Subject: Thoughts on virtualizing task containers
Posted by Paul Menage on Fri, 24 Aug 2007 00:18:15 GMT
View Forum Message <> Reply to Message

I'm interested in people's thoughts about the right user API for virtualizing task containers and resource controllers.

With namespaces, the model is fairly straightforward - you can split a new namespace off from your own, without caring whether you're the top-level namespace or some child namespace. So nested namespaces just work, but at the expense of not being able to see anything that's at a higher level than your current namespace.

For resource controllers and other task container subsystems, that model isn't necessarily desired. For example, tasks running in a cpuset don't see their own cpuset as the root; it's possible (with appropriate permissions) to modify other cpusets outside of your own one.

For doing full virtual server containers, root in the virtual server may well want to make use of task container subsystems, e.g. to control the amount of CPU cycles or memory that groups of processes within the virtual server can use. What kinds of controls do we want to give the host to determine what kind of container operations the guest virtual server can do? What should the guest see in terms of task container support?

Ideally, the guest would have what appears to it to be a full (and isolated) task container implementation to play with, complete with a full complement of subsystems.

But at the same time, some subsystems might not be easily virtualizable, and the host might not want the guest to have access to all subsystems anyway.

One way to handle this might be to have a "virtualize" subsystem that allows you to set virtualization boundaries; by setting a particular container's virtualize control file to "true" you'd enforce the rule that tasks in that container (or child containers) would:

- only be able to mount container hierarchies with task subsystems mounted in that hierarchy

- see that container as their root container

- not be able to increase the resource limits of their own container. (similar to the way that the "mems" and "cpus" files in the root cpuset container directory are read-only, certain control files would

become read-only in the virtualized container directory).

Possibly rather than needing a separate "virtualize" subsystem you could infer virtualization boundaries based on container boundaries in whichever hierarchy had the nsproxy subsystem mounted (if any). If the nsproxy subsystem wasn't mounted anywhere, then no virtualization would occur.

On top of the implementation issues, there are conceptual issues to deal with such as:

- what do you do with subsystems (e.g. freezer, OOM handler, network flow id assigner, etc) that are inherently hard to virtualize in a hierarchical way?

- if the host mounts a container filesystem hierarchy in the guest's filespace, does the guest see the host's view of containers or the guest's view? i.e. is the virtualization associated with the mount point or with the process doing the viewing? (I'm inclined to say the former)

- how much visibility/control does the host have into any child task containers created by the guest?

- can the guest split the subsystems that are visible to it amongst multiple hierarchies? Or do we rule that guests can only have access to a single hierarchy?

Paul

Subject: Re: Thoughts on virtualizing task containers
Posted by serue on Mon, 27 Aug 2007 14:40:26 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> I'm interested in people's thoughts about the right user API for
> virtualizing task containers and resource controllers.
>
> With namespaces, the model is fairly straightforward - you can split a
> new namespace off from your own, without caring whether you're the
> top-level namespace or some child namespace. So nested namespaces just
> work, but at the expense of not being able to see anything that's at a
> higher level than your current namespace.

>
> For resource controllers and other task container subsystems, that
> model isn't necessarily desired. For example, tasks running in a
> cpuset don't see their own cpuset as the root; it's possible (with
> appropriate permissions) to modify other cpusets outside of your own
> one.
>
> For doing full virtual server containers, root in the virtual server
> may well want to make use of task container subsystems, e.g. to
> control the amount of CPU cycles or memory that groups of processes
> within the virtual server can use. What kinds of controls do we want
> to give the host to determine what kind of container operations the
> guest virtual server can do? What should the guest see in terms of
> task container support?
>
> Ideally, the guest would have what appears to it to be a full (and
> isolated) task container implementation to play with, complete with a
> full complement of subsystems.

Agreed, although at k-s last year it was agreed that it's ok if
processes in a guest know that they are in a guest. That was in
relation specifically to /proc virtualization.

So for instance if some vserver is locked to cpus 3 and 4, I think it
would be nice if that vserver thought it was on a 2-cpu system and had
full access to cpus 0 and 1, but it's not necessary.

OTOH, the ability for root in that vserver to restrict some of it's
tasks to cpu 3 (or ideally to half of cpu 3) should definately exist.

> But at the same time, some subsystems might not be easily
> virtualizable, and the host might not want the guest to have access to
> all subsystems anyway.
>
> One way to handle this might be to have a "virtualize" subsystem that
> allows you to set virtualization boundaries; by setting a particular
> container's virtualize control file to "true" you'd enforce the rule
> that tasks in that container (or child containers) would:
>
> - only be able to mount container hierarchies with task subsystems
> mounted in that hierarchy

So long as the available resources are always a subset of the resources
assigned to the container, are there situations where it would be
desirable to not allow the container to sub-partition among it's tasks?

I suppose at some point we might run into performance concerns due to
too much fine-grained partitioning?

But in general I would say no.

> - see that container as their root container

Yes.

> - not be able to increase the resource limits of their own container.
> (similar to the way that the "mems" and "cpus" files in the root
> cpuset container directory are read-only, certain control files would
> become read-only in the virtualized container directory).

Yes.

At some point we might want to allow some capability
CAP_SYS_CONTAINER_OVERRIDE to allow a task to create a container with
more resources than it's parent, but it probably isn't necessary.

> Possibly rather than needing a separate "virtualize" subsystem you
> could infer virtualization boundaries based on container boundaries in
> whichever hierarchy had the nsproxy subsystem mounted (if any). If the
> nsproxy subsystem wasn't mounted anywhere, then no virtualization
> would occur.

A different approach:

Since the controls are implemented as vfs operations, perhaps a natural
way to split these up could be by mounts namespaces.

So you clone(CLONE_NEWNS), then when the child does a mount of
containerfs filesystem, the container hierarchy applies to the child but
not the parent.

That's probably not quite right, since we probably don't want every
CLONE_NEWNS to have that effect. So instead we could introduce
CLONE_CONTAINER_LEVEL, which always implies CLONE_NEWNS, and which means
that any containerfs mounts by the child are not applied to the parent.

In either case the 'virtualize' subsystem would still exist to allow
parents to keep track of containerfs mounts by children.

(perhaps this isn't actually a different approach, but rather detail at
a different level, since we could still use the nsproxy to cache the
container virtualization hierarchy level or whatever the heck we call
it).

> On top of the implementation issues, there are conceptual issues to
> deal with such as:

>
> - what do you do with subsystems (e.g. freezer, OOM handler, network
> flow id assigner, etc) that are inherently hard to virtualize in a
> hierarchical way?

Guess we need to go through them one by one. With the freezer subsystem, if a process does clone(CLONE_CONTAINER_LEVEL) and then does
mount -t containerfs,whatever -o freeze /freezer
then a subsequent echo 1 > /freezer/freezer.freeze should not freeze it's parent process.

> - if the host mounts a container filesystem hierarchy in the guest's
> filespace, does the guest see the host's view of containers or the
> guest's view? i.e. is the virtualization associated with the mount
> point or with the process doing the viewing? (I'm inclined to say the
> former)
>
> - how much visibility/control does the host have into any child task
> containers created by the guest?

I can think of two possibilities:

1. It could depend upon mount propagation. To make this useful we'd probably need to allow forcing a child to mount any containerfs subsystems under a particular path so we can mark it --rshared. That probably still isn't enough, since the child can just mark the tree --rslave.

2. All containerfs mounts in a child after clone(CLONE_CONTAINER_LEVEL) could appear in the parent's hierarchy mounting the 'virtualization' subsystem.

I think I've argued 1 as unworkable so I'll recommend 2 :)

> - can the guest split the subsystems that are visible to it amongst
> multiple hierarchies? Or do we rule that guests can only have access
> to a single hierarchy?

Offhand I can't think of a reason to have such a restriction, unless we think the guests could impact system performance with too much container work.

thanks,
-serge

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: Thoughts on virtualizing task containers
Posted by Paul Menage on Tue, 28 Aug 2007 21:11:28 GMT
View Forum Message <> Reply to Message

On 8/27/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> Agreed, although at k-s last year it was agreed that it's ok if
> processes in a guest know that they are in a guest. That was in
> relation specifically to /proc virtualization.

If that's been already settled on then it simplifies things a fair bit.

>
> So for instance if some vserver is locked to cpus 3 and 4, I think it
> would be nice if that vserver thought it was on a 2-cpu system and had
> full access to cpus 0 and 1, but it's not necessary.

Well, assuming that userspace would normally handle CPU hotplug, then it ought to be able to handle a non-contiguous CPU map, as a real system might appear like that.

>
> So long as the available resources are always a subset of the resources
> assigned to the container, are there situations where it would be
> desirable to not allow the container to sub-partition among it's tasks?

No, but it will probably take a bit of reorganization in the container system.

E.g. suppose that the host doesn't mount subsystem X, and hence all processes in the system are in the root container for subsystem X. If the guest then mounts subsystem X and tries to change the root container parameters for X, that's going to be a problem.

We could get around that by saying that a guest has no control over the parameters of its root container, in the same way that e.g. even root can't add/subtract cpus from the root cpuset, since it represents what's available in hardware. So the guest would have to mount subsystem X, create a subcontainer, and move processes into it in order to change their X limits.

This then brings up the awkward issue of what happens if the host later wants to mount subsystem X itself and apply its own process division (or what if X is already mounted but the host has divided processes in a way that's different from the host/guest process distinction?

These are all reasons why it might be simpler for the host to decide
exactly which subsystems are available to the guest, and to require
that all the guest processes be in the same sub-tree for all the
available subsystems.

>
> Since the controls are implemented as vfs operations, perhaps a natural
> way to split these up could be by mounts namespaces.
>
> So you clone(CLONE_NEWNS), then when the child does a mount of
> containerfs filesystem, the container hierarchy applies to the child but
> not the parent.
>
> That's probably not quite right, since we probably don't want every
> CLONE_NEWNS to have that effect. So instead we could introduce
> CLONE_CONTAINER_LEVEL, which always implies CLONE_NEWNS, and which means
> that any containerfs mounts by the child are not applied to the parent.

How do you define "not applied"?

> > - how much visibility/control does the host have into any child task
> > containers created by the guest?
>
> 2. All containerfs mounts in a child after
> clone(CLONE_CONTAINER_LEVEL) could appear in the parent's
> hierarchy mounting the 'virtualization' subsystem.

It's not so much the containerfs mounts that matter, I think - it's
the actual creation of containers and movement of tasks between them.
If we can manage to share the superblocks between the parent and
child, and just use mount tricks to only let the child mount a subtree
of the hierarchy, then this information would be shared automatically.

The only drawback with that is that a superblock defines a container
hierarchy for a specified set of subsystems, which is why we come back
to the issue of different hierarchy compositions in the parent and the
child causing problems.

Paul

Subject: Re: Thoughts on virtualizing task containers

Posted by serue on Tue, 11 Sep 2007 13:59:25 GMT
View Forum Message <> Reply to Message

Hi Paul,

did any good ideas come up at the mini-summit or k-s, or were any
decisions made?

(If not I can try to jump back into the thread where we left off :)

thanks,
-serge

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

Subject: Re: Thoughts on virtualizing task containers
Posted by Paul Menage on Wed, 26 Sep 2007 05:23:52 GMT
View Forum Message <> Reply to Message

On 9/11/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> Hi Paul,
>
> did any good ideas come up at the mini-summit or k-s, or were any
> decisions made?
>

Discussions were had, but decisions weren't really made.

My vague thoughts on how to do virtualization are below.

1) Add support for a subsystem state object to be shared with its
children. specifically:
- for each subsystem, have a <subsystem.inherit> control file, which
defaults to 0. This can only be changed when the cgroup has no
children
- any children of a cgroup will share subsystem state with the parent
for any subsystems whose <inherit> file is 1

This ties in with a request that Balbir made for being able to share
resource limits between different levels of cgroups, but it's also
useful for virtualization. It's something I wanted to describe in my
OLS talk but didn't really have time for.

2) have a virtualization cgroup subsystem, which like other subsystems
can be included in at most one hierarchy. The virtualization subsystem

might perhaps be the same thing as the nsproxy subsystem?

3) when mounting a cgroup filesystem, if the virtualization subsystem
is mounted, and the caller is not in its root cgroup (i.e. it's a
guest), then:

- the guest can only see subsystems in the same hierarchy, which
additionally have <inherit> set to 0

- the vfsmount returned from cgroup_get_sb() doesn't refer to the root
of the hierarchy, but instead to the cgroup directory that the guest
is in

- the guest can only mount a single hierarchy, (which therefore must
be a subset of the hierarchy that the guest is running in)

- at the time of mount, the <inherit> bits for any subsystems *not*
selected by the guest get set to 1, thus any guest processes share the
same subsystem state for those subsystems (this is analagous to having
the subsystem not be mounted, at the root/host level).

This approach is a little more restrictive than I'd like, but I think
it should support the basic nested virtual server model reasonable
well.

These changes are going to require a little bit of plumbing in the
core cgroup code, but should have very little effect on any subsystems
themselves, except for a few ways:

- each subsystem will now have a private parent/child tree running
through its subsystem states, rather than having to use the main
cgroup tree

- there will no longer be a direct mapping from a subsystem state to a
cgroup. I'm not sure that this will cause anyone a problem. we'll have
to tweak the current cgroup iteration interfaces to instead iterate
across all the processes in a subsystem state, which may include
multiple cgroups

Paul

Subject: Re: Thoughts on virtualizing task containers

Quoting Paul Menage (menage@google.com):
> On 9/11/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> > Hi Paul,
> >
> > did any good ideas come up at the mini-summit or k-s, or were any
> > decisions made?
> >
>
> Discussions were had, but decisions weren't really made.
>
> My vague thoughts on how to do virtualization are below.

Thanks. Actually doesn't seem that vague, and makes sense.
I'd advocate following this path. Have you started any
patches for this?

> 1) Add support for a subsystem state object to be shared with its
> children. specifically:
> - for each subsystem, have a <subsystem.inherit> control file, which
> defaults to 0. This can only be changed when the cgroup has no
> children
> - any children of a cgroup will share subsystem state with the parent
> for any subsystems whose <inherit> file is 1
>
> This ties in with a request that Balbir made for being able to share
> resource limits between different levels of cgroups, but it's also
> useful for virtualization. It's something I wanted to describe in my
> OLS talk but didn't really have time for.
>
> 2) have a virtualization cgroup subsystem, which like other subsystems
> can be included in at most one hierarchy. The virtualization subsystem
> might perhaps be the same thing as the nsproxy subsystem?
>
> 3) when mounting a cgroup filesystem, if the virtualization subsystem
> is mounted, and the caller is not in its root cgroup (i.e. it's a
> guest), then:
>
> - the guest can only see subsystems in the same hierarchy, which

same hierarchy as the virtualization cgroup subsystem, right?

> additionally have <inherit> set to 0
>
> - the vfsmount returned from cgroup_get_sb() doesn't refer to the root
> of the hierarchy, but instead to the cgroup directory that the guest
> is in

>
> - the guest can only mount a single hierarchy, (which therefore must
> be a subset of the hierarchy that the guest is running in)
>
> - at the time of mount, the <inherit> bits for any subsystems *not*
> selected by the guest get set to 1, thus any guest processes share the
> same subsystem state for those subsystems (this is analagous to having
> the subsystem not be mounted, at the root/host level).

So then for subsystems which the guest did select, are there
default values inherited from the parent cgroup, and restrictions based
on those? Or does the guest get to set any values it wants for those
subsystems?

> This approach is a little more restrictive than I'd like, but I think
> it should support the basic nested virtual server model reasonable
> well.
>
> These changes are going to require a little bit of plumbing in the
> core cgroup code, but should have very little effect on any subsystems
> themselves, except for a few ways:
>
> - each subsystem will now have a private parent/child tree running
> through its subsystem states, rather than having to use the main
> cgroup tree
>
> - there will no longer be a direct mapping from a subsystem state to a
> cgroup. I'm not sure that this will cause anyone a problem. we'll have
> to tweak the current cgroup iteration interfaces to instead iterate
> across all the processes in a subsystem state, which may include
> multiple cgroups
>
> Paul

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

---

Subject: Re: Thoughts on virtualizing task containers
Posted by Paul Menage on Mon, 01 Oct 2007 17:36:51 GMT
View Forum Message <> Reply to Message

On 10/1/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> I'd advocate following this path. Have you started any
> patches for this?

Started playing with it, but only really to see what breaks so far.

> > - the guest can only see subsystems in the same hierarchy, which
>
> same hierarchy as the virtualization cgroup subsystem, right?

Right.

>
> So then for subsystems which the guest did select, are there
> default values inherited from the parent cgroup, and restrictions based
> on those? Or does the guest get to set any values it wants for those
> subsystems?

That's something that's going to need to be subsystem-specific, I think. For something like cpusets, that already handles restricting a child to be a subset of a parent, this should be automatic. There's already talk of adding hierarchy to resource counters, which would theoretically solve the problem for all resource counter users.

Paul