Localization Support for Docker Images

Introduction

When a docker container is launched and the image of the container is not present on the node, then `docker run` implicitly pulls the docker image. This increases the time for container launch within YARN.

If NM can provide a way to localize docker images before container launch, then for a YARN service this will reduce the upgrade downtime and potentially the service start time if the service can have multiple containers allocated on the same node.

Requirements

1. Add support to NodeManager to localize docker images.
2. Retrieve the progress of localization.
3. Add support for Docker images cleanup.
4. Nice to have - pull policies with container launch. Not needed for the service upgrade.
    a. ifNotPresent: default, pull image when it is not present locally.
    b. always: pull image always before launch.
    c. never: never pull images before launch. If the image is not present locally, the launch should fail with a relevant message.
5. Nice to have - an API added to NM to list images at that node.

Implementation

Requirement 1: Add Support to NodeManager to localize docker images

NM API to localize docker images

For this, we can utilize the existing NM `localize` API. This requires addition of `DOCKER_IMAGE` type to `LocalResourceType`.

Existing localize API in ContainerManagementProtocol

```
@Public
@Unstable
ResourceLocalizationResponse localize(ResourceLocalizationRequest request)
 throws YarnException, IOException;
```

NOTE: NMClientAsync does not have a `localize` method. This needs to be added.

Localization of image by NM

When NM is requested to localize a docker image, it needs to execute `docker pull`.
YARN-5669 already added a `DockerPullCommand` to NM.

In addition, my proposal is to add a simple interface and a docker implementation for pulling images - ImageManager

```
interface ImageManager {
        pullImage(image info);
}
```

The `DockerImageManager` will implement pull image for `DOCKER_IMAGE` type by executing the `DockerPullCommand`.

```
class DockerImageManager implements ImageManager {
        pullImage(image info) {
                // execute docker pull command
        }
}
```

The flow will be as :

1. ContainerManager.localize() -> dispatches ***LOCALIZE_CONTAINER_RESOURCES*** event
2. ResourceLocalizationService handles ***LOCALIZE_CONTAINER_RESOURCES***
    - If the resource type is DOCKER_IMAGE, then DockerImageManager is used to pull the images.

Requirement 2: Retrieve the progress of localization

Currently NM does not have an API to retrieve the progress of localization. Unless the client can know when the localization of a resource is complete irrespective of the type of the resource, it cannot take any appropriate action.

This leads to a need of an API to retrieve the progress on the localization.

Add an API to ContainerManagementProtocol to retrieve the progress of localization

GetLocalizationStatusesResponse getLocalizationStatuses(GetLocalizationStatusesRequest request) **throws** YarnException,IOException;
where:

- GetLocalizationStatusesRequest: List<ContainerID> containerId
- GetLocalizationStatusesResponse: Map<ContainerID, Map<LocalResource, LocalizationState>>
- LocalizationState: IN_PROGRESS, COMPLETED, FAILED

This will let the client request the localization statuses of multiple containers running running on the same node in one call.

NOTE: This is just a rough idea. There will be changes during implementation. Will also need to add getLocalizationStatuses(...) to NMClient.

Requirement 3: Add Support to removing old docker images

There needs to be a way for NM to know which image can be removed.

There are 2 ways of handling this:

1. **NM maintains an LRU cache which holds meta-info of docker images**
   This requires NM to maintain an LRU cache of docker images. Once an image which is not used and evicted from the cache, it will be eligible for deletion. This will also require storing the images meta-info in NMStateStore using which the cache is reconstructed after NM restart.

   Benefits of this approach:

   - NM only deletes the images that were localized by it.
   - NM will delete an image only if it is least recently used unlike `docker image prune` that removes the image if it is not being used currently.
   - With docker containers, there is a problem that docker containers stick around in zombie state. That will cause deletion of eligible to fail constantly. With this approach we have a potential solution:
     - image meta contains count of running containers. This count is updated when the container is launched.
     - NM already has a service which periodically check the statuses of the container. If the container is stopped/failed, image meta is updated.

- When the image is evicted and count of running containers = 0, it can be force deleted.
- This solution relies on NM's view of containers and images.

Cons of this approach:

- NM stores images meta-data in memory which is backed by NMStateStore. Increases NM memory footprint.

If we go by this approach, we can add an LRU cache to DockerImageManager. Parameters to configure the cache can be discussed during implementation of the cleanup.

2. **Docker provides `docker prune`**
   An example of docker prune command:
   `docker image prune -a --filter "until=24h"  // until works with image creation time.`

   This command will remove all the images that are not being used by any container and were created more than 24 hours ago is. This also will remove any dangling images.

   Benefits of this approach:

   - NM doesn't have to store any meta-data about the images it localized.
   - Nothing special to do during NM restarts.

   Cons of this approach:

   - Does not provide enough options to clean up old images that are least recently used.
   - **NM deletes images that were not localized by it.**
   - `docker prune` will remove of all the images that at that time are not being referenced by any running container.
   - Don't have a solution for zombie containers that keep running.

Based on the image cleanup approach we go with, we can submit either of the below delete tasks to DeletionService at regular intervals:

- Docker images prune task
- Docker rmi deletion task for all eligible images that can be removed.

Additional Requirements

- An “Untouchable” list of images that are never automatically deleted by NM.

Any new APIs add to NM?

- For Requirement 2, add an RPC API to retrieve status on the localization.
- Additional API in the NM to list the images.
    - This can be just an API added to the NM web-service.

Service Upgrade Use case - Minimize service downtime

YARN service upgrade allows the user to upgrade the version of the artifact without stopping the service. During the upgrade, one of the goals is to minimize the downtime of the container.

The upgrade of a YARN Service involves 2 steps:

1. Initializing Service Upgrade
   In this phase, the ServiceMaster identifies all the component instances that need an upgrade and marks them.

2. Upgrade an instance
   This triggers the upgrade of a container. The ServiceMaster invokes the reinit API of NM to upgrade the container.

During reinit, the NM stops the running container, cleans up the working dir, and then starts the container again. For this step, we want to reduce the time it takes to restart the container. Since `docker run` implicitly does a `docker pull` if the new image version is not present, this will take a long time and the container will be down for a longer time.

A solution is to localize the docker images during step 1. When the image is already localized, then the upgrade of a container will take much less time and therefore, the disruption to the service is minimum.

Future Enhancements

1. NM report to RM about the localization progress.
2. API to delete images cluster wide