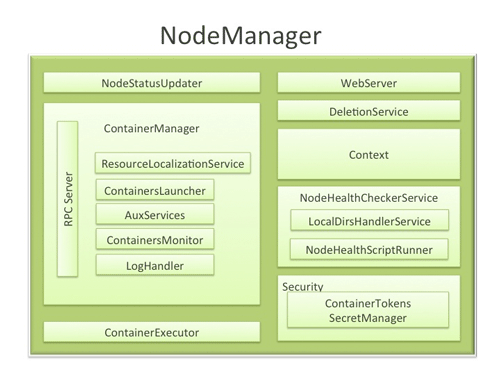
Node Manager

The NodeManager (NM) is YARN’s per-node agent, and takes care of the individual compute nodes in a Hadoop cluster. This includes keeping up-to date with the ResourceManager (RM), overseeing containers’ life-cycle management; monitoring resource usage (memory, CPU) of individual containers, tracking node-health, log’s management and auxiliary services which may be exploited by different YARN applications.

**NodeManager Components**

[](http://hortonworks.com/wp-content/uploads/2012/09/Node-Manager-Diagram.png)

* 1. **NodeStatusUpdater**

On startup, this component registers with the RM and sends information about the resources available on the nodes. Subsequent NM-RM communication is to provide updates on container statuses – new containers running on the node, completed containers, etc.

In addition the RM may signal the NodeStatusUpdater to potentially kill already running containers.

* 1. **ContainerManager**

This is the core of the NodeManager. It is composed of the following sub-components, each of which performs a subset of the functionality that is needed to manage containers running on the node.

* + 1. **RPC server**: ContainerManager accepts requests from Application Masters (AMs) to start new containers, or to stop running ones. It works with ContainerTokenSecretManager (described below) to authorize all requests. All the operations performed on containers running on this node are written to an audit-log which can be post-processed by security tools.
    2. **ResourceLocalizationService**: Responsible for securely downloading and organizing various file resources needed by containers. It tries its best to distribute the files across all the available disks. It also enforces access control restrictions of the downloaded files and puts appropriate usage limits on them.
    3. **ContainersLauncher**: Maintains a pool of threads to prepare and launch containers as quickly as possible. Also cleans up the containers’ processes when such a request is sent by the RM or the ApplicationMasters (AMs).
    4. **AuxServices**: The NM provides a framework for extending its functionality by configuring auxiliary services. This allows per-node custom services that specific frameworks may require, and still sandbox them from the rest of the NM. These services have to be configured before NM starts. Auxiliary services are notified when an application’s first container starts on the node, and when the application is considered to be complete.
    5. **ContainersMonitor**: After a container is launched, this component starts observing its resource utilization while the container is running. To enforce isolation and fair sharing of resources like memory, each container is allocated some amount of such a resource by the RM. The ContainersMonitor monitors each container’s usage continuously and if a container exceeds its allocation, it signals the container to be killed. This is done to prevent any runaway container from adversely affecting other well-behaved containers running on the same node.
    6. **LogHandler**: A pluggable component with the option of either keeping the containers’ logs on the local disks or zipping them together and uploading them onto a file-system.
  1. **ContainerExecutor**

Interacts with the underlying operating system to securely place files and directories needed by containers and subsequently to launch and clean up processes corresponding to containers in a secure manner.

* 1. **NodeHealthCheckerService**

Provides functionality of checking the health of the node by running a configured script frequently. It also monitors the health of the disks specifically by creating temporary files on the disks every so often. Any changes in the health of the system are notified to NodeStatusUpdater (described above) which in turn passes on the information to the RM.

* 1. **Security**
     1. **ApplicationACLsManager**NM needs to gate the user facing APIs like container-logs’ display on the web-UI to be accessible only to authorized users. This component maintains the ACLs lists per application and enforces them whenever such a request is received.
     2. **ContainerTokenSecretManager**: verifies various incoming requests to ensure that all the incoming operations are indeed properly authorized by the RM.
  2. **WebServer**

Exposes the list of applications, containers running on the node at a given point of time, node-health related information and the logs produced by the containers.

**Spotlight on Key Functionality**

* 1. **Container Launch**

To facilitate container launch, the NM expects to receive detailed information about a container’s runtime as part of the container-specifications. This includes the container’s command line, environment variables, a list of (file) resources required by the container and any security tokens.

On receiving a container-launch request – the NM first verifies this request, if security is enabled, to authorize the user, correct resources assignment, etc. The NM then performs the following set of steps to launch the container.

* + 1. A local copy of all the specified resources is created (Distributed Cache).
    2. Isolated work directories are created for the container, and the local resources are made available in these directories.
    3. The launch environment and command line is used to start the actual container.
  1. **Log Aggregation**

Handling user-logs has been one of the big pain-points for Hadoop installations in the past. Instead of truncating user-logs, and leaving them on individual nodes like the TaskTracker, the NM addresses the logs’ management issue by providing the option to move these logs securely onto a file-system (FS), for e.g. HDFS, after the application completes.

Logs for all the containers belonging to a single Application and that ran on this NM are aggregated and written out to a single (possibly compressed) log file at a configured location in the FS. Users have access to these logs via YARN command line tools, the web-UI or directly from the FS.

* 1. **How MapReduce shuffle takes advantage of NM’s Auxiliary-services**

The Shuffle functionality required to run a MapReduce (MR) application is implemented as an Auxiliary Service. This service starts up a Netty Web Server, and knows how to handle MR specific shuffle requests from Reduce tasks. The MR AM specifies the service id for the shuffle service, along with security tokens that may be required. The NM provides the AM with the port on which the shuffle service is running which is passed onto the Reduce tasks.

**Conclusion**

In YARN, the NodeManager is primarily limited to managing abstract containers i.e. only processes corresponding to a container and not concerning itself with per-application state management like MapReduce tasks. It also does away with the notion of named slots like map and reduce slots. Because of this clear separation of responsibilities coupled with the modular architecture described above, NM can scale much more easily and its code is much more maintainable