
Subject: [RFC][patch 1/3] network container subsystem
Posted by [Daniel Lezcano](#) on Tue, 04 Sep 2007 17:00:23 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patch creates the network container subsystem.
It consists for the moment on a single file "network.ipv4".

The interface is pretty simple:

To add an IP address to the container:

```
echo add AB12FFFF > network.ipv4
```

To remove this IP address:

```
-----
```

```
echo del AB12FFFF > network.ipv4
```

To list the addresses:

```
-----
```

```
cat network.ipv4
```

The parameter is an IPV4 address in the hexa format. The parsing of a dotted-decimal parameter is totally painful. If this format hurts someone, I can change it to a dotted format at the risk of having something buggy.

This patch by itself does nothing more than adding/removing elements from a list.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
include/linux/container_subsys.h | 4
init/Kconfig                     | 8 +
kernel/Makefile                  | 1
kernel/container_network.c       | 285 +++++
4 files changed, 298 insertions(+)
```

Index: 2.6-mm/include/linux/container_subsys.h

=====

```
--- 2.6-mm.orig/include/linux/container_subsys.h
+++ 2.6-mm/include/linux/container_subsys.h
@@ -30,3 +30,7 @@
#endif
```

```
/* */
```

```

+
+ifdef CONFIG_CONTAINER_NETWORK
+SUBSYS(network)
+endif
Index: 2.6-mm/init/Kconfig
=====
--- 2.6-mm.orig/init/Kconfig
+++ 2.6-mm/init/Kconfig
@@ -326,6 +326,14 @@
    Provides a simple Resource Controller for monitoring the
    total CPU consumed by the tasks in a container

+config CONTAINER_NETWORK
+    bool "Network container subsystem"
+    depends on CONTAINERS && SECURITY_NETWORK
+    help
+    Provides a network controller to isolate network traffic
+
+    Say N if unsure
+
+config CPUSETS
+    bool "Cpuset support"
+    depends on SMP && CONTAINERS
Index: 2.6-mm/kernel/Makefile
=====
--- 2.6-mm.orig/kernel/Makefile
+++ 2.6-mm/kernel/Makefile
@@ -43,6 +43,7 @@
obj-$(CONFIG_CPUSETS) += cpuset.o
obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
obj-$(CONFIG_CONTAINER_NS) += ns_container.o
+obj-$(CONFIG_CONTAINER_NETWORK) += container_network.o
obj-$(CONFIG_IKCONFIG) += configs.o
obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
Index: 2.6-mm/kernel/container_network.c
=====
--- /dev/null
+++ 2.6-mm/kernel/container_network.c
@@ -0,0 +1,285 @@
+/*
+ * container_network.c - container network subsystem
+ *
+ * Copyright 2006, 2007 IBM Corp
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>

```

```

+#include <linux/fs.h>
+#include <linux/uaccess.h>
+#include <linux/ctype.h>
+#include <linux/list.h>
+#include <linux/spinlock.h>
+
+struct network {
+ struct container_subsys_state css;
+ struct list_head ipv4_list; /* store the IPV4 addresses */
+ rwlock_t ipv4_list_lock;
+};
+
+struct ipv4_list {
+ __be32 address;
+ struct list_head list;
+};
+
+static struct network top_network = {
+ .ipv4_list = LIST_HEAD_INIT(top_network.ipv4_list),
+ .ipv4_list_lock = __RW_LOCK_UNLOCKED(top_network.ipv4_list_lock),
+};
+
+struct container_subsys network_subsys;
+
+enum container_filetype {
+ FILE_IPV4,
+};
+
+static inline struct network *container_network(struct container *container)
+{
+ return container_of(
+ container_subsys_state(container, network_subsys_id),
+ struct network, css);
+}
+
+static struct container_subsys_state *network_create(struct container_subsys *ss,
+ struct container *container)
+{
+ struct network *network;
+
+ /* Don't let anybody do that */
+ if (!capable(CAP_NET_ADMIN))
+ return ERR_PTR(-EPERM);
+
+ /* The current container is the initial container */
+ if (!container->parent)
+ return &top_network.css;
+
+

```

```

+ network = kzalloc(sizeof(*network), GFP_KERNEL);
+ if (!network)
+ return ERR_PTR(-ENOMEM);
+
+ INIT_LIST_HEAD(&network->ipv4_list);
+ network->ipv4_list_lock = __RW_LOCK_UNLOCKED(network->ipv4_list_lock);
+
+ return &network->css;
+}
+
+static void network_destroy(struct container_subsys *ss,
+ struct container *container)
+{
+ struct network *network;
+ struct ipv4_list *entry, *next;
+ struct list_head *l;
+ rwlock_t *lock;
+
+ network = container_network(container);
+ l = &network->ipv4_list;
+ lock = &network->ipv4_list_lock;
+
+ /* flush the ipv4 list */
+ write_lock(lock);
+ list_for_each_entry_safe(entry, next, l, list) {
+ list_del(&entry->list);
+ kfree(entry);
+ }
+ write_unlock(lock);
+
+ kfree(network);
+}
+
+static int network_add_ipv4_address(struct container *container, __be32 address)
+{
+ struct ipv4_list *entry;
+ struct network *network;
+
+ entry = kmalloc(sizeof(*entry), GFP_KERNEL);
+ if (!entry)
+ return -ENOMEM;
+ entry->address = address;
+
+ network = container_network(container);
+ write_lock(&network->ipv4_list_lock);
+ list_add(&entry->list, &network->ipv4_list);
+ write_unlock(&network->ipv4_list_lock);
+
+

```

```

+ return 0;
+}
+
+static int network_del_ipv4_address(struct container *container, __be32 address)
+{
+ struct ipv4_list *entry;
+ struct network *network;
+ int ret = 0;
+
+ network = container_network(container);
+ write_lock(&network->ipv4_list_lock);
+ list_for_each_entry(entry, &network->ipv4_list, list) {
+ if (entry->address != address)
+ continue;
+
+ list_del(&entry->list);
+ goto out_free;
+ }
+ ret = -EINVAL;
+out:
+ write_unlock(&network->ipv4_list_lock);
+ return ret;
+
+out_free:
+ kfree(entry);
+ goto out;
+}
+
+static int network_parse_ipv4_address(struct container *container, char *buffer)
+{
+ int len = strlen(buffer);
+ char *addr;
+ __be32 address;
+
+ /* remove trailing left space */
+ while(isspace(*buffer))
+ buffer++;
+
+ /* remove trailing right space */
+ while(isspace(buffer[len - 1]))
+ buffer[(len--) - 1] = 0;
+
+ len = strlen(buffer);
+ addr = memchr(buffer, ' ', len);
+ if (!addr)
+ return -EINVAL;
+ *addr++ = 0;
+
+

```

```

+ /* remove trailing left space again */
+ while(isspace(*addr))
+   addr++;
+
+ /* Shall I check if the address is setup on the host ? */
+ sscanf(addr, "%X", &address);
+
+ if (!strcmp(buffer, "add"))
+   return network_add_ipv4_address(container, address);
+ else if (!strcmp(buffer, "del"))
+   return network_del_ipv4_address(container, address);
+
+ return -EINVAL;
+}
+
+static int network_fill_ipv4_address(struct container *container, char *buffer)
+{
+   struct network *network;
+   struct ipv4_list *entry;
+   char *s = buffer;
+   network = container_network(container);
+
+   read_lock(&network->ipv4_list_lock);
+   list_for_each_entry(entry, &network->ipv4_list, list)
+     s += sprintf(s, "%X\n", entry->address);
+   read_unlock(&network->ipv4_list_lock);
+
+   return strlen(buffer);
+}
+
+static ssize_t network_write(struct container *container,
+   struct cftype *cft,
+   struct file *file,
+   const char __user *userbuf,
+   size_t nbytes, loff_t *unused_ppos)
+{
+   enum container_filetype type = cft->private;
+   char *buffer;
+   int retval = 0;
+
+   if (!capable(CAP_NET_ADMIN))
+     return -EPERM;
+
+   if (nbytes >= PATH_MAX)
+     return -E2BIG;
+
+   buffer = kmalloc(nbytes + 1, GFP_KERNEL);
+   if (!buffer)

```

```

+ return -ENOMEM;
+
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+   retval = -EFAULT;
+   goto out_free;
+ }
+ buffer[nbytes] = 0;
+
+ container_lock();
+ switch(type) {
+
+ case FILE_IPV4:
+   retval = network_parse_ipv4_address(container, buffer);
+   break;
+
+ default:
+   retval = -EINVAL;
+   break;
+ };
+ container_unlock();
+
+out_free:
+ if (!retval)
+   retval = nbytes;
+
+ kfree(buffer);
+ return retval;
+}
+
+static ssize_t network_read(struct container *container,
+   struct cftype *cft,
+   struct file *file,
+   char __user *userbuf,
+   size_t nbytes, loff_t *ppos)
+{
+   enum container_filetype type = cft->private;
+   char *page;
+   int retval;
+
+   page = (char *)__get_free_page(GFP_TEMPORARY);
+   if (!page)
+     return -ENOMEM;
+
+   container_lock();
+   switch(type) {
+   case FILE_IPV4:
+     retval = network_fill_ipv4_address(container, page);
+     break;

```

```

+
+ default:
+ retval = -EINVAL;
+ };
+ container_unlock();
+
+ retval = simple_read_from_buffer(userbuf, nbytes, ppos, page, retval);
+
+ free_page((unsigned long)page);
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+ .name = "ipv4",
+ .read = network_read,
+ .write = network_write,
+ .private = FILE_IPV4,
+ },
+};
+
+static int network_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_files(cont, ss, files, ARRAY_SIZE(files));
+}
+
+struct container_subsys network_subsys = {
+ .name = "network",
+ .create = network_create,
+ .destroy = network_destroy,
+ .populate = network_populate,
+ .subsys_id = network_subsys_id,
+ .can_attach = NULL,
+ .attach = NULL,
+ .fork = NULL,
+ .exit = NULL,
+};
+
--

```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
