Subject: Re: [ckrm-tech] [PATCH 00/10] Containers(V10): Generic Process Containers

Posted by serge on Fri, 08 Jun 2007 19:42:47 GMT

View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On 6/8/07, Serge E. Hallyn <serue@us.ibm.com> wrote:
> >
> >I do fear that that could become a maintenance nightmare. For instance
> >right now there's the call to fsnotify_mkdir(). Other such hooks might
> >be placed at vfs_mkdir, which we'd then likely want to have placed in
> >our container_mkdir() and container_clone() fns. And of course
> >may_create() is static inline in fs/namei.c. It's trivial, but still if
> >it changes we'd want to change the version in kernel/container.c as
> >well.
>
> Do we need to actually need to respect may_create() in
> container_clone()? I guess it would provide a way for root to control
> which processes could unshare namespaces.
>
> >
> >What would be the main advantage of doing it this way? Do you consider
> >the extra subys->auto_setup() hook to be avoidable bloat?
> >
>
> I was thinking that it would be nice to be able to atomically set up
> the resources in the new container at the point when it's created
> rather than later. But I guess this way can work too. Can we call it
> something like "clone()" rather than "auto_setup()"?
>
> Paul

clone() implies it does the actual cloning, so how about post_clone()
as in the patch below?

I'm still not saying I'm entirely opposed to moving the vfs_mkdir logic
straight into container_clone() - it's more that I would expect other
people to object when they saw that. So if you decide you don't like
the end result with this patch, let me know and I'll give that a shot.

Paul (Jackson), is this comment added in cpusets close enough to what
you were asking for?

thanks,
-serge

>From c2f1a39b231f06cb524c6e95d74de6ddee286f25 Mon Sep 17 00:00:00 2001
From: Serge E. Hallyn <serue@us.ibm.com>

Date: Fri, 8 Jun 2007 15:36:59 -0400
Subject: [PATCH 4/4] containers: minor clone cleanup

rename auto_setup() to post_clone(), and comment the cpusets version.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>

```
---
 Documentation/containers.txt |   10 +++++-----
 include/linux/container.h    |    2 +-
 kernel/container.c           |    4 ++--
 kernel/cpuset.c              |   20 ++++++++++++++++++--
 4 files changed, 26 insertions(+), 10 deletions(-)

diff --git a/Documentation/containers.txt b/Documentation/containers.txt
index 28c9e10..9fdb808 100644
--- a/Documentation/containers.txt
+++ b/Documentation/containers.txt
@@ -514,12 +514,12 @@ 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)
+void post_clone(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.
+Called at the end of 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 d809b41..1a83913 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -213,7 +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 (*post_clone)(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 e0793f4..11e326a 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -2400,8 +2400,8 @@ int container_clone(struct task_struct *tsk, struct container_subsys
*subsys)

  /* do any required auto-setup */
  for_each_subsys(root, ss) {
-  if (ss->auto_setup)
-   ss->auto_setup(ss, child);
+  if (ss->post_clone)
+   ss->post_clone(ss, child);
  }

  /* All seems fine. Finish by moving the task into the new container */
diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index ff01aaa..ecefb1d 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
@@ -1189,7 +1189,23 @@ int cpuset_populate(struct container_subsys *ss, struct container
*cont)
  return 0;
 }

-void cpuset_auto_setup(struct container_subsys *ss,
+/*
+ * post_clone() is called at the end of container_clone().
+ * 'container' was just created automatically as a result of
+ * a container_clone(), and the current task is about to
+ * be moved into 'container'.
+ *
+ * Currently we refuse to set up the container - thereby
+ * refusing the task to be entered, and as a result refusing
+ * the sys_unshare() or clone() which initiated it - if any
+ * sibling cpusets have exclusive cpus or mem.
+ *
+ * If this becomes a problem for some users who wish to
+ * allow that scenario, then cpuset_post_clone() could be
+ * changed to grant parent->cpus_allowed-sibling_cpus_exclusive
+ * (and likewise for mems) to the new container.
+ */
+void cpuset_post_clone(struct container_subsys *ss,
   struct container *container)
 {
  struct container *parent, *child;
@@ -1269,7 +1285,7 @@ struct container_subsys cpuset_subsys = {
  .can_attach = cpuset_can_attach,
  .attach = cpuset_attach,
  .populate = cpuset_populate,
- .auto_setup = cpuset_auto_setup,
+ .post_clone = cpuset_post_clone,
  .subsys_id = cpuset_subsys_id,
  .early_init = 1,
 };
--
1.5.1.1.GIT
```