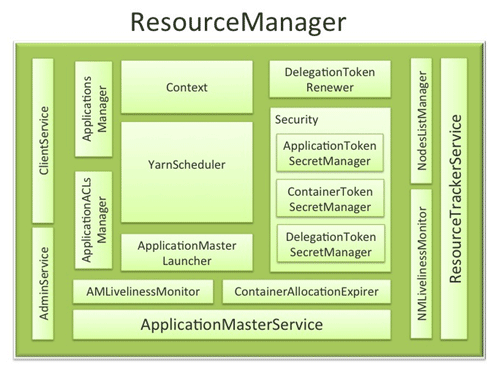
**Apache Hadoop YARN – ResourceManager**

As previously described, **ResourceManager (RM)** is the master that arbitrates all the available cluster resources and thus helps manage the distributed applications running on the YARN system. It works together with the per-node **NodeManagers (NMs)** and the per-application **ApplicationMasters (AMs)**.

1. **NodeManagers** take instructions from the ResourceManager and manage resources available on a single node.
2. **ApplicationMasters** are responsible for negotiating resources with the ResourceManager and for working with the NodeManagers to start the containers.

[](http://hortonworks.com/wp-content/uploads/2012/08/resource_manager.png)

**ResourceManager Components**

The ResourceManager has the following components (see the figure above):

1. **Components interfacing RM to the clients:**
   * **ClientService**: The client interface to the Resource Manager. This component handles all the RPC interfaces to the RM from the clients including operations like application submission, application termination, obtaining queue information, cluster statistics etc.
   * **AdminService**: To make sure that admin requests don’t get starved due to the normal users’ requests and to give the operators’ commands the higher priority, all the admin operations like refreshing node-list, the queues’ configuration etc. are served via this separate interface.
2. **Components connecting RM to the nodes:**
   * **ResourceTrackerService**: This is the component that responds to RPCs from all the nodes. It is responsible for registration of new nodes, rejecting requests from any invalid/decommissioned nodes, obtain node-heartbeats and forward them over to the YarnScheduler. It works closely with NMLivelinessMonitor and NodesListManager described below.
   * **NMLivelinessMonitor**: To keep track of live nodes and specifically note down the dead nodes, this component keeps track of each node’s its last heartbeat time. Any node that doesn’t heartbeat within a configured interval of time, by default 10 minutes, is deemed dead and is expired by the RM. All the containers currently running on an expired node are marked as dead and no new containers are scheduling on such node.
   * **NodesListManager**: A collection of valid and excluded nodes. Responsible for reading the host configuration files specified via yarn.resourcemanager.nodes.include-path and yarn.resourcemanager.nodes.exclude-path and seeding the initial list of nodes based on those files. Also keeps track of nodes that are decommissioned as time progresses.
3. **Components interacting with the per-application AMs:**
   * **ApplicationMasterService**: This is the component that responds to RPCs from all the AMs. It is responsible for registration of new AMs, termination/unregister-requests from any finishing AMs, obtaining container-allocation & deallocation requests from all running AMs and forward them over to the YarnScheduler. This works closely with AMLivelinessMonitor described below.
   * **AMLivelinessMonitor**: To help manage the list of live AMs and dead/non-responding AMs, this component keeps track of each AM and its last heartbeat time. Any AM that doesn’t heartbeat within a configured interval of time, by default 10 minutes, is deemed dead and is expired by the RM. All the containers currently running/allocated to an AM that gets expired are marked as dead. RM schedules the same AM to run on a new container, allowing up to a maximum of 4 such attempts by default.
4. **The core of the ResourceManager – the scheduler and related components:**
   * **ApplicationsManager**: Responsible for maintaining a collection of submitted applications. Also keeps a cache of completed applications so as to serve users’ requests via web UI or command line long after the applications in question finished.
   * **ApplicationACLsManager**: RM needs to gate the user facing APIs like the client and admin requests to be accessible only to authorized users. This component maintains the ACLs lists per application and enforces them whenever an request like killing an application, viewing an application status is received.
   * **ApplicationMasterLauncher**: Maintains a thread-pool to launch AMs of newly submitted applications as well as applications whose previous AM attempts exited due to some reason. Also responsible for cleaning up the AM when an application has finished normally or forcefully terminated.
   * **YarnScheduler**: The Scheduler is responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc. It performs its scheduling function based on the resource requirements of the applications such as memory, CPU, disk, network etc. Currently, only memory is supported and support for CPU is close to completion.
   * **ContainerAllocationExpirer**: This component is in charge of ensuring that all allocated containers are used by AMs and subsequently launched on the correspond NMs. AMs run as untrusted user code and can potentially hold on to allocations without using them, and as such can cause cluster under-utilization. To address this, ContainerAllocationExpirer maintains the list of allocated containers that are still not used on the corresponding NMs. For any container, if the corresponding NM doesn’t report to the RM that the container has started running within a configured interval of time, by default 10 minutes, the container is deemed as dead and is expired by the RM.
5. **TokenSecretManagers (for security):**ResourceManager has a collection of SecretManagers which are charged with managing tokens, secret-keys that are used to authenticate/authorize requests on various RPC interfaces. A future post on YARN security will cover a more detailed descriptions of the tokens, secret-keys and the secret-managers but a brief summary follows:
   * **ApplicationTokenSecretManager**: To avoid arbitrary processes from sending RM scheduling requests, RM uses the per-application tokens called ApplicationTokens. This component saves each token locally in memory till application finishes and uses it to authenticate any request coming from a valid AM process.
   * **ContainerTokenSecretManager**: SecretManager for ContainerTokens that are special tokens issued by RM to an AM for a container on a specific node. ContainerTokens are used by AMs to create a connection to the corresponding NM where the container is allocated. This component is RM-specific, keeps track of the underlying master and secret-keys and rolls the keys every so often.
   * **RMDelegationTokenSecretManager**: A ResourceManager specific delegation-token secret-manager. It is responsible for generating delegation tokens to clients which can be passed on to unauthenticated processes that wish to be able to talk to RM.
6. **DelegationTokenRenewer:** In secure mode, RM is Kerberos authenticated and so provides the service of renewing file-system tokens on behalf of the applications. This component renews tokens of submitted applications as long as the application runs and till the tokens can no longer be renewed.
7. **Conclusion**
8. In YARN, the ResourceManager is primarily limited to scheduling i.e. only arbitrating available resources in the system among the competing applications and not concerning itself with per-application state management. Because of this clear separation of responsibilities coupled with the modularity described above, and with the powerful scheduler API discussed in the previous post, RM is able to address the most important design requirements – scalability, support for alternate programming paradigms.
9. To allow for different policy constraints, the scheduler described above in the RM is pluggable and allows for different algorithms. In a future post of this series, we will dig deeper into various features of CapacityScheduler that schedules containers based on capacity guarantees and queues.