Efficient and fast container execution using image snapshotters

Max Fatouros[1], *Derek* Feichtinger[1], *Clemens* Lange[1,*], *Jakob* Blomer[2], *Amal* Thundiyil[2], and *Valentin* Völkl[2,**]

[1]Paul Scherrer Institute, Forschungsstrasse 111, 5232 Villigen PSI, Switzerland
[2]CERN, Esplanade des Particules 1, 1211 Meyrin 23, Switzerland

Abstract. A large fraction of computing workloads in high-energy and nuclear physics is executed using software containers. For physics analysis use, such container images often have sizes of several gigabytes. Executing a large number of such jobs in parallel on different compute nodes efficiently, demands the availability and use of caching mechanisms and image loading techniques to prevent network saturation and significantly reduce startup time. Using the industry-standard containerd container runtime for pulling and running containers, enables the use of various so-called snapshotter plugins that "lazily" load container images. We present a quantitative comparison of the performance of the CVMFS, SOCI, and Stargz snapshotter plugins. Furthermore, we also evaluate the user-friendliness of such approaches and discuss how such seamlessly containerised workloads contribute to the reusability and reproducibility of physics analyses.

1 Introduction

Containers are executable units of software that package applications along with their dependencies, libraries, and runtime environment. A container image serves as a blueprint for containers, encapsulating everything needed to run the software [1]. For the sake of this study, only container images following the Open Container Initiative (OCI) image specification [2] are considered since this is the de facto industry standard. One significant challenge in using containers is that downloading container images accounts for 76% of the container startup time, even though, on average, only 6.4% of the fetched data is needed for the container to start performing productive tasks [3].

To address the inefficiencies in container startup times, lazy-pulling emerged as a promising solution. Lazy-pulling improves resource utilisation and reduces startup delays by deferring the download of container image data until it is actually required for execution. To manage the filesystem state of container images, so-called image snapshotters are used. Modern container runtimes, such as containerd—widely recognised as the most adopted runtime for managing container lifecycles [4, 5]—provide an extendable architecture through plugins. By using "snapshotter plugins", lazy-pulling can be enabled for containerd. This work extends prior research [7] that compares the performance of the CVMFS [8, 9] and Stargz

*e-mail: clemens.lange@cern.ch
**e-mail: valentin.volkl@cern.ch

[10, 11] snapshotters with the default OverlayFS snapshotter, by evaluating the usability and performance of three prominent snapshotters: CVMFS, Stargz, and SOCI [12].

2 Methods

2.1 Container Images and Workloads

Two container images commonly used in high-energy physics (HEP) analysis, a ROOT and a Python image, are evaluated in the benchmarks. ROOT, an open-source data analysis framework [13], includes a C++ interpreter and Python bindings, and serves as the foundation for many HEP software frameworks. The evaluated workloads include running the bash shell within the ROOT container, printing "Hello World" using the Python interpreter, importing ROOT within Python via PyROOT, and executing the fill random function [14] as a typical ROOT use case.

The Python container image is widely used in scientific computing, particularly for scripting and automation tasks in HEP analysis. The workload evaluated in this case involves printing "Hello World" using the Python interpreter.

2.2 containerd Snapshotter Plugins

This work compares the following containerd snapshotter plugins:

- OverlayFS snapshotter: This is the default, non-lazy-loading snapshotter [15].
- CVMFS snapshotter: This snapshotter assembles all the layers of container images from CVMFS (the CernVM File System) into a stacked file system that containerd can use [9].
- Stargz snapshotter: This snapshotter uses the Stargz image format [16] and uses HTTP range requests to fetch specific file entries on-demand [11].
- SOCI snapshotter: It uses the original, unaltered OCI image and creates a distinct index artifact (the "SOCI index") that coexists with the image in the remote registry. The SOCI snapshotter checks the registry for the SOCI index at container launch time for lazy-loading [12].

2.3 Benchmarking Procedure

The benchmarking process utilizes nerdctl [17] to execute each workload with the snapshotter plugins, including OverlayFS, Stargz, SOCI, and CVMFS. Performance metrics such as creation, execution, and runtime durations are extracted by parsing log files, while data transfer patterns are analysed using tcpdump [18] to monitor network traffic. Caches are cleared between runs for containerd, all snapshotters, and for the CVMFS repository.

Measurements are performed with one machine acting as a client to pull, create, and run containers with nerdctl, while another machine on the same network acts as a server to host the image registry and a Squid proxy [19] for CVMFS. The Squid proxy is pre-cached before each benchmark to ensure the data we pull is available on our proxy, and does not have to be pulled again from the CVMFS server.

2.4 Benchmarking Hardware

The client machine has an Intel® Core™ i5-4670 CPU with 8 GiB of DDR3 RAM at 1600 MHz. The server has an Intel® Xeon™ E3-1246 CPU with 8 GiB of DDR3 RAM at 1333 MHz. Both machines run Ubuntu 24.04 LTS.

3 Results

3.1 Performance

3.1.1 Python

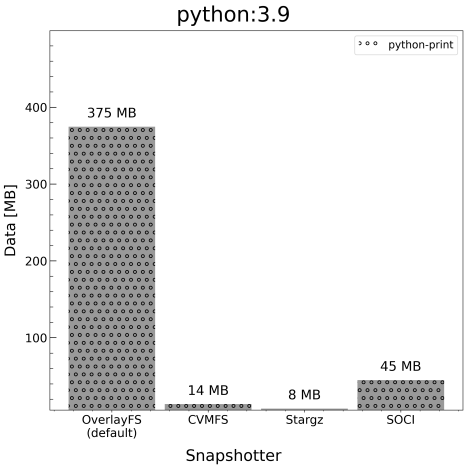

Figure 1. Comparison of data transferred for each snapshotter using the Python image.

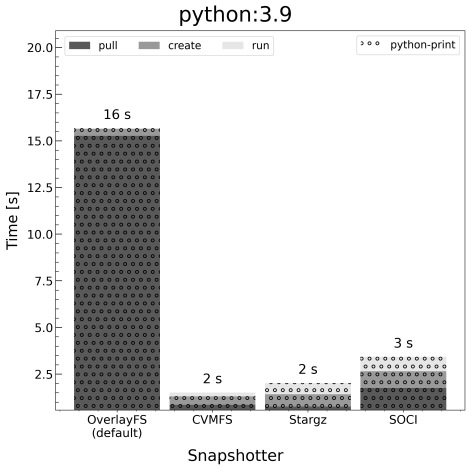

Figure 2. Comparison of time taken for each snapshotter using the Python image.

The time required to start container images is drastically reduced for all lazy-pulling snapshotters. Only a few megabytes of data are downloaded. The SOCI snapshotter loads more data due to its minimum layer size requirement of 10 MB (configurable), below which layers are not lazily pulled. Similarly, the CVMFS snapshotter has a small data overhead from downloading the CVMFS catalogue.

3.1.2 ROOT

The performance of the evaluated snapshotters is comparable in terms of data downloaded. The `import ROOT` operation results in the loading of a large amount of data. The CVMFS snapshotter faster than the other two lazy-pulling snapshotters. For more complex workloads, the time spent pulling the image is negligible compared to the overall execution time. However, significant data savings should be considered in evaluating the overall efficiency.

3.1.3 Note on Client-Server Configuration

As mentioned in Section 2.3, the benchmarks are configured to have one machine on the local network act as a server to host the image registry and also act as the Squid proxy for CVMFS. In this way, network latency and throughput effects can largely be ignored. When using an image registry in a different/remote network, we find that the SOCI and Stargz snapshotters take longer than indicated in Figures 2 and 4.

To mitigate this effect, the SOCI and Stargz snapshotters could also be configured to use a local/close-by proxy or pull-through registry. In case of the CVMFS snapshotter a Squid proxy, which has the same effect of caching data from remote servers, is used by default. The configuration described in Section 2.3 is chosen in an attempt to give an equal comparison of the snapshotters.

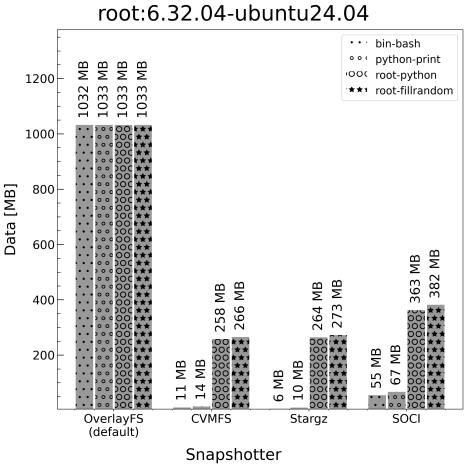

Figure 3. Comparison of data transferred for each snapshotter over different workloads using the ROOT image.

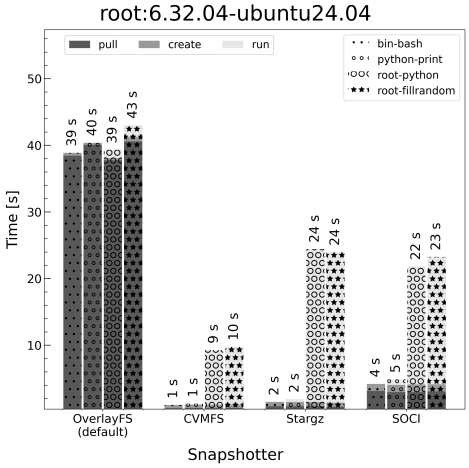

Figure 4. Comparison of time taken for each snapshotter over different workloads using the ROOT image.

3.2 Usability

The usability and adoption of snapshotters introduces varying levels of complexity based on their requirements and compatibility. For use in physics analysis, it is important that the additional effort required by the user is low and potential delays between code changes/commit and image availability are minimal.

The Stargz snapshotter necessitates the conversion of container images into a specific Stargz format programmatically. While this optimization enhances performance, it may pose adoption challenges due to the additional image preparation steps required. However, this can be largely hidden from the user when automatically building images in continuous integration systems.

The SOCI snapshotter introduces minimal modifications to existing container images, making it relatively straightforward to adopt. The possibility of adding the SOCI index after image creation makes it a very flexible approach, which can furthermore be automated. However, the snapshotter's functionality is currently constrained by registry support for additional image artifacts; for instance, it is compatible with the Harbor registry [20], but not with the GitLab registry [21].

The CVMFS snapshotter relies on unpacked images, which introduces a delay between image creation and availability. A dedicated infrastructure needs to be set up for unpacking and distributing the images. Computing systems used in particle physics typically have CVMFS available. Work is ongoing to speed up and simplify the unpacking process.

Very recently, Docker has added support for containerd as a content store [6]. This feature was introduced in Docker-Engine version 24.0 (released in May 2023), and allows completely transparent use of the CVMFS snapshotter with common docker commands. The following minimal configuration is sufficient to enable the use of the snapshotter in Docker on system where it is available:

Listing 1. Docker configuration for the CVMFS snapshotter

```
#/etc/docker/daemon.json
{
    "storage-driver": "cvmfs-snapshotter",
    "features": {
        "containerd-snapshotter": true
    }
}
```

4 Conclusions

Container image snapshotters present new opportunities for improving image distribution and access. The evaluation revealed the following findings:

- Bandwidth and data savings: significant reductions in bandwidth usage and data transfer were observed, with savings exceeding 65% in some cases.
- Time savings: The time savings achieved by using snapshotters varied depending on the specific workload and image characteristics.

In terms of usability and requirements, all evaluated snapshotters (CVMFS, SOCI, and Stargz) have distinct advantages and limitations. However, performance across the snapshotters was generally similar, with the CVMFS snapshotter showing a slight performance edge over the other two methods evaluated.

References

[1] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, Vol. 2014, No. 239, pp. 2, 2014.

[2] Open Container Initiative, 2024 [Online]. Available: https://opencontainers.org/. [Accessed: Dec. 2024].

[3] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy Docker containers," in *14th USENIX Conference on File and Storage Technologies (FAST)*, 2016, pp. 181—195.

[4] Datadog, *The Container Report*, 2023. [Online]. Available: https://www.datadoghq.com/container-report/. [Accessed: Dec. 2024].

[5] Containerd, "containerd," GitHub, 2025. [Online]. Available: https://github.com/containerd/containerd/tree/v1.7.12. [Accessed: Oct. 2024].

[6] Docker Engine, "docker," 2024. [Online]. Available: https://docs.docker.com/engine/release-notes/24.0/. [Accessed: Oct. 2024].

[7] S. Mosciatti, C. Lange, and J. Blomer, "Increasing the execution speed of containerized analysis workflows using an image snapshotter in combination with CVMFS," *Frontiers in Big Data*, Vol. 4, No. 673163, pp. 1–10, 2021. doi:10.3389/fdata.2021.673163.

[8] J. Blomer, C. Aguado Sanchez, P. Buncic, and A. Harutyunyan; S. C. Lin (editor), "Distributing LHC application software and conditions databases using the CernVM file system," J. Phys. Conf. Ser. Vol. 331, pp. 042003, 2011. doi:10.1088/1742-6596/331/4/042003.

[9] J. Blomer et al., "The CernVM File System: v2.12," Oct. 2024, Zenodo, doi:10.5281/zenodo.4114078.

[10] K. Tokunaga, "Lazy pulling with eStargz," *Medium*, 2020. [Online]. Available: https://medium.com/nttlabs/lazy-pulling-estargz-ef35812d73de. [Accessed: Dec. 2024].

[11] Containerd, "Stargz Snapshotter," GitHub, 2024. [Online]. Available: https://github.com/containerd/stargz-snapshotter/tree/v0.15.1. [Accessed: Oct. 2024].

[12] AWS Labs, "SOCI Snapshotter," GitHub, 2025. [Online]. Available: https://github.com/awslabs/soci-snapshotter/tree/v0.8.0. [Accessed: Oct. 2024].

[13] R. Brun and F. Rademakers, "ROOT: An object-oriented data analysis framework," in *Proceedings of the AIHENP 1997 Workshop*, 1997.

[14] CERN, "fillrandom.py," *ROOT Documentation*, [Online]. Available: https://root.cern/doc/v608/fillrandom_8py.html. [Accessed: Dec. 2024].

[15] Containerd, "Snapshotters," *Containerd Documentation*, 2023. [Online]. Available: https://github.com/containerd/containerd/blob/main/docs/snapshotters/README.md. [Accessed: Dec. 2024].

[16] Google, "CRFS: Container Registry Filesystem," GitHub, 2023. [Online]. Available: https://github.com/google/crfs. [Accessed: Dec. 2024].

[17] Containerd, "Nerdctl: Docker-compatible CLI for containerd," GitHub, 2024. [Online]. Available: https://github.com/containerd/nerdctl. [Accessed: Dec. 2024].

[18] tcpdump, "tcpdump: A Powerful Command-Line Packet Analyzer," tcpdump.org, 2024. [Online]. Available: https://www.tcpdump.org. [Accessed: Dec. 2024].

[19] Squid, "Squid: Optimising Web Delivery," squid-cache.org, 2024. [Online]. Available: http://www.squid-cache.org. [Accessed: Dec. 2024].

[20] Harbor registry, 2024 [Online]. Available: https://goharbor.io/. [Accessed: Dec. 2024].

[21] GitLab container registry, 2024 [Online]. Available: https://docs.gitlab.com/user/packages/container_registry/. [Accessed: Dec. 2024].