YARN Localization Support for Docker Image (YARN-3854)

Zhankun 2016/06

1. Introduction

YARN now supports Docker to launch containers through the LinuxContainerExecutor. Currently YARN will delegate the preparation of the Docker image to the Docker engine if the image is not present on the target host. But there are more issues to be considered in a YARN cluster when it comes to having the Docker images ready before launching Docker. We filed this design document for discussion of in-band control of Docker images in YARN (YARN-3854).

2. Problem Definition

As we all know, Docker will try to pull images from docker hub or private repository if no local image found. So currently, to ensure that LinuxContainerExecutor can launch container successfully, one must configure public network for Docker daemon, set up own private repo or prepare the image in advance by “docker load” manually.
Each above choice needs more thinking in it, let’s check them one by one. Firstly, if a user enables public network, the docker pull speed will be a concern. The task will timeout if docker pull operation costs much because of sufficient large image or unstable networking. Solutions which include supporting localization progress query from AM and improvement of localization (for instance, delta download module in NM) for this are discussed in a separate JIRA (YARN-3289).
Secondly, investments on own private repository in an internal network is a good choice and most big company will do this. This private repo is not only for YARN but also CI/CD integration with other existing system to use Docker. I agree with this solution if the user can afford to maintain a private repo. But not all users want this or maybe they just want to use Docker with YARN in a strictly restricted environment. This will bring a requirement that is supporting “docker load” before launching Docker container.
Thirdly, a user can manually load images for cluster in advance or develop own YARN application to do this step. But doing this load stuff manually is obviously unacceptable. And

although a YARN application which spawn a separate task can achieve this, it needs the submitting user has the privilege of docker group. This also has potential security issue.

To sum up, this proposal is meant to enable the docker image localization and “docker load” for YARN app. We can split it into three questions and provide possible solutions for each question in next section:

1. How can YARN know that it needs to load a Docker image package file?
2. Where can YARN find the package file?
3. What’s the timing of execute “docker load”?

3. Possible solutions

3.1 Docker image localization interface

Option 1. Environment variable in ContainerLaunchContext
In order to let YARN know that it needs to load a Docker image package file, one choice is that using an environment variable “YARN_CONTAINER_RUNTIME_DOCKER_IMAGE_FILE” in ContainerLaunchContext as the interface for YARN application to command loading a Docker image file. And the value of this variable would be a regex expression which represent the file names in LocalResources of container. So that the ContainerRuntime can execute “docker load” on each of the local image package file in a blocking way before launching Docker containers.
Option 2. Add an “IMAGE” type in LocalResourceType
This way is straightforward. The YARN application set “LocalResourceType.IMAGE” when create LocalResource. Then the ContainerRuntime can just check this field before launching container to determine whether load this image file.

3.2 Where to execute the “docker load”

Before we discuss where to place the “docker load” logic, I would like to recall the process of current LCE launch container. Once a LocalResource file was fetched successfully in ContainerLocalizer, it will inform ResourceLocalizationService through heatbeat. And the ResourceLocalizationService will send a ResourceLocalizedEvent to LocalizedResource which then send an event to ContainerImpl. And the detailed flow can be viewed in Figure 1.

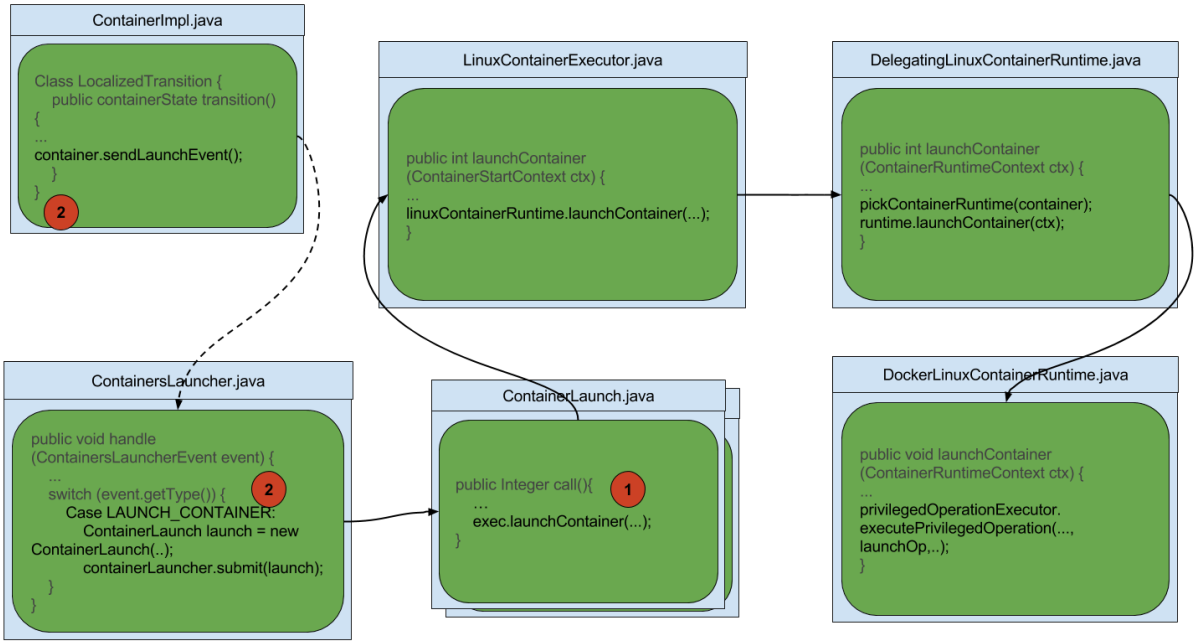

Figure 1. Current LCE launch container process

Current implementation is performing container launch right after all LocalResource is downloaded successfully in ContainerImpl.java. Then we need to find a proper timing to do the “docker load” before launch container.
And it is worth noting that there is an unused callback “void prepareContainer(ContainerRuntimeContext ctx)” in interface ContainerRuntime. We think it’s a better place to implement the “docker load” logic here than other new interfaces. So the next thing is when to invoke the “ContainerRuntime.prepareContainer(...)”. As shown in figure 1 red circles. We have 2 options:

Option 1. Invocation in the “call” method of ContainerLaunch.java
To minimize the changes, we can just invoke “exec.prepareContainer(...)” before “exec.launchContainer(...)”. The efforts of implementing this includes:

- Add a method “prepareContainer(...)” in LinuxContainerExecutor.java
- Implement the “prepareContainer(...)” in DockerLinuxContainerRuntime.java
- Invoke “exec.prepareContainer()” before “exec.launchContainer()” to do the docker load operation

This option is quite straightforward. But the speed may be slow due to the blocking sequence invocation of prepareContainer and launchContainer.

Option 2. Invocation in “handle” method of ContainersLauncher.java
An elegant way to do “docker load” seems to be adding “PREPARING” and “PREPARED” states in ContainerImpl state machine. And the new event type “ContainersLauncherEventType.PREPARE_CONTAINER” handling is also needed in ContainersLauncher. When the ContainerImpl finished the LocalizedTransition, instead of sending launch event to ContainersLauncher, it needs to send “prepare” event and transform itself to “PREPARING” state. Once all preparing stuffs done then it transforms to “PREPARED” state and send container launch event as before.

But adding states to the ContainerImpl state machine is overkilled for me. It has more complex state machine but no obvious benefits for speeding up the whole launch process.

In essence, if we cannot do preparing and localization at the same time as a pipeline to achieve better performance, adding states after “LOCALIZED” in ContainerImpl state machine has no differences between Option1.

If we step back and treat the “prepareContainer” as part of the localization process, the implementation will be simpler and possible for implementing parallel localizing and container preparation. So a better Option 2 seems be just keep the ContainerImpl state machine unchanged. And adding the logic of invoking preparation in LocalizedTransition. The efforts of implementing this includes:

- Add ContainersLauncherEvent (PREPARE type) post operation in LocalizationTransition of ContainerImpl for the finished LocalResource. There will be a preparing queue to track if all type of LocalResource has been processed in “prepareContainer”. Once all LocalResource has been prepared, submit the launch event as before.
- Create a new prepare type of ContainersLauncherEvent and add handling in “handle()” of ContainersLauncher. It will submit a “ContainerPrepare” which implement Callble interface.
- Create a class ContainerPrepare to utilize LCE to do the real “prepareContainer” work. After the prepreation work finished, it will send “RESOURCE_LOCALIZED” to trigger LocalizedTransition of ContainerImpl
- Add a method “prepareContainer(...)” in LinuxContainerExecutor.java
- Implement the “prepareContainer(...)” in DockerLinuxContainerRuntime.java

4. Opinion on best solution

For the Docker image localization interface, the environment variable way is uniformed with current LCE Docker environment interface. It provides a choice to let user specify the “hdfs” as repository directly. We prefer the environment variable way to begin with. But eventually, once we see more adoption, we might see value in adding a new LocalResourceType for Docker images.

For the location of “docker load” logic, we prefer the simplest way which is invocation in the “call” method of ContainerLaunch. This option assumes that the prepareContainer is just a blocking step before launch container. Other options may be reasonable if we met use cases that requires better performance when doing container localization and preparation. So we choose the simplest way to keep it evolving under control.

5. Details on how the solution should be implemented

Based on above opinions on best solution, we'll summarize our design and detailed implementation. Because this docker image localization is a feature based on HDFS distributed cache and maybe related to other modules in the future, we should make it simple and try to use existing facilities as much as possible.
In order to be uniform with current LCE Docker interface, we choose to use an existing environment variable "YARN_CONTAINER_RUNTIME_DOCKER_IMAGE_FILE" for YARN application to set. The environment variable name indicates that YARN should use Docker to load the image file resource. And the value of it which is regex expression represents all the image package files to be loaded into Docker.
During investigation on the timing and location of execute "docker load", we find that the "prepareContainer(...)" callback of ContainerRuntime is suitable for this logic. It's also possible that more preparation task will be done here in the future. And to be simple, we just put the preparation invocation function right before launching container. The efforts of doing the whole implementation are as follows and shown in Figure 2 (red lines):

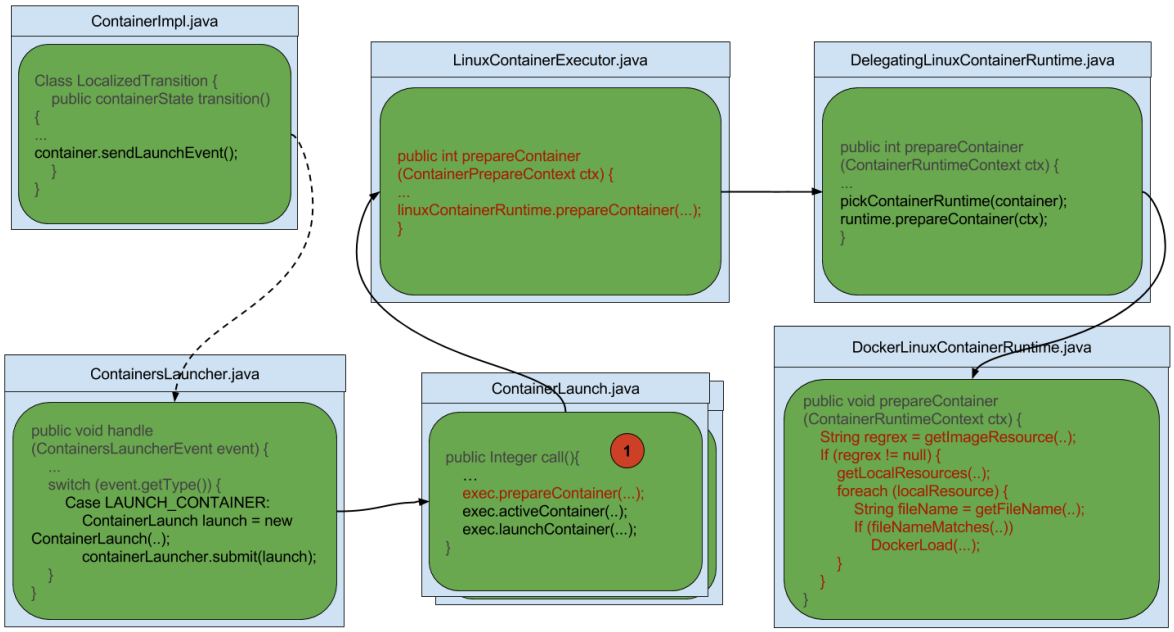

Figure 2. Docker load in prepareContainer()

- Add an interface "prepareContainer(...)" in ContainerExecutor.java
    - This will affect all subclass. Default implementation will just return 0.
- Add an exit code "PREPARE_FAILED(155)" in ContainerExecutor's ExitCode
- Add a method "prepareContainer(...)" in LinuxContainerExecutor.java
- Implement the "prepareContainer(...)" in DockerLinuxContainerRuntime.java
    - Get the regex expression from environment variable "YARN_CONTAINER_RUNTIME_DOCKER_IMAGE_FILE"

    - Get LocaResources and for each of them, if the filename matches the regex expression, run below step. If no filename matches, it fails.
    - Utilize exisiting DockerLoadCommand to build the command file needed by PrivilegedOperation
- Invoke “exec.prepareContainer()” right before “exec.activeContainer()” in ContainerLaunch to do the docker load operation.

6. Future work

The download of docker image will be a huge cost to network bandwidth if thousands of containers are expected to launch in a certain period. Future optimizations to this are as follows:

- Delta download of distributed cache(may be resolved in YARN-3289)
- Optimized scheduling based on distributed cache status