Subject: Re: [ckrm-tech] [PATCH 00/10] Containers(V10): Generic Process Containers
Posted by serue on Thu, 07 Jun 2007 18:01:58 GMT
View Forum Message <> Reply to Message

Quoting Paul Jackson (pj@sgi.com):
> > I suppose as a cleaner alternative we could
> > add a container_subsys->inherit_defaults() handler, to be called at
> > container_clone(), and for cpusets this would set cpus and mems to
> > the parent values - sibling exclusive values.  If that comes to nothing,
> > then the attach_task() is still refused, and the unshare() or clone()
> > fails, but this time with good reason.
>
> Unfortunately, I haven't spent the time I should thinking about
> container cloning, namespaces and such.
>
> I don't know, for the workloads that matter to me, when, how or
> if this container cloning will be used.
>
> I'm tempted to suggest the following.
>
> First, I am assuming that the classic method of creating cpuset
> children will still work, such as the following (which can fail
> for certain combinations of exclusive cpus or mems):
>  cd /dev/cpuset/foobar
>  mkdir foochild
>  cp cpus foochild
>  cp mems foochild
>  echo $$ > foochild/tasks
>
> Second, given that, how about you fail the unshare() or clone()
> anytime that the instance to be cloned has any sibling cpusets
> with any exclusive flags set.

The below patch (on top of my previous patch) does basically that.  But I wasn't able to test it, bc i wasn't able to set cpus_exclusive...

For /cpusets/set0/set1 to have cpu 1 exclusively, does /cpusets/set0 also have to have it exclusively?

If so, then clearly this approach won't work, since if any container has exclusive cpus, then every container will have siblings with exclusive cpus, and unshare still isn't possible on the system.

> The exclusive property is not really on friendly terms with cloning.
>
> Now if the above classic code must be encoded using cloning under
> the covers, then we've got problems, probably more problems than

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

thanks,
-serge

>From 821de58b6ba446e50225606e907baac00130586c Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>
Date: Thu, 7 Jun 2007 13:53:43 -0400
Subject: [PATCH 1/1] containers: implement subsys->auto_setup

container_clone() in one step creates a new container and moves
the current task into it.  Since cpusets do not automatically
fill in the allowed cpus and mems, and do not allow a task to
be attached without these filled in, composing the ns subsystem,
which uses container_clone(), and the cpuset subsystem, results
in sys_unshare() (and clone(CLONE_NEWNS)) always being denied.

To allow the two subsystems to be meaningfully composed, implement
subsystem->auto_setup, called from container_clone() after
creating the new container.

Only the cpuset_auto_setup() is currently implemented.  If any
sibling containers have exclusive cpus or mems, then the cpus
and mems are not filled in for the new container, meaning that
unshare/clone(CLONE_NEWNS) will be denied.  However so long as
no siblings have exclusive cpus or mems, the new container's
cpus and mems are inherited from the parent container.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
 Documentation/containers.txt |    7 +++++++
 include/linux/container.h    |    1 +
 kernel/container.c           |    7 +++++++
 kernel/cpuset.c              |   21 +++++++++++++++++++++
 4 files changed, 36 insertions(+), 0 deletions(-)

diff --git a/Documentation/containers.txt b/Documentation/containers.txt
index ae159b9..28c9e10 100644
--- a/Documentation/containers.txt
+++ b/Documentation/containers.txt
@@ -514,6 +514,13 @@ include/linux/container.h for details).  Note that although this
 method can return an error code, the error code is currently not
 always handled well.

```
+void auto_setup(struct container_subsys *ss, struct container *cont)
+
+Called at container_clone() to do any paramater initialization
+which might be required before a task could attach.  For example
+in cpusets, no task may attach before 'cpus' and 'mems' are
+set up.
+
 void bind(struct container_subsys *ss, struct container *root)
 LL=callback_mutex

diff --git a/include/linux/container.h b/include/linux/container.h
index 37c0bdf..d809b41 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -213,6 +213,7 @@ struct container_subsys {
  void (*exit)(struct container_subsys *ss, struct task_struct *task);
  int (*populate)(struct container_subsys *ss,
    struct container *cont);
+ void (*auto_setup)(struct container_subsys *ss, struct container *cont);
  void (*bind)(struct container_subsys *ss, struct container *root);
  int subsys_id;
  int active;
diff --git a/kernel/container.c b/kernel/container.c
index 988cd8b..e0793f4 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -2316,6 +2316,7 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
  struct inode *inode;
  struct css_group *cg;
  struct containerfs_root *root;
+ struct container_subsys *ss;

  /* We shouldn't be called by an unregistered subsystem */
  BUG_ON(!subsys->active);
@@ -2397,6 +2398,12 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)
   goto again;
  }

+ /* do any required auto-setup */
+ for_each_subsys(root, ss) {
+  if (ss->auto_setup)
+   ss->auto_setup(ss, child);
+ }
+
  /* All seems fine. Finish by moving the task into the new container */
```

```
  ret = attach_task(child, tsk);
  mutex_unlock(&container_mutex);
diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index 0f9ce7d..ff01aaa 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
@@ -1189,6 +1189,26 @@ int cpuset_populate(struct container_subsys *ss, struct container
*cont)
  return 0;
 }

+void cpuset_auto_setup(struct container_subsys *ss,
+  struct container *container)
+{
+ struct container *parent, *child;
+ struct cpuset *cs, *parent_cs;
+
+ parent = container->parent;
+ list_for_each_entry(child, &parent->children, sibling) {
+  cs = container_cs(child);
+  if (is_mem_exclusive(cs) || is_cpu_exclusive(cs))
+   return;
+ }
+ cs = container_cs(container);
+ parent_cs = container_cs(parent);
+
+ cs->mems_allowed = parent_cs->mems_allowed;
+ cs->cpus_allowed = parent_cs->cpus_allowed;
+ return;
+}
+
 /*
  * cpuset_create - create a cpuset
  * parent: cpuset that will be parent of the new cpuset.
@@ -1249,6 +1269,7 @@ struct container_subsys cpuset_subsys = {
  .can_attach = cpuset_can_attach,
  .attach = cpuset_attach,
  .populate = cpuset_populate,
+ .auto_setup = cpuset_auto_setup,
  .subsys_id = cpuset_subsys_id,
  .early_init = 1,
 };
--
1.5.1.1.GIT
```