APACHE HADOOP YARN

YARN:

YARN stands for “Yet-Another-Resource-Negotiator”. It is a new framework that facilitates writing arbitrary distributed processing frameworks and applications. YARN can run applications that do not follow the Map-reduce model.

MR2:

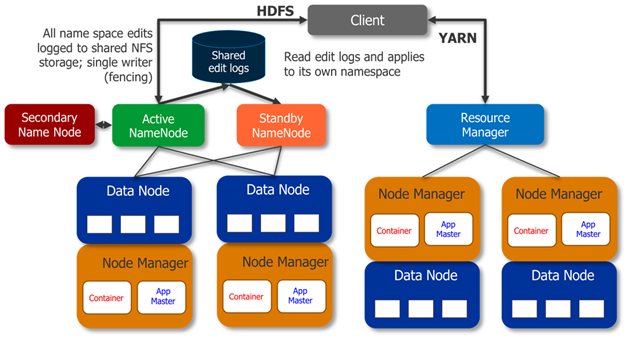
MR2 is one such distributed application that runs the Map-Reduce framework on top of YARN. It is a more isolated and scalable model than the MR1 system where a singular Job-Tracker does all the resource management, scheduling and task monitoring work.

Architecture:

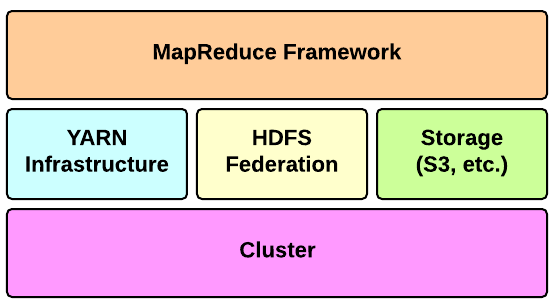
The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs.

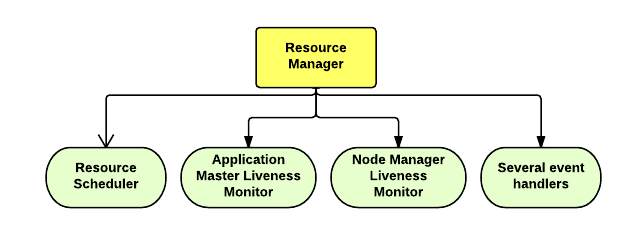
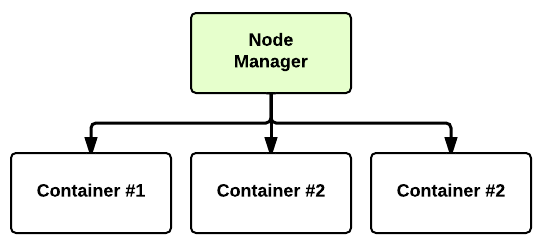
The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.



[**Apache Hadoop**](http://hadoop.apache.org/) is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. There are mainly five building blocks inside this runtime envinroment (from bottom to top):



* the **cluster** is the set of host machines (**nodes**). Nodes may be partitioned in **racks**. This is the hardware part of the infrastructure.
* the **YARN Infrastructure** (Yet Another Resource Negotiator) is the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions. Two important elements are:
  + the **Resource Manager** (one per cluster) is the master. It knows where the slaves are located (Rack Awareness) and how many resources they have. It runs several services, the most important is the **Resource Scheduler** which decides how to assign the resources
  + the **Node Manager** (many per cluster) is the slave of the infrastructure. When it starts, it announces himself to the Resource Manager. Periodically, it sends an heartbeat to the Resource Manager. Each Node Manager offers some resources to the cluster. Its resource capacity is the amount of memory and the number of vcores. At run-time, the Resource Scheduler will decide how to use this capacity: a **Container** is a fraction of the NM capacity and it is used by the client for running a program.
* the **HDFS Federation** is the framework responsible for providing permanent, reliable and distributed storage. This is typically used for storing inputs and output (but not intermediate ones).
* other alternative storage solutions. For instance, Amazon uses the Simple Storage Service (S3).
* the **MapReduce Framework** is the software layer implementing the **[MapReduce paradigm](http://en.wikipedia.org/wiki/MapReduce.html)**.

The YARN infrastructure and the HDFS federation are completely decoupled and independent: the first one provides resources for running an application while the second one provides storage. The MapReduce framework is only one of many possible framework which runs on top of YARN (although currently is the only one implemented).

Limitations of MR1:

**Issue of Availability:**

Hadoop 1.0 Architecture had only one single point of availability i.e. the Job Tracker, so in case if the Job Tracker fails then all the jobs will have to restart.

**Issue of Scalability:**

The Job Tracker runs on a single machine performing various tasks such as Monitoring, Job Scheduling, Task Scheduling and Resource Management. In spite of the presence of several machines (Data Nodes), they were not being utilized in an efficient manner, thereby limiting the scalability of the system.

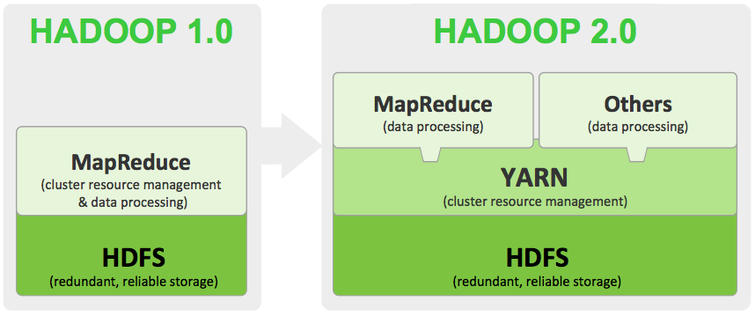
**Cascading Failure Issue:**

In case of Hadoop MapReduce when the number of nodes is greater than 4000 in a cluster, some kind of fickleness is observed. The most common kind of failure that was observed is the cascading failure which in turn could cause the overall cluster to deteriorate when trying to overload the nodes or replicate data via network flooding.

**Multi-Tenancy Issue:**

The major issue with Hadoop MapReduce that paved way for the advent of Hadoop YARN was multi-tenancy. With the increase in the size of clusters in Hadoop systems, the clusters can be employed for a wide range of models.

Hadoop MapReduce devotes the nodes of the cluster in the Hadoop System so that they can be repurposed for other big data workloads and applications. Nevertheless, with Big Data and Hadoop, ruling the data processing applications for cloud deployments, the number of nodes in the cluster is likely to increase and this issue is addressed with a switch from 1.x to 2.x.



The Hadoop 1.0 or the so called MRv1 mainly consists of 3 important components namely:

##### 1)**Resource Management:**

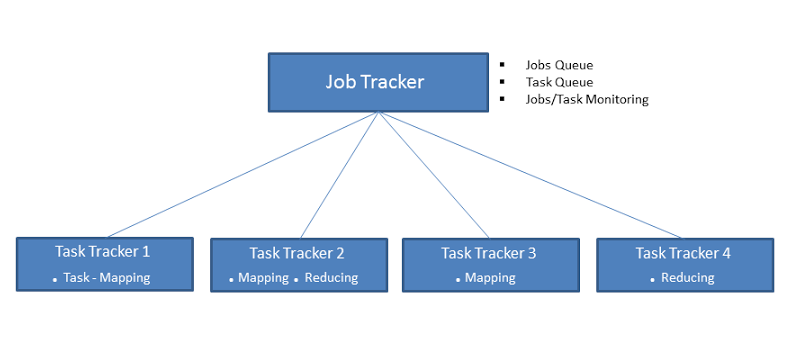
This is an infrastructure component that takes care of monitoring the nodes, allocating the resources and scheduling various jobs.

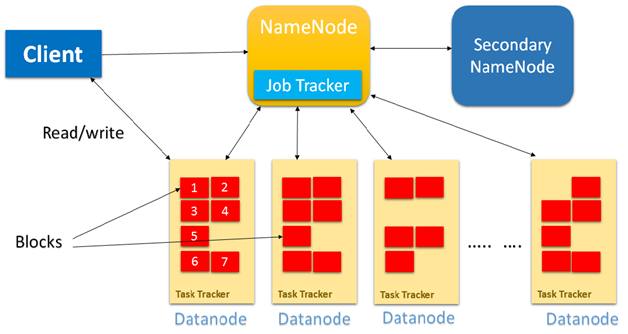
##### **2) Application Programming Interface (API):**

This component is for the users to program various MapReduce applications.

##### **3) Framework:**

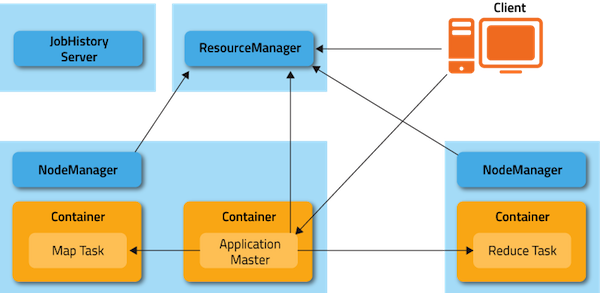
This component is for all the runtime services such as Shuffling, Sorting and executing Map and Reduce processes.

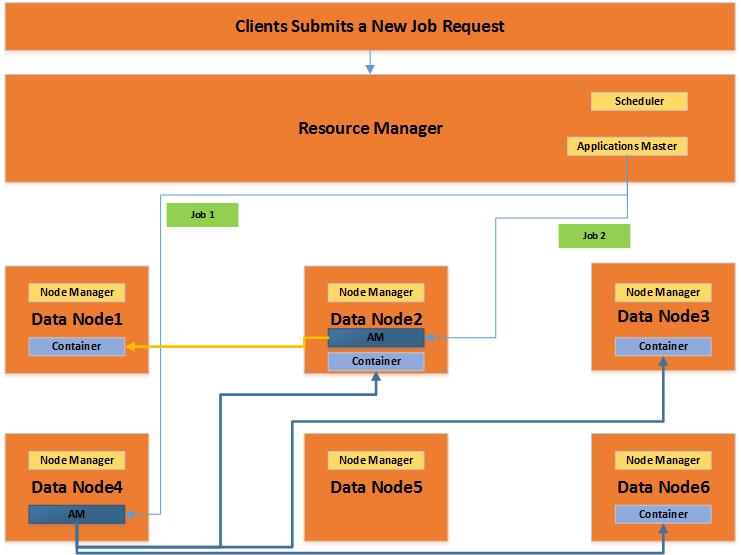




The major difference with Hadoop 2.0 is that, in this next generation of Hadoop the cluster resource management capabilities are moved into YARN.

YARN Solution:





In Hadoop 2.0, the Job Tracker in YARN mainly depends on 3 important components

##### **1. Resource Manager Component:**

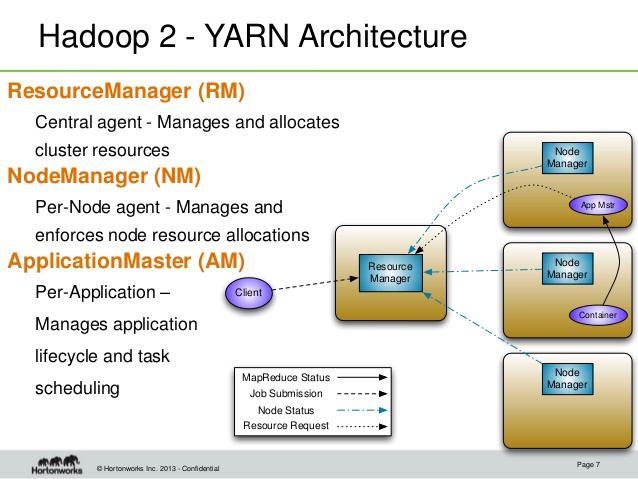
This component is considered as the negotiator of all the resources in the cluster. Resource Manager is further categorized into an Application Manager that will manage all the user jobs with the cluster and a pluggable scheduler. This is a relentless YARN service that is designed for receiving and running the applications on the Hadoop Cluster. In Hadoop 2.0, a MapReduce job will be considered as an application.

##### **2. Node Manager Component:**

This is the job history server component of YARN which will furnish the information about all the completed jobs. The NM keeps a track of all the users’ jobs and their workflow on any particular given node.

##### **3. Application Master Component (User Job Life Cycle Manager):**

This is the component where the job actually resides and the Application Master component is responsible for managing each and every Map Reduce job and is concluded once the job completes processing.



**RM-Resource Manager**

1.It is the global resource scheduler

2.It runs on the Master Node of the Cluster

3.It is responsible for negotiating the resources of the system amongst the competing applications.

4.It keeps a track on the heartbeats from the Node Manager

**NM-Node Manager**

1.Node Manager communicates with the resource manager.

2.It runs on the Slave Nodes of the Cluster

**AM-Application Master**

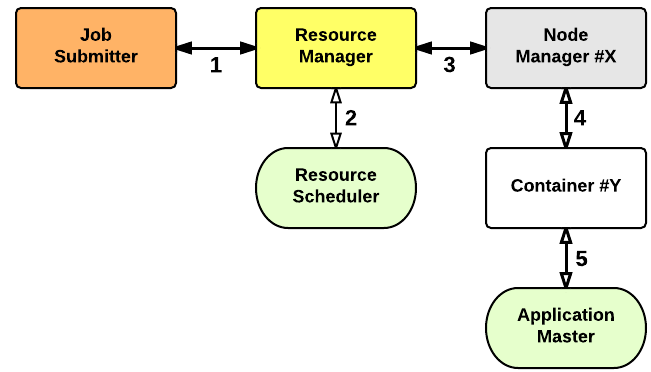
1.There is one AM per application which is application specific or framework specific.

2.The AM runs in Containers that are created by the resource manager on request.

YARN Application StartUp:

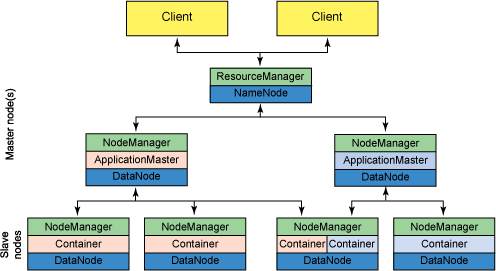
The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the **Application Master**



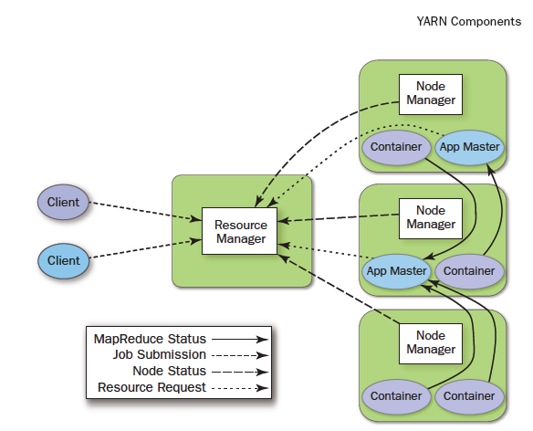
The Application Master is responsible for the execution of a single application. It asks for containers to the Resource Scheduler (Resource Manager) and executes specific programs (e.g., the main of a Java class) on the obtained containers. The Application Master knows the application logic and thus it is framework-specific. The MapReduce framework provides its own implementation of an Application Master.

The Resource Manager is a single point of failure in YARN. Using Application Masters, YARN is spreading over the cluster the metadata related to running applications. This reduces the load of the Resource Manager and makes it fast recoverable.



MRv2 The Job Submission Process:

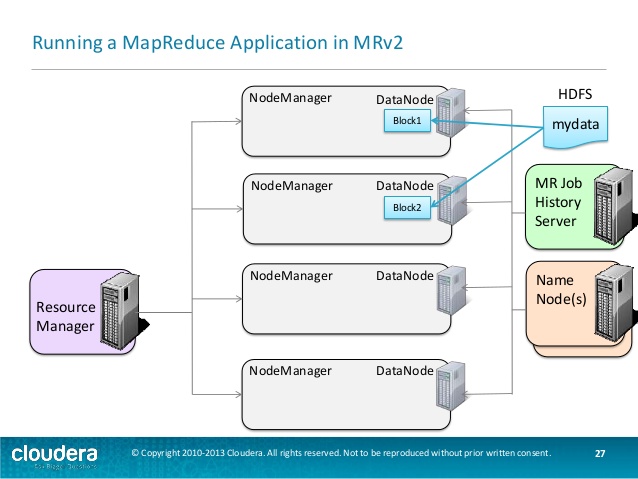
1. A client program *submits* the application, including the necessary specifications to *launch the application-specific ApplicationMaster* itself.
2. The ResourceManager assumes the responsibility to negotiate a specified container in which to start the ApplicationMaster and then *launches* the ApplicationMaster.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

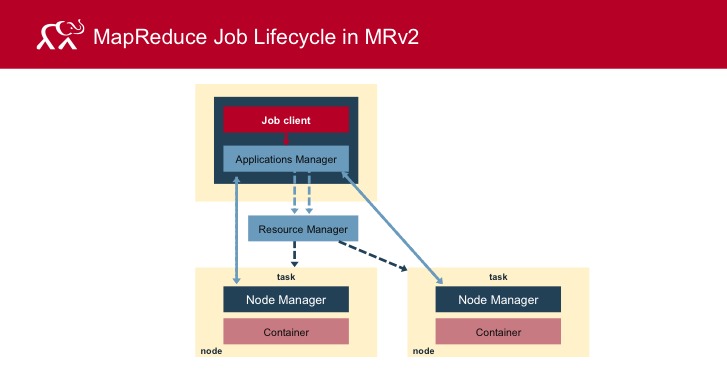


Key Differences Between MR1 and MR2:

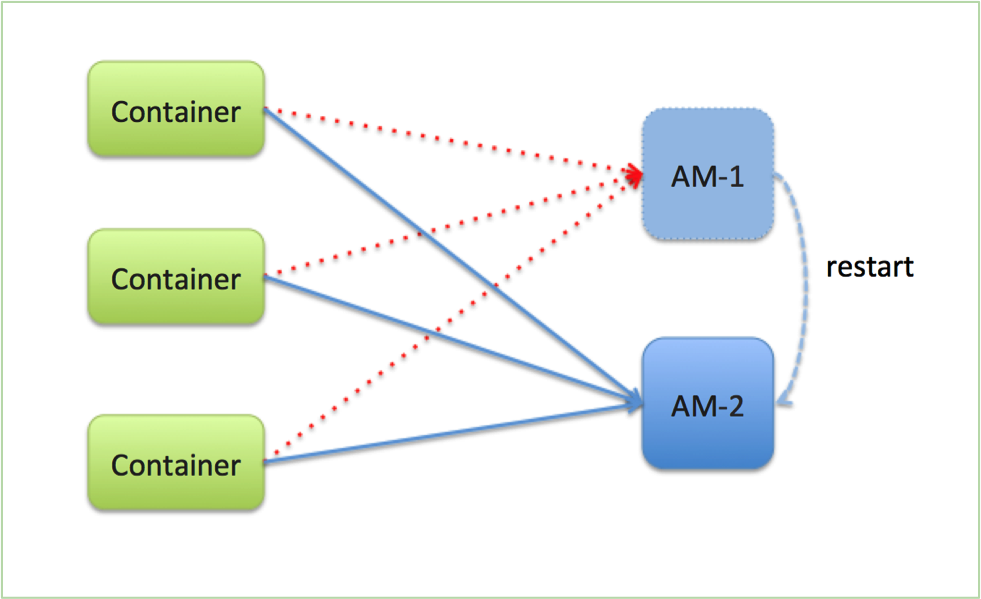
* Old map-reduce version known as a MR1 and new map-reduce version know as MR2.
* Job Tracker and Task Tracker no more with Map-reduce new version.
* The new model is more isolated and scalable as compared to the earlier MR1 system. MapReduce perform data processing via YARN. Other tools can also perform data processing via YARN. Hence Yarn execution model is more generic than earlier MapReduce model.
* MR1 was not able to do so. It would only run MapReduce applications.

MR Job Lifecycle on YARN cluster:

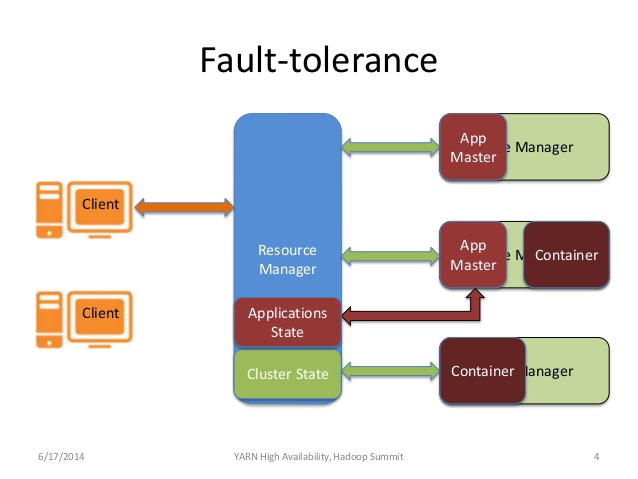


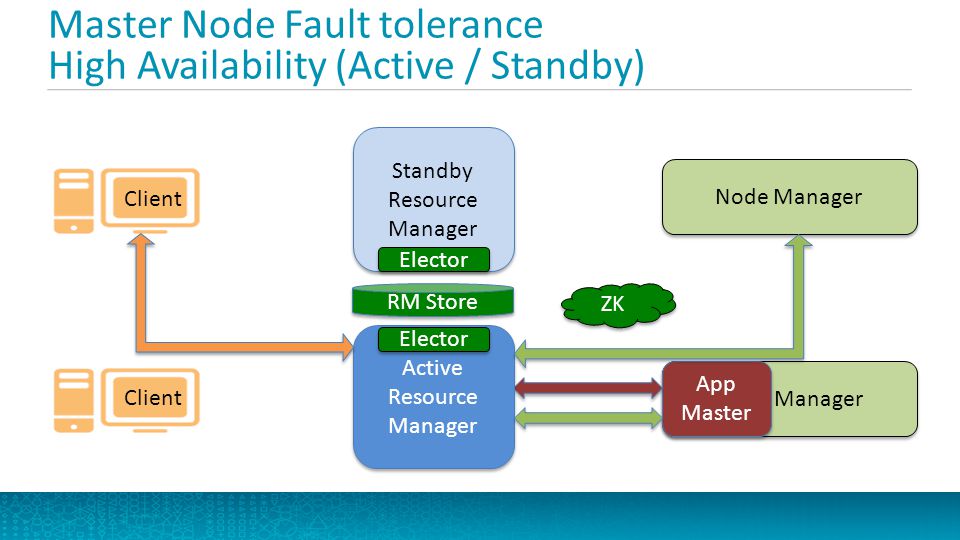


Fault Tolerance:



* **Task(Container)**
* MRAppMaster will re-attempt that complete with exceptions or stop responding(4 times by default)
* Application 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 default)
* MRAppMaster optional setting: job recovery
* If false all task will re-run
* If true MRAppMaster retrieves state of tasks when it restarts, only incomplete tasks will be re-run
* **Resource Manager**
* No application and Task will be launch if RM is not available
* RM should be configure with the HA
* **Node Manager**
* If NM stop to send heartbeats to RM, it will remove NM from the active node list Task on the node treated as a fail by AppMaster
* AppMaster node fail it will be treated as failed application



****