RunC Container Repository V2

Problem Statement

The current runc container repository design has scalability and usability issues which will likely limit widespread adoption. This proposal aims to address some of these issues with a new design.

Current Structure

The current `RuncContainerRuntime` makes use of two plugin interfaces, `RuncImageTagToManifestPlugin` (which defines how to resolve container image coordinates or "tags" to container manifests) and `RuncManifestToResourcesPlugin` (which maps container manifests to a list of YARN `LocalResource` objects to localize). Together, these in effect define an "interface" to a container repository. The default implementations (`ImageTagToManifestPlugin` and `HdfsManifestToResourcesPlugin`, respectively) of these contracts effectively work together to form what from here on will be called the Repository V1 format.

The Repository V1 format (including default paths) looks like this in HDFS:

- `/runc-root/` (repository root)
  - `image-tag-to-hash` (mapping file which translates tags to manifest hashes)
  - `config/` (OCI container configs)
    - {sha256}... (individual container config stored by sha256 hash)
  - `layers/`(OCI image layers)
    - `{sha256}.sqsh` (filesystem layers in squashfs format, named from original Docker sha256)
  - `manifests/` (OCI image manifests)
    - `{sha256}...` (individual manifests stored by sha256 of content)

The `ImageTagToManifestPlugin` class also performs periodic downloading of the `image-tag-to-hash` file (by default every 10 minutes) to keep a local cache of tag to manifest mappings.

Challenges

Manageability

Because the `image-tag-to-hash` file contains all mappings between tags and images, it must be updated *atomically* and *without fail* to prevent denial of service (i.e. failure to launch) of YARN containers. It is also not amenable to multiple concurrent writers.

Scalability (large numbers of images)

The flat structure of the current repository layout requires that all image layers (and to a lesser extend configurations and manifests) must be stored in the same directory, resulting in extremely large file counts and NameNode pressure when listing contents. A complex image may have dozens (or even hundreds) of image layers, so with tens of thousands of images, it's not unforeseeable to find hundreds of thousands or even millions of image files in a single repository.

Scalability (large clusters)

The current implementation downloads the image-tag-to-hash file every 10 minutes, whether updates have happened or not, and caches the results in memory. This happens on every NodeManager instance, leading to unnecessary HDFS traffic.

Consistency (image tags)

Because of the previously mentioned caching, newly created or modified image tags will not have guaranteed visibility within the cluster until at l0 minutes + the time required for the slowest NodeManager to refresh its cache. This can result in failure to launch runc containers, or "split brain" execution where multiple runc containers for an application (even spawned simultaneously) may execute in different versions of the same-named image.

Consistency (naming)

The current `config`, `layers`, and `manifests` directories do not have consistent casing (two plural, one singular), leading to confusion for administrators.

Consistency (reproducibility)

The current design does not preserve the original Docker image layers, making re-import if necessary problematic.

V2 repository design

We propose the following filesystem layout and behavior:

- `/runc-repository/` (repository root)
  - `meta/` (metadata go here)
    - `{namespace}/` (image namespace directory)
      - `{name@tag}.properties` (tag name meta-file in properties format)
  - `config/`
    - `{hash}/` (first two digits of configs contained within)
      - `{sha256}` (OCI container config by SHA-256 of content)
  - `layer/`
    - `{hash}/` (first two digits of layers contained within)
      - `{sha256}.tar.gz` (Original Docker image layer)
      - `{sha256}.sqsh` (Squashfs image, same base name as Docker image)
  - `manifest/`
    - `{hash}/` (first two digits of manifests contained within)
      - `{sha256}` (OCI layer manifest by SHA-256 of content)

Example (only files shown)

NOTE: Only files shown, not directories, and sha-256 hash files are abbreviated.

- `runc-repository/meta/apache/hadoop@3.3.0.properties`
- `runc-repository/config/fd/fd2ba622...d23bf8ce93`
- `runc-repository/layer/ba/ba378b8e...6871d6ae`
- `runc-repository/manifest/85/85962bab...5a0ef7a5`

The meta property files contain key/value pairs and are meant to be extensible. For now, we will define the following (with example):

```
runc.import.type=docker
runc.import.source=repository.docker.com/apache/hadoop:3.3.0
runc.import.time=2020-11-17T16:44:23.387Z
runc.manifest=sha256:85962bab...5a0ef7a5
```

Only `runc.manifest` is required, but the other values are defined here to support lineage tracking, etc.

Manageability

Because metadata is now stored per image in separate files, a new image may be deployed without risk to other images. Adding a namespace abstraction allows for clean separation of images used for different purposes.

An optional namespace can be specified as part of the runc image:

YARN_CONTAINER_RUNTIME_RUNC_IMAGE=[namespace]/<image_name>[:tag]

Examples:

YARN_CONTAINER_RUNTIME_RUNC_IMAGE=busybox

/runc/root/meta/library/busybox@latest.properties

YARN_CONTAINER_RUNTIME_RUNC_IMAGE=hadoop/busybox:latest

/runc/root/meta/hadoop/busybox@latest.properties

YARN_CONTAINER_RUNTIME_RUNC_IMAGE=shared/busybox:1.0.0

/runc/root/meta/shared/busybox@1.0.0.properties

Scalability (large numbers of images)

Scalability of V2 is improved with respect to large number of images because the number of files in any given directory will be reduced by a factor of 256, while not introducing any additional per-image or per-layer overhead for retrieval, as the hashes are all known in advance. In the event that a directory needs to be listed, the number of files in it will be reasonably small.

Scalability (large clusters)

Scalability in large, busy clusters can be addressed by caching just tag to manifest mappings on a per-tag basis (and for a limited time). If a tag is unknown to a NodeManager upon container localization, it will attempt to retrieve the associated metadata file from the repository. Once that mapping is known, it can be cached for a period of time to prevent needing to download (or even stat) the metadata file again. Even relatively short cache times can beneficial in large clusters; it is highly likely that multiple containers using a given image spin up in rapid succession.

Consistency (image tags)

Consistency in V2 is in the worst-case (caching enabled) no worse than V1. In fact, it will be significantly better as a newly created tag WILL be resolvable by all NodeManagers immediately. Only changed tags may be inconsistent in the face of caching. For deployments where changing of already existing tags may be the norm, we will provide a facility to disable tag caching and fall back to HDFS files.

Consistency (reproducibility)

V2 will allow for existing Docker image layers (in tar.gz format) to be preserved, allowing re-conversion if necessary without requiring that the source Docker repository remained unchanged.

Upgrades / Backwards compatibility

It is our contention that the current runc container runtime is used in very limited fashion today, as it was only released with Hadoop 3.3.0 and is somewhat experimental. However, we may have existing users who would be impacted by a change of this magnitude. We propose keeping the existing implementations in place, but changing the default configuration to specify the new V2 repository layout plugins. This will enable a consistent, supported layout moving forward that other tooling (such as an import tool) can be built upon. Existing users who wish to do a seamless upgrade would need to update their configuration pre-upgrade to explicitly mention the old implementations and/or convert to the new format during upgrade in a new repository path.