CernVM-FS Container Image Integration

Simone Mosciatti, Jakob Blomer, Gerardo Ganis, Radu Popescu
CERN, Espl. des Particules 1, 1211 Meyrin
E-mail: simone.mosciatti@cern.ch

Abstract. Linux containers have gained widespread use in high energy physics, be it for services using container engines such as containerd/kubernetes, for production jobs using container engines such as Singularity or Shifter, or for development workflows using Docker as a local container engine. Thus the efficient distribution of the container images, whose size usually ranges from a few hundred megabytes to a few tens of gigabytes, is becoming a pressing concern. Because container images show similar characteristics than scientific application stacks, unpacking the images in CernVM-FS can remedy the distribution issues provided that the container engine at hand is able to use such unpacked images from CernVM-FS. In this contribution, we willl report on recent advances in the integration of Singularity, Docker, and containerd with CernVM-FS. We show improvements in the publishing of container images from a Docker registry that rely on new means of directly ingesting image tarballs. Well also show a repository file system structure for storing container images that are optimized for storing both container engines using flat root file systems (Singularity) as well as container engines using layers (containerd, Docker). To evaluate the benefits of our approach, we show concrete use cases and figures for production and development images from LHC experiments stored in the recently created unpacked.cern.ch repository.

1. Introduction

Containers orchestration tools, such as kubernetes, provide a solution to the problem of managing runtime dependencies of complex software stacks. They are in widespread use in industry and quite attractive also in the high energy physic field. The great size of HEP software stacks, routinely bigger than 1 GB, however, makes difficult to deploy containers on a large number of machines. Indeed a standard solution in this case, especially in HEP, is to use CernVM-FS [2] to deploy the software to a central server and lazily load only the strictly necessary pieces needed to run the computation. The singlularity.opensciencegrid.org CernVM-FS repository has successfully shown how to distribute singularity containers at Worldwide LHC Computing Grid scale. For layer-based container engines, such as docker and containerd, previous work introduced the concept of a thin-image [1], blending together the container layer abstraction with the efficient file-level distribution provided by CernVM-FS. A thin-image is a small JSON file that contains the location of the docker layers, previously stored in CernVM-FS. The container runtime can access the files in the layers directly from CernVM-FS without the need to download big files from a central repository. Our contribution is to provide a container publishing service to serve both Singularity and Docker thin-images containers in a completely automatic way, starting from a standard Docker images. This allows HEP researchers to run their containerized computing jobs even on big clusters with ease. The design of the container publishing service aims at supporting thousands of images per year, summing up to billions of files.

2. Our approach

The container publishing service is based on the assumption that the initial container development and the maintenance of user containers is done with Docker. The new service provides a declarative interface for the users to specify which Docker images they desire to provide on CernVM-FS for large-scale deployment. Upon specification, the Docker source image is converted to both a Singularity compatible directory tree (a flat root file system) and a Docker compatible directory structure (a set of directories each containing part of the whole filesystem) and published into a CernVM-FS repository. The declarative interface consists of a simple YAML file, the wishlist, that specifies:

- The target Docker registry for storing the CernVM-FS thin image
- The credentials of the target Docker registry
- The naming scheme for the thin images
- The list of the Docker images to convert

The use of such declaration file was inspired by the successful experience reported by Open Science Grid, indeed, the wishlist can be stored in git so that multiple stakeholders can visualize it and request modifications. The modifications are rejected and accepted by the owner of the repository. To convert the regular Docker source image into a Docker thin images we begin by ingesting every Docker layer into CVMFS, after each layer has been ingested we proceed to create the thin-images descriptor [1]. Finally, the thin-image descriptor is pushed as Docker images into the Docker registry specified in the wishlist. To convert the Docker source image into a Singularity image we exploit the capability of Singularity to run images directly from a simple directory containing the root file system. Hence we unpack the whole image into a directory that is then published in CVMFS.

3. Scalability

The container publishing service is designed with the assumption to support a large number of images, in the order of one image for every physic analysis. In order to support the anticipated scale, the directory tree of the CernVM-FS repository hosting the images is carefully crafted.

In particular, we exploit the sub-catalogs of CernVM-FS and its deduplication capability.

3.1. Deduplication

Images typically share a lot of identical files. For instance, the same operating system is easily shared between a lot of different images, but also some basic software and configuration files can be identical between different images. This peculiarity is accentuated by the fact that Docker images tend to be built on top of identical base images in order to exploit the Docker cache capabilities. Moreover, the Singularity version of the image contains the same files than the Docker version. These identical files are automatically deduplicated by CernVM-FS using its internal Content Addressable Storage engine, in this way we avoid storing multiple times the same file.

3.2. Sub-catalogs and directory tree structure

The directory tree is structured to make effective use of the CernVM-FS sub-catalogs.

There are two conflicting constraint to consider while designing the filesystem structure to make efficient use of CernVM-FS sub-catalogs. From on side we want to maximize the amount of catalogs, so that each sub-catalog contains the least number of entries and it is small in size. From the other side we want to minimize the amount of catalogs necessary to access a file, in this way fewer round-trip connection to the server are necessary but the catalogs grow in size.

We decide to create a sub-catalog for each Singularity filesystem and one for each Docker layer. While this approach is not perfect it strikes a good balance between the two constraints. With this approach we also avoid the need to download additional catalogs during the runtime of the container.

If we stored all the Docker layers in the same directory, the size of the sub-catalog in such directory would grew too big for most practical purposes. In order to mitigate this problem we decide to partition the docker layers in different directories. We decide to partition the docker layer using the first two characters of their hexadecimal hash as partition key, this create 256 partition slots. This amount of slots is a good balance between the size of each catalog and the number of catalogs.

3.3. Metadata and garbage collections

When the original Docker images are updated, the container publishing service will download the new images and replace the old one. Hence some layer of the previously published images will become obsoletes and it will be possible to remove them. In this case, two problems arise:

- Users could still rely on old versions of image that use obsolete layers.
- Some unrelated image could still be using those layers.

Unfortunately is not possible to know what version of the image the users have downloaded locally, so it is impossible to know with certainty if an old layer is still required by some users, hence we decide to provide a grace period of 30 days. After the grace period, the layer is removed. A similar policy is applied to the Singularity root file-system.

For what concerns layers shared between different images, we keep metadata about each layer. The metadata consists of a list of images that use such layer, when we delete an image, we delete also the reference of that image from each layer metadata. As soon as a layer is not referenced by any image, the layer can be deleted after the grace period.

4. Summary

In this work we introduce a container publishing service to automatically manage the distribution of containers using CVMFS. We show how we addressed the challenges related to the anticipated scale in terms of the number of distributed container images. The service has been released in cvmfs 2.6, and a first working prototype deployment has been set up as the unpacked.cern.ch repository.

References

[1] N Hardi and J Blomer and G Ganis and R Popescu 2018Journal of Physics: Conference Series Making containers lazy with Docker and CernVM-FS

[2] Blomer, Jakob and Buncic, Predrag and Fuhrmann, Thomas 2011CERN-THESIS-2011-251DecentralizedData Storage and Processing in the Context of the LHC Experiments at CERN