Subject: [RFC] [PATCH 0/3] containers: introduction
Posted by serue on Wed, 20 Dec 2006 06:01:28 GMT
View Forum Message <> Reply to Message

Following is a small patchset implementing what I was describing
about earlier, namely semantics for a hierarchical container
naming scheme.

What works:

1. ls -l /proc/$$/container
   shows the full hierarchical name of the container;

2. mount -t containerfs none /container
   results in a file tree under /container representing the
   full container hierarchy

3. cd /proc/$$/container; ls
   results in a listing of child containers

What doesn't work:
1. The /proc/$$/container link always appears dead (red
in bash on my fedora test system) because it points
into a kern_mounted fs.

2. Features like

cd /proc/$$/container
mv container_3 my_child_container

to rename a container or

cd /proc/$$/container
rm container_3

to kill all processes a container are unimplemented.

3. Semantics for entering a namespace are not only
unimplemented, but entirely unconsidered thus far.
I suppose one cool way to enter a container would
be

ln -s /proc/$$/container/child_container /proc/$$/container

but that

a. Does not provide the ability to switch only
some of the namespaces, as Herbert wants.

b. May be unimplementable using proc support
as is - not sure.

thanks,
-serge

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

Subject: [RFC] [PATCH 1/3] container: implement 'containers' as a namespace naming device
Posted by serue on Wed, 20 Dec 2006 06:02:02 GMT
View Forum Message <> Reply to Message

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [RFC] [PATCH 1/3] container: implement 'containers' as a namespace naming device

Implement containers and their reference counting, as the device
for providing simple, hierarchical naming of namespaces.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
include/linux/container.h | 24 ++++++++++
include/linux/init_task.h | 13 +++++
include/linux/nsproxy.h | 2 +
kernel/Makefile | 2 -
kernel/container.c | 108 ++
kernel/nsproxy.c | 11 ++++-
6 files changed, 158 insertions(+), 2 deletions(-)

diff --git a/include/linux/container.h b/include/linux/container.h
new file mode 100644
index 0000000..fcd85f3
--- /dev/null
+++ b/include/linux/container.h
@@ -0,0 +1,24 @@
+#ifndef _LINUX_CONTAINER_H
+#define _LINUX_CONTAINER_H
+
+#include <linux/spinlock.h>
+#include <linux/list.h>
+#include <linux/kref.h>
+
+struct nsproxy;
+
+struct container {

```
+ struct container *parent;
+ char *name;
+ struct nsproxy *nsproxy;
+ struct list_head children;
+ struct list_head peers;
+ struct kref ref;
+};
+extern struct container init_container;
+
+void put_container(struct container *c);
+struct container *new_container(struct container *parent,
+      struct nsproxy *nsproxy);
+
+#endif
diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index a2d95ff..445a556 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -70,6 +70,18 @@ #define INIT_SIGNALS(sig) {      \
  { .__session      = 1},      \
 }

+/* presumably the init container name will come from .config */
+#define INIT_CONTAINER_NAME "init_container"
+extern struct container init_container;
+#define INIT_CONTAINER(container) {     \
+ .parent  = &init_container,    \
+ .name  = INIT_CONTAINER_NAME,    \
+ .nsproxy = &init_nsproxy,    \
+ .children = LIST_HEAD_INIT(container.children),  \
+ .peers  = LIST_HEAD_INIT(container.peers),  \
+ .ref  = {.refcount = ATOMIC_INIT(1)},   \
+}
+
 extern struct nsproxy init_nsproxy;
 #define INIT_NSPROXY(nsproxy) {      \
  .pid_ns  = &init_pid_ns,     \
@@ -77,6 +89,7 @@ #define INIT_NSPROXY(nsproxy) {      \
  .nslock  = __SPIN_LOCK_UNLOCKED(nsproxy.nslock),  \
  .uts_ns  = &init_uts_ns,     \
  .mnt_ns  = NULL,      \
+ .container = &init_container,    \
  INIT_IPC_NS(ipc_ns)      \
 }

diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 0b9f0dc..30c9876 100644
--- a/include/linux/nsproxy.h
```

```
+++ b/include/linux/nsproxy.h
@@ -8,6 +8,7 @@ struct mnt_namespace;
 struct uts_namespace;
 struct ipc_namespace;
 struct pid_namespace;
+struct container;

 /*
  * A structure to contain pointers to all per-process
@@ -28,6 +29,7 @@ struct nsproxy {
  struct ipc_namespace *ipc_ns;
  struct mnt_namespace *mnt_ns;
  struct pid_namespace *pid_ns;
+ struct container *container;
 };
 extern struct nsproxy init_nsproxy;

diff --git a/kernel/Makefile b/kernel/Makefile
index 839a58b..e5d82c1 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -8,7 +8,7 @@ obj-y     = sched.o fork.o exec_domain.o
      signal.o sys.o kmod.o workqueue.o pid.o \
      rcupdate.o extable.o params.o posix-timers.o \
      kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
-     hrtimer.o rwsem.o latency.o nsproxy.o srcu.o
+     hrtimer.o rwsem.o latency.o nsproxy.o srcu.o container.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
diff --git a/kernel/container.c b/kernel/container.c
new file mode 100644
index 0000000..ed3269f
--- /dev/null
+++ b/kernel/container.c
@@ -0,0 +1,108 @@
+/*
+ *  Copyright (C) 2006 IBM Corporation
+ *
+ *  Author: Serge Hallyn <serue@us.ibm.com>
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License as
+ *  published by the Free Software Foundation, version 2 of the
+ *  License.
+ *
+ *  Jun 2006 - namespaces support
+ *             OpenVZ, SWsoft Inc.
```

```
+ *             Pavel Emelianov <xemul@openvz.org>
+ */
+
+#include <linux/module.h>
+#include <linux/version.h>
+#include <linux/container.h>
+#include <linux/init_task.h>
+
+struct nsproxy;
+
+struct container init_container = INIT_CONTAINER(init_container);
+
+/*
+ * free_container: called from rcu_call when all references
+ * are gone
+ * all references won't be gone until all children are already
+ * freed.
+ */
+static void free_container(struct kref *ref)
+{
+ struct container *c = container_of(ref, struct container, ref);
+
+ if (c->parent != c)
+  kfree(c->name);
+ if (!list_empty(&c->peers))
+  list_del(&c->peers);
+ kfree(c);
+}
+
+
+/*
+ * get a container reference
+ * we also grab a reference to all it's parents
+ */
+struct container *get_container(struct container *c)
+{
+ struct container *c2 = c;
+
+ if (c) {
+  kref_get(&c2->ref);
+  while (c2->parent != c2) {
+   c2 = c2->parent;
+   kref_get(&c2->ref);
+  }
+ }
+ return c;
+}
+
```

```
+/*
+ * put a container
+ * when we put a container, we also put all it's parents.
+ */
+void put_container(struct container *c)
+{
+ struct container *parent;
+ if (!c)
+  return;
+ parent = c->parent;
+ while (parent != c) {
+  kref_put(&c->ref, free_container);
+  c = parent;
+  parent = parent->parent;
+ }
+ kref_put(&c->ref, free_container);
+}
+
+/* i expect this lock to be moved into the containers... */
+static DEFINE_SPINLOCK(container_lock);
+static int last_unnamed_container = 1;
+
+struct container *new_container(struct container *parent,
+     struct nsproxy *nsproxy)
+{
+ struct container *c;
+
+ c = kmalloc(sizeof(*c), GFP_KERNEL);
+ if (!c)
+  return NULL;
+ c->name = kzalloc(20, GFP_KERNEL);
+ if (!c->name) {
+  kfree(c);
+  return NULL;
+ }
+ spin_lock(&container_lock);
+ sprintf(c->name, "container_%d", last_unnamed_container++);
+ spin_unlock(&container_lock);
+ INIT_LIST_HEAD(&c->peers);
+ c->parent = parent;
+ get_container(parent);
+ list_add_tail(&c->peers, &parent->children);
+ INIT_LIST_HEAD(&c->children);
+ c->nsproxy = nsproxy;
+ kref_init(&c->ref);
+
+ return c;
+}
```

```
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f5b9ee6..0cfa4c4 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -20,6 +20,7 @@ #include <linux/init_task.h>
 #include <linux/mnt_namespace.h>
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>
+#include <linux/container.h>

 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

@@ -46,8 +47,14 @@ static inline struct nsproxy *clone_name
  struct nsproxy *ns;

  ns = kmemdup(orig, sizeof(struct nsproxy), GFP_KERNEL);
- if (ns)
+ if (ns) {
   atomic_set(&ns->count, 1);
+  ns->container = new_container(orig->container, ns);
+  if (!ns->container) {
+   kfree(ns);
+   ns = NULL;
+  }
+ }
  return ns;
 }

@@ -145,5 +152,7 @@ void free_nsproxy(struct nsproxy *ns)
   put_ipc_ns(ns->ipc_ns);
  if (ns->pid_ns)
   put_pid_ns(ns->pid_ns);
+ put_container(ns->container);
+ ns->container->nsproxy = NULL;
  kfree(ns);
 }
--
1.4.1
```

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

Subject: [RFC] [PATCH 2/3] container: create containerfs
Posted by serue on Wed, 20 Dec 2006 06:02:13 GMT

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [RFC] [PATCH 2/3] container: create containerfs

Create containerfs as a view of the hierarchy of containers.

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

```
---
 include/linux/container.h |    1
 kernel/container.c        |  175 +++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 176 insertions(+), 0 deletions(-)

diff --git a/include/linux/container.h b/include/linux/container.h
index fcd85f3..c224a53 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -13,6 +13,7 @@ struct container {
  struct nsproxy *nsproxy;
  struct list_head children;
  struct list_head peers;
+ struct dentry *dentry;
  struct kref ref;
 };
 extern struct container init_container;
diff --git a/kernel/container.c b/kernel/container.c
index ed3269f..6206e72 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -17,10 +17,18 @@ #include <linux/module.h>
 #include <linux/version.h>
 #include <linux/container.h>
 #include <linux/init_task.h>
+#include <linux/fsnotify.h>
+#include <linux/fs.h>
+#include <linux/mount.h>
+
+#define CONTAINERFS_MAGIC 0xb6663caf

 struct nsproxy;

 struct container init_container = INIT_CONTAINER(init_container);
+static struct vfsmount *containerfs_mount;
+static void containerfs_remove(struct dentry *dentry);
+static struct dentry *containerfs_create_dir(struct container *container);

 /*
  * free_container: called from rcu_call when all references
@@ -32,6 +40,7 @@ static void free_container(struct kref *
```

```
 {
  struct container *c = container_of(ref, struct container, ref);

+ containerfs_remove(c->dentry);
  if (c->parent != c)
   kfree(c->name);
  if (!list_empty(&c->peers))
@@ -103,6 +112,172 @@ struct container *new_container(struct c
  INIT_LIST_HEAD(&c->children);
  c->nsproxy = nsproxy;
  kref_init(&c->ref);
+ containerfs_create_dir(c);

  return c;
 }
+
+static struct inode *containerfs_get_inode(struct super_block *sb, int mode, dev_t dev)
+{
+ struct inode *inode = new_inode(sb);
+
+ if (inode) {
+  inode->i_mode = mode;
+  inode->i_uid = 0;
+  inode->i_gid = 0;
+  inode->i_blocks = 0;
+  inode->i_atime = inode->i_mtime = inode->i_ctime = CURRENT_TIME;
+  switch (mode & S_IFMT) {
+  default:
+   printk("%s: whoa: non-dirs for containerfs should not exist\n",
+       __FUNCTION__);
+  case S_IFDIR:
+   inode->i_op = &simple_dir_inode_operations;
+   inode->i_fop = &simple_dir_operations;
+
+   /* directory inodes start off with i_nlink == 2
+    * (for "." entry) */
+   inc_nlink(inode);
+   break;
+  }
+ }
+ return inode;
+}
+
+static int containerfs_mknod(struct inode *dir, struct dentry *dentry,
+    int mode, dev_t dev)
+{
+ struct inode *inode;
+ int error = -EPERM;
```

```
+
+ if (dentry->d_inode)
+  return -EEXIST;
+
+ inode = containerfs_get_inode(dir->i_sb, mode, dev);
+ if (inode) {
+  d_instantiate(dentry, inode);
+  dget(dentry);
+  error = 0;
+ }
+ return error;
+}
+
+#define IS_ROOT_CONTAINER(x) (x->parent == x)
+
+static int containerfs_mkdir(struct inode *dir, struct dentry *dentry, int mode)
+{
+ int res;
+
+ mode = (mode & (S_IRWXUGO | S_ISVTX)) | S_IFDIR;
+ res = containerfs_mknod(dir, dentry, mode, 0);
+ if (!res) {
+  inc_nlink(dir);
+  fsnotify_mkdir(dir, dentry);
+ }
+ return res;
+}
+
+static struct dentry *containerfs_create_dir(struct container *container)
+{
+ struct dentry *dentry = NULL;
+ struct dentry *parent;
+ int error;
+
+ if (IS_ROOT_CONTAINER(container))
+  parent = containerfs_mount->mnt_root;
+ else
+  parent = container->parent->dentry;
+
+ mutex_lock(&parent->d_inode->i_mutex);
+ dentry = lookup_one_len(container->name, parent,
+     strlen(container->name));
+ if (!IS_ERR(dentry)) {
+  error = containerfs_mkdir(parent->d_inode, dentry, S_IRUGO|S_IXUGO);
+  dput(dentry);
+ } else
+  error = PTR_ERR(dentry);
+ mutex_unlock(&parent->d_inode->i_mutex);
```

```
+
+ if (error) {
+  dentry = NULL;
+  goto exit;
+ }
+
+ container->dentry = dentry;
+exit:
+ return dentry;
+}
+
+static inline int containerfs_positive(struct dentry *dentry)
+{
+ return dentry->d_inode && !d_unhashed(dentry);
+}
+
+static void containerfs_remove(struct dentry *dentry)
+{
+ struct dentry *parent;
+ int ret = 0;
+
+ if (!dentry)
+  return;
+
+ parent = dentry->d_parent;
+ if (!parent || !parent->d_inode)
+  return;
+
+ mutex_lock(&parent->d_inode->i_mutex);
+ if (containerfs_positive(dentry)) {
+  if (dentry->d_inode) {
+   dget(dentry);
+   ret = simple_rmdir(parent->d_inode, dentry);
+   if (ret)
+    printk(KERN_ERR
+     "ContainerFS rmdir on %s failed : "
+     "directory not empty.\n",
+     dentry->d_name.name);
+   else
+    d_delete(dentry);
+   dput(dentry);
+  }
+ }
+ mutex_unlock(&parent->d_inode->i_mutex);
+}
+
+static int container_fill_super(struct super_block *sb, void *data, int silent)
+{
```

```
+ static struct tree_descr container_files[] = {{""}};
+
+ return simple_fill_super(sb, CONTAINERFS_MAGIC, container_files);
+}
+
+static int container_get_sb(struct file_system_type *fs_type,
+   int flags, const char *dev_name,
+   void *data, struct vfsmount *mnt)
+{
+ return get_sb_single(fs_type, flags, data, container_fill_super, mnt);
+}
+
+static struct file_system_type containerfs_type = {
+ .owner = THIS_MODULE,
+ .name =  "containerfs",
+ .get_sb = container_get_sb,
+ .kill_sb = kill_litter_super,
+};
+
+static int __init containerfs_init(void)
+{
+ int retval;
+
+ retval = register_filesystem(&containerfs_type);
+
+ if (retval)
+  return retval;
+ containerfs_mount =  kern_mount(&containerfs_type);
+ if (IS_ERR(containerfs_mount))
+  return PTR_ERR(containerfs_mount);
+ containerfs_create_dir(&init_container);
+
+ return 0;
+}
+
+core_initcall(containerfs_init);
--
1.4.1
```


Subject: [RFC] [PATCH 3/3] containers: hook /proc/$$/container into containerfs
Posted by serue on Wed, 20 Dec 2006 06:02:34 GMT

From: Serge E. Hallyn <serue@us.ibm.com>
Subject: [RFC] [PATCH 3/3] containers: hook /proc/$$/container into containerfs

Create a symlink from /proc/$$/container into the containerfs.
What works:

1. ls -l /proc/$$/container
   shows the full hierarchical name of the container;

2. mount -t containerfs none /container
   results in a file tree under /container representing the
   full container hierarchy

3. cd /proc/$$/container; ls
   results in a listing of child containers

What doesn't work:
The /proc/$$/container link always appears dead (red
in bash on my fedora test system) because it points
into a kern_mounted fs.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
---
fs/proc/base.c | 18 ++++++++++++++++++
include/linux/container.h | 1 +
kernel/container.c | 2 +-
3 files changed, 20 insertions(+), 1 deletions(-)

diff --git a/fs/proc/base.c b/fs/proc/base.c
index 77a57b5..00c7618 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -73,6 +73,7 @@ #include <linux/audit.h>
#include <linux/poll.h>
#include <linux/nsproxy.h>
#include <linux/oom.h>
+#include <linux/container.h>
#include "internal.h"

/* NOTE:
@@ -189,6 +190,21 @@ static int proc_root_link(struct inode *
return result;
}

+static int proc_container_link(struct inode *inode, struct dentry **dentry,
+ struct vfsmount **mnt)
+{

```
+ struct task_struct *task = get_proc_task(inode);
+ int result = -ENOENT;
+
+ if (task) {
+  *dentry = dget(task->nsproxy->container->dentry);
+  *mnt = mntget(containerfs_mount);
+  put_task_struct(task);
+  result = 0;
+ }
+ return result;
+}
+
 #define MAY_PTRACE(task) \
  (task == current || \
  (task->parent == current && \
@@ -1879,6 +1895,7 @@ #endif
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
+ LNK("container",  container),
 };

 static int proc_tgid_base_readdir(struct file * filp,
@@ -2157,6 +2174,7 @@ #endif
 #ifdef CONFIG_FAULT_INJECTION
  REG("make-it-fail", S_IRUGO|S_IWUSR, fault_inject),
 #endif
+ LNK("container",  container),
 };

 static int proc_tid_base_readdir(struct file * filp,
diff --git a/include/linux/container.h b/include/linux/container.h
index c224a53..d5d143c 100644
--- a/include/linux/container.h
+++ b/include/linux/container.h
@@ -17,6 +17,7 @@ struct container {
  struct kref ref;
 };
 extern struct container init_container;
+extern struct vfsmount *containerfs_mount;

 void put_container(struct container *c);
 struct container *new_container(struct container *parent,
diff --git a/kernel/container.c b/kernel/container.c
index 6206e72..5ce82b1 100644
--- a/kernel/container.c
+++ b/kernel/container.c
@@ -26,7 +26,7 @@ #define CONTAINERFS_MAGIC 0xb6663caf
```

```
 struct nsproxy;

 struct container init_container = INIT_CONTAINER(init_container);
-static struct vfsmount *containerfs_mount;
+struct vfsmount *containerfs_mount;
 static void containerfs_remove(struct dentry *dentry);
 static struct dentry *containerfs_create_dir(struct container *container);

--
1.4.1
```

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by ebiederm on Sat, 23 Dec 2006 02:55:10 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Following is a small patchset implementing what I was describing
> about earlier, namely semantics for a hierarchical container
> naming scheme.
>
> What works:
>
> 1. ls -l /proc/$$/container
>   shows the full hierarchical name of the container;
>
> 2. mount -t containerfs none /container
>   results in a file tree under /container representing the
>   full container hierarchy
>
> 3. cd /proc/$$/container; ls
>   results in a listing of child containers
>
> What doesn't work:
> 1. The /proc/$$/container link always appears dead (red
> in bash on my fedora test system) because it points
> into a kern_mounted fs.

Just a quick comment. I am not at all comfortable exporting
internal kernel mounts without something explicit happening.
I played with that in one of my earlier patches and the corner cases
are just extremely weird and mess with the usual unix guarantees

about the namespace. Two specific examples are that .. fails to work properly,
as does sys_getcwd.

My gut feel is that we need something like union mounts so we can
glue these kinds of things but that will mount and unmount as
a unit. So we can preserve backwards compatibility with existing
filesystems. I haven't had a chance to look at what it would
take to implement this kind of hidden union mount though.

With the mount tree cloning code I believe we are quite close (at least
if we don't need the union property).

Eric

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by serue on Tue, 26 Dec 2006 14:27:15 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Following is a small patchset implementing what I was describing
> > about earlier, namely semantics for a hierarchical container
> > naming scheme.
> >
> > What works:
> >
> > 1. ls -l /proc/$$/container
> >   shows the full hierarchical name of the container;
> >
> > 2. mount -t containerfs none /container
> >   results in a file tree under /container representing the
> >   full container hierarchy
> >
> > 3. cd /proc/$$/container; ls
> >   results in a listing of child containers
> >
> > What doesn't work:
> > 1. The /proc/$$/container link always appears dead (red
> > in bash on my fedora test system) because it points
> > into a kern_mounted fs.
>
> Just a quick comment. I am not at all comfortable exporting

> internal kernel mounts without something explicit happening.
> I played with that in one of my earlier patches and the corner cases
> are just extremely weird and mess with the usual unix guarantees
> about the namespace. Two specific examples are that .. fails to work properly,
> as does sys_getcwd.
>
> My gut feel is that we need something like union mounts so we can
> glue these kinds of things but that will mount and unmount as
> a unit. So we can preserve backwards compatibility with existing
> filesystems. I haven't had a chance to look at what it would
> take to implement this kind of hidden union mount though.

Or we could go ahead and fully implement it in procfs. As you'd said
earlier, that really maps best into what we want. Containerfs was
just much simpler and quicker to implement for demonstrating the semantics.

> With the mount tree cloning code I believe we are quite close (at least
> if we don't need the union property).

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by ebiederm on Thu, 28 Dec 2006 21:37:46 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> Or we could go ahead and fully implement it in procfs. As you'd said
> earlier, that really maps best into what we want. Containerfs was
> just much simpler and quicker to implement for demonstrating the semantics.

That sounds like the safe bet. I'm keeping the other idea around
in case we find something that needs a little more power :)

Eric

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by ebiederm on Tue, 09 Jan 2007 01:43:12 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Or we could go ahead and fully implement it in procfs. As you'd said
> earlier, that really maps best into what we want. Containerfs was
> just much simpler and quicker to implement for demonstrating the semantics.

Well for what it is worth I just notices that nfs is currently and automounter
that transparently unmounts it's children when you unmount it. I don't think
that is quite enough to split /proc into two but it does have some potential
when it comes to new features.

Using itty bity purpose built file systems if there is an automounter for them
because much easier for user space.

Eric

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by serue on Wed, 10 Jan 2007 21:34:38 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Or we could go ahead and fully implement it in procfs. As you'd said
> > earlier, that really maps best into what we want. Containerfs was
> > just much simpler and quicker to implement for demonstrating the semantics.
>
> Well for what it is worth I just notices that nfs is currently and automounter
> that transparently unmounts it's children when you unmount it. I don't think
> that is quite enough to split /proc into two but it does have some potential
> when it comes to new features.
>
> Using itty bity purpose built file systems if there is an automounter for them
> because much easier for user space.

I'm not parsing the last sentence.

Are you suggesting that we may be able to stick with a custom fs,
using autofs to automount it if the symlink /proc/$$/container is
dereferenced while only a kernel mount of /containers exists?

I suppose a simpler solution is to not define /proc/$$/container,
but rather just let /container in the containerfs symlink to

the current process' container. That way you can't reference
/containers/container unless containerfs is already mounted under
/containers, and we avoid the problem completely.

-serge

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by serue on Wed, 10 Jan 2007 21:42:46 GMT
View Forum Message <> Reply to Message

Quoting Serge E. Hallyn (serue@us.ibm.com):
> Following is a small patchset implementing what I was describing
> about earlier, namely semantics for a hierarchical container
> naming scheme.
>
> What works:
>
> 1. ls -l /proc/$$/container
>   shows the full hierarchical name of the container;
>
> 2. mount -t containerfs none /container
>   results in a file tree under /container representing the
>   full container hierarchy
>
> 3. cd /proc/$$/container; ls
>   results in a listing of child containers
>
> What doesn't work:
> 1. The /proc/$$/container link always appears dead (red
> in bash on my fedora test system) because it points
> into a kern_mounted fs.
>
> 2. Features like
>
>   cd /proc/$$/container
>   mv container_3 my_child_container
>
>   to rename a container or
>
>   cd /proc/$$/container
>   rm container_3
>
>   to kill all processes a container are unimplemented.

>
> 3. Semantics for entering a namespace are not only
> unimplemented, but entirely unconsidered thus far.
> I suppose one cool way to enter a container would
> be
>
> ln -s /proc/$$/container/child_container /proc/$$/container
>
> but that
>
> a. Does not provide the ability to switch only
> some of the namespaces, as Herbert wants.
> b. May be unimplementable using proc support
> as is - not sure.

A conversation with Cedric today, we were thinking perhaps the
way to achieve this is to create files under each container
directory for each namespace type.

For instance,

```
d /containers/init_container/
f /containers/init_container/network
f /containers/init_container/uts
f /containers/init_container/user
f /containers/init_container/pid
d /containers/init_container/vserver1/
f /containers/init_container/vserver1/network
f /containers/init_container/vserver1/uts
f /containers/init_container/vserver1/user
f /containers/init_container/vserver1/pid
```

Note that if I want to enter just the network namespace of vserver1,
it's not quite right to say you're entering vserver1 at all, since
it consists of each namespace therein. Rather, you might

```
mkdir /containers/init_container/vserver2
ln -s /containers/init_container/vserver1/network \
 /containers/init_container/vserver2/
echo /containers/init_container/vserver2 > /proc/$$/container
exec /bin/sh
```

What happened? Well, we created a new container with no tasks.
We linked vserver2's network namespace in there, then requested
that we enter the container. Since no other namespaces had been
linked in, all other namespaces will be inherited from our own
namespace.

Thoughts?

-serge

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by ebiederm on Wed, 10 Jan 2007 22:00:58 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>
>> > Or we could go ahead and fully implement it in procfs. As you'd said
>> > earlier, that really maps best into what we want. Containerfs was
>> > just much simpler and quicker to implement for demonstrating the semantics.
>>
>> Well for what it is worth I just notices that nfs is currently and automounter
>> that transparently unmounts it's children when you unmount it. I don't think
>> that is quite enough to split /proc into two but it does have some potential
>> when it comes to new features.
>>
>> Using itty bity purpose built file systems if there is an automounter for them
>> because much easier for user space.
>
> I'm not parsing the last sentence.
>
> Are you suggesting that we may be able to stick with a custom fs,
> using autofs to automount it if the symlink /proc/$$/container is
> dereferenced while only a kernel mount of /containers exists?
>
> I suppose a simpler solution is to not define /proc/$$/container,
> but rather just let /container in the containerfs symlink to
> the current process' container. That way you can't reference
> /containers/container unless containerfs is already mounted under
> /containers, and we avoid the problem completely.

I am saying:
autofs is not special. Doing automounting the nfs way
you can add and remove mounts transparently to the user.

A very good use for this would be to mount/unmount things like
/proc/sys/fs/binfmt_misc/.

That technique may have an implication for the design of a container
filesystem.

The result is that if something is more simply implemented as a
separate filesystem, that is a possibility.

Eric

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

Subject: Re: [RFC] [PATCH 0/3] containers: introduction
Posted by serue on Thu, 11 Jan 2007 16:24:52 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >>
> >> > Or we could go ahead and fully implement it in procfs. As you'd said
> >> > earlier, that really maps best into what we want. Containerfs was
> >> > just much simpler and quicker to implement for demonstrating the semantics.
> >>
> >> Well for what it is worth I just notices that nfs is currently and automounter
> >> that transparently unmounts it's children when you unmount it. I don't think
> >> that is quite enough to split /proc into two but it does have some potential
> >> when it comes to new features.
> >>
> >> Using itty bity purpose built file systems if there is an automounter for them
> >> because much easier for user space.
> >
> > I'm not parsing the last sentence.
> >
> > Are you suggesting that we may be able to stick with a custom fs,
> > using autofs to automount it if the symlink /proc/$$/container is
> > dereferenced while only a kernel mount of /containers exists?
> >
> > I suppose a simpler solution is to not define /proc/$$/container,
> > but rather just let /container in the containerfs symlink to
> > the current process' container. That way you can't reference
> > /containers/container unless containerfs is already mounted under
> > /containers, and we avoid the problem completely.
>

> I am saying:
> autofs is not special. Doing automounting the nfs way
> you can add and remove mounts transparently to the user.

I see, thanks.

We don't want the kernel to know about magic pathname strings,
so as long as we are willing to mount containerfs under a known
location in procfs, this becomes trivial. Otherwise, I guess we
need to talk about convention.

I suppose just not having the kernel-mount, having the symlink,
and making /sys/containers the known location, isn't bad. Then
if /sys/containers isn't mounted and doesn't exist so we can't
automount it, /proc/$$/containers is just a bad link.

> A very good use for this would be to mount/unmount things like
> /proc/sys/fs/binfmt_misc/.
>
> That technique may have an implication for the design of a container
> filesystem.
>
> The result is that if something is more simply implemented as a
> separate filesystem, that is a possibility.

That's what's holding me back here - I'm still not sure whether
to proceed with a separate implementation, proceed with the
current implementation of Paul's containers, or wait for an
update from Paul responding to your feedback.

But both the standalone and paul-based approaches were easy to
implement so I guess it's not a big deal to just proceed with
my own and port to containers if/when appropriate.

-serge