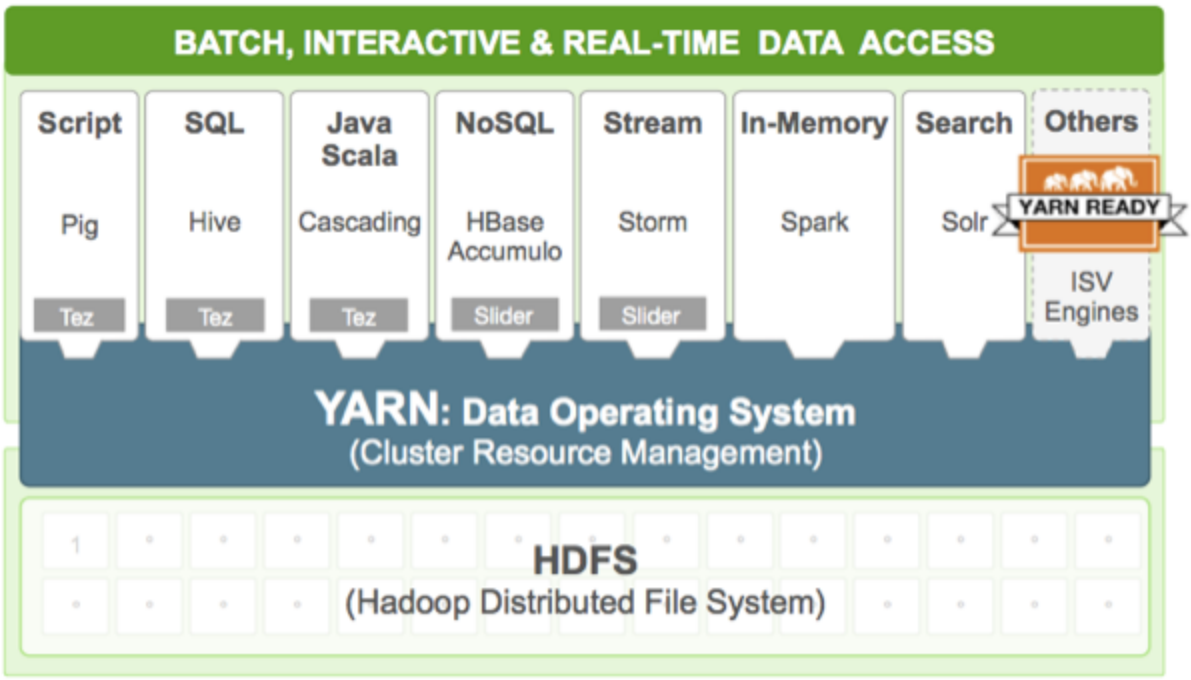
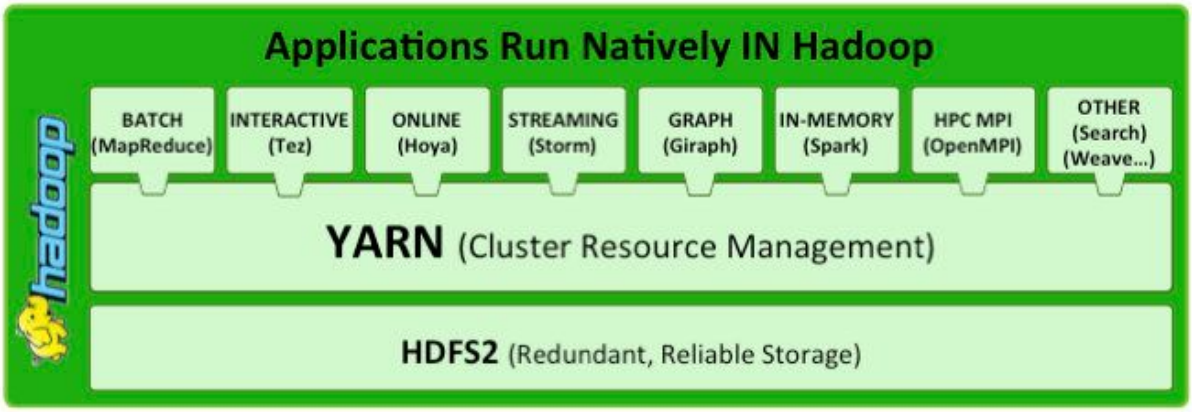
Hadoop Yarn

YARN - Yet Another Resource Negotiator





YARN is the architectural center of Hadoop that allows multiple data processing engines such as interactive SQL, real time streaming, data science and batch processing to handle data stored in a single platform, unlocking an entirely new approach to analytics.

Multi-tenancy	- YARN allows multiple access engines (either open-source or proprietary) to use Hadoop as the common standard for batch, interactive and real-time engines that can simultaneously access the same data set.

Cluster utilization	- YARN's dynamic allocation of cluster resources improves utilization over more static MapReduce rules used in early versions of Hadoop
Scalability	- YARN's ResourceManager focuses exclusively on scheduling and keeps pace as clusters expand to thousands of nodes managing petabytes of data.
Compatibility	- Existing MapReduce applications developed for Hadoop 1 can run YARN without any disruption to existing processes that already work

Features	Hadoop 1.x	Hadoop 2.0
HDFS Federation	One Namenode and a Namespace	Mutiple Namenode and Namespaces
Namenode Availability	Not present	High Availability
YARN - processing control and multi-tenancy	JobTracker and TaskTracker	Resource Manager, Node Manager, App Master, Capacity Scheduler

How YARN works

two major responsibilities of the JobTracker/TaskTracker into separate entities:

- a global **ResourceManager**
- a per-application **ApplicationMaster**
- a per-node slave **NodeManager**
- a per-application **Container** running on a NodeManager

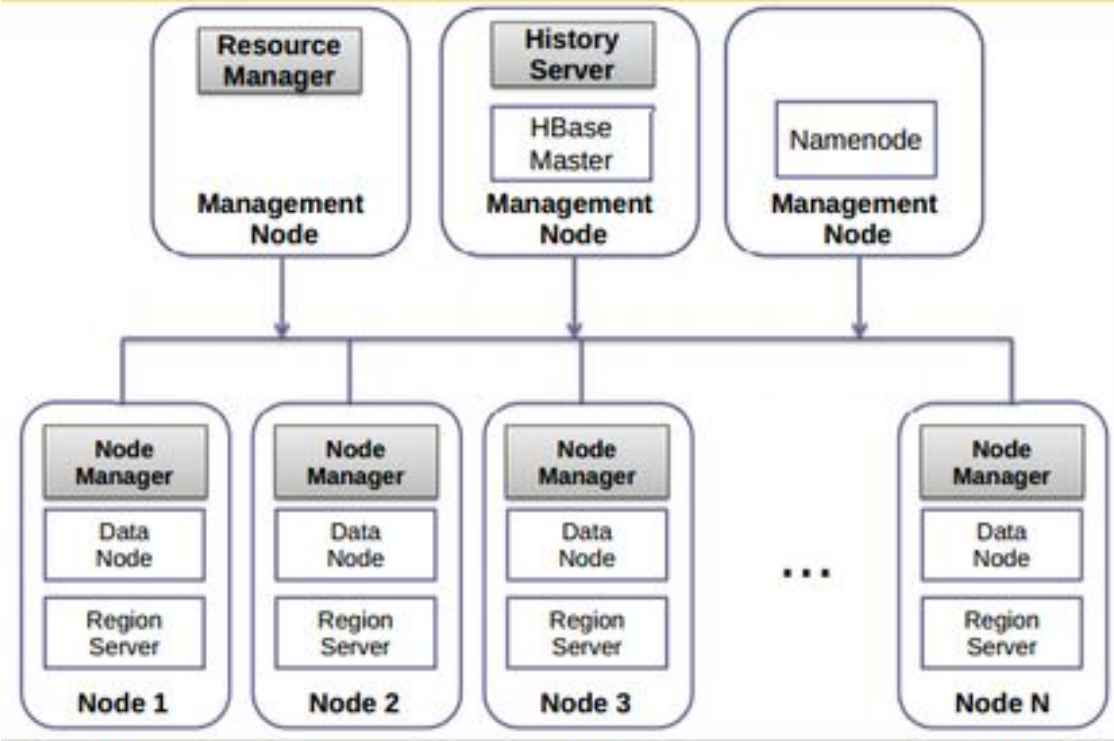
* The **ResourceManager** is the ultimate authority that arbitrates resources among all applications in the system.

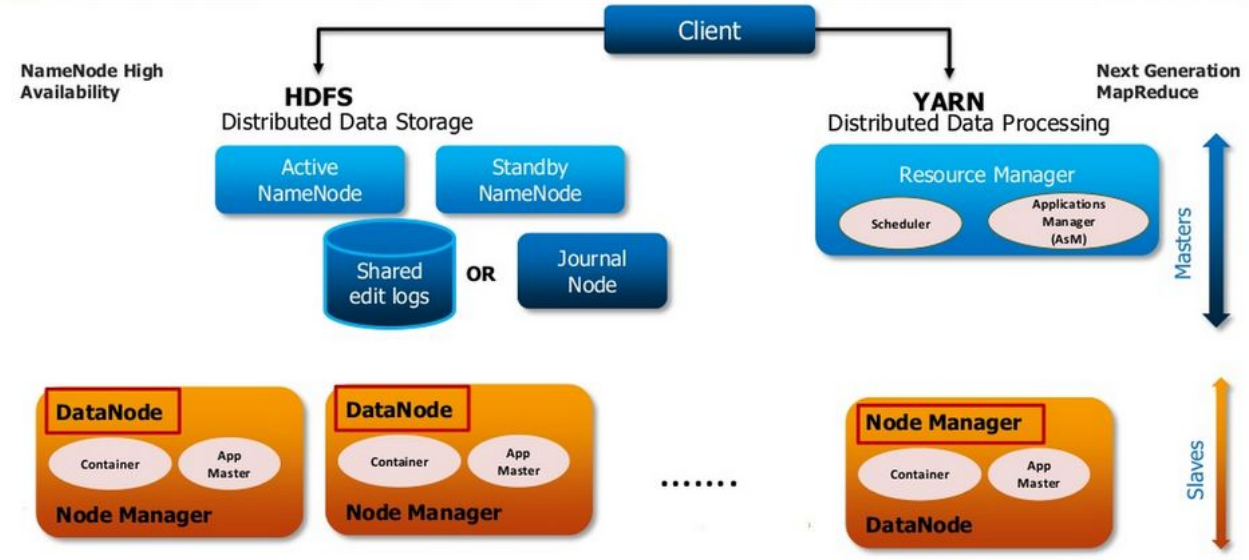
The ResourceManager has a scheduler, which is responsible for allocating resources to the various applications running in the cluster, according to constraints such as queue capacities and user limits. The scheduler schedules based on the resource requirements of each application.

* The **ApplicationMaster** is a framework-specific entity that negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor the component tasks.

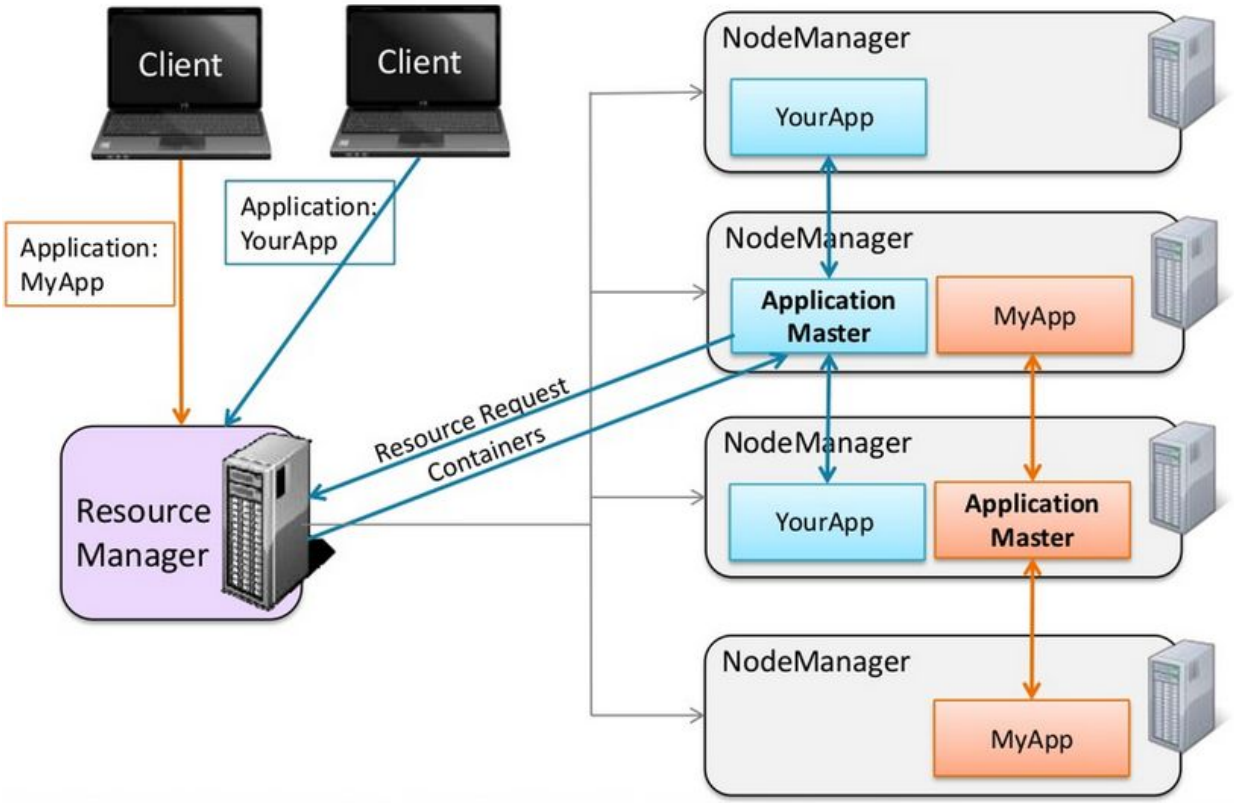
* Each ApplicationMaster has responsibility for negotiating appropriate resource containers from the scheduler, tracking their status, and monitoring their progress. From the system perspective, the ApplicationMaster runs as a normal container.

* The **NodeManager** is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager.



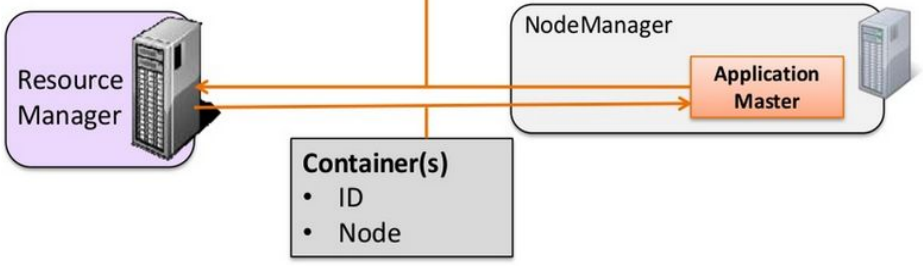


YARN Cluster: Running an Application



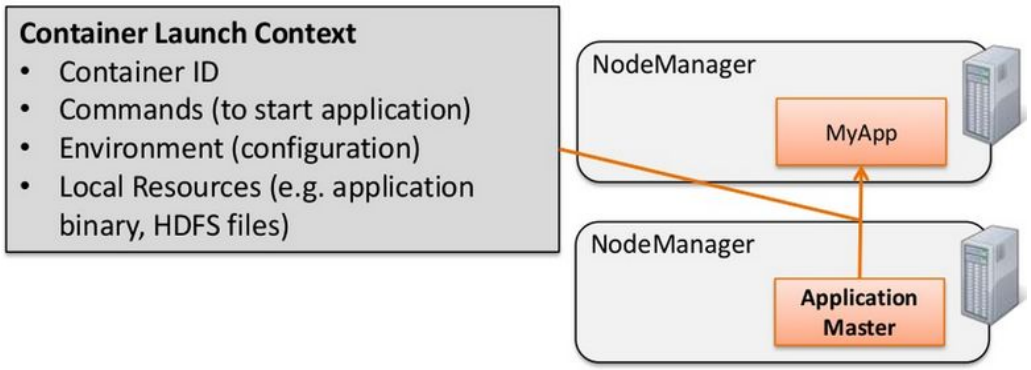
Resource Requests

Resource Request
- Resource name (hostname, rackname or *)
- Priority (within this application, not between applications)
- Resource requirements
  - memory (MB)
  - CPU (# of cores)
  - more to come, e.g. disk and network I/O, GPUs, etc.
- Number of containers



Container(s)
- ID
- Node

Launch Container



YARN solutions

- No slots
- Support MR and non-MR running on the same cluster
- Most Job Tracker functions moved to Application Master - One cluster can have many Application Masters

Resource Manager

- Run on Master Node
- Global resource scheduler
- Arbitrates system resources between competing applications
- Manages Nodes
    - Tracks heartbeats from NodeManger
- Manages Containers
    - Handles AM requests for resources
    - de-allocates containers when they expires or the application complets
- Manage ApplicationMasters
    - Creates a containers for AMs and tracks heatbeats
- Manage security

Node Manager

- Run on slave nodes
- communicates with RM
    - Registers and provides info on node resources
    - Sends heartbeats and container status
- Manages processes in containers
    - Launches AMs on requests from the RM
    - Launches application process on request from AM
    - Monitors usage by containers; kills run-way processes
- Provides logging services to applications
    - aggregates logs for an application and saves them to HDFS
- Runs auxiliary services

Containers

- Created by the RM upon the request
- Allocate a certain amount of resources (memory, cpu) on a slave node
- Applications run in one or more containers

Application Master(AM)

- One per application
- Framework / application specific
- Runs in a container
- Requests more container to run applications tasks

YARN Schedulers

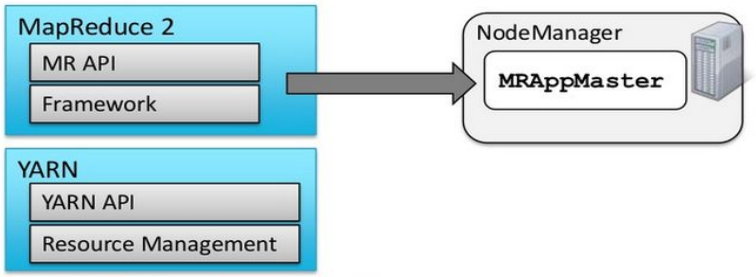
- Pluggable in Resource Manager
- YARN includes two schedulers
    - Capacity schedules
    - Fair schedulers

- How are these different than MR1 scheduler ?
    - Support any YARN application, not just MR
    - No more slots : tasks are allocated based on resources
    - Fairscheduler : pools are now called queues
- Hierarchical queues
    - Queues can contain sub-queues
    - Sub-queues share the resources assigned to queues

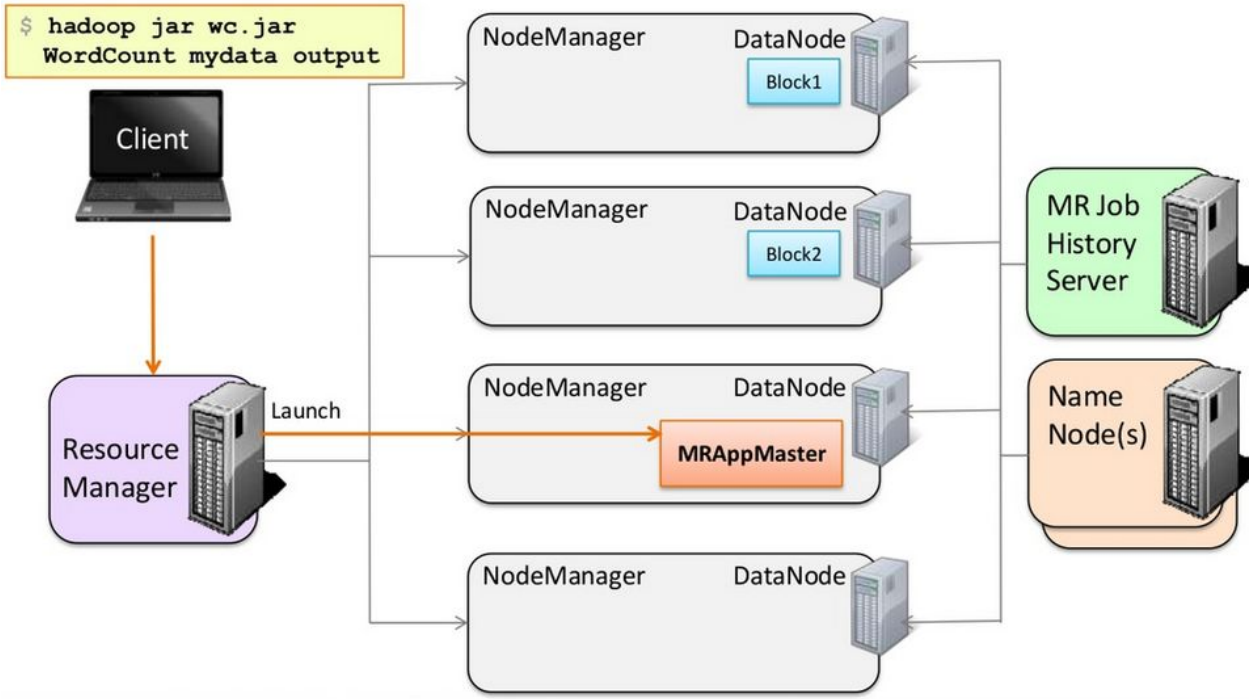
Ref : http://www.slideshare.net/cloudera/introduction-to-yarn-and-mapreduce-2

YARN and MapReduce

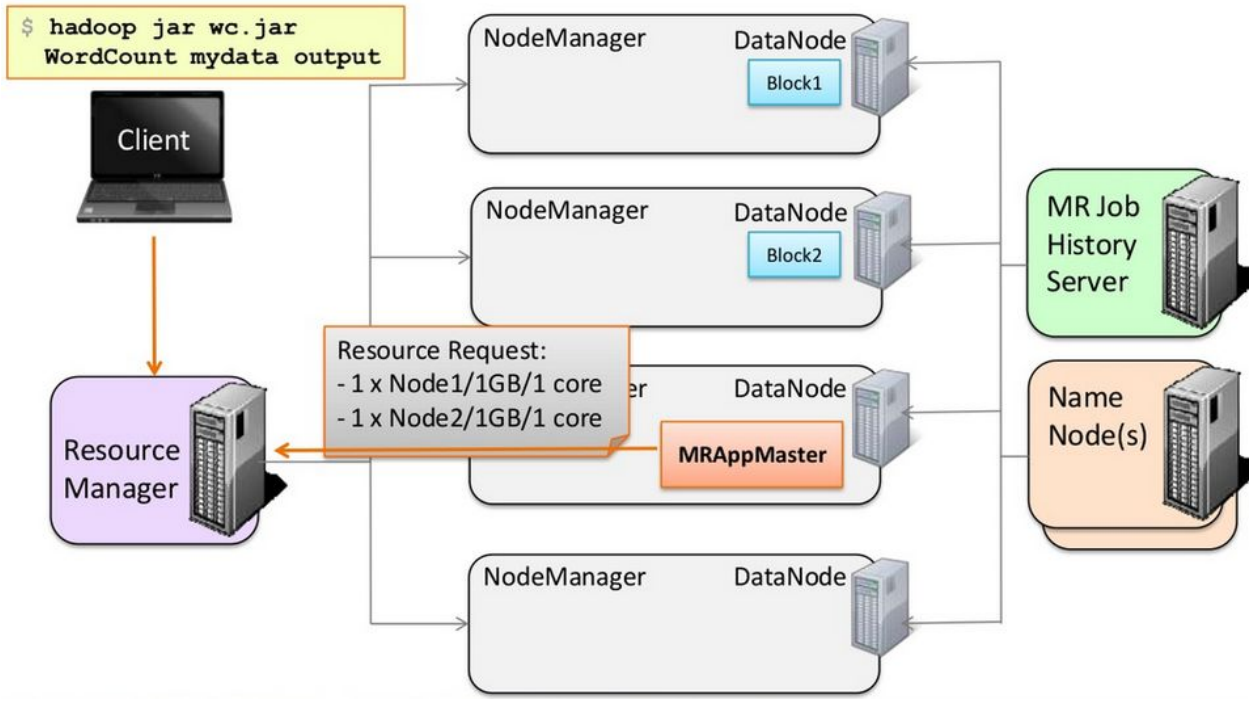
- **YARN does not know or care what kind of application it is running**
  - Could be MR or something else (e.g. Impala)
- **MR2 uses YARN**
  - Hadoop includes a MapReduce ApplicationMaster (MRAppMaster) to manage MR jobs
  - Each MapReduce job is an a new instance of an application



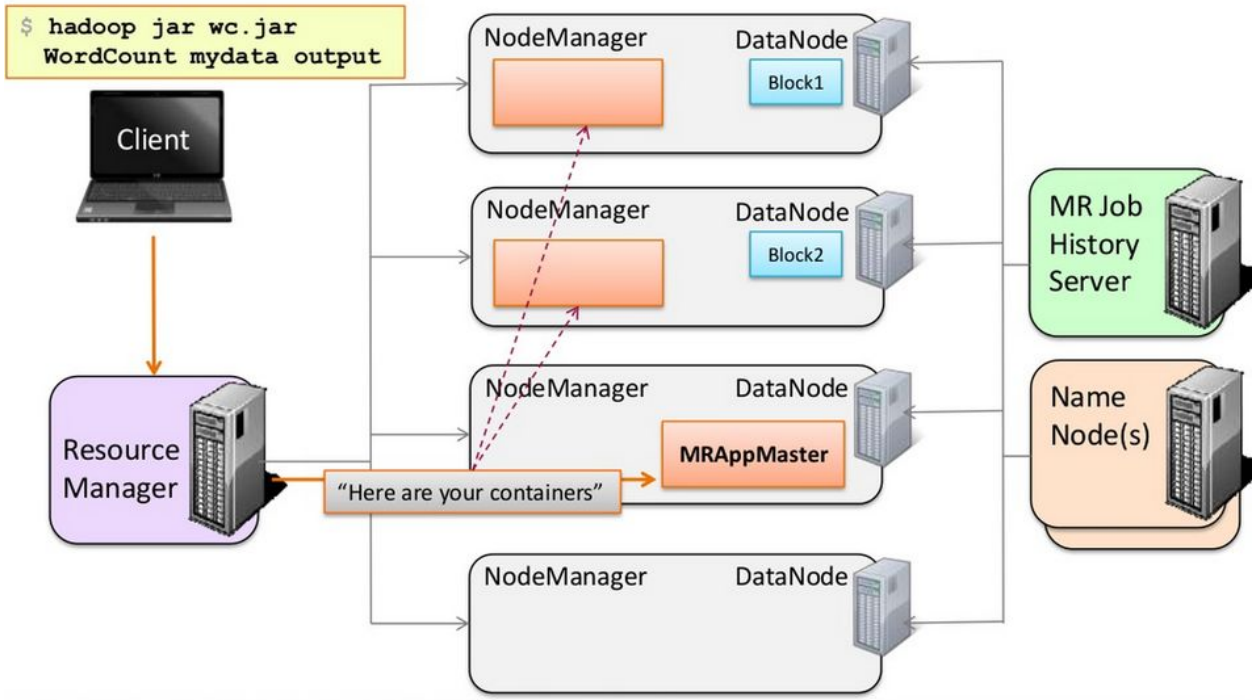
Running a MapReduce Application in MRv2



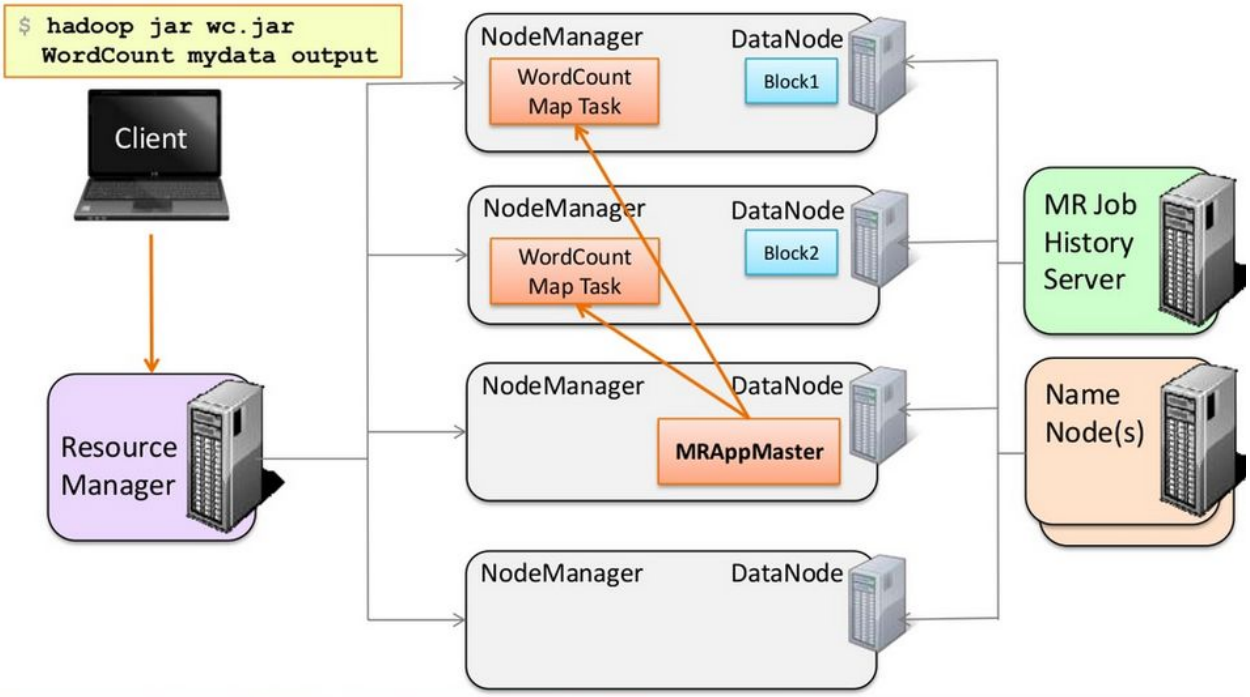
Running a MapReduce Application in MRv2



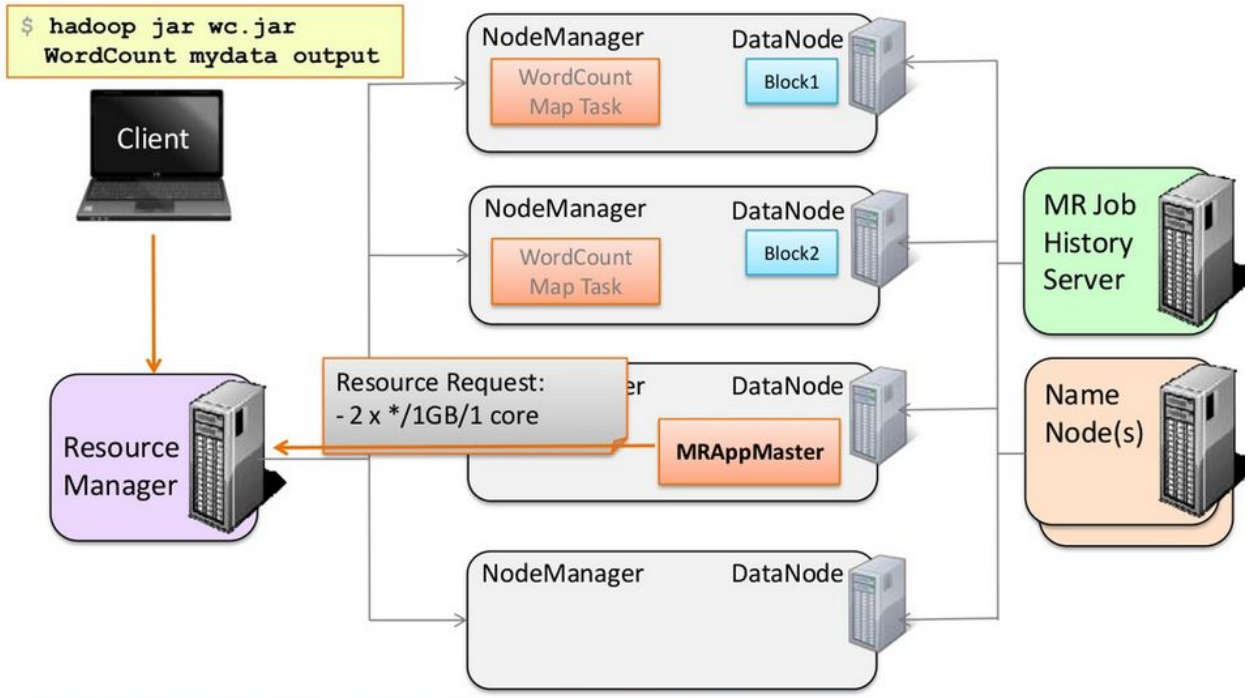
Running a MapReduce Application in MRv2



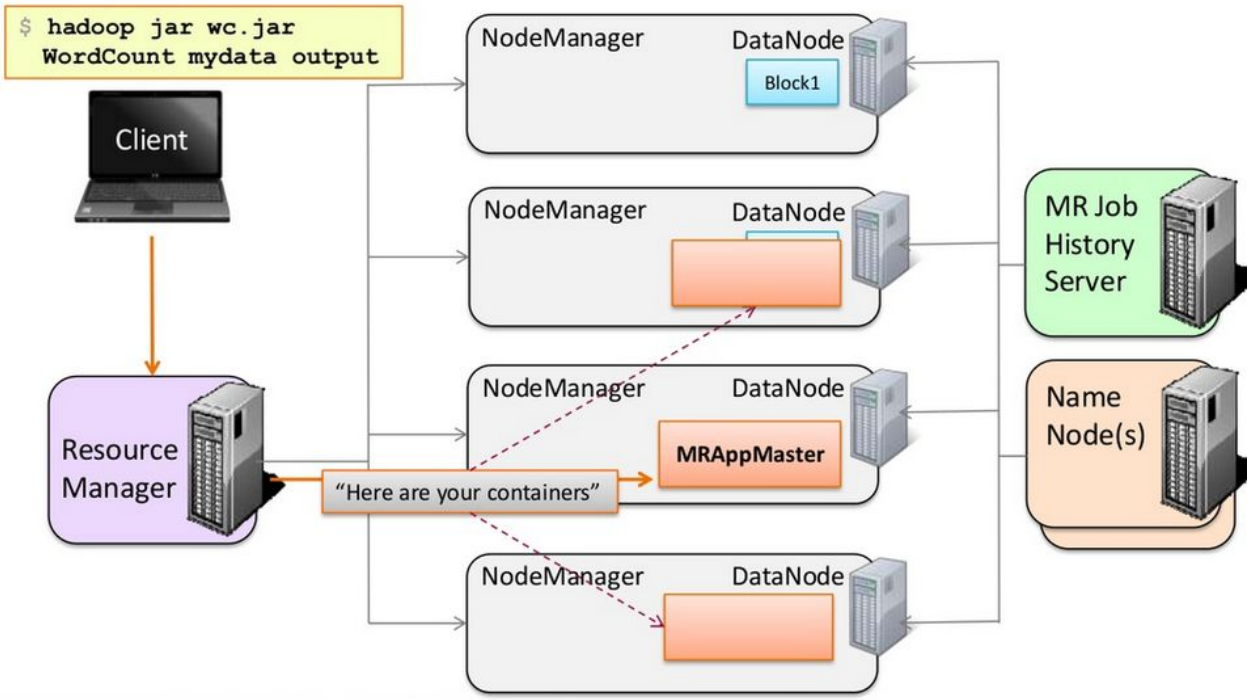
Running a MapReduce Application in MRv2



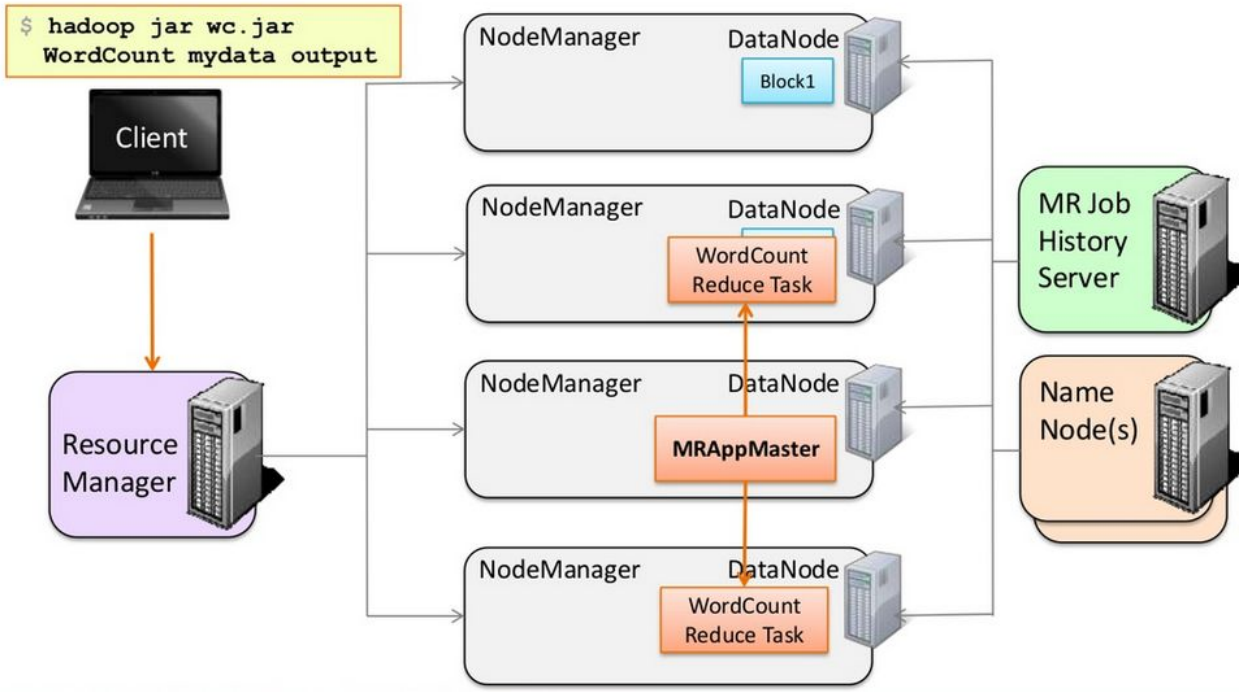
Running a MapReduce Application in MRv2



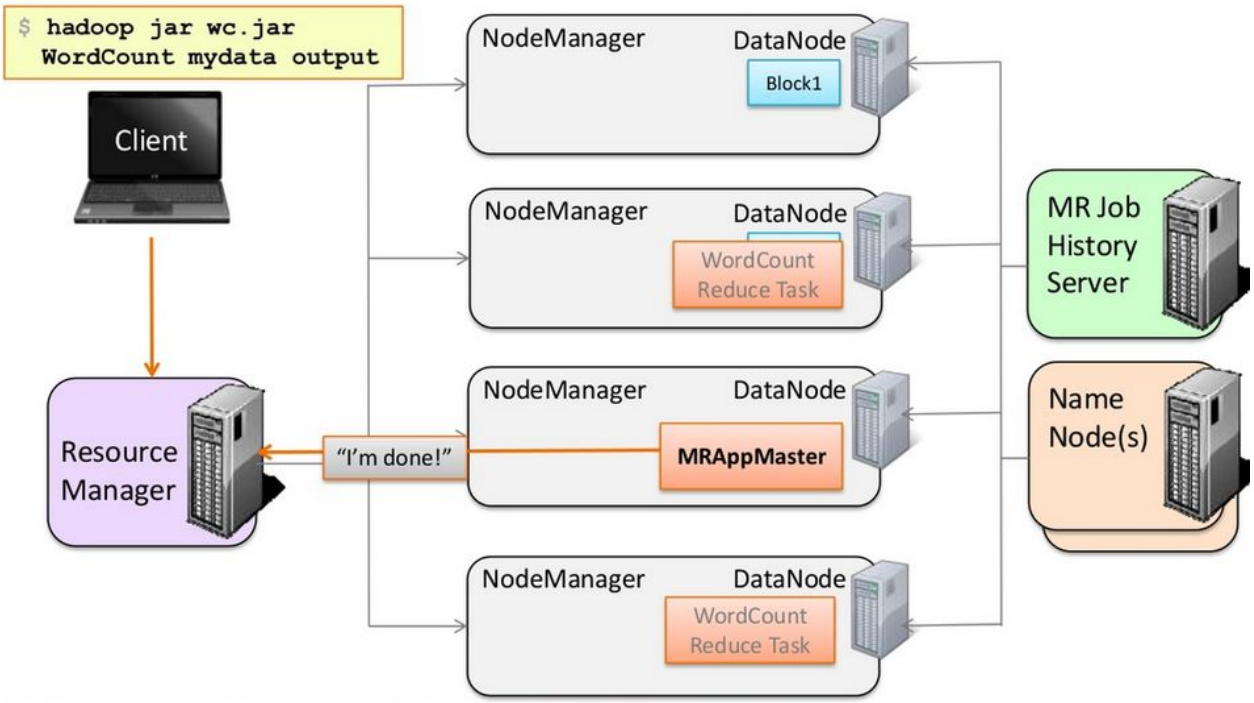
Running a MapReduce Application in MRv2



Running a MapReduce Application in MRv2

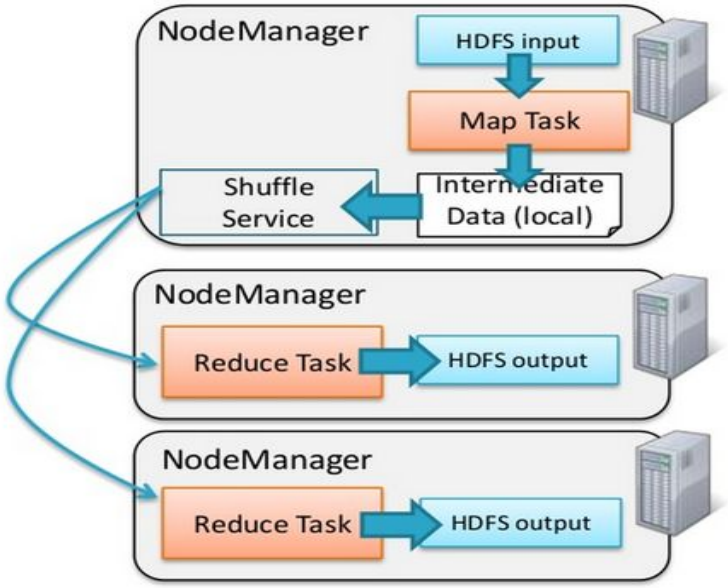


Running a MapReduce Application in MRv2



The MapReduce Framework on YARN

- **In YARN, Shuffle is run as an auxiliary service**
  - Runs in the NodeManager JVM as a persistent service



Fault Tolerance

- **Any of the following can fail**
  - Task (Container) – Handled just like in MRv1
    - MRAppMaster will re-attempt tasks that complete with exceptions or stop responding (4 times by default)
    - Applications with too many failed tasks are considered failed
  - Application Master
    - If application fails or if AM stops sending heartbeats, RM will re-attempt the whole application (2 times by default)
    - MRAppMaster optional setting: Job recovery
      - if false, all tasks will re-run
      - if true, MRAppMaster retrieves state of tasks when it restarts; only incomplete tasks will be re-run

Fault Tolerance

- **Any of the following can fail**
  - NodeManager
    - If NM stops sending heartbeats to RM, it is removed from list of active nodes
    - Tasks on the node will be treated as failed by MRAppMaster
    - If the App Master node fails, it will be treated as a failed application
  - ResourceManager
    - No applications or tasks can be launched if RM is unavailable
    - Can be configured with High Availability