Container Migration Around The World

October 12, 2017 | Adrian Reber

< Back to all posts Tags: *Infrastructure*

In this article I want to talk about a runC container which I want to migrate around the world while clients stay connected to the application.

In my previous Checkpoint/Restore In Userspace (CRIU) articles I introduced CRIU (From Checkpoint/Restore to Container Migration) and in the follow-up I gave an example how to use it in combination with containers (Container Live Migration Using runC and CRIU). Recently Christian Horn published an additional article about CRIU which is also a good starting point.

In my container I am running Xonotic. Xonotic calls itself 'The Free and Fast Arena Shooter'. The part that is running in the container is the server part of the game to which multiple clients can connect to play together. In this article the client is running on my local system while the server and its container is live migrated around the world.

This article also gives detailed background information about

my upcoming talk at the Open Source Summit Europe: Container Migration Around The World.

Container Setup

The first step in my setup is to install the necessary files into the container. The systems involved in the container migration are using Red Hat Enterprise Linux 7.4. As Xonotic is part of Extra Packages for Enterprise

Linux (EPEL) the EPEL repository has to be enabled and then the following command is necessary to install the container:

```
#  mkdir -p /runc/containers/xonotic/rootfs
# yum install --releasever 7.4 --installroot /runc/containers/xonoti
c/rootfs xonotic-server
```

Once the files for the container are installed the container configuration needs to be created. Just as in my previous article I am using oci-runtime-tools to generate the configuration and therefore the second step is to install *oci-runtime-tools*:

```
# export GOPATH=/some/dir
# mkdir -p $GOPATH
# go get github.com/opencontainers/runtime-tools
# cd $GOPATH/src/github.com/opencontainers/runtime-tools/
# make
# make install
```

Once the tool is installed I am using it in my third step to create the container configuration file:

```
# cd /runc/containers/xonotic
# oci-runtime-tool generate \
  --args "/usr/bin/darkplaces-dedicated" \
  --args "-userdir" --args "/tmp" --tmpfs /tmp \
  --rootfs-readonly  \
  --linux-namespace-remove network \
  | jq 'del(.linux.seccomp)' > config.json
```

The '--args' parameters are used to tell the container what to start. In this case it is:

```
/usr/bin/darkplaces-dedicated -userdir /tmp
```

The '--tmpfs' parameter is telling the container to mount a *tmpfs* at */tmp* and the parameter '--rootfs-readonly' configures the container as read-only. This is important for the actual migration. More details on that later. The last parameter '--linux-namespace-remove network' is used to use the host system's network instead of a network namespace. The output of *oci-runtime-tools* is piped through *jq* to remove all the *seccomp* configuration as the combination of RHEL, CRIU, runC and *seccomp* does not yet work.

Before starting the container two more things are necessary. First it is necessary to create the following link as Xonotic does not start without it:

```
# cd /runc/containers/xonotic/rootfs/usr/share/xonotic/
# ln -s data id1
```

The last step is to tell the Xonotic server which IP address to bind to:

```
# echo "net_address 192.168.122.99" > /runc/containers/xonotic/rootf
s/usr/share/xonotic/data/server.cfg
```

This is necessary as we have to use a floating IP address on all systems during the migration to keep the connection alive between client and server. Once the container is installed and configured I am starting it with the following command:

```
# runc run xonotic -d -b /runc/containers/xonotic/ &> /dev/null < /d
ev/null
```

The redirection of *stdout* and *stdin* is necessary for the migration so that the container has no relation to the terminal it was started in. To check if the container is running I can type 'runc list' which gives me the following output

```
ID       PID   STATUS  BUNDLE                   CREATED
     OWNER
xonotic  30207 running /runc/containers/xonotic 2017-08-04T15:19:15.
35Z  root
```

Now that the container is running in my virtual machine I can connect with my local Xonotic client to the server running in that container:

```
xonotic-glx +connect 192.168.122.99 +vid_fullscreen 0 +mastervolume
0 +_cl_name tedric
```

This starts the Xonotic client in window mode with audio turned off and setting the player name to 'tedric' and it looks like this:

image4

An important indicator of where the container with the server is running is the time listed in the 'ping' column. Right now it is low with 15ms as the client is running on the same system as the virtual machine with the container. It will be a larger number once the container is further away.

Local Migration

In a first step to demonstrate the container migration I will migrate the container from one virtual machine to another virtual machine. Both virtual machines are running on the same system as the client from above. As I was using the parameter '--rootfs-readonly' the file system of the container does not need to be migrated during the actual migration. To make the container file system available on all involved systems I was using rsync to copy the container file system.

As I want to keep the client's connection to the server alive during the migration I also have to make sure that the IP address of the container is also moved during the migration process. To move the IP address from one system to another I am using *keepalived*. The floating IP address in this example is 192.168.122.99.

The actual migration is performed by two python scripts built for this demo. One is used on the system where the container is currently running. It is called *migrate*:

```
#   migrate
Usage: migrate [container id] [destination] [pre-copy] [post-copy]
```

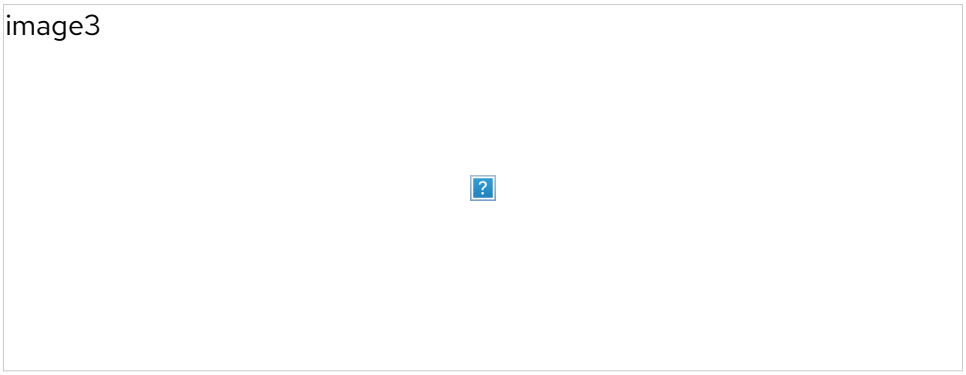
I made the script available at https://people.redhat.com/areber/criu/migrate. This script and the other one include some hard-coded assumptions and probably cannot be used without additional changes.

The migration destination system needs to run the corresponding server script. The server script started by this systemd service file, listens to the commands from above's *migrate* script. To start the migration from my first virtual machine (rhel01) to the second virtual machine (rhel02) following command is necessary:

```
# migrate xonotic rhel02
runc checkpoint --image-path image  xonotic finished after 0.58 seco
nd(s) with 0
Giving floating IP to rhel02
DUMP size: 366M    /runc/containers/xonotic/image
Transferring DUMP to rhel02
DUMP transfer time 0.17 seconds
runc restored xonotic successfully
```

The *migrate* script will do all the necessary steps to checkpoint the container, to transfer the checkpoint (which is fast as it is using a shared NFS), move the IP address and to restart the container on the destination. The *migrate* script can also perform migration optimizations (**pre-copy** and **post-copy**) which is described in more details below. Before and after the migration the client is connected to the server. During the actual migration, for a short amount of time, the network connection is disrupted which is visualized by an indicator like in the following screenshot:

image3



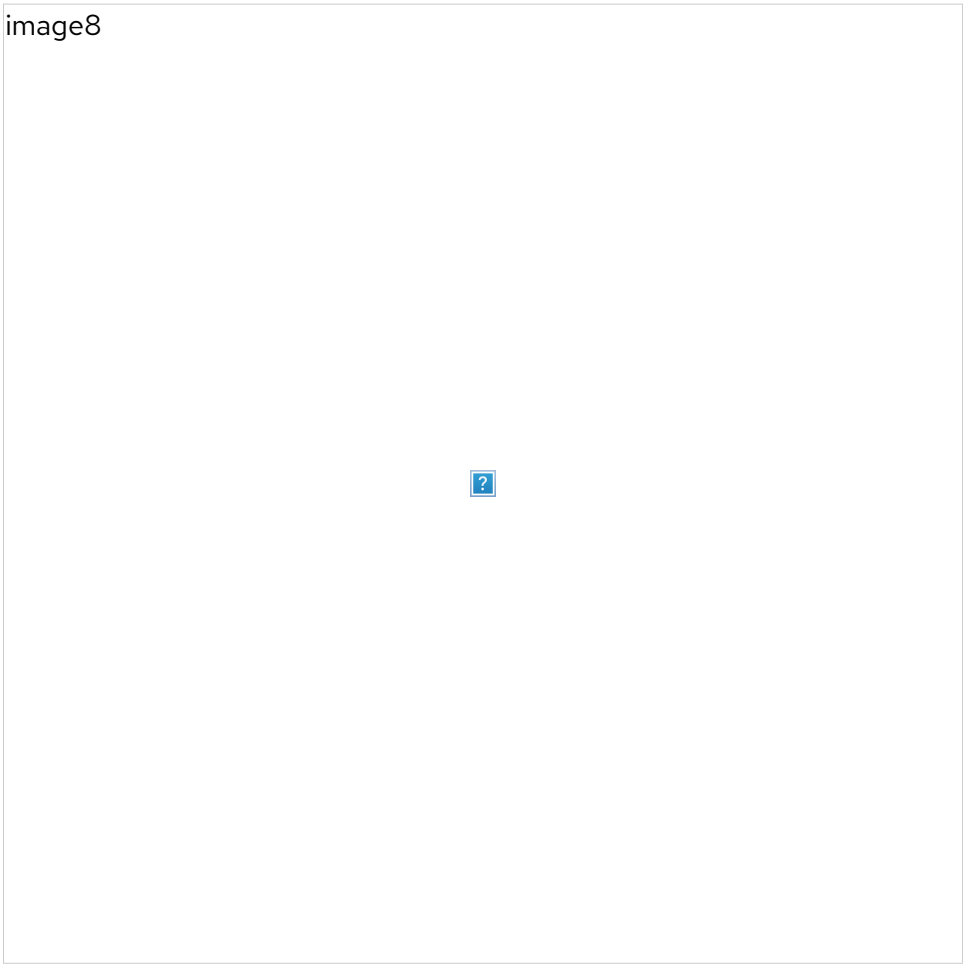
Good, so far the migration works. Up to this point the packages included in Red Hat Enterprise Linux 7.4 provide all the features needed.

So it is nice that it is possible to live migrate a running container from one local virtual machine to another local virtual machine but let's try to migrate the container over some distance.

Remote Migration (same continent)

The next step is to migrate the same container to a system about 150 kilometers away. I will now migrate the container from a virtual machine on my system near Stuttgart, Germany to a system in Strasbourg, France:

image8



Trying to migrate the running container from my local virtual machine to the system in Strasbourg with the same command as above works but the connection from the client to the server will be disrupted as the migration takes too long. This is related to the size of the processes dumped, which is about 400MB, in combination with the bandwidth of 10Mbit/s of my network uplink.

Fortunately CRIU includes optimizations to decrease the unavailability of processes (or containers in my case) during the migration. One of those possible optimizations is pre-copy migration. CRIU uses the Linux kernel's possibility to track memory changes to decrease migration downtime by iteratively transferring only memory pages which have changed since the previous pre-copy cycle. This reduces the container's downtime to the transfer time of the last memory pages delta transfer.

As runC already has support for CRIU's pre-copy functionality I was able to easily integrate pre-copy into my migrate script:

```
#  migrate xonotic rhelfr true
runc checkpoint --pre-dump --image-path parent xonotic finished afte
r 0 second(s) with 0
PRE-DUMP size: 351M    /runc/containers/xonotic/parent
Transferring PRE-DUMP to rhelfr
PRE-DUMP transfer time 22.95 seconds
runc checkpoint --image-path image  --parent-path ../parent xonotic
finished after 0.15 second(s) with 0
Giving floating IP to rhelfr
DUMP size: 20M    /runc/containers/xonotic/image
Transferring DUMP to rhelfr
DUMP transfer time 1.09 seconds
runc restored xonotic successfully
```

The third parameter to my migrate script (true) tells the script to first do a **pre-dump** and keep the container running while the initial dump (351MB in this case) is transferred (using rsync with compression) to the destination system. Now the container is running in Strasbourg which results in a different value in the 'ping' column:

image6

So instead of 15ms as with the container running on a virtual machine on the same system it has now increased to 50ms.

Remote Migration (different continent)

In the next step of the journey around the world the container is migrated to another continent: from Strasbourg in France to Montreal in Canada:

image2

```
# migrate xonotic rhelca true
runc checkpoint  --pre-dump --image-path parent xonotic finished aft
er 0 second(s) with 0
PRE-DUMP size: 351M    /runc/containers/xonotic/parent
Transferring PRE-DUMP to rhelca
PRE-DUMP transfer time 7.39 seconds
runc checkpoint --image-path image  --parent-path ../parent xonotic
finished after 0.23 second(s) with 0
Giving floating IP to rhelca
DUMP size: 21M    /runc/containers/xonotic/image
Transferring DUMP to rhelca
DUMP transfer time 1.36 seconds
runc restored xonotic successfully
```

This migration changes the value in the ‘ping’ column from 50ms to 133ms:

image1.png

Remote Migration (another different continent)

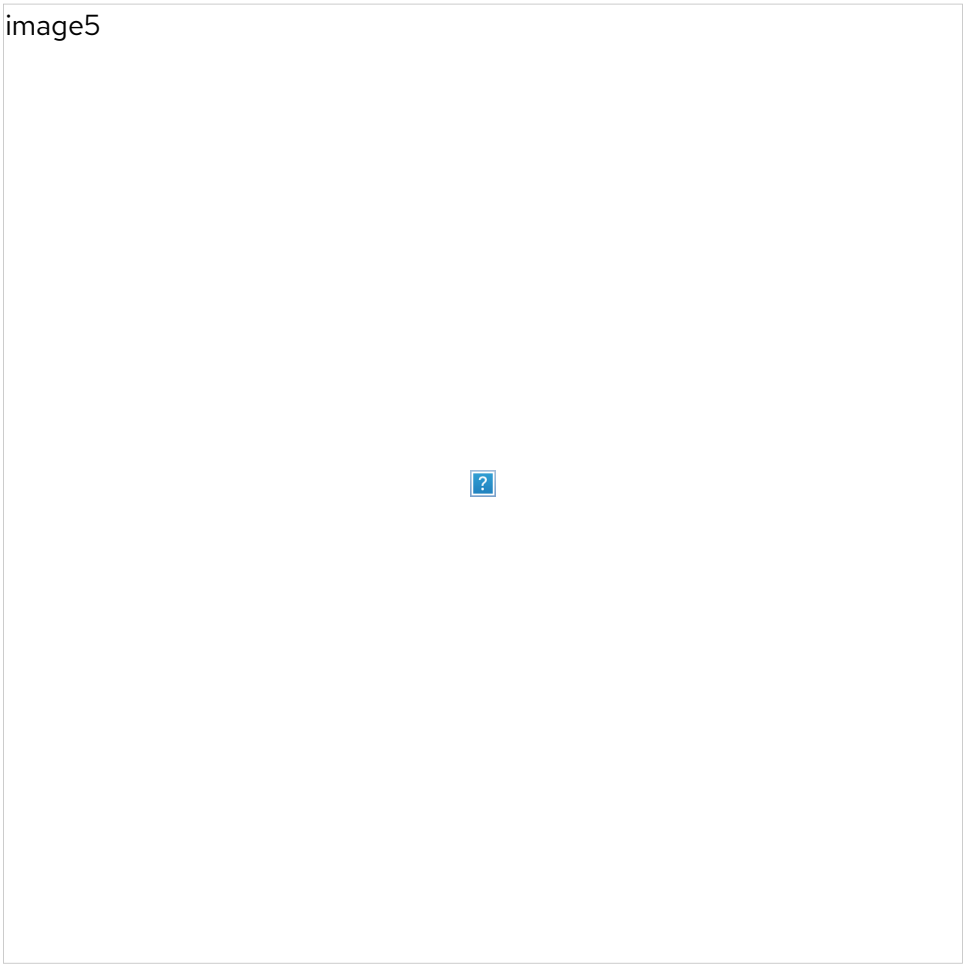
The next step in the container's journey around the world is Singapore:

image9

```
# migrate xonotic rhelsg true
runc checkpoint  --pre-dump --image-path parent xonotic finished aft
er 0 second(s) with 0
PRE-DUMP size: 351M    /runc/containers/xonotic/parent
Transferring PRE-DUMP to rhelsg
PRE-DUMP transfer time 11.75 seconds
runc checkpoint --image-path image  --parent-path ../parent xonotic
finished after 0.19 second(s) with 0
Giving floating IP to rhelsg
DUMP size: 21M    /runc/containers/xonotic/image
Transferring DUMP to rhelsg
DUMP transfer time 3.15 seconds
runc restored xonotic successfully
```

Using the Xonotic server in Singapore increases the value in the 'ping' column even more from my client location. Instead of the previous 133ms it is now up to 283ms.

image5



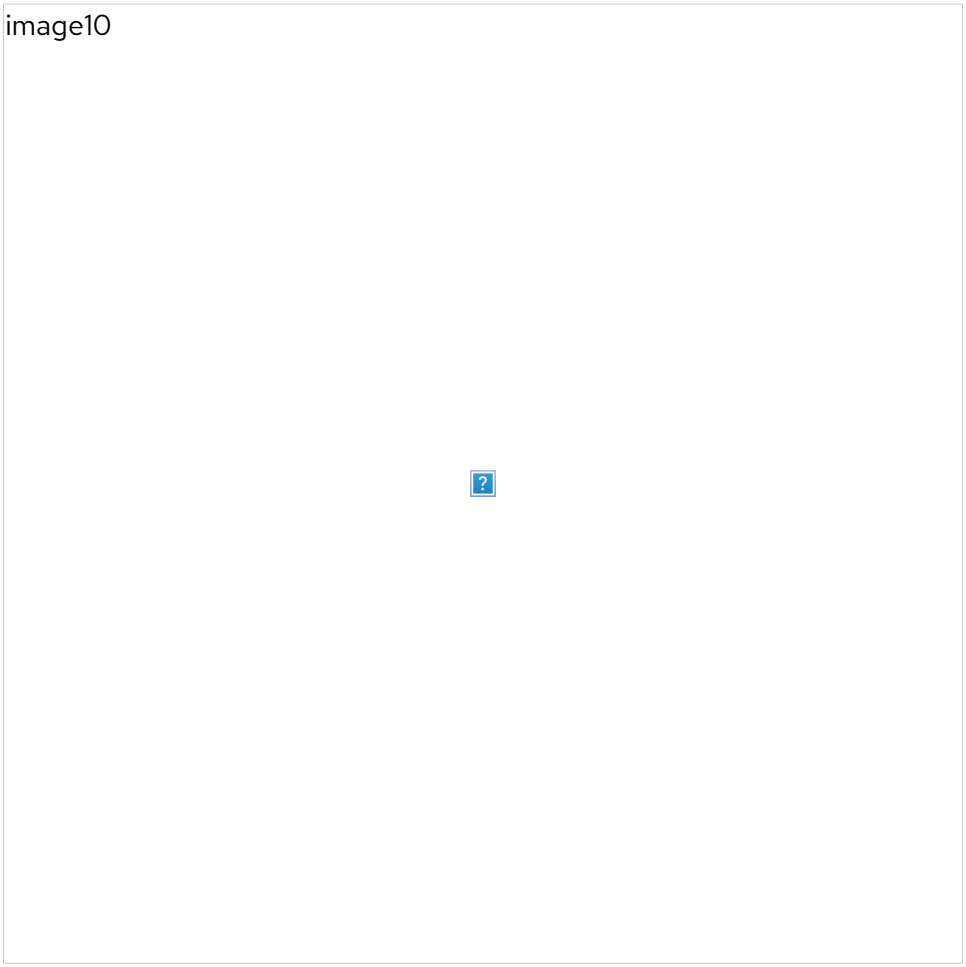
Back again

The last step in my container's migration around the world is back to Strasbourg:

image7

```
# migrate xonotic rhelfr true
runc checkpoint  --pre-dump --image-path parent xonotic finished aft
er 3 second(s) with 0
PRE-DUMP size: 351M    /runc/containers/xonotic/parent
Transferring PRE-DUMP to rhelfr
PRE-DUMP transfer time 13.76 seconds
runc checkpoint --image-path image  --parent-path ../parent xonotic
finished after 0.59 second(s) with 0
Giving floating IP to rhelfr
DUMP size: 21M    /runc/containers/xonotic/image
Transferring DUMP to rhelfr
DUMP transfer time 3.88 seconds
runc restored xonotic successfully
```

image10



With the container back in Strasbourg the value in the 'ping' column is now back to 50ms.

Thanks to the integration of pre-copy optimization in CRIU and runC it was possible to migrate my Xonotic container once around the world while the client's connection remains intact.

The nice thing about this demonstration is that all involved software components are taken without changes from upstream. All involved VMs are running the default RHEL kernel and userspace. CRIU is running from a git checkout of the *criu-dev* development branch and runC is running from git checkout of the *master* branch.

With this setup not only pre-copy migration is possible but also post-copy migration which is also called lazy migration. Especially the combination of pre-copy optimization and lazy migration is interesting as it offers the lowest downtimes during container migration. Using my migration script this is possible by setting the third (**pre-copy**) and fourth parameter (**post-copy**) to 'true':

```
#  migrate xonotic rhel02 true true
runc checkpoint  --pre-dump --image-path parent xonotic finished aft
er 0 second(s) with 0
PRE-DUMP size: 351M    /runc/containers/xonotic/parent
Transferring PRE-DUMP to rhel02
PRE-DUMP transfer time 0.1 seconds
runc checkpoint --image-path image  --parent-path ../parent --lazy-p
ages --page-server localhost:27 --status-fd /tmp/postcopy-pipe xonot
ic
Ready for lazy page transfer
runc checkpoint --image-path image  --parent-path ../parent --lazy-p
ages --page-server localhost:27 --status-fd /tmp/postcopy-pipe xonot
```

PREVIOUS OLDER POST

Red Hat, Google Cloud, and other industry leaders join together to standardize Kubernetes service component auditing and policy enforcement

NEXT NEWER POST

Friday Five — October 13, 2017

with Red Hat Enterprise Linux 7.4 out of the box without any additional requirements. For the optimized cases (pre-copy and post-copy) I am using the latest upstream git checkouts. All presented features are, however, fully merged in the corresponding upstream projects and can easily be tested. I have also recorded a video demonstrating the migration around the world. If you are going to this year's Open Source Summit Europe in Prague please come to my talk Container Migration Around The World for more details, live demonstrations and answers for your questions.