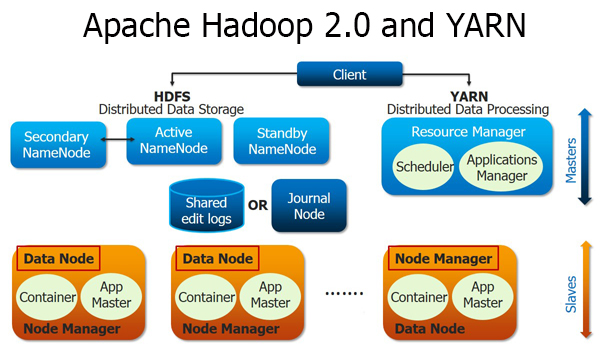
**HADOOP 2.X Architecture**

Hadoop 2.x Architecture has two sets of daemons:

1. HDFS 2.x Daemons: Name Node, Secondary Name node, Data Node

2. MAPREDUCE 2.x Daemons: YARN (Resource Manager, Node Manager)



***HDFS 2.x Daemons:***

HDFS has master slave architecture for job processing.

**NAME NODE:**

* Name node is the master in HDFS 2.x which keeps the directory tree of all the files in the HDFS cluster.
* It tracks where across the cluster files have been located.
* It doesn’t store the data but keeps the tracks of data nodes (slaves).

**SECONDARY NAME NODE:**

* Secondary Name Node works concurrently with the primary Name Node as a helper daemon.
* Secondary Name Node constantly reads all the file systems and metadata from the RAM of the Name Node and writes it into the hard disk or the file system.
* It is responsible for combining the EditLogs with FsImage from the Name Node.
* It downloads the EditLogs from the Name Node at regular intervals and applies to FsImage. The new FsImage is copied back to the Name Node, which is used whenever the Name Node is started the next time.
* Secondary Name Node performs regular checkpoints in HDFS. Therefore, it is also called Checkpoint Node.

**DATA NODE:**

* Data node work like a slave in the HDFS 2.x architecture.
* A single name node can have multiple data nodes.
* It assists the name node in job processing and reports back to name node once job completes.
* Data Nodes transmit periodic heartbeats, block reports and handles commands from the Name Nodes.

***MAPREDUCE 2.x Daemons***

Previously, Hadoop 1.x has Job Tracker and Task Tracker, but it has been replaced by YARN in Hadoop 2.x Architecture.

YARN is responsible for resource management in Hadoop 2.x.

YARN consists of RESOURCE MANAGER and NODE MANAGER.

**RESOURCE MANAGER:**

* This daemon runs on Master node.
* Resource Manager communicates with Node Manager in order to track the resource utilization.
* Resource Manager utilizes two processes named as APPLICATION MANAGER & SCHEDULER for MapReduce task and resource management.
* It is responsible for job submission from client and job scheduling on cluster, monitoring of executing jobs on cluster.
* It is responsible for resource allocation on the slave node.
* SCHEDULER is responsible for resource allocation to various running applications.
* APPLICATION MANAGER is responsible for accepting job submissions, negotiating first CONTAINER for APPLICATION MASTER and provides the service for restarting the APPLICATION MASTER ‘s CONTAINER on failure.

**NODE MANAGER:**

* This daemon runs on Slave node.
* Node manager communicates with Resource Manager to report the resource utilization.
* Node Manager utilizes two processes names as APPLICATION MASTER & CONTAINER for MapReduce Task Scheduling and job execution on the slave node.
* APPLICATION MASTER is created for each application running in the cluster. It provides task level scheduling and monitoring.

CONTAINER represents an allocated resource in the cluster. The allocated CONTAINER resides on a single node always and has a unique ContainerID.It has a specific amount of Resource allocated.

**Execution Sequence of YARN:**

1. A client program submits the application, including the necessary specifications to launch the application-specific Application Master itself
2. The Resource Manager assumes the responsibility to negotiate a specified container in which to start the Application Master and then launches the Application Master.
3. The Application Master, on boot-up, registers with the Resource Manager – the registration allows the client program to query the Resource Manager for details, which allow it to directly communicate with its own Application Master.
4. During normal operation the Application Master negotiates appropriate resource containers via the resource-request protocol.
5. On successful container allocations, the Application Master launches the container by providing the container launch specification to the Node Manager. The launch specification, typically, includes the necessary information to allow the container to communicate with the Application Master itself.
6. The application code executing within the container then provides necessary information (progress, status etc.) to its Application Master via an application-specific protocol.
7. During the application execution, the client that submitted the program communicates directly with the Application Master to get status, progress updates etc. via an application-specific protocol.
8. Once the application is complete, and all necessary work has been finished, the Application Master deregisters with the Resource Manager and shuts down, allowing its own container to be repurposed.