YARN-9834: Allow using a pool of local users to run Yarn Secure Container in secure mode

1. Introduction

Yarn Secure Container allows separation of different user's local files and container processes running on the same node manager. This depends on an out of band service such as SSSD/Winbind to sync all domain users to local machine that runs Yarn node manager. **Hadoop code only works with local users**.

Winbind/SSSD user sync has lots of overhead, especially for large corporations. Also if running Yarn node manager inside Kubernetes cluster (meaning node managers running inside Docker container), it doesn't make sense for each Docker container to domain join with Active Directory and sync a whole copy of domain users to the Docker container.

We need an optional light-weighted approach to enable Yarn Secure Container in secure mode, as an alternative to AD domain join and SSSD/Winbind based user-sync service.

Today, class *LinuxContainerExecutor* already supports running Yarn container process as one designated local user in non-secure mode.

We can add new configurations to Yarn, such that with *LinuxContainerExecutor* we can **pre-create a pool of local users on each Yarn node manager. At runtime, Yarn node manager allocates a local user to run the container process, for the domain user that submits the application**. When all containers of that user are finished and all files belonging to that user are deleted, we can release the allocation and allow other users to use the same local user to run their Yarn containers.

2. Design

We propose to extend *LinuxContainerExecutor* to support pool-user in secure mode. *LinuxContainerExecutor* is the main class alongside with other classes and the container-executor binary to implement the Yarn Secure Container feature.

For *LinuxContainerExecutor*, there are existing configurations for non-secure mode:

```
yarn.nodemanager.linux-container-executor.nonsecure-mode.limit-users, defaults to false
yarn.nodemanager.linux-container-executor.nonsecure-mode.local-user, defaults to "nobody"
```

We propose to add these new configurations for secure mode:

```
yarn.nodemanager.linux-container-executor.secure-mode.use-pool-user, defaults to false
yarn.nodemanager.linux-container-executor.secure-mode.pool-user-prefix, defaults to "user"
yarn.nodemanager.linux-container-executor.secure-mode.pool-user-count, defaults to -1, meaning the value of yarn.nodemanager.resource.cpu-vcores are used.
```

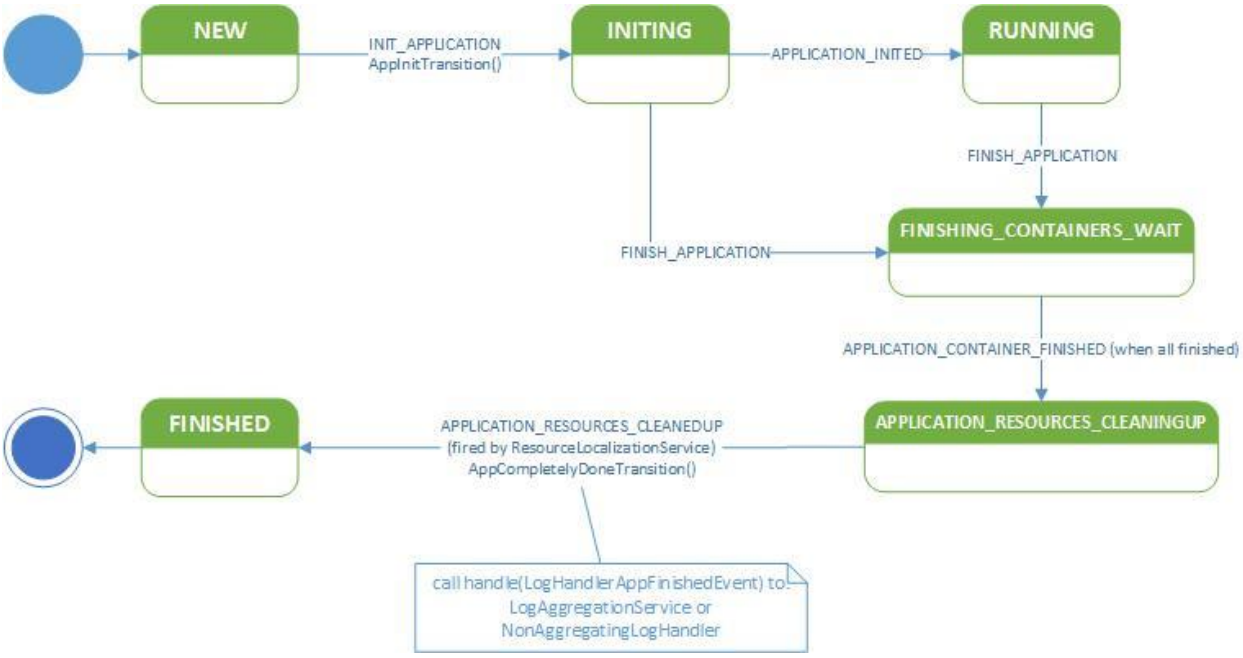
By default, this feature is turned off. If we enable it, with pool-user-prefix set to "user", then we expect there are pre-created local users user0 - usern, where the total number of local users equals to pool-user-count. The default pool-user-count equals to cpu-vcores, because in theory that's the maximum number of concurrent containers running on a given yarn node manager.

2.1 Design Details

2.1.1 Pool user allocation and deallocation

Now when to add the mapping and when to remove it?

In node manager, *ApplicationImpl* implements the state machine for a Yarn app life cycle in node manager, only if the app has at least 1 container running on that node manager. We can hook up the code to add the mapping during application initialization.



For removing the mapping, we need to wait for 3 things:

1) All applications of the same user is completed;
2) All log handling of the applications (*LogAggregationService* or *NonAggregatingLogHandler*) is done;
3) All pending *FileDeletionTask* that use the user's identity is finished.

Note that all operation to these reference counting should be synchronized operation.

2.1.2 The Allocator

The API for SecureModeLocalUserAllocator:
public String getRunAsLocalUser(String appUser)
public void allocate(String appUser, String appId)
public void deallocate(String appUser, String appId)
public void incrementFileOpCount(String appUser)
public void decrementFileOpCount(String appUser)
public void incrementLogHandlingCount(String appUser)
public void decrementLogHandlingCount(String appUser)

We can use an in-memory allocator (basically a hashtable) to keep the app user to local user mapping.

2.1.3 What happens if local user pool exhausted?

If all of our local users in the pool are allocated, we'll return "nonexistuser" as runas user, this will cause the container to fail to execute and Yarn will relaunch it in other nodes.

2.1.4 What happens if node manager restarts?

During Yarn node manager execution, it maintains 3 local folders:
- nmPrivate: all files are created by the yarn user who starts node manager.
- filecache: all public resources.
- usercache: all private and application resources, and working directory for Yarn containers for each app user.

At startup time, Yarn node manager calls *ResourceLocalizationService.init()*, which renames all 3 root folders above if they exist, then schedules asynchronous FileDeletionTask to delete the content of these folders. For usercache folder, the *FileDeletionTask* is issued as the owner (local pool users) of the local files.

This creates a security hole if we allocate a local user, e.g. user1 to a new application, before old content in usercache owned by user1 has not been deleted yet.

The solution is to block these local users from being allocated to new applications, until all the *FileDeletionTask* for that user are done. To achieve this, we allow local pool user allocation during calls to *incrementFileOpCount(appUser)*, which is during *FileDeletionTask* creation. And if appUser matches a local pool user, we allocate that specific user to the same named *appUser*, preventing new containers to reuse the same local pool user, until all the *FileDeletionTask* belonging to that user are done.

2.1.5 What happens if user code running in Yarn container creates other local folders?

Local users in the pool should not have home directory. If we allow local user to write to /tmp, then we need to let application developer know that files written to /tmp is not secured. We can also disable local pool users to write in any other folders (including /tmp) other than the container working directory, using Linux file ACLs.

3. Limitations

1) This feature does not support PRIVATE visibility type of resource allocation. Because PRIVATE type of resources are potentially cached in the node manager for a very long time, supporting it will be a security problem that a user might be able to peek into previous user's PRIVATE resources. We can modify code to treat all PRIVATE type of resource as APPLICATION type.

2) It is recommended to enable DominantResourceCalculator so that no more than "cpu-vcores" number of concurrent containers running on a node manager:

```
yarn.scheduler.capacity.resource-calculator
= org.apache.hadoop.yarn.util.resource.DominantResourceCalculator
```

3) Currently this feature does not work with Yarn Node Manager recovery. This is because the mappings are kept in memory, it cannot be recovered after node manager restart.