International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 01 | Jan 2022 www.irjet.net p-ISSN: 2395-0072

Understanding Migration Mechanisms of Containers using CRIU

Avijit Kumar Dash

Student, Dept. Of Computer Science and Engineering, Indian Institute of Technology Bombay, Powai, Mumbai, Maharashtra 400076

Abstract - *Operating system-level virtualization has grown in popularity due to isolating multiple user-space environments. The container is one of the most popular virtualization environments. Containers are lightweight virtualization technology compared to Virtual Machines and have seen widespread adoption in recent years. Container-type virtualization can run many isolated processes, identified as containers, under a single kernel instance. Such isolation presents the opportunity to save the whole state and restart it later. Checkpointing itself can be used for various migration techniques. This report presents the survey reports of different checkpointing and restarts components for containers.*

Key Words: **Virtualization, Container, Live Migration, Cold Migration, CRIU,**

1.INTRODUCTION

Virtualization techniques are now extensively applied in today's data centers. Virtualization is a technology that enables one to build multiple virtualized environments or reserved resources from a particular physical hardware system. It refers to running concurrent operating systems in a single environment. Virtualization has allowed the commoditization of cloud computing. The hardware resources have become possible to run various environments and share computing resources amongst multiple enterprises. Two standard technology types are used in virtualization: hypervisor-based virtualization and container-based virtualization.

1.1 Why Virtualization is needed

1.1.1 Resource Distribution

Sometimes, there is a need to limit the set of resources a group of processes can access to execute without any problem. Limiting resources include the size of memory, the number of CPU cores, set of network interfaces and devices, or other similar resources.

1.1.2 Migration

Virtualized machines are distributed among clusters of several physical machines within data centers and other related approaches. These Virtualized machines must be distributed among the devices to ensure optimal use of resources and avoid bottlenecks. Sometimes, the present device needs to shut down for maintenance work, so moving the ongoing execution to another device is beneficial. Because Virtualized machines run on abstracted hardware and do not directly communicate with the host OS, so it is possible to migrate them to another machine transparently.

2. BACKGRAOUND

Containers are a lightweight virtualization technology. The container is a standard package of software that contains all of the necessary components to run in any environment. A container is illustrated in Figure 1.

The significant difference between VMs and containers is that when VM needs the complete copy of the operating system (OS), containers do not need an entire OS to be installed within the container to operate; instead, they share the host OS kernel. Containers can run with a minimum number of resources to perform the task.

Two mechanisms are necessary to build a container. Those are:

(1) Namespaces

(2) Cgroups

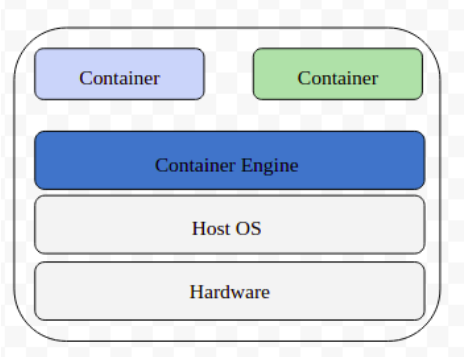

Fig-1: Container

2.1 Namespaces

Namespaces [1] are one of the principal characteristics of the Linux kernel; they carry out the separation between kernel resources. It makes sure that a process can only see the particularized collection of resources. Namespaces are global system resources that create an illusion that they have their isolated instance of the global resource within the namespace. Six namespaces are out there. Namely:

1. Mount namespace
2. PID namespace
3. Network namespace
4. UTS namespace
5. User namespace
6. IPC namespace

2.2 Cgroups

Cgroups are a way by which it is possible to set resource limits on a group of processes. By taking help from Cgroups, it is possible to divide processes into groups and subgroups hierarchically and assign resource limits for each group/subgroup.

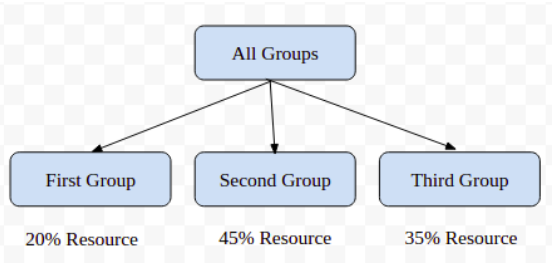

Figure 1: Cgroup resource allocation

2.3 Container in practice

Several tools have been developed over the years for container creation, deployment, and management. Linux Containers (LXC) [3] is a method to build privileged and unprivileged containers using the command line. LXD is a project that uses LXC for a more user-friendly and advanced container experience. Docker [4] is a container engine with its own build and packaging systems. It is provided towards the creation of application containers. Docker previously used LXC for its backend, but then it shifted to its implementation called libcontainer [5]. Docker uses a union mount filesystem where the storage is created from a stack of image files.

2.4 Container migration using CRIU

CRIU [6] is a Checkpoint-Restore tool in Userspace. Using the CRIU tool, migrating a running container or a process from one node to another is possible. It is possible to dump a container using CRIU and use it later, on the same machine or any other machine and can be resumed where it left off.

CRIU needs to freeze the application or group of processes to be migrated before they can be checkpointed. In the meantime, the application or the group of processes should not be aware of the checkpointing and restoring procedure; it must be migrated transparently. The cgroup freezer [7] subsystem uses to make the freezing system transparent. The cgroup freezer uses the kernel's group of freezer functions to pause the cgroup resources temporarily.

Once the application freezes, it begins extracting its state and memory and dumping them to image files. The state is read from the /proc file system. This set of procedures is completed iteratively for the whole process group by obtaining the process's threads in /proc/PID/tasks and their children in their child directories.

CRIU injects a parasite code into the process to collect memory contents. A memory mapping procedure is used to map areas and dump pages into a pipe using vmsplice and then written to an image file using splice. The CRIU restorer process shares parameters with the parasite code by opening the shared memory area allocated by the mmap.

The restore process is straightforward. The restorer process forks the entire process hierarchy by reading all dumped image files and finding out which processes share which resources. Then, the namespaces are restored, besides sockets are also opened. It recreates all processes in the tree by calling fork (). Memory contents are written into a separate address space from the images. Memory data map into its correct position by restorer code before the checkpoint. Lastly, timers and threads are restored.

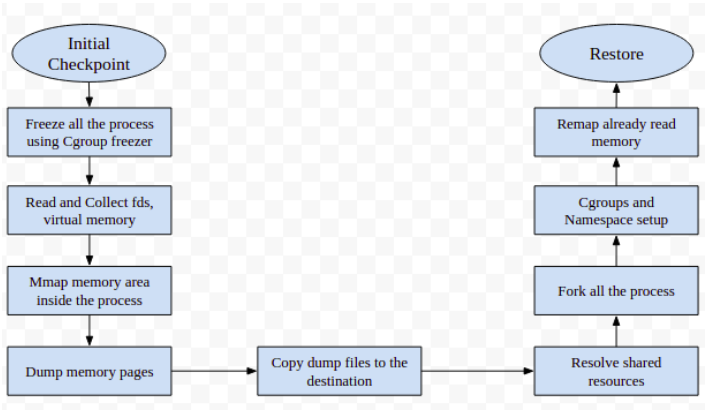

FIg-2: CRIU checkpoint/restore process flowchart

Migration with CRIU has some limitations:

1. One common limitation is that CRIU can only checkpoint and restore processes using inter-process communication (IPC).
2. Existing parent-child connections in process trees must be kept intact.

3. CONTAINER MIGRATION

3.1 Checkpointing and Restore procedure

The checkpointing and restore method is started from the user level, but it is implemented at the kernel level. Thus, it provides transparency of the checkpointing and restoring process. Checkpoint and restore consist of three major stages (i) Pre-dump (Mostly used in Live migration) (ii) Dump. (iii) Restore.

I. **Pre-Dump:** Pre-dumps only contain the memory images. At this phase, criu sends memory pages to the destination node, but it does not stop the container while doing this stage. Pre-dump can be performed many times.

II. **Dump:** The dump phase stopped the application and saved the application's state to restart later.

III. **Restore:** At this stage container or process can be restored that is already checkpointed.

Pre-dump vs. Dump

I. It is possible to perform more than one Pre-dump for a container or process, but dump can be achieved only once.

II. Pre-dump only contain memory contents on the other hand dump save the entire state of the container or process.

The checkpointing and restoring procedure consists of the following stages [9]:

3.1.1 Checkpointing Procedure

I. **Freeze processes:** Freeze all the processes of the container and lock the network.

II. **Dump the container:** Gather and preserve the entire state of the container's processes and dump the container into an image file.

III. **Stop the container:** Suspend executing the container processes.

3.1.2 Restore procedure

I. **Settle shared resources:** At first, CRIU reads all the img files and determines which processes share which resources. Then map those files again for the next step.

II. **Fork the process tree:** At this step, CRIU fork () all the processes tree before restoring.

III. **Restore the container:** Restore the container with the same state as earlier saved in a dump file.

IV. **Restore all processes:** Restore all the processes inside the frozen container.

V. **Resume the container:** Finally, resume execution of the container and unlock the network. After that, the container continues its normal execution.

All the container's processes should be saved to a consistent state. All process dependencies should be saved and restored during restore. Dependencies include the process hierarchy (Figure), identifiers (PGID, SID, etc.), and shared resources. During the restore, all such resources and identifiers must be set accurately.

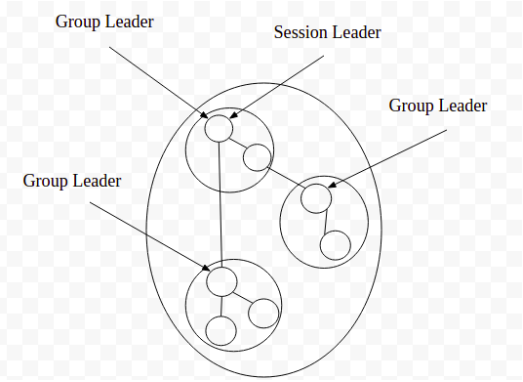

Fig-3: Process Hierarchy

3.1.1 Live Migration

The live migration refers to Checkpoint, a running container in one node and transfer to another node and restart without rebooting on the source node. The main concern of live migration is minimizing downtime. Live migration [9,10,11,12,13] is nothing more than checkpointing the container process or the group of processes, transferring it to a destination system, restoring the process, and resuming it when it is on active mode. There are several types of live migration proposed over the years. This is also known as iterative live migration [14]. The most common types of them are:

I. **Pre-Copy Migration:** Pre-copy migration [10,15] is begun by copying the memory state to the goal side. Pre-copy migration transfers most of the state before freezing the container for the last dump. It is also identified as iterative migration since it may execute the pre-copy phase through many iterations. As the pre-dump phase is conducted in many iterations, some pages might be modified during this phase. The changed memory pages are called dirty pages. The pre-copy step usually terminates when a destined number of iterations has arrived. After that, the container is discontinued on the origin node to capture the last dirty pages and copy them at the goal node without the container changing the

state again. After transferring all the pages, the container started to run on the goal node.

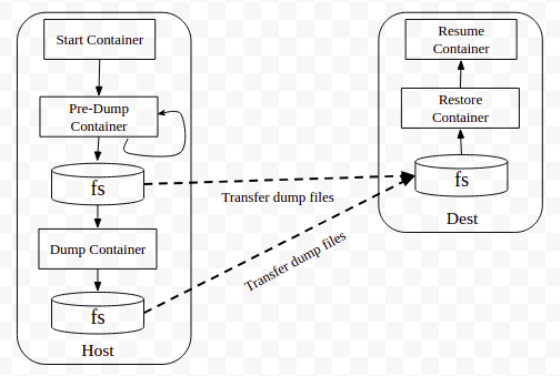

Fig-4: Pre-copy Migration procedure

II. **Post-copy:** Post-copy migration is the reverse process of pre-copy migration. Post-copy migration [10,15] begins by suspending the migrating container at the origin side and then copies the minimal processor state to the target side, resumes the container, and brings memory pages over the network from the origin. Each memory page is copied only once in post-copy migration.

III.

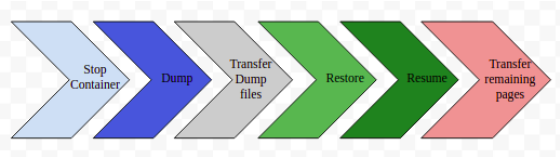

Fig-4: Post-copy Migration procedure

3.1.2 Cold Migration

In this technique [9], the source node freezes the container or group of processes and dumps it. Then all the dump files need to transfer to the destination node. This transfer is done using scp or rsync. After completing the transfer procedure, the container or the group of processes is restored and resumes on the destination side. This migration technique is known as the cold migration technique because it:

I. First, freeze the container and make sure that modification is not possible after freezing the state.

II. Now dumps the entire state and transfers it.

III. Finally, restore the container at destination and resume all its states.

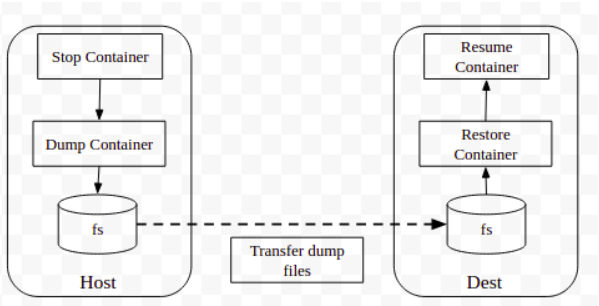

Fig-6: Cold Migration Procedure

3.1.3 Image Cache/Proxy

Two auxiliary processes are run in this procedure, one on either side of the migration process, namely, Image-proxy (at source) and Image-cache (at the destination) [10]. These processes perform over a TCP connection.

I. **Image-proxy:** This presents an in-memory cache where the entire process snapshot dump from the CRIU process is stored instead of being stored as image files on the disk. This process is responsible for pushing the snapshots to the destination over the network.

II. **Image-cache:** It gets the snapshots and stores them in memory until the CRIU restore process is ready to fetch them to recreate the process group at the target. Another advantage of these components, apart from avoiding disk read/write it provide for concurrent transferring snapshots to the destination with the snapshot creation process. It is also thus possible to start the restoration process at the target before the snapshot process has even concluded at the source.

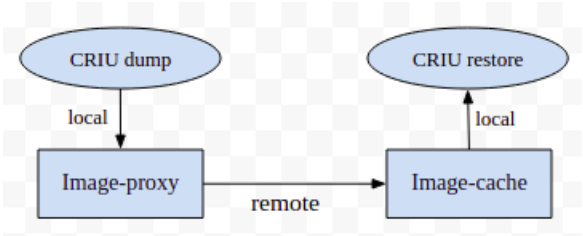

Fig-7: Image Cache/Proxy Migration Procedure

Table -1: Migration techniques and their procedure.

Migration Techniques	Procedure Followed
Live Migration	I. Pre-copy II. Post-Copy
Cold Migration	Stop and Copy

Image Cache/Proxy	No persistent Storage needed

4. CONCLUSIONS

Container migration is necessary in many cases, such as if the host machine started to work abnormally or needs to be upgraded. CRIU was developed to migrate the running form of a container or a process. It checkpointed a container and stored it checkpointed files as collection image files to persistent storage. Then transfer those files to the destination for restoring the container. Various container migration techniques can be performed with CRIU, such as Live migration, Image-cache/Image-proxy. Though live migration disadvantages, image files must be reserved and regained multiple times from persistent storage. This disadvantage can be eliminated by leveraging the image-cache/image-proxy migration technique. Though image-cache/image-proxy migration technique still needs to be improved to get better performance. When the container migration is performed with higher workloads, it is slower than a container with smaller workloads. The number of CPU cores affects the CPU usage of the checkpoint and restore. If the number of cores is more, CPU usage will be lower. Lastly, Pre-copy and post-copy migrations can be the best choices under different cases.

REFERENCES

1. Overview of Linux namespaces https://man7.org/linux/man-pages/man7/namespaces.7.html
2. Introduce io. latency io controller for cgroups. https://lwn.net/Articles/758697/.
3. Linux Containers. https://linuxcontainers.org/
4. Docker. https://www.docker.com/
5. libcontainer. https://github.com/opencontainers/runc/tree/master/libcontainer
6. Checkpoint/restore in userspace. https://criu.org/Main_Page.
7. Cgroup freezer subsystem. https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt
8. Chen, Yang. (2015). Checkpoint and Restore of Micro-service in Docker Containers. 10.2991/icmii-15.2015.160
9. Live migration. https://criu.org/Live_migration
10. Radostin Stoyanov & Martin Kollingbaum. (2018). Efficient Live Migration of Linux Containers. Conference: 13th Workshop on Virtualization in High -Performance Cloud Computing VHPC '18At: Frankfurt, Germany
11. Simon Pickartz, Niklas Eiling, Stefan Lankes, Lukas Razik, Antonello Monti. Migrating LinuX Containers Using CRIU. International Conference on High Performance Computing, ISC High Performance 2016: High Performance Computing pp 674-684.
12. Andrey Mirkin, Alexey Kuznetsov, Kir Kolyshkin (2010). Containers checkpointing and live migration. In Ottawa Linux Symposium
13. Adityas Widjajarto, Deden Witarsyah Jacob, Muharman Lubis. Live migration using checkpoint and restore in userspace (CRIU): Usage analysis of network, memory and CPU. Bulletin of Electrical Engineering and Informatics, Vol. 10, No. 2, April 2021, pp. 837~847ISSN: 2302-9285, DOI: 10.11591/eei. v10i2.2742
14. Iterative Migration. https://criu.org/Iterative_migration
15. Adrian Reber. https://lisas.de/~adrian/posts/2016-Oct-14-combining-pre-copy-and-post-copy-migration.html
16. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F.; Puliafito, A. Container Migration in the Fog: A Performance Evaluation. Sensors 2019,19, 1488. https://doi.org/10.3390/s19071488
17. Image cache/proxy. https://criu.org/CLI/opt/--remote