Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Oren Laadan on Sat, 05 Apr 2008 00:30:33 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Fri, 2008-04-04 at 11:56 -0400, Oren Laadan wrote:
>> Matt Helsley wrote:
>>> On Thu, 2008-04-03 at 16:49 -0700, Paul Menage wrote:
>>>> On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:
>>>>> * "freezer.kill"
>>>>>
>>>>> writing <n> will send signal number <n> to all tasks
>>>>>
>>>> My first thought (not having looked at the code yet) is that sending a
>>>> signal doesn't really have anything to do with freezing, so it
>>>> shouldn't be in the same subsystem. Maybe a separate subsystem called
>>>> "signal"?
>>>>
>>>> And more than that, it's not something that requires any particular
>>>> per-process state, so there's no reason that the subsystem that
>>>> provides the "kill" functionality shouldn't be able to be mounted in
>>>> multiple hierarchies.
>>>>
>>>> How about if I added support for stateless subsystems, that could
>>>> potentially be mounted in multiple hierarchies at once? They wouldn't
>>>> need an entry in the css set, since they have no state.
>>> This seems reasonable to me. A quick look at Cedric's patches suggests
>>> there's no need for such cgroup subsystems to be tied together -- the
>>> signalling is all done internally to the freeze_task(), refrigerator(),
>>> and thaw_process() functions from what I recall.
>>>
>>>> * Usage :
>>>>
>>>> # mkdir /containers/freezer
>>>> # mount -t container -ofreezer freezer /containers/freezer
>>>> # mkdir /containers/freezer/0
>>>> # echo $some_pid > /containers/freezer/0/tasks
>>>>
>>>> to get status of the freezer subsystem :
>>>>
>>>> # cat /containers/freezer/0/freezer.freeze
>>>> RUNNING
>>>>
>>>> to freeze all tasks in the container :
>>>>
>>>> # echo 1 > /containers/freezer/0/freezer.freeze
>>>> # cat /containers/freezer/0/freezer.freeze
>>>> FREEZING

>>>>> # cat /containers/freezer/0/freezer.freeze
>>>>> FROZEN
>>>> Could we separate this out into two files? One called "freeze" that's
>>>> a 0/1 for whether we're intending to freeze the subsystem, and one
>>>> called "frozen" that indicates whether it is frozen? And maybe a
>>>> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
>>>> human-readable way?
>>> 3 files seems like overkill. I think making them human-readable is good
>>> and can be done with two files: "state" (read-only) and
>>> "state-next" (read/write). Transitions between RUNNING and FROZEN are
>>> obvious when state-next != state. I think the advantages are it's pretty
>>> human-readable, you don't need separate strings and files for the
>>> transitions, it's clear what's about to happen (IMHO), and it only
>>> requires 2 files. Some examples:
>>>
>>> To initiate freezing:
>>>
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # echo "FROZEN" > /containers/freezer/0/freezer.state-next
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # cat /containers/freezer/0/freezer.state-next
>>> FROZEN
>>> # sleep N
>>> # cat /containers/freezer/0/freezer.state
>>> FROZEN
>>> # cat /containers/freezer/0/freezer.state-next
>>> FROZEN
>>>
>>> So to cancel freezing you might see something like:
>>>
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # cat /containers/freezer/0/freezer.state-next
>>> FROZEN
>>> # echo "RUNNING" > /containers/freezer/0/freezer.state-next
>>> # cat /containers/freezer/0/freezer.state-next
>>> RUNNING
>>>
>>> If you wanted to know if a group was transitioning:
>>>
>>> # diff /containers/freezer/0/freezer.state /containers/freezer/0/freezer.state-next
>>>
>>> Or:
>>> # current=`cat /containers/freezer/0/freezer.state`
>>> # next=`cat /containers/freezer/0/freezer.state-next`
>>> # ["$current" != "$next"] && echo "Transitioning"

>>> # ["$current" == "RUNNING" -a "$next" == "FROZEN"] && echo "Freezing"
>>> # ["$current" == "FROZEN" -a "$next" == "RUNNING"] && echo "Thawing"
>>> # ["$current" == "RUNNING" -a "$next" == "RUNNING"] && echo "No-op"
>>> # ["$current" == "FROZEN" -a "$next" == "FROZEN"] && echo "No-op"
>> First, I totally agree with Serge's comment (oh well, it's about my
>> own suggestion, so I must) - for checkpoint/restart we'll need more
>> states if we are to use the same subsystem.
>
> I don't have an upper limit on how many more states we will need and I
> think that number impacts the interface significantly. Can you give us
> an estimate?

In Zap there are (using the current terminology):
RUNNING - running
FREEZING - transition to FROZEN
FROZEN - frozen
THAWING - transition to RUNNING
CKPTING - being checkpointed (needs special semantics* for some ops)
RSTRTING - being restarted (may need special semantics* for some ops)
ABORTING - transition to DEAD (mainly when aborting a failed restart)
DEAD - dead (but not fully cleaned up yet)

There is also "SPECIAL" in which some operations are not allowed;
this simplifies dealing with a bunch of races related to checkpoint/
restart, but I'm not sure it's a must. If anything, it only stays
for very short times (like an uninterruptible sleep) saying "don't
mess with this container now, it's busy".

I have very good justifications for almost all the states, a good
reasoning for DEAD, and a case for SPECIAL (although there may be
a way to do without it).

Despite the "many" states, there are very few transitions: CKPTING
can only be reached from- and changed to- FROZEN. A similar rule holds
for RSTRTING. ABORTING is reached from RSTRTING, and leads to DEAD.
The only one I don't cover is reaching DEAD from any other state
(except SPECIAL) but I never saw a reason to explicitly encode that.
Something like this (without SPECIAL):

```
        ->  FREEZING  ->         <-> CKPTING
RUNNING                  FROZEN
        <-  THAWING   <-         <-> RSTRTING -> ABORTING -> DEAD
```

Out of curiosity - why does the number of states impact the interface
so much ?

>
>> Second, my gut feeling is that a single, atomic operation to get the

>> status is preferred over multiple (non-atomic) operations. In other
>> words, I suggest a single state file instead of two. You can encode
>> every possible transition in a single state. It's not that the kernel
>
> If the transitions are to be human-readable and there are more than a
> small number of states it may not be desirable to encode transitions as
> states. Paul's reason for suggesting the additional file(s), as best I
> could tell, was to keep the interface human-readable.

The scheme above has very few transitions. Whatever be the final scheme,
I would prefer not to have many possible transition (the full matrix).
It probably isn't necessary either.

The main idea behind limiting the transitions above, is that checkpoint
requires to first freeze the container to be able to capture a consistent
view of the state of its processes; that means, for example, that we also
would like to prevent signals from being delivered to tasks in a frozen
state (if you do want to signal - thaw it first).

There is also the issue of a pre-checkpoint (a.k.a live migration) where
significant state (mainly memory) of the container is recorded while the
container is still running, and when it's finally frozen little state
remains to be saved, reducing the application downtime. I didn't see a
need for a special state for this case; instead Zap uses a status flag
that belongs to the container.

Oren.

>
> Cheers,
> -Matt Helsley
>
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers