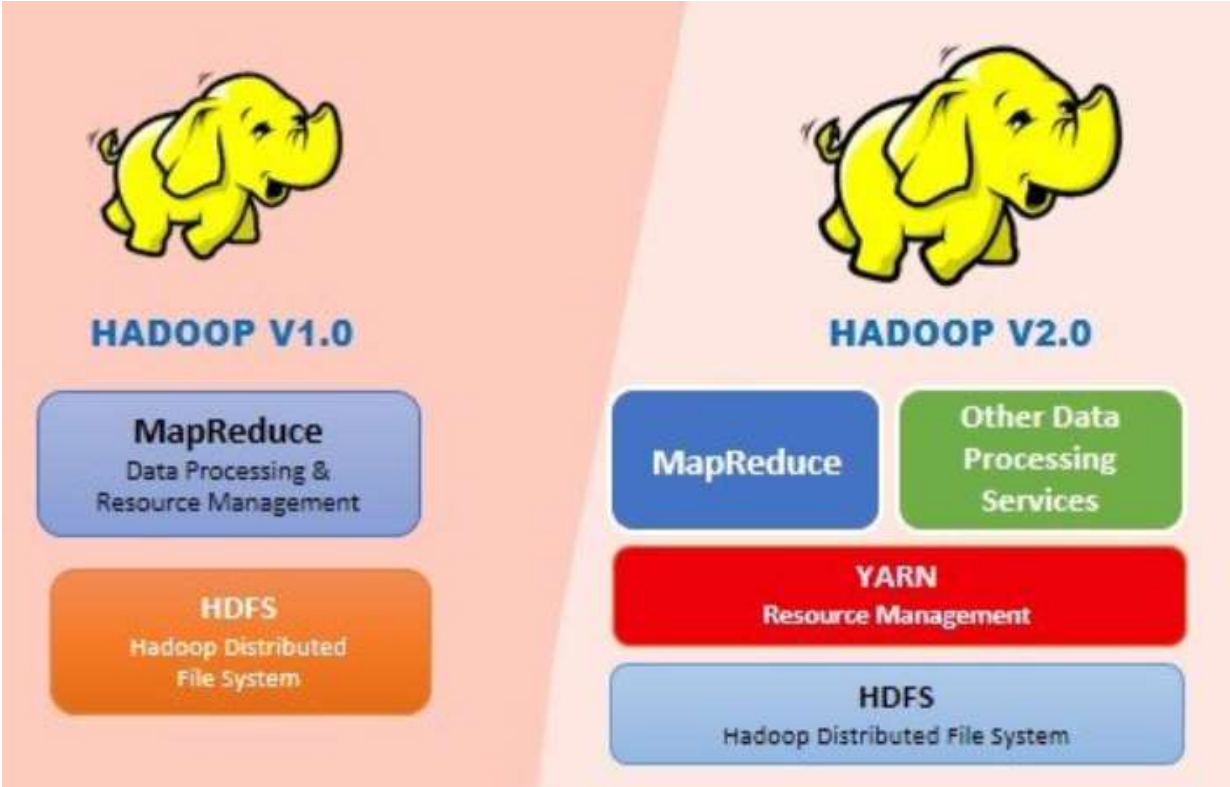
HDFS – MapReduce - YARN

- **HDFS** stores the data while **YARN** coordinates the resources needed to process that data through containers.
- **MapReduce** jobs are executed on the resources allocated by YARN and make use of the **HDFS** for storing input and output data.
- **YARN** handles **resource management** (scheduling, allocation, and monitoring of tasks across the cluster).

YARN

- YARN (Yet Another Resource Negotiator) is a resource management layer for the Hadoop ecosystem.
- It is responsible for managing and scheduling resources across the Hadoop cluster and overseeing the execution of applications in a distributed environment.



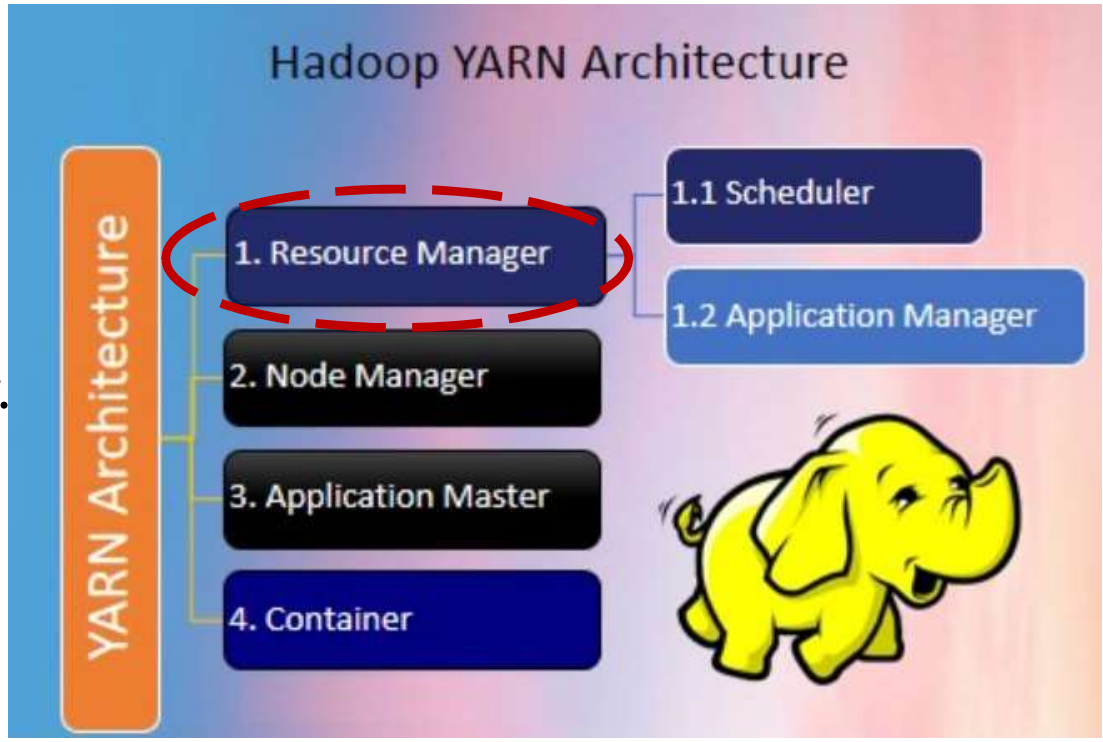
- YARN was introduced in Hadoop 2.x to overcome the limitations of the original MapReduce framework in Hadoop 1.x.

YARN operates as a resource management layer that decouples the resource management and job scheduling aspects of Hadoop, which were previously handled by the **JobTracker** and **TaskTracker** in Hadoop 1.x. *YARN enables Hadoop to run multiple types of distributed applications, not just MapReduce.*

YARN Components

1. Resource Manager (RM):

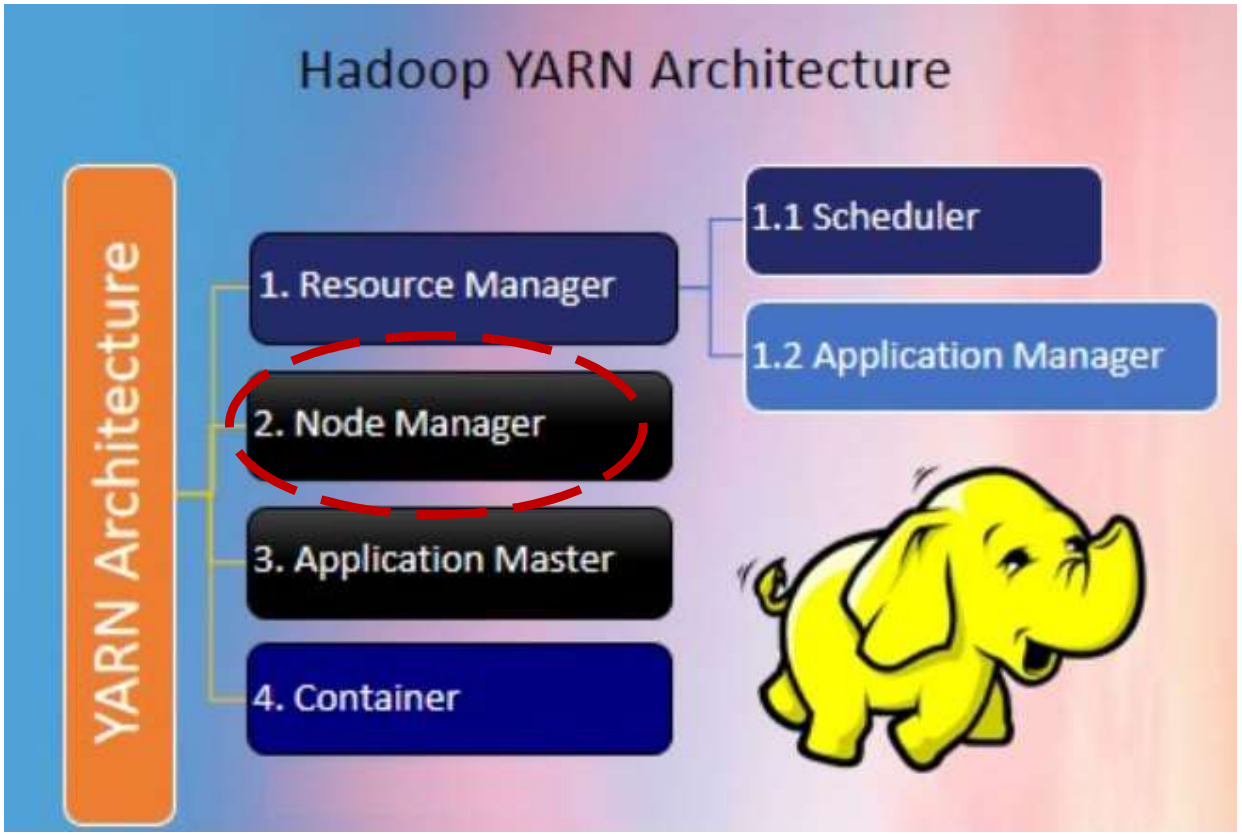
- It is the central authority responsible for managing and allocating resources to different applications running on the cluster.
- It decides where each task should run based on available resources, cluster capacity, and application needs.



It has two main components:

- **Scheduler:** Decides how to allocate resources to different applications based on policies (e.g., capacity, fairness).
- **Application Manager:** Manages the lifecycle of applications and their jobs (launching and terminating applications, handling failures, etc.). It is responsible for accepting job submission. Negotiates the first container from RM for executing the application specific.

YARN Components



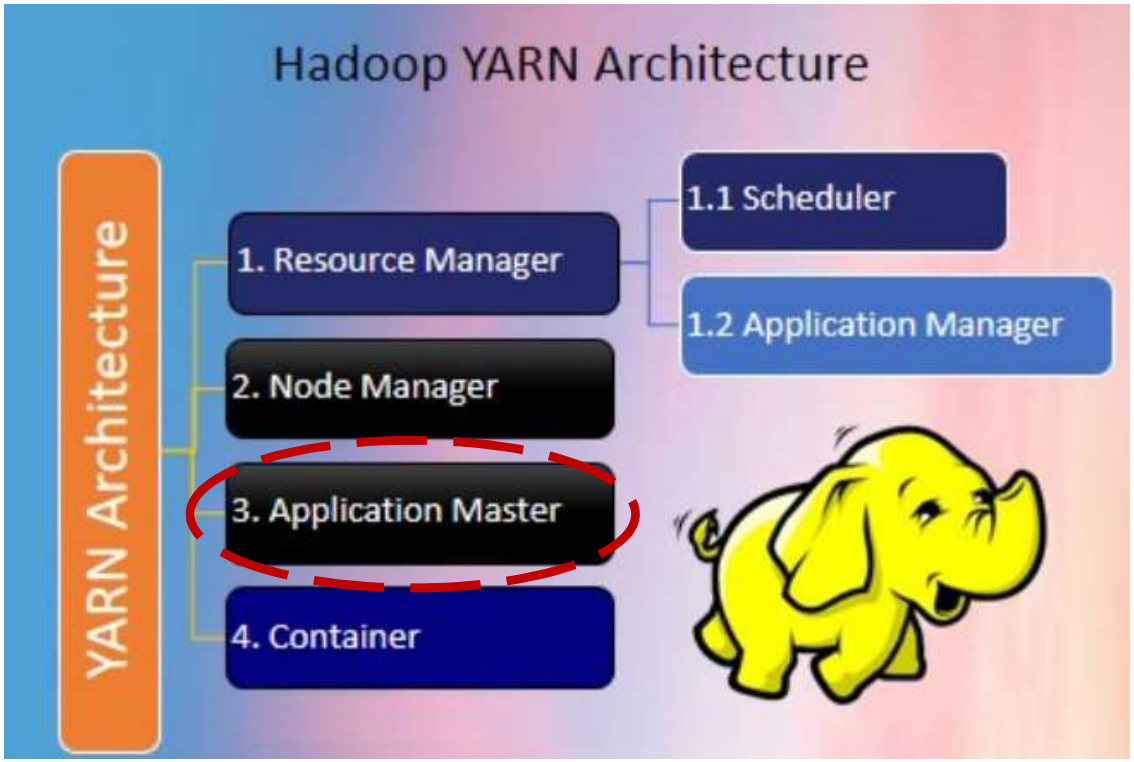
2. Node Manager (NM):

- Each node in the cluster runs a Node Manager. The Node Manager is responsible for:
  - Monitoring the resource usage (CPU, memory, disk) on the node.
  - Running containers where tasks (like MapReduce jobs) are executed.
  - Reporting resource usage to the **Resource Manager**.
  - Managing the lifecycle of containers.

YARN Components

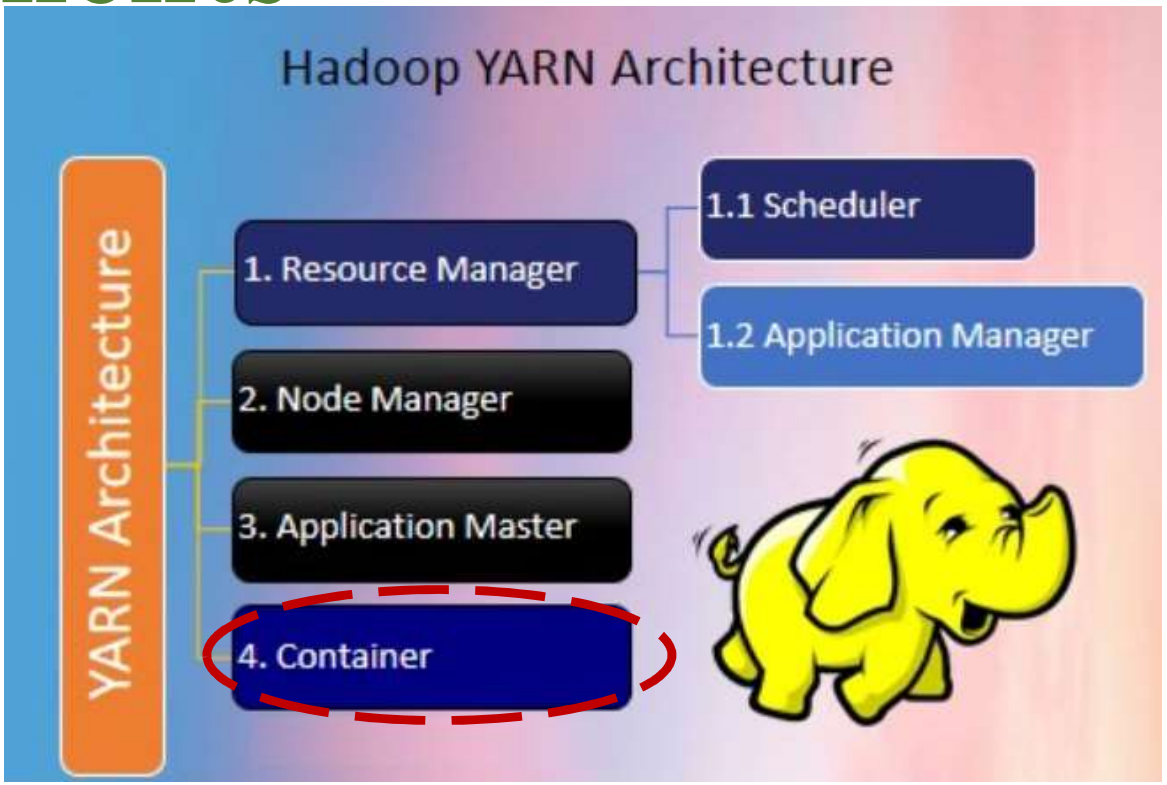
3. Application Master (AM):

- Each application (e.g., a MapReduce job) has its own Application Master. The Application Master is responsible for:



- Negotiating resources with the **Resource Manager**.
- Launching and managing tasks (executing containers) on various nodes in the cluster.
- Monitoring the status of tasks and handling failures (by restarting tasks if needed).
- Reporting the status of the application to the client.

YARN Components



4. Containers:

- A container is a collection of resources (such as memory and CPU) allocated by the Resource Manager for running tasks. The NodeManager on each node manages these containers and ensures the tasks run within their allocated resources.
- Each task runs in a separate container, which is isolated from other tasks, ensuring resource separation and preventing conflicts.

How YARN Work?

1. A client submits an application (e.g., a MapReduce job) to the Resource Manager.
2. The Resource Manager allocates resources (containers) for the application and sends the request to the Node Manager on a particular node.
3. The Node Manager starts a container and launches the Application Master.
4. The Application Master negotiates resources for each task of the job, executes them in containers, and monitors their progress.
5. Once all tasks are completed, the Application Master terminates, and the resources are released.

YARN essentially separates the resource management and job execution responsibilities, making the Hadoop ecosystem more efficient, scalable, and adaptable for various processing frameworks.

How YARN Interacts with MapReduce

YARN provides foundation for running **MapReduce** jobs in a distributed manner and interact in the following ways:

1. **Client Submission**: A client submits a MapReduce job to the Resource Manager. Job is typically packaged with the logic for the Mapper and Reducer.

2. **Resource Manager Allocation**:

   - **Resource Manager** receives the request and allocates resources for the job. It identifies which **Node Managers** have the available resources to run the job's tasks (mappers and reducers).

   - **Resource Manager** launches an **Application Master** for MapReduce job.

3. **Application Master Management**:

   - **Application Master** negotiates with the **Resource Manager** for resources to run **containers** on available nodes. The Application Master is responsible for the coordination of the MapReduce job execution.

   - It requests **containers** from the **Resource Manager** on various **Node Managers** to execute Map and Reduce tasks.

4. **Task Execution:**
    - The **NodeManager** on each node launches containers to run the **Map** tasks (for Mapper) and **Reduce** tasks (for Reducer).
    - Each **Map task** operates on a chunk of data from **HDFS** and processes it.
    - The **Reduce task** operates on the output of the Map tasks, aggregating or reducing the results as per the job requirements.
5. **Data Movement**:
    - **HDFS** (Hadoop Distributed File System) stores the data and provides a distributed storage layer. The **Map tasks** process data stored in **HDFS**, and the **Reduce tasks** fetch the intermediate results from **HDFS** after the **Map tasks** have completed.
    - After the reduce phase, the results are stored back into **HDFS**.
6. **Completion**:
    - Once the MapReduce job is complete, the **ApplicationMaster** notifies the client of job completion, and the resources are released.

How YARN Interacts with HDFS

YARN and HDFS interact as follows:

1. **Storage and Data Availability**:
   - **HDFS** is responsible for storing the data required for any job, including MapReduce jobs. Data is split into blocks (default size 128MB) and distributed across multiple nodes in cluster to ensure fault tolerance and high availability.
   - When a job (e.g., MapReduce) is submitted, **the application tasks running on YARN containers access the data stored in HDFS for processing**.
2. **Data Processing**:
   - **MapReduce tasks** (running under YARN) read input data from **HDFS** and process it. The data might be split into smaller chunks (blocks) for processing by multiple mappers.
   - **Intermediate results** of the Map phase are written back to **HDFS** before being picked up by the **Reducer** tasks.
3. **Fault Tolerance**:
   - HDFS provides fault tolerance by replicating data blocks across multiple nodes. If one node fails, another replica of the data is used.
   - YARN handles fault tolerance by re-scheduling tasks in case of node failures or task failures, ensuring the job continues to progress.

YARN- MapReduce -HDFS

- **YARN** handles **resource management** (scheduling, allocation, and monitoring of tasks across the cluster).
- **MapReduce** jobs are executed on the resources allocated by YARN and make use of the **HDFS** for storing input and output data.
- **HDFS** stores the data while **YARN** coordinates the resources needed to process that data through containers.

What is a Web Server Log?

A web server log entry

127.0.0.1 - Jake [10/Oct/2021:13:55:36 -0700] "GET /monkey_pb.gif HTTP/1.0" 200 2326

In this illustration, a client with the IP address 127.0.0.1, presumably Jake, requested the file /monkey_pb.gif at a specific time and date.

The server's response was a 200 status code (reflecting success), and it transmitted a file of 2326 bytes.

Another example of a web server log's entry could be:

192.0.2.1 - - [07/Dec/2021:11:45:26 -0700] "GET /index.html HTTP/1.1" 200 4310

This excerpt shows a user, with the IP address 192.0.2.1, triggered a request on 7th December 2021 at 11:45:26.

The requested page was 'index.html', using the HTTP/1.1 protocol.

The server successfully engaged with the request (status code: 200), responding with 4310 bytes.

Web server logs are generally maintained in an easily decipherable text manner. But their size can become extensive, particularly for heavily trafficked sites necessitating parsing and analyzing applications or tools.

What is a Web Server Log?

- This ledger is akin to the server's event history, documenting every request handled, every document delivered, and every glitch faced.
- **Access Logs**: Logging all the requests handled by the server. It reveals who accessed what, when, and in what manner.
- **Error Logs**: Logging encountered server errors. These can be key to diagnosing and identifying issues within the server or the site.
- **Security Logs**: Logging all security-associated incidents, like unsuccessful login attempts or doubtful activity. They are vital tools in upholding the server and the site's security robustness.