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
https://lists.linux-foundation.org/mailman/listinfo/containers