Linux Containers

Linux Containers (LXC) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a single control host (LXC host). It does not provide a virtual machine, but rather provides a virtual environment that has its own CPU, memory, block I/O, network, etc. space and the resource control mechanism. This is provided by **namespaces** and **cgroups** features in Linux kernel on LXC host. It is similar to a chroot, but offers much more isolation.

Alternatives for using containers are **systemd-nspawn**, **docker** or `rkt (https://www.archlinux.org/packages/?name=rkt)`.

Related articles

AirVPN

ABS

Cgroups

Docker

LXD

OpenVPN

OpenVPN (client) in Linux containers

OpenVPN (server) in Linux containers

PeerGuardian Linux

systemd-nspawn

Contents

Privileged containers or unprivileged containers

LXCs can be setup to run in either *privileged* or *unprivileged* configurations.

In general, running an *unprivileged* container is **considered safer (https://www.stgraber.org/2014/01/17/lxc-1-0-unprivileged-containers)** than running a *privileged* container since *unprivileged* containers have an increased degree of isolation by virtue of their design. Key to this is the mapping of the root UID in the container to a non-root UID on the host which makes it more difficult for a hack within the container to lead to consequences on host system. In other words, if an attacker manages to escape the container, he or she should find themselves with no rights on the host.

The Arch packages currently provide out-of-the-box support for *privileged* containers. *Unprivileged* containers are only available for the system administrator without additional kernel configuration. This is due to the current Arch **`linux (https://www.archlinux.org/packages/?name=linux)`** kernel shipping with user namespaces disabled for normal users. This article contains information for users to run either type of container, but additional setup is required to use *unprivileged* containers.

An example to illustrate unprivileged containers

To illustrate the power of UID mapping, consider the output below from a running, *unprivileged* container. Therein, we see the containerized processes owned by the containerized root user in the output of `ps` :

```
[root@unprivileged_container /]# ps -ef | head -n 5
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 17:49 ?        00:00:00 /sbin/init
root        14     1  0 17:49 ?        00:00:00 /usr/lib/systemd/systemd-journald
dbus        25     1  0 17:49 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
systemd+    26     1  0 17:49 ?        00:00:00 /usr/lib/systemd/systemd-networkd
```

On the host however, those containerized root processes are running as the mapped user (ID>100000) on the host, not as the root user on the host:

```
[root@host /]# lxc-info -Ssip --name sandbox
State:          RUNNING
PID:            26204
CPU use:        10.51 seconds
BlkIO use:      244.00 KiB
Memory use:     13.09 MiB
KMem use:       7.21 MiB
```

```
[root@host /]# ps -ef | grep 26204 | head -n 5
UID        PID  PPID  C STIME TTY          TIME CMD
100000   26204 26200  0 12:49 ?        00:00:00 /sbin/init
100000   26256 26204  0 12:49 ?        00:00:00 /usr/lib/systemd/systemd-journald
100081   26282 26204  0 12:49 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
100000   26284 26204  0 12:49 ?        00:00:00 /usr/lib/systemd/systemd-logind
```

Setup

Required software

Installing `lxc (https://www.archlinux.org/packages/?name=lxc)` and `arch-install-scripts (https://www.archlinux.org/packages/?name=arch-install-scripts)` will allow the host system to run privileged lxcs.

Enable support to run unprivileged containers (optional)

Users wishing to run *unprivileged* containers need to complete several additional setup steps.

Firstly, a kernel is required that has support for **User Namespaces**. However, due to more general security concerns, the default Arch kernel does ship with User Namespaces enabled only for the *root* user. You have multiple options to use a kernel with `CONFIG_USER_NS` and thereby create *unprivileged* containers:

- Use **`linux (https://www.archlinux.org/packages/?name=linux)`** (v4.14.5 or later) or **`linux-hardened (https://www.archlinux.org/packages/?name=linux-hardened)`** and start your unprivileged containers only as *root*
- Use **`linux (https://www.archlinux.org/packages/?name=linux)`** (v4.14.5 or later) or **`linux-hardened (https://www.archlinux.org/packages/?name=linux-hardened)`** and enable the *sysctl* setting `kernel.unprivileged_userns_clone` to allow normal users to run unprivileged containers. This can be done for the current session with `sysctl kernel.unprivileged_userns_clone=1` and can be made permanent with **`sysctl.d(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/sysctl.d.5)`**
- Compile and install your own custom kernel with `CONFIG_USER_NS` enabled. See **Kernels/Arch Build System** for more information on compiling a custom kernel.

Secondly, modify `/etc/lxc/default.conf` to contain the following lines:

```
lxc.idmap = u 0 100000 65536
lxc.idmap = g 0 100000 65536
```

Finally, create both `/etc/subuid` and `/etc/subgid` to contain the mapping to the containerized uid/gid pairs for each user who shall be able to run the containers. The example below is simply for the root user (and systemd system unit):

```
/etc/subuid
---
root:100000:65536
```

```
/etc/subgid
---
root:100000:65536
```

Host network configuration

LXCs support different virtual network types and devices (see **lxc.container.conf(5) (https://linuxcontainers.org/lxc/manpages//man5/lxc.container.conf.5.html)**). A bridge device on the host is required for most types of virtual networking.

LXC comes with its own NAT Bridge (lxcbr0).

Note: A NAT bridge is a standalone bridge with a private network that is not bridged to the host eth0 or a physical network. It exists as a private subnet in the host.

Tip: This is quite useful when WIFI is the only option. There have been various attempts of creating Bridges on WIFI without much success.

To use LXC's NAT Bridge you need to create its configuration file:

```
/etc/default/lxc-net
```

```
# Leave USE_LXC_BRIDGE as "true" if you want to use lxcbr0 for your
# containers.  Set to "false" if you'll use virbr0 or another existing
# bridge, or mavlan to your host's NIC.
USE_LXC_BRIDGE="true"

# If you change the LXC_BRIDGE to something other than lxcbr0, then
# you will also need to update your /etc/lxc/default.conf as well as the
# configuration (/var/lib/lxc/<container>/config) for any containers
# already created using the default config to reflect the new bridge
# name.
# If you have the dnsmasq daemon installed, you'll also have to update
# /etc/dnsmasq.d/lxc and restart the system wide dnsmasq daemon.
LXC_BRIDGE="lxcbr0"
LXC_ADDR="10.0.3.1"
LXC_NETMASK="255.255.255.0"
LXC_NETWORK="10.0.3.0/24"
LXC_DHCP_RANGE="10.0.3.2,10.0.3.254"
LXC_DHCP_MAX="253"
# Uncomment the next line if you'd like to use a conf-file for the lxcbr0
# dnsmasq.  For instance, you can use 'dhcp-host=mail1,10.0.3.100' to have
# container 'mail1' always get ip address 10.0.3.100.
#LXC_DHCP_CONFILE=/etc/lxc/dnsmasq.conf

# Uncomment the next line if you want lxcbr0's dnsmasq to resolve the .lxc
# domain.  You can then add "server=/lxc/10.0.3.1' (or your actual $LXC_ADDR)
# to your system dnsmasq configuration file (normally /etc/dnsmasq.conf,
# or /etc/NetworkManager/dnsmasq.d/lxc.conf on systems that use NetworkManager).
# Once these changes are made, restart the lxc-net and network-manager services.
# 'container1.lxc' will then resolve on your host.
#LXC_DOMAIN="lxc"
```

Tip: Make sure the bridges ip-range does not interfere with your local network.

Then we need to modify the LXC container template so our containers use our bridge:

```
/etc/lxc/default.conf
```

```
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
```

You also need to install **Dnsmasq** which is a dependency for lxcbr0.

```
pacman -S dnsmasq
```

Then we can start the bridge:

```
systemctl start lxc-net
```

If you want the bridge to start at boot-time

```
systemctl enable lxc-net
```

For further information including Host bridges users are referred to the **Network bridge** article.

Container creation

For *privileged* containers, simply select a template from `/usr/share/lxc/templates` that matches the target distro to containerize. Users wishing to containerize non-Arch distros will need additional packages on the host depending on the target distro:

- Debian-based: **debootstrap (https://www.archlinux.org/packages/?name=debootstrap)**
- Fedora-based: **yum (https://aur.archlinux.org/packages/yum/)** AUR

Run `lxc-create` to create the container, which installs the root filesystem of the LXC to `/var/lib/lxc/CONTAINER_NAME/rootfs` by default. Example creating an Arch Linux LXC named "playtime":

```
# lxc-create -n playtime -t /usr/share/lxc/templates/lxc-archlinux
```

Users wishing to run *unprivileged* containers should use the -t download directive and select from the images that are displayed. For example:

```
# lxc-create -n playtime -t download
```

Alternatively, create a *privileged* container, and see: **#Converting a privileged container to an unprivileged container**.

Tip: Users may optionally install **haveged (https://www.archlinux.org/packages/?name=haveged)** and **start** `haveged.service` to avoid a perceived hang during the setup process while waiting for system entropy to be seeded. Without it, the generation of private/GPG keys can add a lengthy wait to the process.

Tip: Users of **Btrfs** can append `-B btrfs` to create a Btrfs subvolume for storing containerized rootfs. This comes in handy if cloning containers with the help of `lxc-clone` command. **ZFS** users may use `-B zfs`, correspondingly.

Container configuration

The examples below can be used with *privileged* and *unprivileged* containers alike. Note that for unprivileged containers, additional lines will be present by default which are not shown in the examples, including the `lxc.idmap = u 0 100000 65536` and the `lxc.idmap = g 0 100000 65536` values optionally defined in the **#Enable support to run unprivileged containers (optional)** section.

Basic config with networking

Note: With the release of lxc-1:2.1.0-1, many of the configuration options have changed. Existing containers need to be updated; users are directed to the table of these changes in the **v2.1 release notes (https://discuss.linuxcontainers.org/t/lxc-2-1-has-been-released/487)**.

System resources to be virtualized/isolated when a process is using the container are defined in `/var/lib/lxc/CONTAINER_NAME/config`. By default, the creation process will make a minimum setup without networking support. Below is an example config with networking:

```
/var/lib/lxc/playtime/config
```

```
# Template used to create this container: /usr/share/lxc/templates/lxc-archlinux
# Parameters passed to the template:
# For additional config options, please look at lxc.container.conf(5)

## default values
lxc.rootfs.path = /var/lib/lxc/playtime/rootfs
lxc.uts.name = playtime
lxc.arch = x86_64
lxc.include = /usr/share/lxc/config/archlinux.common.conf

## network
lxc.net.0.type = veth
lxc.net.0.link = br0
lxc.net.0.flags = up
lxc.net.0.name = eth0
lxc.net.0.hwaddr = ee:ec:fa:e9:56:7d
# uncomment the next two lines if static IP addresses are needed
# leaving these commented will imply DHCP networking
#
#lxc.net.0.ipv4.address = 192.168.0.3/24
#lxc.net.0.ipv4.gateway = 192.168.0.1
```

Note: The lxc.network.hwaddr entry is optional and if skipped, a random MAC address will be created automatically. It can be advantageous to define a MAC address for the container to allow the DHCP server to always assign the same IP to the container's NIC (beyond the scope of this article but worth mentioning).

Mounts within the container

For *privileged* containers, one can select directories on the host to bind mount to the container. This can be advantageous for example if the same architecture is being containerize and one wants to share pacman packages between the host and container. Another example could be shared directories. The syntax is simple:

```
lxc.mount.entry = /var/cache/pacman/pkg var/cache/pacman/pkg none bind 0 0
```

Note: This will not work without filesystem permission modifications on the host if using *unprivileged* containers.

Xorg program considerations (optional)

In order to run programs on the host's display, some bind mounts need to be defined so that the containerized programs can access the host's resources. Add the following section to `/var/lib/lxc/playtime/config`:

```
## for xorg
lxc.mount.entry = /dev/dri dev/dri none bind,optional,create=dir
lxc.mount.entry = /dev/snd dev/snd none bind,optional,create=dir
lxc.mount.entry = /tmp/.X11-unix tmp/.X11-unix none bind,optional,create=dir,ro
lxc.mount.entry = /dev/video0 dev/video0 none bind,optional,create=file
```

If you still get a permission denied error in your LXC guest, then you may need to call `xhost +` in your host to allow the guest to connect to the host's display server. Take note of the security concerns of opening up your display server by doing this. In addition you might need to add the following line

```
lxc.mount.entry = tmpfs tmp tmpfs defaults
```

before the bind mount lines.

Note: This will not work if using *unprivileged* containers.

OpenVPN considerations

Users wishing to run **OpenVPN** within the container are direct to either **OpenVPN (client) in Linux containers** and/or **OpenVPN (server) in Linux containers**.

Managing containers

Basic usage

To list all installed LXC containers:

```
# lxc-ls -f
```

Systemd can be used to **start** and to **stop** LXCs via `lxc@CONTAINER_NAME.service`. **Enable** `lxc@CONTAINER_NAME.service` to have it start when the host system boots.

Users can also start/stop LXCs without systemd. Start a container:

```
# lxc-start -n CONTAINER_NAME
```

Stop a container:

```
# lxc-stop -n CONTAINER_NAME
```

To login into a container:

```
# lxc-console -n CONTAINER_NAME
```

If when login you get pts/0 and lxc/tty1 use:

```
# lxc-console -n CONTAINER_NAME -t 0
```

Once logged, treat the container like any other linux system, set the root password, create users, install packages, etc.

To attach to a container:

```
# lxc-attach -n CONTAINER_NAME --clear-env
```

It works nearly the same as lxc-console, but you are automatically accessing root prompt inside the container, bypassing login. Without the `--clear-env` flag, the host will pass its own environment variables into the container (including `$PATH`, so some commands will not work when the containers are based on another distribution).

Advanced usage

LXC clones

Users with a need to run multiple containers can simplify administrative overhead (user management, system updates, etc.) by using snapshots. The strategy is to setup and keep up-to-date a single base container, then, as needed, clone (snapshot) it. The power in this strategy is that the disk space and system overhead are truly minimized since the snapshots use an overlayfs mount to only write out to disk, only the differences in data. The base system is read-only but changes to it in the snapshots are allowed via the overlayfs.

For example, setup a container as outlined above. We will call it "base" for the purposes of this guide. Now create 2 snapshots of "base" which we will call "snap1" and "snap2" with these commands:

```
# lxc-copy -n base -N snap1 -B overlayfs -s
# lxc-copy -n base -N snap2 -B overlayfs -s
```

Note: If a static IP was defined for the "base" lxc, that will need to manually changed in the config for "snap1" and for "snap2" before starting them. If the process is to be automated, a script using sed can do this automatically although this is beyond the scope of this wiki section.

The snapshots can be started/stopped like any other container. Users can optionally destroy the snapshots and all new data therein with the following command. Note that the underlying "base" lxc is untouched:

```
# lxc-destroy -n snap1 -f
```

Systemd units and wrapper scripts to manage snapshots for **pi-hole** and **OpenVPN** are available to automate the process in `lxc-snapshots (https://aur.archlinux.org/packages/lxc-snapshots/)`AUR.

Converting a privileged container to an unprivileged container

Once the system has been configured to use unprivileged containers (see, **#Enable support to run unprivileged containers (optional)**), `nsexec-bzr (https://aur.archlinux.org/packages/nsexec-bzr/)`AUR contains a utility called `uidmapshift` which is able to convert an existing *privileged* container to an *unprivileged* container to avoid a total rebuild of the image.

Warning:

- It is recommended to backup the existing image before using this utility!
- This utility will not shift UIDs and GIDs in ACL, you will need to shift them on your own.

Invoke the utility to convert over like so:

```
# uidmapshift -b /var/lib/lxc/foo 0 100000 65536
```

Additional options are available simply by calling `uidmapshift` without any arguments.

Running Xorg programs

Either attach to or SSH into the target container and prefix the call to the program with the DISPLAY ID of the host's X session. For most simple setups, the display is always 0.

An example of running Firefox from the container in the host's display:

```
$ DISPLAY=:0 firefox
```

Alternatively, to avoid directly attaching to or connecting to the container, the following can be used on the host to automate the process:

```
# lxc-attach -n playtime --clear-env -- sudo -u YOURUSER env DISPLAY=:0 firefox
```

Troubleshooting

Root login fails

If you get the following error when you try to login using lxc-console:

```
login: root
Login incorrect
```

And the container's `journalctl` shows:

```
pam_securetty(login:auth): access denied: tty 'pts/0' is not secure !
```

Add `pts/0` to the list of terminal names in `/etc/securetty` on the **container** filesystem, see **[1] (http://unix.stackexchange.com/questions/41840/effect-of-entries-in-etc-securetty/41939#41939)**. You can also opt to delete `/etc/securetty` on the **container** to allow always root to login, see **[2] (https://github.com/systemd/systemd/issues/852)**.

Alternatively, create a new user in lxc-attach and use it for logging in to the system, then switch to root.

```
# lxc-attach -n playtime
[root@playtime]# useradd -m -Gwheel newuser
[root@playtime]# passwd newuser
[root@playtime]# passwd root
[root@playtime]# exit
# lxc-console -n playtime
[newuser@playtime]$ su
```

No network-connection with veth in container config

If you cannot access your LAN or WAN with a networking interface configured as **veth** and setup through `/etc/lxc/`*`containername`*`/config`. If the virtual interface gets the ip assigned and should be connected to the network correctly.

```
ip addr show veth0
inet 192.168.1.111/24
```

You may disable all the relevant static ip formulas and try setting the ip through the booted container-os like you would normaly do.

Example *`container`*`/config`

```
...
lxc.net.0.type = veth
lxc.net.0.name = veth0
lxc.net.0.flags = up
lxc.net.0.link =  bridge
...
```

And then assign your IP through your preferred method **inside** the container, see also Network configuration#Configure the IP address[broken link: invalid section].

Error: unknown command

The error may happen when you type a basic command (*ls*, *cat*, etc.) on an attached container that have different Linux distribution from the host system (e.g. Debian container in Arch Linux host system). When you attach, use the argument `--clear-env`:

```
# lxc-attach -n container_name --clear-env
```

See also

- LXC 1.0 Blog Post Series (https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series/)
- LXC@developerWorks (http://www.ibm.com/developerworks/linux/library/l-lxc-containers/)
- Docker Installation on ArchLinux (http://docs.docker.io/en/latest/installation/archlinux/)
- LXC articles on l3net (http://l3net.wordpress.com/tag/lxc/)

Retrieved from "https://wiki.archlinux.org/index.php?title=Linux_Containers&oldid=510375"

- This page was last edited on 10 February 2018, at 12:51.
- Content is available under GNU Free Documentation License 1.3 or later unless otherwise noted.