Container Live Migration

Adrian Reber
Red Hat

2019

Container live migration has already been implemented by some container runtimes. This article provides details what was necessary to implement container live migration in Podman. The article starts with details about how CRIU manages to checkpoint and restore processes and containers. What changes were necessary to CRIU and all the tools and libraries used by Podman to implement container migration. The article continues with how container live migration can be used in Podman and finishes with an outlook on what features could be implemented next to further improve Podman's container live migration capabilities.

I. Introduction

Container live migration in the context of this document means the process of transferring a running container from one system (source) to another system (destination). One possible way of container live migration is using checkpoint and restore to serialize the container state to transfer it from the source system to the destination system. This document will focus on Checkpoint/Restore In Userspace (CRIU)[1] as the technique to serialize processes. CRIU was first discussed in 2011 on the Linux Plumbers Conference[2] and has been integrated in multiple container runtimes to provide the possibility to live migrate containers.

To migrate a container from one system to another CRIU will pause all processes in the container and write the state and all memory pages the processes in the container are using to its image files. In addition to the image files it is also necessary to transfer the file-system the container is using to the destination system. Once image files and container file-system state are available on the destination system the container can be restarted using CRIU. Depending on the used container runtime CRIU will re-create used namespaces, mount file-systems, configure control groups (cgroups) and restore network connections.

The time when CRIU first pauses all processes in a container on the source system until all processes are running again on the destination system is the container downtime. CRIU offers memory pages pre-copy and post-copy mechanisms which can be used to reduce the container downtime during migration.

II. CRIU Details

The concept of checkpointing and restoring a process exists already much longer than CRIU. It has been implemented in different operating systems and there are also multiple implementations in Linux. Other checkpoint/restore implementations in Linux, however, have different limitations. Either the other implementations require to instrument the runtime environment or they rely on out-of-tree Linux kernel changes. Both approaches have their advantages but also drawbacks (see [3] for further discussions). One of the goals of CRIU has always been to be as transparent as possible so that it is easy to checkpoint and restore any process. This also means that CRIU does not require any specially prepared environment like pre-loading libraries or out-of-tree kernel modules.

Instead CRIU relies on existing Linux kernel interfaces to retrieve all necessary information to checkpoint a process. Not always did the existing interfaces provide CRIU with the necessary information and had to be extended. This extension of Linux kernel interfaces, most of the time, has led to new ways to interact with the Linux kernel not only for CRIU, but has also been useful for other use cases.

i. Checkpointing

CRIU makes heavy use of the information which can be read from `/proc/PID/*`. This includes information about mounts, namespaces, opened files, used memory regions.

The basic steps CRIU does during checkpointing is to use `ptrace()` to pause the task. CRIU then collects all the information about the process and writes it to disk. The result of a checkpoint are different files which include all the information required to restore the process later. This includes, for example, the open file descriptors as well as the content of the process registers for each involved process, as well as all relevant memory pages. Once all the information is written to disk the process can continue to run or the process is terminated.

It is important to know that CRIU always operates on a process tree. The user or the container runtime tells CRIU which process identifier (PID) should be checkpointed and CRIU will checkpoint the given PID and all its child processes. It is not possible (not easily at least) to restore processes with another PID. The parent-child relationship of the restored process tree also needs to be the same during restore as it was during checkpointing. In the container use case with a PID namespace this should not be any problem, but using CRIU without a PID namespace can lead to PID collisions. If one of the to be restored PIDs is already in use, CRIU will abort the restore.

One of the more complex tasks, and at the same time more interesting tasks, is the extraction of all required memory pages. One possible way to get access to the content of a process's memory pages is using `ptrace()`. Extracting a large amount of memory pages out of a process using `ptrace()` is, unfortunately, slow. To be able to get the content of the process's memory out on disk much faster, CRIU uses a parasite code. Once the process is paused parts of the process is replaced with the parasite code and the parasite code starts to run in the process. The parasite code waits for commands from the main CRIU process and thus it is possible to extract information about the process from within the process's address space. Once CRIU has finished checkpointing the process the parasite code is removed and the process never knows that it was under CRIU's control. It keeps on running just as before.

What makes the parasite code concept even more interesting is the container use case in a SELinux environment. When checkpointing a container with Podman[4] the container is labeled with something similar to `container_t`. Podman and the tools it calls out to (like CRIU) are running with the `container_runtime_t` label. This means that CRIU will run with another SELinux context than the process into which CRIU inserted the parasite code. When the parasite code tries to `connect()` to a socket of the main CRIU process to receive its commands, SELinux will deny this connection as a container process labeled with `container_t` tries to connect to something on the outside of that container: CRIU running with `container_runtime_t`.

The implemented solution in CRIU (3.12) is to use `setsockcreatecon()`[5] to tell SELinux to label the parasite code socket with the same label the process in the container is running with. This way the parasite code which is running as `container_t`, will connect to a socket which is also labeled with `container_t`.

In addition to the parasite code socket CRIU must also correctly checkpoint the process context of all involved processes (and threads) as well as other sockets opened by the process in such a way that CRIU can set all these labels correctly on restore.

ii. Restoring

Once a process or container has been checkpointed all required information to restore this process or container is in the *image directory* which has been written by CRIU. The simplest form of migration is transferring this directory to the destination system where the process can then be restored with the help of CRIU. For container migration additional steps are required which are discussed in the next chapter.

A high level description of how CRIU restores a process is that CRIU morphs itself into the processes to be restored. For each process to restore CRIU does a `clone()`[6] to re-create the same process tree as before checkpointing. All file descriptors and sockets will be opened and positioned just as they were before checkpointing. Memory pages are copied from the *image directory* into the newly created processes and they will be re-mapped to the right location just before giving back control to the process being restored.

One of the last changes CRIU does to the processes being restored are security related changes. During restore CRIU runs with as many privileges as possible which are all changed to the state they had during checkpointing just before the restore finishes. This includes *seccomp*, *capabilities* and *AppArmor* or *SELinux*.

Restoring a process with SELinux labels requires to restore the process context as well as the socket context. One of the first unusual things when restoring SELinux process context is that CRIU morphs itself into the restored process. This means that the restored process has to change its own label during restore and changing its own SELinux process context during runtime is not very common for a process. In the container runtime use case this also means that changing the SELinux process context must happen as late as possible, because to restore a process CRIU performs a lot of operations which require much more privileges than a container process should have.

Once all security related confinements have been restored CRIU gives the control back to the original process and all processes continue running on the destination system just as they were before starting the whole migration.

III. Container Runtime Integration

CRIU based container migration exists already for some time in different container runtimes. The following is about the integration of the possibility to migrate a container with Podman.

The first Podman changes towards container migration were integrated into Podman in October 2018[7][8]. This initial support was, however, only to checkpoint and restore containers on the same host. No migration yet.

To support checkpointing and restoring containers with Podman required a few changes to runc[9] and CRIU. Other container runtimes supporting container migration with the help of CRIU delegated the task to re-create all necessary namespaces to CRIU. For the network namespace Podman, however, uses Container Network Interface (CNI)[10]. So the first step was to teach CRIU and runc[11] to restore a container into an existing network namespace. For the initial start of a container as well as for container restore, Podman lets CNI create a new and completely configured network namespace and in the case of a restore CRIU restores the container into that network namespace.

Once CRIU, conmon[12] and runc were able to handle checkpoint/restore including external network namespaces, the actual Podman changes for checkpoint/restore were merged.

At this point, however, it was still not possible to migrate a Podman container. The initial idea was that once CRIU has written the checkpoint to the *image directory* it should be easy to implement migration. Take the *image directory*, copy it from the source to the destination system and restore the container. Unfortunately that did not work. Using Podman's initial checkpoint/restore implementation, Podman still had metadata of the checkpointed container. This metadata easily survived a reboot of the system, but not migration. So before telling CRIU to restore the container Podman needs to re-create the missing container metadata to be able to restore it.

To provide an easy user interface the actual checkpoint data as well as the container metadata is exported in one single file, which can be easily transferred from the source to the destination system. To checkpoint and export a container in Podman following command is needed:

```
# podman container checkpoint -l
-e checkpoint.tar.gz
```

Copying the file `checkpoint.tar.gz` from the source to the destination system it can be used by Podman to restore a container. Thus migrating the container:

```
# podman container restore -i
checkpoint.tar.gz
```

The container metadata includes the container ID, the container name, the container IP address and which container images needs to be downloaded from the registry, if that container is not yet available locally. During the restore Podman uses the metadata to re-create the container and then CRIU can restore the processes into this newly set up container.

Currently this is limited to containers which do not change their file-system. Having a read-only container file-system makes sure that nothing can be written to the container. If a container with a modified file-system is restored it can either fail immediately or at some later time when the processes in the container are trying to access files which have the wrong content or do not exist.

This also uncovered an interesting runc behaviour. Even for read-only container file-systems, runc creates missing mount-points, which is unexpected for a read-only container file-system. On container restore, however, runc did not modify the file system. This means that it was not possible to restore a container from a freshly pulled container image, even if it was exactly the same as used during checkpointing. After some discussion[13] a runc change was merged to have the same behaviour during restore as on initial container creation.

With the necessary changes merged into runc it was possible to implement container migration in Podman[14] as described above. The most difficult part was to make sure everything works correctly in combination with SELinux. Restoring a container with CRIU and Podman required additional changes, especially in CRIU, to make sure the restored container has the same SELinux properties as during checkpointing. Once CRIU was able to handle SELinux correctly it was also necessary to adapt the different involved SELinux policies which required additional coordination.

In addition to the examples of how to migrate a container with Podman above, it is also possible to restore a container with a different name. This enables restoring a container multiple times on multiple systems, which could be useful for container which require a long time to start.

Start the container once and wait for a long running initialization to finish. Tell Podman to do a checkpoint of the container and export it. The exported checkpoint can then be used to start the container multiple times with different names much faster as all complicated initializations have already been performed and startup time is reduced to the time CRIU needs to copy the memory pages to the right locations.

The implemented container migration support in Podman can now be used to actually migrate containers by copying the exported checkpoint archive to the destination system and tell Podman to restore the container from that checkpoint archive. It can also be used to create multiple copies of already running containers which can be especially useful for containers which require a long time to start up.

IV. Optimizations

The current implementation has the limitation, that it cannot handle changes being done to the container's file system. Either the container has to have a read-only file-system or changes have to be written to a `tmpfs` as CRIU will be able to migrate the content of all in the container mounted `tmpfs`. To open up container migration also for container file-systems which are changing during runtime, Podman needs to be able to export all file-system related changes which have been made to the file-system on top of the container image which was used to start the container.

This optimization is mainly to make it easier for users to migrate a container. Right now the user has to know that the file-system cannot change or the migration might fail. The expectation is, that this should be easy to implement as a next step.

Other possible optimizations are to decrease the container downtime during migration. CRIU provides the possibility to use pre-copy migration, post-copy migration as well as the combination of pre-copy and post-copy migration. These concepts have already been discussed[3] and CRIU has all the necessary functionality already implemented.

V. Conclusion

One of the main conclusions is probably that implementing container migration in Podman took a lot of time. Getting container migration working required changes in many different components: CRIU, runc, conmon, selinux-policy, container-selinux and Podman. Most of the time was not spent on the actual implementation but, in this case, on coordination and waiting. Once it was clear what had to be done it required the actual code changes and getting these necessary changes included into the corresponding upstream's, which most of the time required some discussions. Then it was necessary to wait until these changes are available to the other involved projects and then additional changes were no longer blocked. So a lot of communicating, coordinating and waiting was required. All involved projects were always helpful and open to any of these changes, but it still required time.

With the current implementation it is possible to migrate containers which do not change their file-system and this initial implementation of container live migration allows the possibility to add support for containers which are changing their file-systems. This implementation also allows to improve container live migration with the help of CRIU's support for pre-copy and post-copy process migration. This implementation also opens the possibility to implement container live migration in higher layers which are using Podman to control containers.

References

[1] *Checkpoint/Restore In Userspace (CRIU).* [Online; accessed 2019-05-03]. URL: `https://criu.org/`.

[2] *Checkpoint/restart in the userspace.* [Online; accessed 2019-05-03]. URL: `http://blog.linuxplumbersconf.org/2011/ocw/sessions/831`.

[3] Adrian Reber. "Process migration in a parallel environment". In: (2016). URL: `http://dx.doi.org/10.18419/opus-8791`.

[4] *Podman.* [Online; accessed 2019-05-03]. URL: `https://podman.io/`.

[5] *setsockcreatecon(3).* [Online; accessed 2019-05-03]. URL: `http://man7.org/linux/man-pages/man3/setsockcreatecon.3.html`.

[6] *clone(2).* [Online; accessed 2019-05-03]. URL: `http://man7.org/linux/man-pages/man2/clone.2.html`.

[7] *Add support to checkpoint/restore containers.* [Online; accessed 2019-05-03]. URL: `https://github.com/containers/libpod/pull/469`.

[8] *Adding checkpoint/restore support to Podman.* [Online; accessed 2019-05-03]. URL: `https://podman.io/blogs/2018/10/10/checkpoint-restore.html`.

[9] *runc.* [Online; accessed 2019-05-03]. URL: `https://github.com/opencontainers/runc`.

[10] *Container Network Interface - CNI.* [Online; accessed 2019-05-03]. URL: `https://github.com/containernetworking/cni`.

[11] *Add support to checkpoint and restore into external network namespaces.* [Online; accessed 2019-05-03]. URL: `https://github.com/opencontainers/runc/pull/1849`.

[12] *conmon: add support to restore a container.* [Online; accessed 2019-05-03]. URL: `https://github.com/cri-o/cri-o/pull/1427`.

[13] *Create bind mount mountpoints during restore.* [Online; accessed 2019-05-03]. URL: `https://github.com/opencontainers/runc/pull/1968`.

[14] *Add support to migrate containers.* [Online; accessed 2019-05-03]. URL: `https://github.com/containers/libpod/pull/2272`.