Allow whitelisted users to disable user re-mapping/squashing when launching docker containers

Zhankun 2016/09

1. Introduction

At present, YARN leverages the Docker "--user" option to run Docker container as the username configured (non-secure mode) or submitting user (secure mode). This brings in a limitation that YARN cannot run the Docker container with arbitrary user setup. We need a way to allow a predefined set of users to run Docker containers without "--user" option. And we also to need solve some implications of dropping this option: log aggregation, file/directory permission etc.

2. Problem Definition

The implications of dropping "--user" option is caused by the permissions of directories/files bind-mounted into Docker container when launching it. First, we should clarify the reasons for below unexpected behaviors:

- LinuxContainerExecutor will failed to launch Docker container due to launch_script.sh and the parent directories permissions in "$x/nm-local-dir/usercache/$username"
  Reason: The container-executor creates the "$username" directory in usercache with permission 2750 and copy token file and launch container script file from "$x/nm-local-dir/nmPrivate/$applicationID/$containerID" (644) to usercache directory with permission 600 and 700 respectively. Because of this, command "bash $pathToLaunchScript" passed to docker may fail due to the user (if has different UID) in Docker container won't be allowed to do this.

Note that we should also consider the situation that an application set environment variable YARN_CONTAINER_RUNTIME_DOCKER_RUN_OVERRIDE_DISABLE to true. In this case, the entrypoint of Docker image is not set up by LCE and the application should ensure it works fine. An improvement here is that we can pass the logdir to container with Docker “-e” option.

- Even if Docker container successfully launched, process in it cannot write stdout or stderr log file to the mounted log directory.
  Reason: The container-executor creates log directory in $x/logs/userlogs with permission 2750. Or the root logdir mounted into Docker container already exists and doesn’t allow write of the process running user.
- Log aggregation may also encounter issues in secured mode.

We can use “bash -c “ instead of “bash <script_file>” when launch container to bypass the first permission check. But there are several host dependencies in launch-container.sh. For instance, symbolic link creation involves directory creation operations in container working directory. If the working directory is overridden to bind-mounted host directory, then the host directory DAC settings (discretionary access control) will block the link and mkdir operations. If the working directory is not overridden by YARN, then the container may failed to write logs to bind-mounted host log directory or failed to execute application logic due to unable to read the security token (MRv2).

3. Possible solutions

3.1 Change the permission settings of bind-mounted host directory for whitelisted users

The solution is extending the needed directories permission for the whitelisted users to read and write.
From my testings, this will work. The code changes are mainly resides in container-executor to reach below goals:

- launch-container.sh created for whitelisted users can be read and executed by all users
- Container working directory of whitelisted users can be written by all users
- Container tokens of whitelisted users applications can be read by all users
- Application/container log directories created for whitelisted users can be written by all users

The drawbacks of this solution is that it brings potential security risks when application launches yarn simple docker instead of Docker container. The application process may modify other applications data. We may avoid this by a sophisticated designed container-executor but that would be complex.

3.2 Adapt the container user's UID to match the run-as user in host at runtime

We can keep current directories permission but utilize "usermod" to change the container user's UID before any container process start. This allows the user in container to present identical UID to the bind-mounted host directories/files and act as the host run-as user.

It's worth noted that the "usermod" need root access, so we need to start the Docker container with "--user=root" option and adapt the container user before execute the real entry point. Details steps are as follows:

1. Get the $UID of host run-as user
2. Get the $username set up in Docker image through "Docker inspect [Image]"
3. Get the $entrypoint set up in Docker image through "Docker inspect [Image]"
4. When the container started, it will first check the $username:
   If equals root, just execute the launch_container.sh or $entrypoint
   If not root:
   - Change container user's UID by "usermod -u $UID $username"
   - Execute the launch_container.sh or $entrypoint as original user by "su $username -c"

The changed UID container user still have same access on its own directories but also can access the bind-mounted directories from host (because Linux FS only cares about the UID/GID).

4. Opinion on best solution

Although solution 3.1 works but is hard to maintain. In my opinion, the solution 3.2 can meets our needs but also avoid changes to current YARN directories permission settings. The other implications caused by just drop --user option also disappears.
In essence, the 3.2 solution have same effect with current "--user=[run-as username]" mechanism but don't require the Docker image to customize user's UID statically. Instead, YARN does this "normalize user" work at runtime.

5. Details on how the solution should be implemented

Generally, we need to find out a suitable place for the shell script snippet which adapts the container user UID. And it should be executed before current procedure in "launch_container.sh". I think we have two ways to do this. And both ways needs Docker inspect to get the Docker image information.

5.1 A generic launch container script serving all type of containers

One way to do that is extending this generic launch_container.sh to accept parameters containing new UID , container username and original entry point. Base on the parameters, the script will first handle this “normalize user” work and may also run the original entry point instead of YARN generated scripts.

Fistly, we’ll change the launch_container.sh as below (just a example). Once there are arguments passed in, the usermod will be executed if necessary. The default yarn container will get zero arguments so it will not be affected in theory.

```
#!/bin/bash
container_run_as_user=`whoami`
if [ "$#" -eq 2 ]; then
new_uid=$1
adapted_user=$2
old_id=`id -u $new_user`
If [“$old_id” != “$new_uid”]; then
  usermod -o -u $new_uid $adapted_user
  container_run_as_user=$adapted_user
fi
fi
sudo -u $container_run_as_user bash << EOF
export ...
export ...
…
exec /bin/bash -c "$JAVA_HOME/bin/java ...
EOF
```

Secondly, modify the DockerLinuxContainerRuntime#launchContainer, if the run-as user is in whitelist, then we will first get container info (including container username and entrypoint) through “Docker inspect”. Then use “--user=root” option and set the container entrypoint command to “bash .../launch_container.sh $containerUsername“.

5.2 Modifying the launch container script file only at Docker launch phase

Anther way to do this is modify the generated launch_container.sh in DockerLinuxContainerRuntime#launchContainer. This will have extra cost of read the script content from file system, add lines and write it back.

6. Future work

TBD