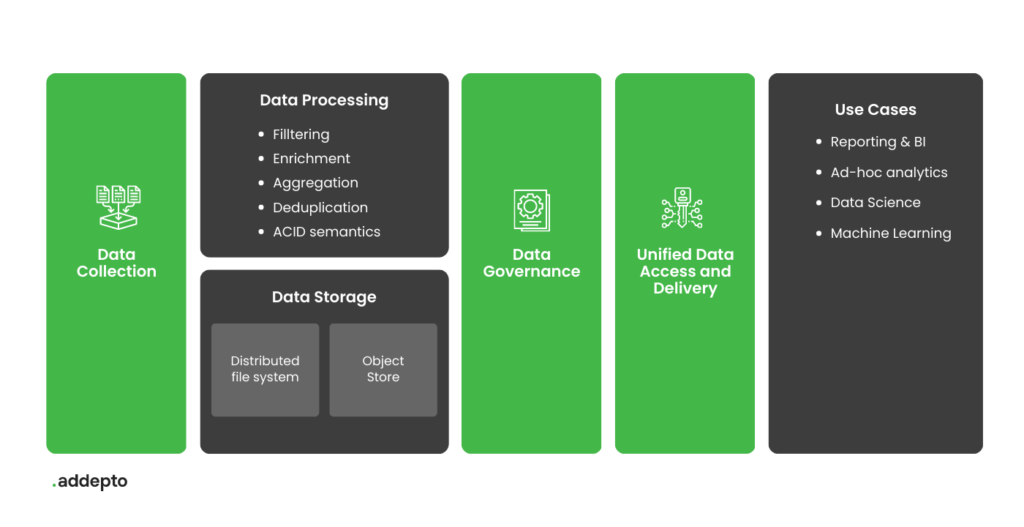
**Big Data Platform**

Big Data platform workflow can be divided into the following stages:

1. **Data Collection**  
   Big Data platforms collect data from various sources, such as sensors, weblogs, social media, and other databases.
2. **Data Storage**  
   Once the data is collected, it is stored in a repository, such as Hadoop Distributed File System (HDFS), Amazon S3, or Google Cloud Storage.
3. **Data Processing**  
   Data Processing involves tasks such as filtering, transforming, and aggregating the data. This can be done using distributed processing frameworks, such as Apache Spark, Apache Flink, or Apache Storm.
4. **Data Analytics**  
   After data is processed, it is then analyzed with analytics tools and techniques, such as machine learning algorithms, predictive analytics, and data visualization.
5. **Data Governance**  
   Data Governance (data cataloging, data quality management, and data lineage tracking) ensures the accuracy, completeness, and security of the data.
6. **Data Management**  
   Big data platforms provide management capabilities that enable organizations to make backups, recover, and archive.



These stages are designed to derive meaningful business insights from raw data from multiple sources such as website analytic systems, CRM, ERP, loyalty engines, etc. Processed data stored in a unified environment can be used in preparing static reports and visualizations but also for other analytics and – for example – building Machine Learning models.

## Big Data Platform examples

### Apache Hadoop

Hadoop is an open-source programming architecture and server software. It is employed to store and analyze large data sets very fast with the assistance of thousands of commodity servers in a clustered computing environment[6]. In case of one server or hardware failure, it can replicate the data leading to no loss of data.

 it is commonly employed on Ubuntu and other variants of Linux.

### Cloudera

Cloudera is a big data platform based on Apache’s Hadoop system. It can handle huge volumes of data. Enterprises regularly store over 50 petabytes in this platform’s Data Warehouse, which handles data such as text, machine logs, and more. Cloudera’s DataFlow also enables real-time data processing.

Cloudera platform is based on the Apache Hadoop ecosystem and includes components such as HDFS, Spark, Hive, and Impala, among others. Cloudera provides a comprehensive solution for managing and processing big data and offers features such as data warehousing, machine learning, and real-time data processing. The platform can be deployed on-premise, in the cloud, or as a hybrid solution.

### Apache Spark

Apache Spark is an open-source data-processing engine designed to deliver the computational speed and scalability required for streaming data, graph data, machine learning, and artificial intelligence applications. Spark processes and keeps the data in memory without writing to or reading from the disk, which is why it is way faster than the alternatives such as Apache Hadoop.

The solution can be deployed on-premise, in addition to being available on cloud platforms such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure.

### Databricks

Databricks is a cloud-based platform for big data processing and analysis based on Apache Spark. It provides a collaborative work environment for data scientists, engineers, and business analysts offering features such as an interactive workspace, distributed computing, machine learning, and integration with popular big data tools.

### Snowflake

Snowflake is a cloud-based data warehousing platform that provides data storage, processing, and analysis capabilities. It supports structured and semi-structured data and provides a SQL interface for querying and analyzing data.

It provides a fully managed service, which means that the platform handles all infrastructure and management tasks, including automatic scaling, backup and recovery, and security. It supports integrating various data sources, including other cloud-based data platforms and on-premise databases.

### Datameer

Datameer is a data analytics platform that provides big data processing and analysis capabilities designed to support end-to-end analytics projects, from data ingestion and preparation to analysis, visualization, and collaboration.

Datameer provides a visual interface for designing and executing big data workflows and includes built-in support for various data sources and analytics tools. The platform is optimized for use with Hadoop, and provides integration with Apache Spark and other big data technologies.

The service is available as a cloud-based platform and on-premise. The on-premise version of Datameer provides the same features as the cloud-based platform but is deployed and managed within an organization’s own data center.

### Apache Storm

Apache Storm is a free and open-source distributed processing system designed to process high volumes of data streams in real-time, making it suitable for use cases such as real-time analytics, online machine learning, and IoT applications.

Storm processes data streams by breaking them down into small units of work, called “tasks,” and distributing those tasks across a cluster of machines. This allows Storm to process large amounts of data in parallel, providing high performance and scalability.

Apache Storm is available on cloud platforms such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure, but it is possible to deploy it also on-premise.

## Complex Cloud Big Data Platform: AWS, GCP, Azure

Complex Cloud Big Data platforms refer to the cloud-based services offered by the major cloud providers Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. They are designed for processing and analyzing large, complex data sets.

#### AWS

AWS provides you with access to a broader ecosystem of tools that comprises many additional tools and features, e.g., AWS Lambda microservices, Amazon OpenSearch Service for search capabilities, Amazon Cognito for user authentication, AWS Glue for data transformation, and Amazon Athena for data analysis, Amazon EMR for processing and analyzing big data, Amazon Kinesis for real-time data processing, and Amazon Redshift for data warehousing, to name a few.

Amazon facilitates the whole process of building a data lake on the cloud and adjusting it to your needs. They automatically configure the core AWS services allowing you to tag, search, share, transform, analyze, and govern specific subsets of data. The AWS solution deploys a console that users can access to search and browse available datasets. Here, you will find additional details.

#### GCP

Google Cloud Platform provides a series of modular cloud services, including computing, data storage, data analytics, and machine learning. According to Google, you can govern purpose-built data and analytic open-source software clusters such as Apache Spark in as little as 90 seconds.

**Hadoop Distributed File System – HDFS**

The HDFS forms the underlying basis of all Hadoop installations. Files, or more generally data, is **stored i**n HDFS and **accessed** by the nodes of Hadoop.

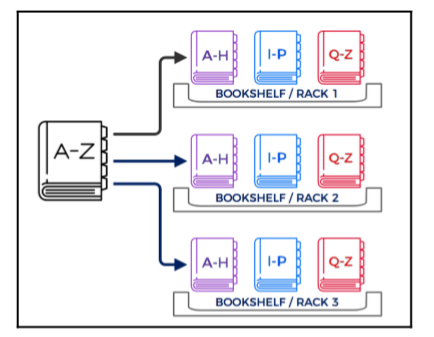
HDFS performs two main functions:

1. Namespaces: Provides namespaces that hold cluster metadata, that is, the location of data in the Hadoop cluster
2. Data storage: Acts as storage for data used in the Hadoop cluster

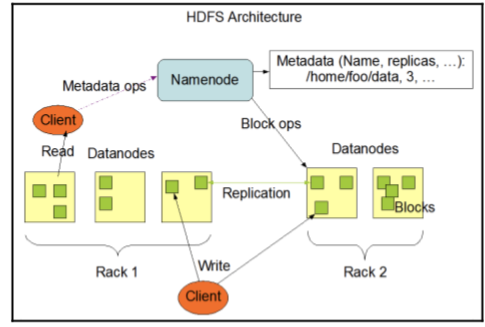
**Analogy**

The filesystem is termed as distributed since the data is stored in chunks across multiple servers.

In ordinary filesystems, the entire book would be stored as a single file on the disk. In HDFS, the book would be split into smaller chunks, say a chunk for Chapters A - H, another for I - P, and a third one for Q - Z.



* A process can **read the book in parallel** by querying the parts from different servers. This reduces I/O contention and is a very fitting example of the proper use of parallelism.
* If any of the racks are **not available**, we can retrieve the chapters from any of the other racks as there are multiple copies of each chapter available on different racks
* selective **access rights** to different chapter groups
* access a single chapter



HDFS backend of Hadoop consists of

NameNode:

* This can be considered **the master node**. The NameNode contains cluster metadata and is aware of what data is stored in which location - in short, it holds the namespace.
* In Hadoop 2, there can be more than one NameNode. A secondary NameNode can be created that acts as a helper node to the primary

DataNode:

* The DataNodes are the **individual servers that are responsible for storing chunks** of the data and performing compute operations when they receive a new request.
* These are primarily **commodity servers** that are less powerful in terms of resource and capacity than the NameNode that stores the cluster metadata.

**Data storage process in HDFS**

All data in HDFS is written in blocks, usually of size 128 MB. Thus, a single file of say size 512 MB would be split into four blocks (4 \* 128 MB). These blocks are then written to DataNodes.

The general process of writing data into HDFS is as follows:

1.The **NameNode receives a request** to write a new file to HDFS.

2.Since the data has to be written in blocks or chunks, the **HDFS client** (the entity that made the request) **begins caching data** into a local buffer and once the buffer reaches the allocated chunk size (for example, 128 MB), it informs the NameNode that it is ready to write the first block (chunk) of data.

3.The NameNode, based on information available to it about the state of the HDFScluster, responds with information on the destination DataNode where the block needs to be stored.

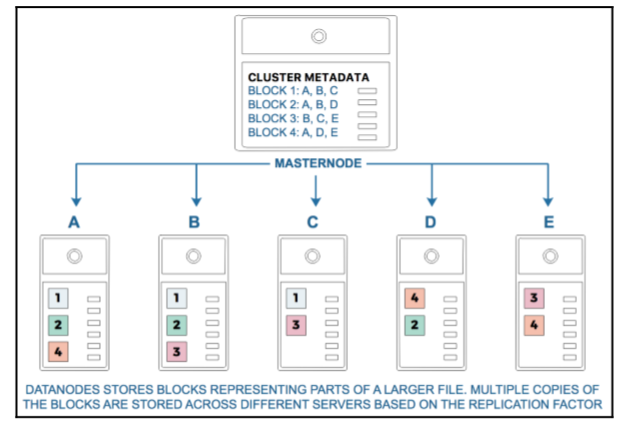
4.The **HDFS client writes data to the target DataNode** and **informs the NameNode** once the write process for the block has completed.

5. The **target DataNode**, subsequently, **begins copying its copy of the block of data to a second DataNode**, which will serve as a replica for the current block.

6. Once the second DataNode completes the write process, it sends the block of data to the third DataNode.

7.This process repeats until all the blocks corresponding to the data (orequivalently, the file) are copied across different nodes. Note that the number of chunks will depend on the file size.

The following image illustrated the distribution of the data across 5 datanodes.



The HDFS architecture in the **first release of Hadoop**, also known as Hadoop had the following characteristics:

**Single NameNode**: Only one NameNode was available, and as a result it also acted as a single point of failure since it stored all the cluster metadata.

**Multiple DataNodes** :that stored blocks of data, processed client requests, and performed I/O operations (create, read, delete, and so on) on the blocks.

The HDFS architecture in the **second release of Hadoop**, also known as Hadoop 2, provided all the benefits of the original HDFS design and also added some new features, most notably, the ability to have **multiple NameNodes that can act as primary and secondary NameNodes.**

**Hadoop MapReduce**

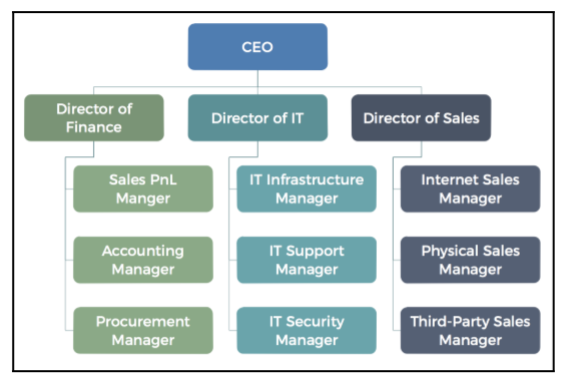
MapReduce works on the **principle of dividing larger tasks into smaller subtasks.**

Instead of delegating a single machine to compute a large task, **a network of smaller machines** can instead be used to complete the smaller subtasks.

By distributing the work in this manner, the task can be completed much more efficiently relative to using a single-machine architecture

**Analogy**

The CEO wants to know how many new hires have joined the company. The CEO sends a request to his or her directors to report back the number of hires in their departments. The directors in turn send a request to managers in their individual departments to provide the number of new hires. The managers provide the number to the directors, who in turn send the final value back to the CEO.



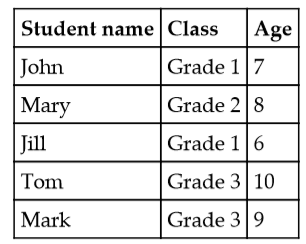
A technical understanding of MapReduce MapReduce, as the name implies, has a map phase and a reduce phase.

The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

Key-value: A key-value pair establishes a relationship. For example, if John is 20 years old, a simple key-value pair could be (John, 20). In MapReduce, the map operation produces such key-value pairs that have an entity and the value assigned to the entity.

The reduce phase takes the key-value input from the map function and performs a summarization operation. For example, consider the output of a map operation that contains the ages of students in different grades in a school:



our key-value pairs would be (Grade 1, 7), (Grade 2, 8), (Grade 1, 6), (Grade 3, 10), and (Grade 3, 9).

For example, Server A would receive the tuples (Grade 1, 7) and (Grade 1, 6), Server B would receive the tuple (Grade 2, 8), Server C would receive the tuples (Grade 3, 10) and (Grade 3, 9). Each of the servers, A, B, and C, would then find the average of the tuples and report back (Grade 1, 6.5), (Grade 2, 8), and (Grade 3, 9.5).

In Hadoop, the following process takes place during MapReduce:

1.The client sends a request for a task.

2.NameNode allocates DataNodes (individual servers) that will perform the map operation and ones that will perform the reduce operation. **The servers where the data resides can only perform the map operation.**

3.DataNodes perform the map phase and produce key-value (k,v) pairs.

As the mapper produces the (k,v) pairs, they are sent to these reduce nodes based on the keys the node is assigned to compute.

**By delegating multiple individual nodes to perform computations independently, the Hadoop architecture can perform very large-scale data processing effectively.**

**For example, given a 512 MB file and a 128 MB block size, four blocks would be needed to store the entire file. Hence, a MapReduce operation will at a minimum require four map tasks whereby each map operation would be applied to each subset of the data**

**MapReduce programming offers several benefits to help you gain valuable insights from your big data:**

**Scalability**. Businesses can process petabytes of data stored in the Hadoop Distributed File System (HDFS).

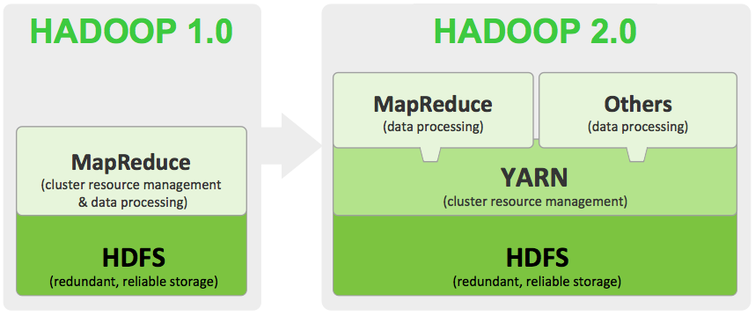
**Flexibility**. Hadoop enables easier access to **multiple sources of data and multiple types of data.**

**Speed**. **With parallel processing and minimal data movement**, Hadoop offers fast processing of massive amounts of data.

**Simple**. Developers can write code in a choice of languages, including **Java, C++ and Python.**

**Yarn in Hadoop - Arcitecture, Components and Working**

In Hadoop 1, the process of managing jobs and monitoring them was performed by processes known as JobTracker and TaskTracker(s). NameNodes that ran the JobTracker daemon (process) would submit jobs to the DataNodes which ran TaskTracker daemons (processes).



The JobTracker was responsible for the co-ordination of all MapReduce jobs and served as a central administrator for managing processes**, handling server failure, re-allocating to new DataNodes,** and so on. The TaskTracker monitored the execution of jobs local to its own instance in the DataNode and provided feedback on the status to the JobTracker as shown in the following:

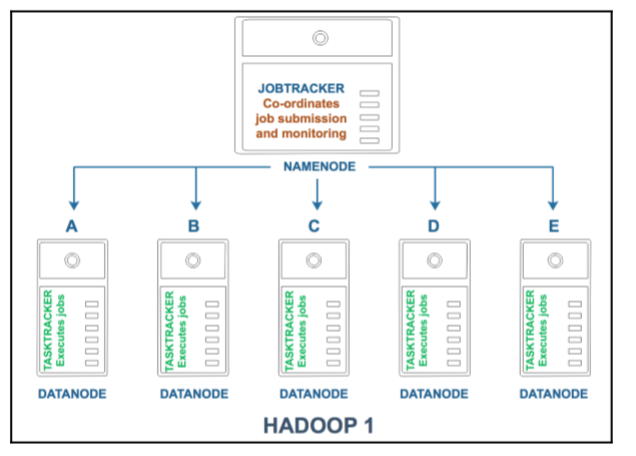
In Hadoop 1, the NameNode, and consequently the JobTracker process, managed both job scheduling and resource monitoring. In the event the NameNode failed, all activities in the cluster would cease immediately

Hadoop 2 alleviated all these concerns:

**The process of job management, scheduling, and resource monitoring was decoupled and delegated to a new framework/module called YARN**

A secondary NameNode could be defined which would act as a helper for the primary NameNode

Further, Hadoop 2.0 would accommodate frameworks beyond MapReduce Instead of fixed map and reduce slots,

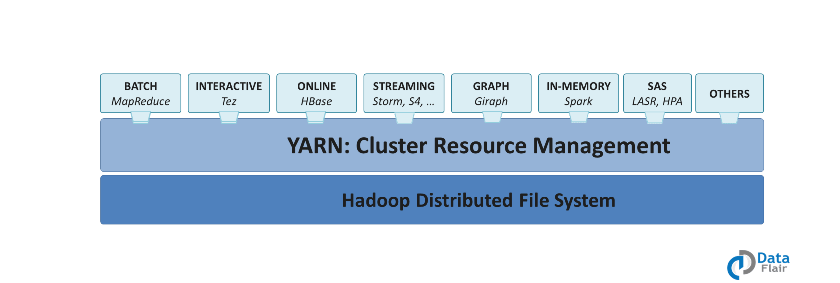


**What is YARN in Hadoop?**

* Apache YARN (Yet Another Resource Negotiator) is a resource management layer in Hadoop.
* YARN came into the picture with the introduction of **Hadoop 2.x.**
* It allows various data processing engines such as **interactive processing, graph processing, batch processing, and stream processing to run and process** data stored in HDFS (Hadoop Distributed File System).

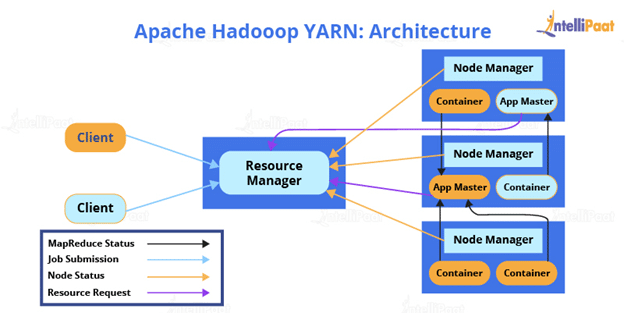
**Why is YARN in Hadoop used?**

* Hadoop 1.x had some shortcomings like **delays in batch processing**, **scalability issues**, etc. as it **relied on MapReduce** for processing big datasets.
* With YARN, Hadoop is now able to support a variety of processing approaches and has a larger array of applications.



* Hadoop YARN clusters are now able to run stream data processing and interactive querying side by side with MapReduce batch jobs.
* YARN framework runs even the non-MapReduce applications, thus overcoming the shortcomings of Hadoop 1.x.

**Hadoop YARN Architecture**



Apache YARN framework contains a

* **Resource Manager (master daemon),**
* **Node Manager (slave daemon), and**
* **An Application Master.**

Resource Manager

Resource Manager is the master daemon of YARN. It is responsible for managing several other applications, along with the **global assignments of resources such as CPU and memory**. It is used for **job scheduling**. Resource Manager has two components:

* Scheduler: Schedulers’ task is to distribute resources to the running applications. It only deals with the **scheduling of tasks** and hence it performs no tracking and no monitoring of applications.
* Application Manager: The application Manager **manages applications** running in the cluster. Tasks, such as the **starting of Application Master** or monitoring, are done by the Application Manager.

Node Manager

Node Manager is the slave daemon of YARN. It has the following responsibilities:

* Node Manager has to **monitor the container’s resource usage**, along with **reporting it to the Resource Manager**.
* The **health of the node** on which YARN is running is tracked by the Node Manager.
* It takes **care of each node** in the cluster while managing the workflow, along with user jobs on a particular node.
* It keeps the data in the Resource Manager updated
* Node Manager can also destroy or kill the container if it gets an order from the Resource Manager to do so.

The third component of Apache Hadoop YARN is the Application Master.

Application Master

**Every job submitted to the framework is an application, and every application has a specific Application Master associated with it.**

Application Master performs the following tasks:

It coordinates the execution of the application in the cluster, along with managing the faults.

It negotiates resources from the Resource Manager.

It works with the Node Manager **for executing and monitoring** other components’ tasks.

At regular intervals, **heartbeats are sent to the Resource Manager** for checking its health, along with updating records according to its resource demands.

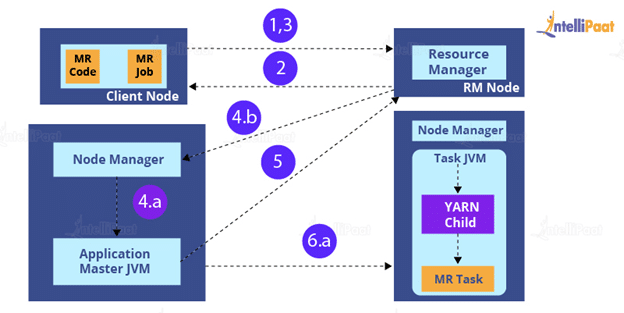
Container

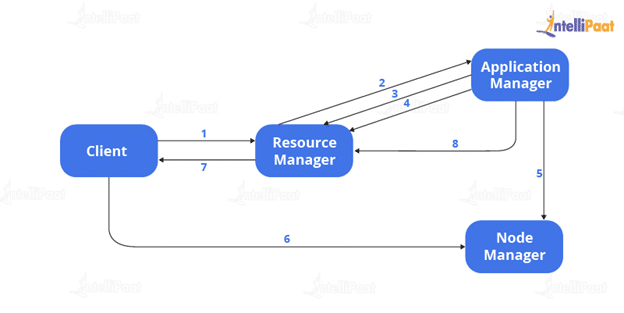
A container is a **set of physical resources (CPU cores, RAM, disks, etc.)** on a single node. The tasks of a container are listed below:

It grants the right to an application to use a specific amount of resources (memory, CPU, etc.) on a specific host.

YARN containers are particularly managed by a Container Launch context which is Container Life Cycle (CLC). This record contains a map of environment variables, dependencies stored in remotely accessible storage, security tokens, the payload for Node Manager services, and the command necessary to create the process.

YARN separates HDFS and MapReduce, making the Hadoop environment more suitable for applications that can’t wait for the batch processing jobs to get finished.





**Workflow of an Application in YARN**

1. **Submission of the application by Client**
2. **Container allocation** for **starting Application Manager**
3. **Registering the Application Manager** with Resource Manager
4. Application Manager **asks for containers** from Resource Manager
5. Application Manager notifies **Node Manager** to launch containers
6. Application code **gets executed in the container**
7. Client contacts Resource Manager/Application Manager to monitor the status of the application
8. Application Manager gets disconnected with Resource Manager

**Features of YARN**

* High-degree compatibility: Applications created use the MapReduce framework that can be run easily on YARN.
* Better cluster utilization: YARN **allocates all cluster resources efficiently and dynamically**, which leads to better utilization of Hadoop as compared to the previous version of it.
* Utmost scalability: Whenever there is an **increase in the number of nodes** in the Hadoop cluster, the YARN Resource Manager assures that it meets the user requirements.
* Multi-tenancy: Various engines that access data on the Hadoop cluster **can efficiently work together** all because of YARN as it is a highly versatile technology.