Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Matt Helsley on Thu, 08 Mar 2007 06:10:09 GMT
View Forum Message <> Reply to Message

On Thu, 2007-03-08 at 16:32 +-1300, Sam Vilain wrote:

+ADw-snip+AD4

+AD4 Kirill, 06032418:36+-03:
+AD4 +AD4 I propose to use +ACI-namespace+ACI naming.
+AD4 +AD4 1. This is already used in fs.
+AD4 +AD4 2. This is what IMHO suites at least OpenVZ/Eric
+AD4 +AD4 3. it has good acronym +ACI-ns+ACI.
+AD4
+AD4 Right. So, now I'll also throw into the mix:
+AD4
+AD4   - resource groups (I get a strange feeling of d+AOk-j+AOA v+APo there)

+ADw-offtopic+AD4
Re: d+AOk-j+AOA v+APo: yes+ACE

It's like that Star Trek episode ... except we can't agree on the name
of the impossible particle we will invent which solves all our problems.
+ADw-/offtopic+AD4

At the risk of prolonging the agony I hate to ask: are all of these
groupings really concerned with +ACI-resources+ACI?

+AD4   - supply chains (think supply and demand)
+AD4   - accounting classes

CKRM's use of the term +ACI-class+ACI drew negative comments from Paul Jackson
and Andrew Morton about this time last year. That led to my suggestion
of +ACI-Resource Groups+ACI. Unless they've changed their minds...

+AD4 Do any of those sound remotely close? If not, your turn :)

I'll butt in here: task groups? task sets? confuselets? +ADs)

Cheers,
 -Matt Helsley

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by ebiederm on Thu, 08 Mar 2007 06:44:58 GMT
View Forum Message <> Reply to Message

Matt Helsley <matthltc@us.ibm.com> writes:

> On Thu, 2007-03-08 at 16:32 +-1300, Sam Vilain wrote:
>
> +ADw-snip+AD4
>
> +AD4 Kirill, 06032418:36+-03:
> +AD4 +AD4 I propose to use +ACI-namespace+ACI naming.
> +AD4 +AD4 1. This is already used in fs.
> +AD4 +AD4 2. This is what IMHO suites at least OpenVZ/Eric
> +AD4 +AD4 3. it has good acronym +ACI-ns+ACI.
> +AD4
> +AD4 Right. So, now I'll also throw into the mix:
> +AD4
> +AD4   - resource groups (I get a strange feeling of d+AOk-j+AOA v+APo there)
>
> +ADw-offtopic+AD4
> Re: d+AOk-j+AOA v+APo: yes+ACE
>
> It's like that Star Trek episode ... except we can't agree on the name
> of the impossible particle we will invent which solves all our problems.
> +ADw-/offtopic+AD4
>
> At the risk of prolonging the agony I hate to ask: are all of these
> groupings really concerned with +ACI-resources+ACI?
>
> +AD4   - supply chains (think supply and demand)
> +AD4   - accounting classes
>
> CKRM's use of the term +ACI-class+ACI drew negative comments from Paul Jackson
> and Andrew Morton about this time last year. That led to my suggestion
> of +ACI-Resource Groups+ACI. Unless they've changed their minds...
>
> +AD4 Do any of those sound remotely close? If not, your turn :)
>
> I'll butt in here: task groups? task sets? confuselets? +ADs)

Generically we can use subsystem now for the individual pieces without
confusing anyone.

I really don't much care as long as we don't start redefining
container as something else. I think the IBM guys took it from
solaris originally which seems to define a zone as a set of
isolated processes (for us all separate namespaces). And a container

as a set of as a zone that uses resource control. Not exactly how
we have been using the term but close enough not to confuse someone.

As long as we don't go calling the individual subsystems or the
process groups they need to function a container I really don't care.

I just know that if we use container for just the subsystem level
it makes effective communication impossible, and code reviews
essentially impossible. As the description says one thing the
reviewer reads it as another and then the patch does not match
the description. Leading to NAKs.

Resource groups at least for subset of subsystems that aren't
namespaces sounds reasonable. Heck resource group, resource
controller, resource subsystem, resource just about anything seems
sane to me.

The important part is that we find a vocabulary without doubly
defined words so we can communicate and a small common set we can
agree on so people can work on and implement the individual
resource controllers/groups, and get the individual pieces merged
as they are reading.

Eric

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Herbert Poetzl on Fri, 09 Mar 2007 01:06:28 GMT
View Forum Message <> Reply to Message

On Wed, Mar 07, 2007 at 11:44:58PM -0700, Eric W. Biederman wrote:
> Matt Helsley <matthltc@us.ibm.com> writes:
>
> > On Thu, 2007-03-08 at 16:32 +-1300, Sam Vilain wrote:
> >
> > +ADw-snip+AD4
> >
> > +AD4 Kirill, 06032418:36+-03:
> > +AD4 +AD4 I propose to use +ACI-namespace+ACI naming.
> > +AD4 +AD4 1. This is already used in fs.
> > +AD4 +AD4 2. This is what IMHO suites at least OpenVZ/Eric
> > +AD4 +AD4 3. it has good acronym +ACI-ns+ACI.
> > +AD4

> > +AD4 Right. So, now I'll also throw into the mix:
> > +AD4
> > +AD4   - resource groups (I get a strange feeling of d+AOk-j+AOA v+APo there)
> >
> > +ADw-offtopic+AD4
> > Re: d+AOk-j+AOA v+APo: yes+ACE
> >
> > It's like that Star Trek episode ... except we can't agree on the name
> > of the impossible particle we will invent which solves all our problems.
> > +ADw-/offtopic+AD4
> >
> > At the risk of prolonging the agony I hate to ask: are all of these
> > groupings really concerned with +ACI-resources+ACI?
> >
> > +AD4   - supply chains (think supply and demand)
> > +AD4   - accounting classes
> >
> > CKRM's use of the term +ACI-class+ACI drew negative comments from Paul Jackson
> > and Andrew Morton about this time last year. That led to my suggestion
> > of +ACI-Resource Groups+ACI. Unless they've changed their minds...
> >
> > +AD4 Do any of those sound remotely close? If not, your turn :)
> >
> > I'll butt in here: task groups? task sets? confuselets? +ADs)
>
> Generically we can use subsystem now for the individual pieces without
> confusing anyone.
>
> I really don't much care as long as we don't start redefining
> container as something else. I think the IBM guys took it from
> solaris originally which seems to define a zone as a set of
> isolated processes (for us all separate namespaces). And a container
> as a set of as a zone that uses resource control. Not exactly how
> we have been using the term but close enough not to confuse someone.
>
> As long as we don't go calling the individual subsystems or the
> process groups they need to function a container I really don't care.
>
> I just know that if we use container for just the subsystem level
> it makes effective communication impossible, and code reviews
> essentially impossible. As the description says one thing the
> reviewer reads it as another and then the patch does not match
> the description. Leading to NAKs.
>
> Resource groups at least for subset of subsystems that aren't
> namespaces sounds reasonable. Heck resource group, resource
> controller, resource subsystem, resource just about anything seems
> sane to me.

>
> The important part is that we find a vocabulary without doubly
> defined words so we can communicate and a small common set we can
> agree on so people can work on and implement the individual
> resource controllers/groups, and get the individual pieces merged
> as they are reading.

from my personal PoV the following would be fine:

spaces (for the various 'spaces')

- similar enough to the old namespace
- can be easily used with prefix/postfix
  like in pid_space, mnt_space, uts_space etc
- AFAIK, it is not used yet for anything else

container (for resource accounting/limits)

- has the 'containment' principle built in :)
- is used in similar ways in other solutions
- sounds similar to context (easy to associate)

note: I'm also fine with other names, as long as
we find some useable vocabulary soon, as the
different terms start confusing me on a regular
basis, and we do not go for already used names,
which would clash with Linux-VServer or OpenVZ
terminology (which would confuse the hell out of
the end-users :)

best,
Herbert

> Eric
> ____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers
____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Paul Jackson on Fri, 09 Mar 2007 04:37:08 GMT

Matt wrote:
> It's like that Star Trek episode ... except we can't agree on the name

Usually, when there is this much heat and smoke over a name, there is
really an underlying disagreement or misunderstanding over the meaning
of something.

The name becomes the proxy for meaning ;).

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Sam Vilain on Sat, 10 Mar 2007 09:06:11 GMT

View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Wed, Mar 07, 2007 at 11:44:58PM -0700, Eric W. Biederman wrote:
>
>> I really don't much care as long as we don't start redefining
>> container as something else. I think the IBM guys took it from
>> solaris originally which seems to define a zone as a set of
>> isolated processes (for us all separate namespaces). And a container
>> as a set of as a zone that uses resource control. Not exactly how
>> we have been using the term but close enough not to confuse someone.
>>
>> As long as we don't go calling the individual subsystems or the
>> process groups they need to function a container I really don't care.
>> [...]
>> Resource groups at least for subset of subsystems that aren't
>> namespaces sounds reasonable. Heck resource group, resource
>> controller, resource subsystem, resource just about anything seems
>> sane to me.
>>
>> The important part is that we find a vocabulary without doubly
>> defined words so we can communicate and a small common set we can
>> agree on so people can work on and implement the individual
>> resource controllers/groups, and get the individual pieces merged

>> as they are reading.
>>
>
> from my personal PoV the following would be fine:
>
> spaces (for the various 'spaces')
>
>   - similar enough to the old namespace
>   - can be easily used with prefix/postfix
>   like in pid_space, mnt_space, uts_space etc
>   - AFAIK, it is not used yet for anything else
>
> container (for resource accounting/limits)
>
>   - has the 'containment' principle built in :)
>   - is used in similar ways in other solutions
>   - sounds similar to context (easy to associate)
>
> note: I'm also fine with other names, as long as
> we find some useable vocabulary soon, [...]

I like these a lot, particularly in that "mount space" could be a
reasonable replacement for "namespace".

As a result of this discussion, I see the sense in Paul Menage's
original choice of term.

There's just one problem. We'd have to rename the mailing list to
"spaces and containers" :-)

Sam.

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Paul Jackson on Sun, 11 Mar 2007 21:15:55 GMT

View Forum Message <> Reply to Message

Sam, responding to Herbert:
> > from my personal PoV the following would be fine:
> >
> > spaces (for the various 'spaces')
> >   ...
> > container (for resource accounting/limits)

> >   ...
>
> I like these a lot ...

Hmmm ... ok ...

Let me see if I understand this.

We have actors, known as threads, tasks or processes, which use things, which are instances of such classes of things as disk partitions, file systems, memory, cpus, and semaphores.

We assign names to these things, such as SysV id's to the semaphores, mount points to the file systems, pathnames to files and file descriptors to open files. These names provide handles that are typically more convenient and efficient to use, but alas less persistent, less ubiquitous, and needing of some dereferencing when used, to identify the underlying thing.

Any particular assignment of names to some of the things in particular class forms one namespace (aka 'space', above). For each class of things, a given task is assigned one such namespace. Typically many related tasks (such as all those of a login session or a job) will be assigned the same set of namespaces, leading to various opportunities for optimizing the management of namespaces in the kernel.

This assignment of names to things is neither injective nor surjective nor even a complete map.

For example, not all file systems are mounted, certainly not all possible mount points (all directories) serve as mount points, sometimes the same file system is mounted in multiple places, and sometimes more than one file system is mounted on the same mount point, one hiding the other.

In so far as the code managing this naming is concerned, the names are usually fairly arbitrary, except that there seems to be a tendency toward properly virtualizing these namespaces, presenting to a task the namespaces assigned it as if that was all there was, hiding the presence of alternative namespaces, and intentionally not providing a 'global view' that encompasses all namespaces of a given class.

This tendency culminates in the full blown virtual machines, such as Xen and KVM, which virtualize more or less all namespaces.

Because the essential semantics relating one namespace to another are rather weak (the namespaces for any given class of things are or can be pretty much independent of each other), there is a preference and a tradition to keep such sets of namespaces a simple flat space.

Conclusions regarding namespaces, aka spaces:

A namespace provide a set of convenient handles for things of a particular class.

For each class of things, every task gets one namespace (perhaps a Null or Default one.)

Namespaces are partial virtualizations, the 'space of namespaces' is pretty flat, and the assignment of names in one namespace is pretty independent of the next.

===

That much covers what I understand (perhaps in error) of namespaces.

So what's this resource accounting/limits stuff?

I think this depends on adding one more category to our universe.

For the purposes of introducing yet more terms, I will call this new category a "metered class."

Each time we set about to manage some resource, we tend to construct some more elaborate "metered classes" out of the elemental classes of things (partitions, cpus, ...) listed above.

Examples of these more elaborate metered classes include percentages of a networks bandwidth, fractions of a nodes memory (the fake numa patch), subsets of the systems cpus and nodes (cpusets), ...

These more elaborate metered classes each have fairly 'interesting' and specialized forms. Their semantics are closely adapted to the underlying class of things from which they are formed, and to the usually challenging, often conflicting, constraints on managing the usage of such a resource.

For example, the rules that apply to percentages of a networks bandwidth have little in common with the rules that apply to sets of subsets of a systems cpus and nodes.

We then attach tasks to these metered classes. Each task is assigned one metered instance from each metered class. For example, each task is assigned to a cpuset.

For metered classes that are visible across the system, we tend

to name these classes, and then use those names when attaching
tasks to them. See for example cpusets.

For metered classes that are only privately visible within the
current context of a task, such as setrlimit, set_mempolicy,
mbind and set_mempolicy, we tend to implicitly attach each task
to its current metered class and provide it explicit means
to manipulate the individual attributes of that metered class
by direct system calls.

Conclusions regarding metered classes, aka containers:

Unlike namespaces, metered classes have rich and varied semantics,
sometimes elaborate inheritance and transfer rules, and frequently
non-flat topologies.

Depending on the scope of visibility of a metered class, it may
or may not have much of a formal name space.

===

My current understanding of Paul Menage's container patch is that it is
a useful improvement for some of the metered classes - those that could
make good use of a file system like hierarchy for their interface.
It probably doesn't benefit all metered classes, as they won't all
benefit from a file system like hierarchy, or even have a formal name
space, and it doesn't seem to benefit the name space implementation,
which happily remains flat.

I could easily be wrong in this understanding.

===

For those metered classes which have system wide names, it may -seem-
that attaching tasks to selected instances of those classes is much
the same operation as is attaching a task to a namespace.   Perhaps
the sense that this was so has been the driving force behind trying
to unite namespaces and containers.

However, as I've described above, these seem rather different to me.
The underlying semantics, topology, and variety are different, and
some of these differences are necessarily exposed at the point that
we attach tasks to namespaces or containers.

Moreover, each of these namespaces and each of these metered classes
typically has its own dedicated API across the kernel-user boundary,
so sharing of kernel implementation internals is mostly just a
private matter for the kernel.

Conclusions:

  We're discussing kernel internal optimizations here, not
  user visible API issues. As Serge keeps reminding us, this is
  just an optimization.

  I tend to favor keeping spaces and containers relatively separate,
  minimizing the entwining of them to where there is a clear
  performance (space, time, or scaling) win, and to where that
  optimization doesn't depend on trying to force fit either spaces
  or containers into the mold of the other.

--
                  I won't rest till it's the best ...
                  Programmer, Linux Scalability
                  Paul Jackson <pj@sgi.com> 1.925.600.0401

____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Sam Vilain on Mon, 12 Mar 2007 09:35:17 GMT
View Forum Message <> Reply to Message

Allow me to annotate your nice summary. A lot of this is elaborating on
what you are saying; and I think where we disagree, the differences are
not important.

Paul Jackson wrote:
> We have actors, known as threads, tasks or processes, which use things,
> which are instances of such classes of things as disk partitions,
> file systems, memory, cpus, and semaphores.
>
> We assign names to these things, such as SysV id's to the semaphores,
> mount points to the file systems, pathnames to files and file
> descriptors to open files. These names provide handles that
> are typically more convenient and efficient to use, but alas less
> persistent, less ubiquitous, and needing of some dereferencing when
> used, to identify the underlying thing.
>
> Any particular assignment of names to some of the things in particular
> class forms one namespace (aka 'space', above). For each class of
> things, a given task is assigned one such namespace. Typically many
> related tasks (such as all those of a login session or a job) will be

> assigned the same set of namespaces, leading to various opportunities
> for optimizing the management of namespaces in the kernel.
>
> This assignment of names to things is neither injective nor surjective
> nor even a complete map.
>
> For example, not all file systems are mounted, certainly not all
> possible mount points (all directories) serve as mount points,
> sometimes the same file system is mounted in multiple places, and
> sometimes more than one file system is mounted on the same mount point,
> one hiding the other.
>

Right, which is why I preferred the term "mount space" or "mount
namespace". The keys in the map are not as important as the presence of
the independent map itself.

Unadorned "namespaces" is currently how they are known, and short of
becoming the Hurd I don't think this term is appropriate for Linux.

> In so far as the code managing this naming is concerned, the names are
> usually fairly arbitrary, except that there seems to be a tendency
> toward properly virtualizing these namespaces, presenting to a task
> the namespaces assigned it as if that was all there was, hiding the
> presence of alternative namespaces, and intentionally not providing a
> 'global view' that encompasses all namespaces of a given class.
>
> This tendency culminates in the full blown virtual machines, such as
> Xen and KVM, which virtualize more or less all namespaces.
>

Yes, these systems, somewhat akin to microkernels, virtualize all
namespaces as a byproduct of their nature.

> Because the essential semantics relating one namespace to another are
> rather weak (the namespaces for any given class of things are or can
> be pretty much independent of each other), there is a preference and
> a tradition to keep such sets of namespaces a simple flat space.
>

This has been the practice to date with most worked implementations,
with the proviso that as the feature becomes standard people may start
expecting spaces within spaces to work indistinguishably from top level
spaces; in fact, perhaps there should be no such distinction between a
"top" space and a subservient space, other than the higher level space
is aware of the subservient space.

Consider, for instance, that BIND already uses Linux kernel features

which are normally only attributed to the top space, such as adjusting
ulimits and unsetting capability bits. This kind of application
self-containment may become more commonplace.

And this perhaps begs the question: is it worth the performance penalty,
or must there be one?

> Conclusions regarding namespaces, aka spaces:
>
> A namespace provide a set of convenient handles for things of a
> particular class.
>
> For each class of things, every task gets one namespace (perhaps
> a Null or Default one.)
>

Conceptually, every task exists in exactly one space of each type,
though that space may see and/or administer other spaces.

> Namespaces are partial virtualizations, the 'space of namespaces'
> is pretty flat, and the assignment of names in one namespace is
> pretty independent of the next.
>
> ===
>
> That much covers what I understand (perhaps in error) of namespaces.
>
> So what's this resource accounting/limits stuff?
>
> I think this depends on adding one more category to our universe.
>
> For the purposes of introducing yet more terms, I will call this
> new category a "metered class."
>

The term "metered" implies "a resource which renews over time". How does
this apply to a fixed limit? A limit's nominal unit may not be delimited
in terms of time, but it must be continually maintained, so it can be
"metered" in terms of use of that limit over time.

For instance, a single system in scheduling terms is limited to the use
of the number of CPUs present in the system. So, while it has a "limit"
of 2 CPUs, in terms of a metered resource, it has a maximum rate of 2
CPU seconds per second.

> Each time we set about to manage some resource, we tend to construct
> some more elaborate "metered classes" out of the elemental classes
> of things (partitions, cpus, ...) listed above.

>
> Examples of these more elaborate metered classes include percentages
> of a networks bandwidth, fractions of a nodes memory (the fake numa
> patch), subsets of the systems cpus and nodes (cpusets), ...
>

Indeed, and some of the key performance benefits of the containers
approach is that the resource limits may be implemented in a "soft"
fashion, that can makes available system resource in demand by some
containers. For instance, available RAM, or space in a filesystem.

> These more elaborate metered classes each have fairly 'interesting'
> and specialized forms. Their semantics are closely adapted to the
> underlying class of things from which they are formed, and to the
> usually challenging, often conflicting, constraints on managing the
> usage of such a resource.
>
> For example, the rules that apply to percentages of a networks
> bandwidth have little in common with the rules that apply to sets of
> subsets of a systems cpus and nodes.
>
> We then attach tasks to these metered classes. Each task is assigned
> one metered instance from each metered class. For example, each task
> is assigned to a cpuset.
>
> For metered classes that are visible across the system, we tend
> to name these classes, and then use those names when attaching
> tasks to them. See for example cpusets.
>
> For metered classes that are only privately visible within the
> current context of a task, such as setrlimit, set_mempolicy,
> mbind and set_mempolicy, we tend to implicitly attach each task
> to its current metered class and provide it explicit means
> to manipulate the individual attributes of that metered class
> by direct system calls.
>
> Conclusions regarding metered classes, aka containers:
>
>   Unlike namespaces, metered classes have rich and varied semantics,
>   sometimes elaborate inheritance and transfer rules, and frequently
>   non-flat topologies.
>

I think this is most true from the perspective of the person managing
the system. They may set up arbitrarily complex rules to manage the
systems they are responsible for.

Namespaces may transfer and inherit properties as well. For instance in

the case of mount namespaces, clones of a mountspaces may in existing
kernels be set to receive updates of mounts in the mountspace it was
cloned from. In the case of a level 3 network space (netspace?
ipspace?), the parent namespace is responsible for the routing and
master iptables, and there may be rules determined about the
interoperation between, for instance the parent iptables and the
iptables which processes in the subservient space can affect.

The difference is that these relationships would not be expected to be
tuned, so much as a few common arrangements of inheritance and transfer
selected between. For instance, not every level 3 network space should
be able to access control iptables rules for its addresses.

>   Depending on the scope of visibility of a metered class, it may
>   or may not have much of a formal name space.
>
> ===
>
> My current understanding of Paul Menage's container patch is that it is
> a useful improvement for some of the metered classes - those that could
> make good use of a file system like hierarchy for their interface.
> It probably doesn't benefit all metered classes, as they won't all
> benefit from a file system like hierarchy, or even have a formal name
> space, and it doesn't seem to benefit the name space implementation,
> which happily remains flat.
>

Utmost care should be taken to checkpoint/migration as this filesystem
is developed; nothing should be revealed which would not be possible to
keep after a checkpoint + migration, as there may be running processes
which are inspecting the data available through the interface.

> I could easily be wrong in this understanding.
>
> ===
>
> For those metered classes which have system wide names, it may -seem-
> that attaching tasks to selected instances of those classes is much
> the same operation as is attaching a task to a namespace.   Perhaps
> the sense that this was so has been the driving force behind trying
> to unite namespaces and containers.
>
> However, as I've described above, these seem rather different to me.
> The underlying semantics, topology, and variety are different, and
> some of these differences are necessarily exposed at the point that
> we attach tasks to namespaces or containers.
>

Different, I would say, largely because to be similar it would require
the abstractions and systems of the kernel to have been built using the
container / space as the starting point. So corresponding concepts meet
varying semantics on either side, so any attempt to build the systems
together would be fraught with difficulty.

> Moreover, each of these namespaces and each of these metered classes
> typically has its own dedicated API across the kernel-user boundary,
> so sharing of kernel implementation internals is mostly just a
> private matter for the kernel.
>
> Conclusions:
>
> We're discussing kernel internal optimizations here, not
> user visible API issues. As Serge keeps reminding us, this is
> just an optimization.
>
> I tend to favor keeping spaces and containers relatively separate,
> minimizing the entwining of them to where there is a clear
> performance (space, time, or scaling) win, and to where that
> optimization doesn't depend on trying to force fit either spaces
> or containers into the mold of the other.

Keep the last word.
Sam.

____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Paul Menage on Mon, 12 Mar 2007 10:00:25 GMT
View Forum Message <> Reply to Message

On 3/11/07, Paul Jackson <pj@sgi.com> wrote:
>
> My current understanding of Paul Menage's container patch is that it is
> a useful improvement for some of the metered classes - those that could
> make good use of a file system like hierarchy for their interface.
> It probably doesn't benefit all metered classes, as they won't all
> benefit from a file system like hierarchy, or even have a formal name
> space, and it doesn't seem to benefit the name space implementation,
> which happily remains flat.

Well, what I was aiming at was a generic mechanism that can handle
"namespaces", "metered classes" and other ways of providing

per-task-group behaviour. So a system-call API doesn't necessarily
have the right flexibility to implement the possible different kinds
of subsystems I envisage.

For example, one way to easily tie groups of processes to different
network queues is to have a tag associated with a container, allow
that to propagate to the socket/skbuf priority field, and then use
standard Linux traffic control to pick the appropriate outgoing queue
based on the skbuf's tag.

This isn't really a namespace, and it isn't really a "metered class".
It's just a way of associating a piece of data (the network tag) with
a group of processes.

With a filesystem-based interface, it's easy to have a file as the
method of reading/writing the tag; with a system call interface, then
either the interface is sufficiently generic to allow this kind of
data association (in which case you're sort of implementing a
filesystem in the system call) or else you have to shoehorn into an
unrelated API (e.g. if your system call talks about "resource limits"
you might end up having to specify the network tag as a "maximum
limit" since there's no other useful configuration data available).

As another example, I'd like to have a subsystem that shows me all the
sockets that processes in the container have opened; again, easy to do
in a filesystem interface, but hard to fit into a
resource-meteting-centric or namespace-centric system call API.

Paul

Subject: Re: [ckrm-tech] [PATCH 0/2] resource control file system - aka containers on top of nsproxy!
Posted by Herbert Poetzl on Mon, 12 Mar 2007 23:21:43 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 03:00:25AM -0700, Paul Menage wrote:
> On 3/11/07, Paul Jackson <pj@sgi.com> wrote:
> >
> > My current understanding of Paul Menage's container patch is that it is
> > a useful improvement for some of the metered classes - those that could
> > make good use of a file system like hierarchy for their interface.
> > It probably doesn't benefit all metered classes, as they won't all
> > benefit from a file system like hierarchy, or even have a formal name

> > space, and it doesn't seem to benefit the name space implementation,
> > which happily remains flat.
>
> Well, what I was aiming at was a generic mechanism that can handle
> "namespaces", "metered classes" and other ways of providing
> per-task-group behaviour. So a system-call API doesn't necessarily
> have the right flexibility to implement the possible different kinds
> of subsystems I envisage.
>
> For example, one way to easily tie groups of processes to different
> network queues is to have a tag associated with a container, allow
> that to propagate to the socket/skbuf priority field, and then use
> standard Linux traffic control to pick the appropriate outgoing queue
> based on the skbuf's tag.
>
> This isn't really a namespace, and it isn't really a "metered class".
> It's just a way of associating a piece of data (the network tag) with
> a group of processes.
>
> With a filesystem-based interface, it's easy to have a file as the
> method of reading/writing the tag; with a system call interface, then
> either the interface is sufficiently generic to allow this kind of
> data association (in which case you're sort of implementing a
> filesystem in the system call) or else you have to shoehorn into an
> unrelated API (e.g. if your system call talks about "resource limits"
> you might end up having to specify the network tag as a "maximum
> limit" since there's no other useful configuration data available).
>
> As another example, I'd like to have a subsystem that shows me all the
> sockets that processes in the container have opened; again, easy to do
> in a filesystem interface, but hard to fit into a
> resource-meteting-centric or namespace-centric system call API.

why? you simply enter that specific space and
use the existing mechanisms (netlink, proc, whatever)
to retrieve the information with _existing_ tools,
no need to do anything unusual via the syscall API

and if you support a 'spectator' context or capability
(which allows to see the _whole_ truth) you can also
get this information for _all_ sockets with existing
tools like netstat or lsof ...

best,
Herbert

> Paul
> ____

> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

____