Add Service Container type to NodeManager in YARN

1. Motivation

From our work to support running OpenMPI on YARN (https://issues.apache.org/jira/browse/MAPREDUCE-2911), we found that it's important to have framework specific daemon process manage the tasks on each node directly. The daemon process, most likely similar in other frameworks as well, provides critical services to tasks running on that node(for example "wireup", spawn user process in large numbers at once etc) . In YARN, it's hard, if not possible, to have the those processes to be managed by YARN.

We propose to extend the container model on NodeManager side to support "Service Container" to run/manage such framework daemon/services process. We believe this is very useful to other application framework developers as well.

2. Idea

Currently in YARN during job execution time, as far as YARN is concerned, there is no distinction among all the processes/containers on each node, all are directly managed by each node manager, regardless the process roles each one plays in the application.

We propose to add a "new" container type modeled after daemon process/framework services on NodeManager side to provide the necessary constructs that framework developers can use. This new container:

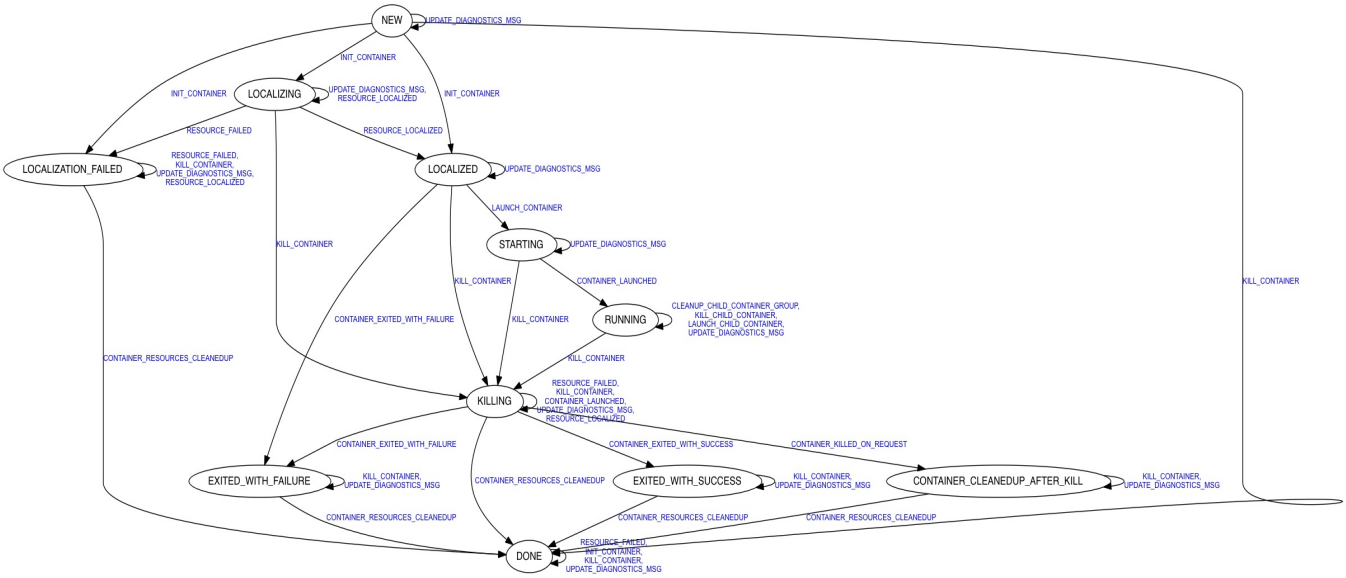
1. Can be used to run/track framework services
2. It has capability to handle the lifecycle of other containers, in another word, NM can delegate container management to framework services containers.
3. Accordingly, container management protocol needs to be extended to allow start, stop, query the containers that are managed by the framework services containers.
4. Above are core requirements, should be efficient for process launching, killing etc.

To differentiate from current NM container, we call this new container “Service Container”, the ordinary task process accordingly as “Task Containers”. The notion of “Service Container” vs. “Task Container” is conceptual, in implementation, in NodeManager, “Service Container” can be implemented by extending current NM ContainerImpl code.

To minimize the change needed and to be backwards compatible, we propose following implementation.

3. Implementation

3.1 Container State



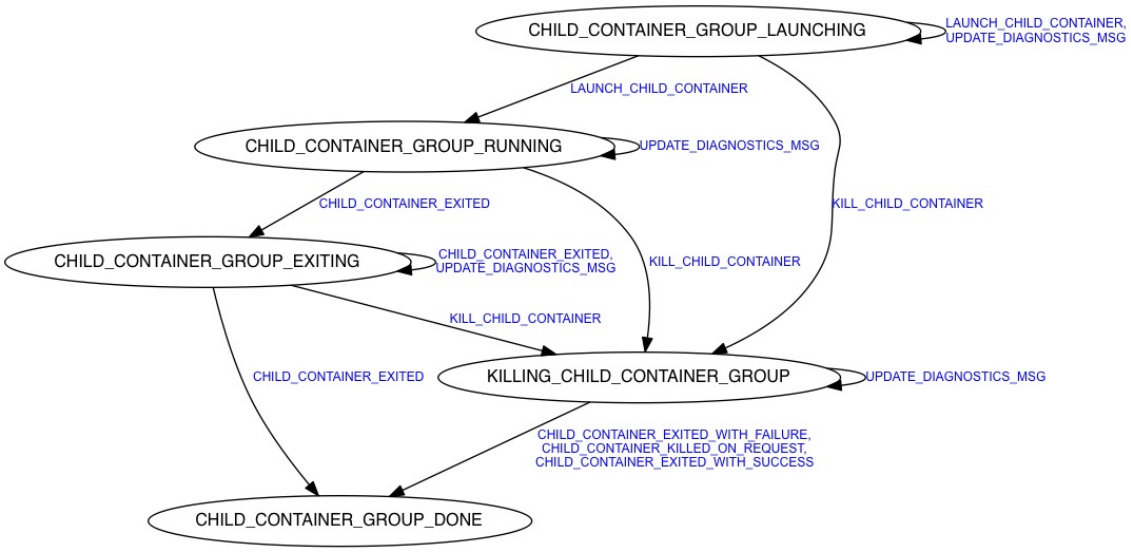
As in above diagram, we extends current NM container code that:

1. At RUNNING state, service containers can accept additional events to support managing task containers(child containers in the diagram), those key events are: LAUNCH_CHILD_CONTAINER, KILL_CHILD_CONTAINER
2. Nice-to-have enhancements: a new STARTING state to allow start daemon process asynchronously.

Service container maintains the set of child containers it manages. In this design, we further propose that the service container can manage child containers in groups. Container Group:

1. is created by Service Container when user wants to start and manage the child containers in groups
2. by default, all child containers are in the same group managed by the service container

3. container group has group events(can be used as sync point) when tasks are managed in group
4. When task group is not used, all tasks are in the default group, which always will be CHILD_CONTAINER_GROUP_RUNNIN state as long as the Service Container exists



3.2 Container Launch

We propose to reuse the current container manager protocol and only extends ContainerLaunchContext code as following:

1. AM starts/stop a service container just as normal containers as today, no changes expected

2. AM allocate resources and start task containers, in CLC,

a. AM specifies the container ID this tasks container is serviced by, servicedBy=containtainerId

b. AM can further specify GroupID, this can be application framework generated value.

c. If container is managed in group, additional attributes can be added pertain to the group(number of processes to start together)

3. stop and get child container state can be done in the same as today, no changes expected

4. Container Status – service container will return additional information regarding the Child Container Group state