LXC (Linux Containers)

Introduction

Linux Containers, or LXC, is operating system-level method for running multiple separate isolated Linux installations (containers) on a single host. Rather than simulating the computer hardware as in true virtualization, LXC uses the cgroups and namespaces functionalities of the host's Linux kernel to provide strong isolation of the container. It is an intermediate solution between chroots and full virtualization, having a small impact on system resource usage similar to chroots, but providing better isolation. It provides a very convenient way to, among other things, maintain a clean build environment or test software against different [Linux] OS versions.

Setting up a Network Bridge

Before creating your first container, it is helpful to do some prep work. When the container is first created, only a minimal set of packages will be installed, so you will want to be able to use slackpkg or wget to round out your system. Typically, a bridge is created on the host, and the container connects to this bridge using a virtual ethernet interface.

While it is possible to set up the network manually, thankfully LXC contains a utility called *lxc-net* that can do it for you. As root, open up the file /etc/default/lxc-net, or create it if it doesn't exist, and add this line:

```
USE_LXC_BRIDGE="true"
```

Then, to bring up the network bridge, simply enter the command:

```
/usr/libexec/lxc/lxc-net start
```

Note that you may need to create the directory /var/lib/misc first for this to work. If it worked, there should not be any error messages or other output. You can check that it worked with ifconfig:

```
# ifconfig lxcbr0
lxcbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.0.3.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::7c17:8ff:fe09:cdcb  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:00:00  txqueuelen 1000  (Ethernet)
        RX packets 818240  bytes 45813772 (43.6 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1304901  bytes 3605721321 (3.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The name of the bridge, gateway, netmask, and many other parameters can be modified by setting the proper variables in /etc/default/lxc/lxc-net. For a complete list, simply open up /usr/libexec/lxc/lxc-net in your favorite editor or pager; there is a comment indicating which variables can be changed. Set them in /etc/default/lxc/lxc-net rather than modifying /usr/libexec/lxc/lxc-net directly.

To actually use this network within the container, there are a few steps that will need to be taken during the initial creation and setup of the container, which will be covered in the next section.

Creating a Container

To create a new container, the lxc-create command will be used. However, there are some initial configuration steps you may want to take first. By default, all containers will go in /var/lib/lxc. If you're like me, /var is mounted on / and doesn't have enough space for multiple full container installs of Slackware! You can change the container path in /etc/lxc/lxc.conf (e.g., by pointing it to /home):

```
lxc.lxcpath = /home/lxc_containers
```

The basic command to create a new container is as follows:

```
lxc-create -n container_name -t template_name -f /path/to/config
```

The config file, specified after the -f flag, can be fairly simple. Here is a sample, which sets only some network parameters:

```
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = lxcbr0
```

This sample config uses a virtual ethernet interface connecting to the bridge on the host called lxcbr0. For additional available configuration options, see the lxc.container.conf manpage. Note that once the container is created, a new config file will be created within the container's top-level directory containing the full range of config settings; this file is read upon container startup and can be edited to modify the container's behavior after creation.

The template file, specified after the -t flag, can be any of the ones present in /usr/share/lxc/templates. For example, for Slackware, you would use:

```
lxc-create -n container_name -t slackware -f /path/to/config
```

The Slackware template accepts a number of environment variables. The most important of these are release and arch. release defaults to current, and arch defaults to the architecture of the host machine. It is also possible to specify the slackpkg mirror as an environment variable. For example, here is how one could create a container for Slackware64-14.2 using a US mirror:

```
arch=x86_64 release=14.2 MIRROR=http://mirrors.us.kernel.org/slackware lxc-
create -n Slackware64-14.2 -t slackware -f /path/to/config
```

When lxc-create is executed, the container will be created at $lxcpath/container_name, and slackpkg will download and install the packages needed for a minimal install. The mirror will also be appended to /etc/slackpkg/mirrors.

Container Configuration

After creating the container, you are almost ready to go. First start the container:

```
lxc-start -n container_name
```

To log in, use:

```
lxc-console -n container_name
```

or

```
lxc-attach -n container_name
```

The first variant will give you a login console on the container, whereas the second will just give you a root shell without asking for a password. The first thing to do is to create a new user; use the standard Slackware tools such as *adduser*. It makes sense to create a user with the same username as a regular user on the host, because it makes ownership of shared directories easier to manage (more on that later).

Network

Even if you successfully set up a bridge network on the host and configured your container to use a virtual ethernet interface as described above, at this point you will probably still not have network access in the container. The reason is that you still need to set up your container's startup scripts to use the network. If you use lxc-net to set up the bridge network on the host, then you can just have your container get an IP address via DHCP. Open /etc/rc.d/rc.inet1.conf and set this:

```
USE_DHCP[0]="yes"
```

If desired, it should be possible to use a static IP address in the container as well.

The minimal Slackware installation set up by lxc-create does not have a lot of choice in editors, but *vi* (symlink to *elvis*) is available. If that's not your cup of tea, you can always edit the container's config files from the host using whichever editor you prefer. The container's entire filesystem is at $lxcpath/container_name/rootfs.

Once you have edited /etc/rc.d/rc.inet1.conf, get out of the container using Ctrl+a then q. Stop the container and then restart it as follows:

```
lxc-stop -n container_name
lxc-start -n container_name
```

Wait a minute or so to give the container plenty of time to “boot” and connect to the network. Then, you can use lxc-ls to check on the status of the container, as follows:

```
# lxc-ls --fancy
NAME                STATE   AUTOSTART GROUPS IPV4      IPV6
Slackware-14.2      STOPPED 0         -      -         -
Slackware64-14.2    RUNNING 0         -      10.0.3.65 -
Slackware64-current STOPPED 0         -      -         -
```

If the container successfully connected to the network, you should see that an IP address has been assigned. Log into the container again, and you should be able to ping outside, use slackpkg, etc. At this point, you can install any additional software you desire on the container. If you are using the container as a build environment, for example, you can do a full install:

```
slackpkg install slackware
```

or

```
slackpkg install slackware64
```

Custom Container Boot Process

In order to customize the boot process you can add or modify the init scripts listed in /usr/share/lxc/scripts/slackware. Any changes you make to the existing scripts will be copied to /etc/rc.d/ in each freshly created LXC container. This will allow you to easily customize the boot process if you create many LXC containers. A good example is if you add a custom rc.inet1.conf. It can become a bit tedious if you wish to create several LXC containers that all use DHCP network addressing. The solution is to create your own rc.inet1.conf, set **USE_DHCP[0]="yes"**, named it rc.inet1.conf.lxc, and save it in /usr/share/lxc/scripts/slackware. Your custom rc.inet1.conf will be copied to each new container.

Sharing Directories with the Host

Because an LXC container's filesystem is just a directory on the host, if you only need to transfer files from the host to the container, you can simply copy them over. However, often you will want to share the files both ways, or rather, be able to access *the same files* within the container without having to actually transfer anything, and this is a little more difficult due to the isolation of the container. LXC provides utilities to mount host directories on containers for this purpose. We will assume that the directory to be shared is at /home/username/foo on the host. To share a user directory with the host, first log into the container and create the top-level of the directory to be shared. Then log out and stop the container. Add the following lines to the container's config file ($lxcpath/container_name/config):

```
lxc.mount.entry = /home/username/foo home/container_user/foo none bind 0 0
```

Note that the second path intentionally lacks a leading slash. This is because it is a relative path – relative to the container's rootfs. Once you start the container again and log in, you should see that the directory has been mounted at /home/container_user/foo. In general, it is best if the username on the host is the same as the container's username, because it avoids any conflicts in file ownership between the host and container user.

Running GUI Applications

Without taking additional steps, it will not be possible to run GUI applications installed in the container. The simplest way to accomplish this is to run them using SSH with X forwarding to the host. However, there are still a few steps required to make this happen. In the container, in /etc/ssh/sshd_config, set the following:

```
X11Forwarding yes
X11UseLocalhost yes
```

This will allow X forwarding via SSH from the container. (Note: X11UseLocalhost yes is required; otherwise X will give an error about the DISPLAY not existing.) It is also necessary to bind-mount the host's /tmp/.X11-unix directory in the container. Add this line to the container's config file ($lxcpath/container_name/config):

```
lxc.mount.entry = /tmp/.X11-unix tmp/.X11-unix none bind,optional,create=dir
```

Next, restart the container. Upon restart, use `lxc-ls –fancy` to determine the container's IP address. You can then run GUI apps from the container as follows:

```
ssh -Y user@IP appname
```

This is probably the simplest way to run GUI apps in the container. However, more advanced usage is possible, including using LXC to “sandbox” applications in an unprivileged container and/or running GUI applications directly in the container without connecting over SSH. For more information, the following page is a good start:

LXC 1.0: GUI in containers [9/10]

See Also

Wikipedia

LXC homepage

Debian wiki

LQ thread

LXC 1.0: GUI in containers [9/10]

Sources

* Originally written by montagdude

howtos