YARN-8012

UNMANAGED CONTAINER CLEANUP (UCC)

An unmanaged container / leaked container is a container which is no longer managed by NM. Thus, it is cannot be managed / leaked by YARN, too.

Current implementation is marked as GREEN.

REPRO CONDITIONS

There are many cases a YARN managed container can become unmanaged, such as:

- NM service is disabled or removed on the node.
- NM is unable to start up again on the node, such as depended configuration, or resources cannot be ready.
- NM local leveldb store is corrupted or lost, such as bad disk sectors.
- NM has bugs, such as wrongly mark live container as complete.

Note, they are caused or things become worse if work-preserving NM restart enabled, see YARN-1336.

BAD IMPACTS

Bad impacts of unmanaged container, such as:

1. Resource cannot be managed for YARN on the node:
    - Cause YARN on the node resource leak
    - Cannot kill the container to release YARN resource on the node to free up resource for other urgent computations on the node.
2. Container and App killing is not eventually consistent for App user:
    - App which has bugs can still produce bad impacts to outside even if the App is killed for a long time

GOAL

The bad impacts mainly about CPU, Mem and constantly impacts, so prioritize below:

1. Cleanup container job object [P0]
2. *Cleanup container disk data -> This logic is duplicated with NM [TBD]*

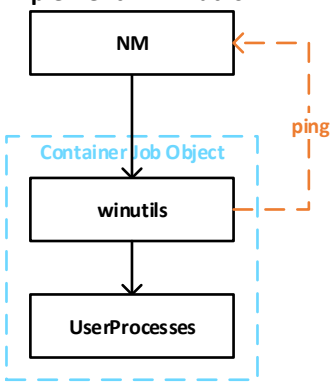
ARCHITECTURE

Prerequisite:

The container job object is set to be JOB_OBJECT_LIMIT_KILL_ON_JOB_CLOSE, so after winutils process exit, all the processes in the container job object will be killed. See https://msdn.microsoft.com/en-us/library/windows/desktop/ms684161(v=vs.85).aspx

Design Choices:

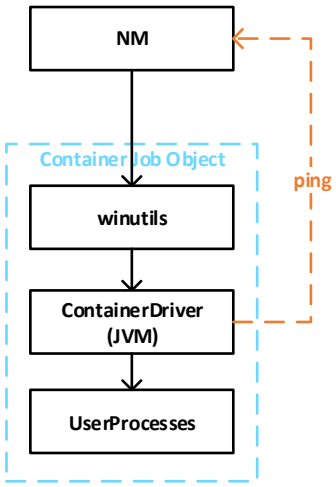
1. **Implement in winutils:**



Cons:

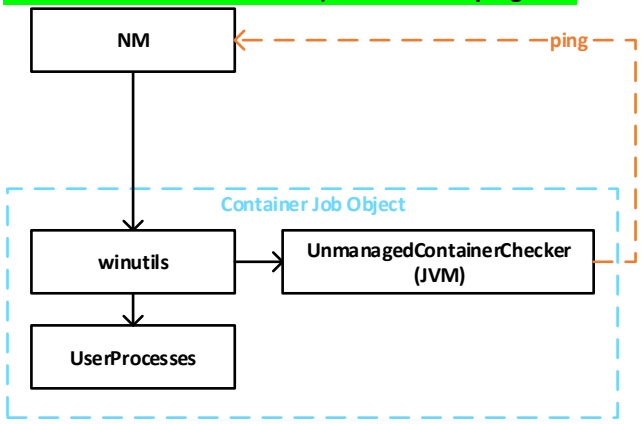
1. Write in C language: Hard to implement IPC, configuration read, json deserialization and extend new features with existing YARN components
2. This is a general feature for all OS platform and containers. However, Linux container cannot leverage the feature, since it is in winutils.

2. **Implement in a new executable: ContainerDriver.jar, it act as a process co-live with container:**
ContainerDriver is **ONLY** used for features that NM cannot handle since NM cannot co-live with container due to YARN-1336:



Cons:
- Need to consider more to eliminate side effects: pass through user process executing env, return results, etc

3. **Reuse NM jar to include UnmanagedContainerChecker feature, and winutils or other OS container executer (such as Linux bin container-executer) can use it to ping NM:**



Pros:
1. Resolved all the **Cons** in choice 1 and choice 2.

Current implementation is based on choice 3.

UNMANAGED CONTAINER DETECTION

Container will be considered as unmanaged when:

1. **NM is dead: Ping NM timeout**

   Failed to check whether container is managed by NM within timeout.

   Notes:

   To avoid judging NM lost inconsistently with RM, the timeout is suggest to be the same as yarn.nm.liveness-monitor.expiry-interval-ms. However, since RM will forget the last NM heartbeat time after RM restart, so, in this case the timeout suggested is more reliable to meet the NM lost expiry configuration and the inconsistency is expected.

   So, to be large than yarn.nm.liveness-monitor.expiry-interval-ms is meaningless.

2. **NM is alive but container not found:**

   The container is COMPLETE or not found in the NM container list.

   Notes:

   - Only considers API Container COMPLETE state as unmanaged, because:
     1. Other NM Container States may allow running container process to exist, such as NEW, LOCALIZING, LOCALIZED.
     2. The COMPLETE state is from API which means it is to be reported to the application, so we can safely consider the container is unmanaged.
     3. Use it to tolerate "NM has bugs. (Such as wrongly mark live container as complete)"
   - NM container list is always complete, because:
     NM starts its WebServer only if all containers managed by it are recovered, so the returned ContainersInfo is ensured to be complete.

TBD:

3. **UnmanagedContainerChecker may be killed by external unexpectedly. So, retry UnmanagedContainerChecker if it exits unexpectedly. And if retried many times in a short period, we stop retrying and consider it is unmanaged.**

   Prefer NOT. Because:

   1. Current winutils also cannot avoid it, it can be killed external unexpectedly and then the whole container will be killed.
   2. Add the unmanaged container judgement logic (retrypolicy) in winutils is not suitable, it should be in UnmanagedContainerChecker.
   3. Add the logic into outside boostrap batch: NM restart may regenerate incomplete batch in the configuration dir, add logic after waiting the UnmanagedContainerChecker is unreliable.

4. **NM is alive and container found, but not synced with RM: NM StatusUpdate timeout**

   → http://NM_ADDR/ws/v1/node/info (Need to add new field to distinguish with HealthReportTime)

   It can be caused by:

   1. NM StatusUpdate thread hang:
      We should consider container is unmanaged and cleanup it
   2. RM down for a long time:
      Whether we should consider container is unmanaged and cleanup it?
      Prefer NOT. Because:
      1. Current RM expire NM behavior is also not, i.e. after RM restart, the node update time is reset.
      2. It may cause all containers in a cluster killed, too bad for long running service.
      3. Current design only covers container unmanaged by NM. It should be NM's responsibility to consider node/ container unmanaged by RM.

In the first stage, prefer to only use trigger condition 1 and 2, to see if we need 3 or 4 in future.

REVISION HISTORY

Date	Modifier	Description
2017.12.04	Yuqi Wang	Draft