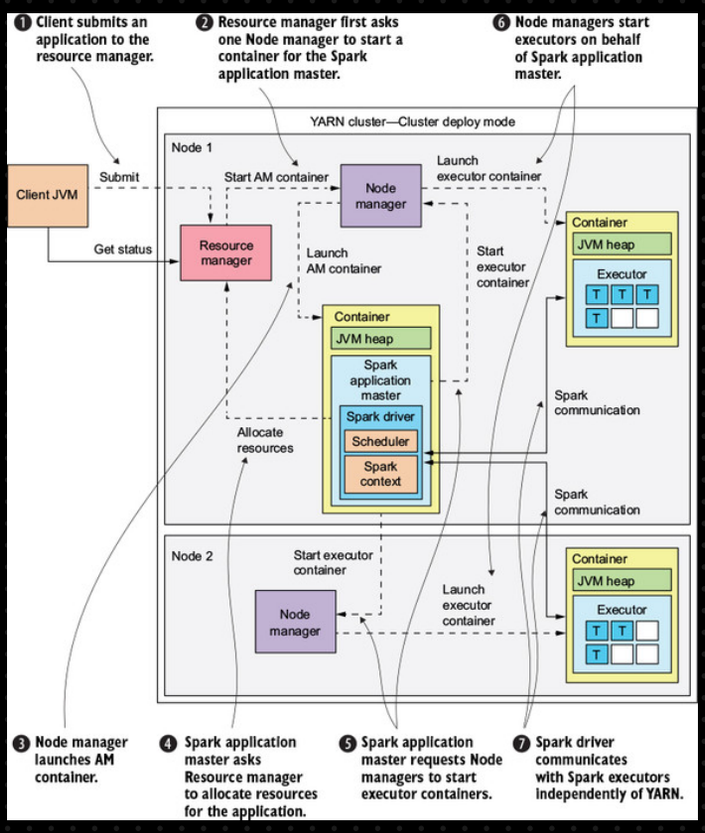
Execution flow

1. A Spark application is submitted to the Master.
2. The Master allocates resources on the Worker nodes.
3. The Master tells the Workers to launch Executors.
4. The Driver program (in its JVM) divides the application into tasks.
5. The Scheduler assigns tasks to the Executors.
6. The Executors execute the tasks in parallel.
7. The results are communicated back to the Driver.
8. The History Server records the job execution details.

Spark on YARN architecture

YARN as a cluster manager





1. Driver Program
   - → submit jobs
   - → create DAGs
   - → managing execution
     - ↙ client
     - ↘ cluster ✓

Difference Between Spark Standalone and YARN architecture

Aspect	Standalone Architecture	YARN Architecture
Cluster Manager	Built-in Spark cluster manager.	YARN (part of Hadoop ecosystem).
Resource Management	Spark directly manages resources.	YARN manages resources and assigns containers.
Scalability	Suitable for small to medium-scale clusters.	Designed for large-scale production clusters.
Driver Location	Runs on the master node of the Spark cluster.	Can run on a client machine (client mode) or YARN (cluster mode).
Fault Tolerance	Limited to Spark's built-in mechanisms.	Relies on YARN's fault tolerance and resource allocation.
Ease of Use	Simple to set up and manage.	Requires integration with Hadoop YARN.
Integration	Primarily used as a standalone framework.	Seamlessly integrates with Hadoop tools like Hive and HBase.
Executor Management	Managed directly by Spark.	Executors run inside YARN-managed containers.
Application Monitoring	Uses Spark UI for job monitoring.	Uses both Spark UI and YARN Web UI for monitoring.

→ In spark standalone cluster, Spark has full control over resource allocation, making it simpler but less suited for large shared cluster

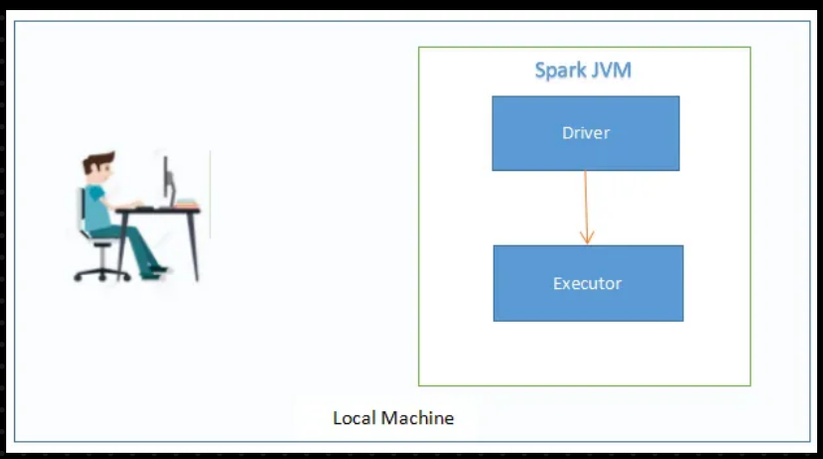
→ In spark on YARN, yarn provides robust resource management and scalability, ideal for production-grade application.

Spark Deployement Mode

Apache spark offers multiple deployement modes, allowing flexibility in running applications based on the use case and underlying infrastructure

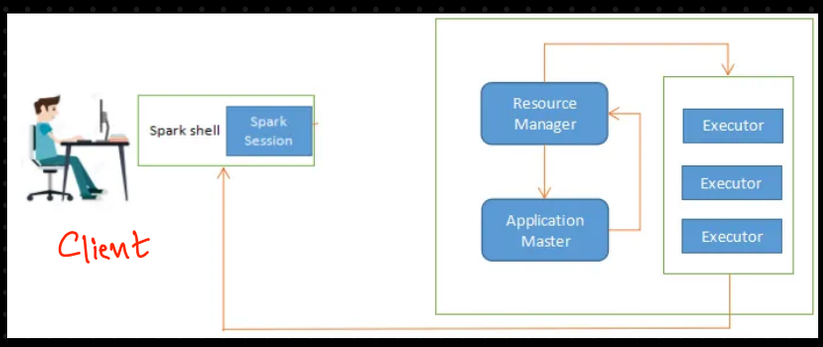
These modes dictate where the driver program runs and how executors interact with the driver.

1. Local Mode



Spark runs on a single machine using one or more threads for parallelism.

2. Client mode

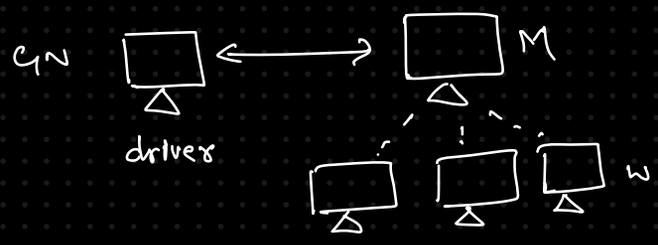


driver program runs on the client machine

- Jupyter notebooks
- Jobs requiring close monitoring of driver output

Good for debugging & line to line run

Gateway Node



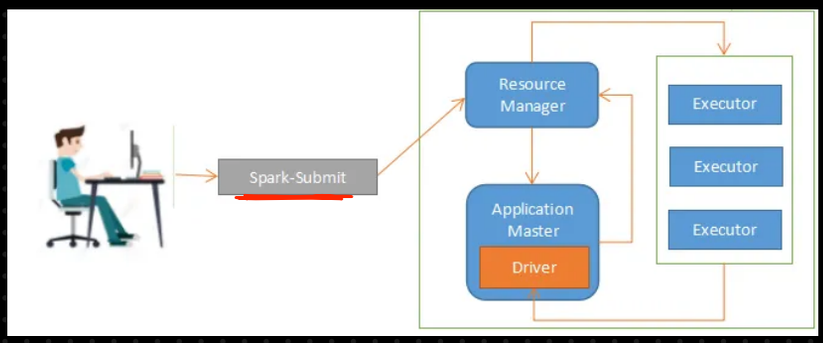
3. Cluster Mode

python app.py →

driver program runs within the cluster alongside the executors.

→ Production grade jobs

→ Jobs requiring high availability and fault tolerance.



Aspect	Local Mode	Client Mode	Cluster Mode
Driver Location	Same machine as the application	On the client machine (job submission node)	On one of the cluster nodes
When to Use	Development/testing	Interactive applications or monitoring jobs	Production-grade, automated jobs
Resource Management	Limited to local machine	Executors on cluster, driver on client	Both driver and executors on the cluster
Fault Tolerance	Minimal	Depends on cluster manager (executor fault-tolerant)	High (driver is fault-tolerant in cluster)
Communication Latency	Low	Higher due to driver-cluster communication	Minimal latency
Cloud Examples	Local VM for development	Jupyter on Dataproc or EMR	Spark-submit jobs on AWS EMR/GCP Dataproc