Performance Analysis of Containerized OrangeFS in HPC Environment.

Izzet Yildirim, Meng Tang, Anthony Kougkas, and Xian-He Sun
Illinois Institute of Technology, Chicago
{iyildirim,mtang11}@hawk.iit.edu,{akougkas,sun}@iit.edu

1 EXTENDED ABSTRACT

Due to their ability to create predictable environments isolated from other applications and flexibility on deployment, and given the fact that they are able to run virtually everywhere, containers have been widely adopted in cloud solutions. Particularly, data analysis and machine learning applications have benefited from containerization. And since such applications' moving to high performance computing (HPC) environments, the need for container support has become preeminent. Hence, containers face crucial performance and I/O challenges in HPC setups. However, given the new challenges they face in such setups, their impact and performance have not been thoroughly investigated, especially from the I/O operations point of view.

Our aim is to fill this gap by providing an empirical analysis of some of the widely used container solutions in HPC environments. We also examine how containers interact with the OrangeFS parallel storage system in an HPC environment. In this endeavor, we make use of the Chameleon experimental platform. We detail the design and implementation of our analysis setup that captures CPU, memory, network, and I/O statistics from the nodes. And we present the key insights from our analysis, e.g. how Singularity is better suitable for such environments in terms of performance and mobility, how Docker falls behind in HPC-related tasks despite its popularity in the container world. Our evaluation shows that containers that are designed specifically for HPC environments produce promising results that indicate that they can be safely employed for I/O intensive HPC applications too.

2 CONTAINERS & ORANGEFS

Containers offer different virtualization aspects than that of other virtualization methods like KVM which explicitly virtualizes the hardware and runs separate kernels. Without the heavy resource utilization of a VM, containers that are running on the same host share the same kernel and are able to provide isolation through cgroups and namespaces. As a result, they consume less resources and allow the host to run many more containers in parallel than if it were to run VMs. Figure 1 shows the difference on the architecture of both approaches and how VMs place a guest OS on top of the virtualization layer.

To provide the ability to perform I/O on a directory on the host, containers support bind-mounting. When used, a bind-mount mounts a file or directory on the host machine into the container. In fact, this particular ability is the prime source of inspiration for this study.

In addition to isolation and the ability to perform I/O on a host directory, one of the most praised benefits of containers is the simplicity of packaging the application and all of its dependencies in a readily deployable format. This is specifically advantageous for I/O intensive HPC applications where faulty nodes have to be easily recovered, and downtimes are not in any way welcomed.

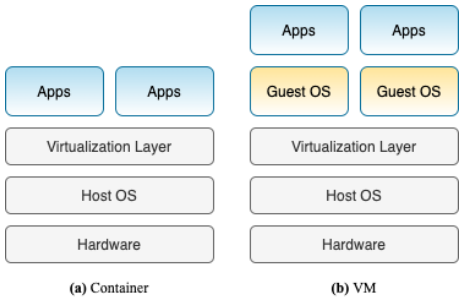

Figure 1: Architectures of containers and VMs

Our solution consists of the following components:

(1) **Docker**: is undoubtedly the most popular container solution. It quickly became unrivaled due to being a better and more efficient solution for providing isolation and mobility for applications in the cloud computing setting. However, when security concerns, the necessity for root user privileges for most of the tasks, and the lack of distributed storage integration are considered, Docker's further adoption in HPC becomes questionable.
(2) **Singularity**: is specifically crafted for the HPC ecosystem. [1] It leverages bind mounts for mapping host directories inside the container, as well as user namespaces and mapping users to specific containers without root access. Although such capabilities may look similar to that of Charliecloud, it differs in wrapping all images in a single file and therefore providing ease of deployment across different resources.
(3) **LXC**: is one of the early examples of containers and was developed around 2008. It is built on top of the idea of layering user space tooling via cgroups and namespaces, similarly to container solutions that came after. Considering the fact that it provides operating system-level virtualization, we want to examine how it performs along with newer container solutions.
(4) **Podman**: is a container engine without a daemon for developing, managing, and running containers. What makes Podman suitable for HPC environments is that it can run containers either as root or in rootless mode.
(5) **Charliecloud**: is designed to handle the workload on HPC clusters [2]. It comes with an ability to make use of the Docker images by converting them into their own image format. And since Charliecloud utilizes user namespaces to avoid requiring root privileges, it provides a quick solution for HPC users who want to migrate to an HPC-focused container from Docker.