Linux Containers

3 languages

日本語
Português
中文（简体）

Related articles

Cgroups
Docker
Linux Containers/Using VPNs
LXD
systemd-nspawn

Linux Containers (https://linuxcontainers.org/) (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 the namespaces and cgroups features in the Linux kernel on the LXC host. It is similar to a chroot, but offers much more isolation.

LXD can be used as manager for LXC. This page deals with using LXC directly.

Alternatives for using containers are systemd-nspawn or Docker.

1 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 within the container to a non-root UID on the host, which makes it more difficult for a hack inside the container to lead to consequences on the host system. In other words, if an attacker manages to escape the container, they should find themselves with limited or no rights on the host.

The Arch `linux` (https://archlinux.org/packages/?name=linux), `linux-lts` (https://archlinux.org/packages/?name=linux-lts) and `linux-zen` (https://archlinux.org/packages/?name=linux-zen) kernel packages currently provide out-of-the-box support for *unprivileged* containers. Similarly, with the `linux-hardened` (https://archlinux.org/packages/?name=linux-hardened) package, *unprivileged* containers are only available for the system administrator; with additional kernel configuration changes required, as user namespaces are disabled by default for normal users there.

This article contains information for users to run either type of container, but additional steps may be required in order to use *unprivileged* containers.

1.1 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 --a
ddress=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 actually shown to be running as the mapped user (ID>99999), rather than the host's actual root user:

```
[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 --a
ddress=systemd: --nofork --nopidfile --systemd-activation
100000   26284 26204  0 12:49 ?        00:00:00 /usr/lib/systemd/systemd-logind
```

2 Setup

2.1 Required software

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

2.1.1 Enable support to run unprivileged containers (optional)

Modify `/etc/lxc/default.conf` to contain the following lines:

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

In other words, map a range of 65536 consecutive uids, starting from container-side uid 0, which shall be uid 100000 from the host's point of view, up to and including container-side uid 65535, which the host will know as uid 165535. Apply that same mapping to gids.

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
```

In addition, running unprivileged containers as an unprivileged user only works if you delegate a **cgroup** in advance (the cgroup2 delegation model enforces this restriction, not liblxc). Use the following *systemd* command to delegate the cgroup (per **LXC - Getting started: Creating unprivileged containers as a user (https://linuxcontainers.org/lxc/getting-started/#creating-unprivileged-containers-as-a-user)**):

```
$ systemd-run --unit=myshell --user --scope -p "Delegate=yes" lxc-start container_name
```

This works similarly for other lxc commands.

Alternatively, delegate *unprivileged* **cgroups** by creating a *systemd* unit (per **Rootless Containers: Enabling CPU, CPUSET, and I/O delegation (https://rootlesscontaine.rs/getting-started/common/cgroup2/#enabling-cpu-cpuset-and-io-delegation)**):

/etc/systemd/system/user@.service.d/delegate.conf

```
[Service]
Delegate=cpu cpuset io memory pids
```

2.1.1.1 Unprivileged containers on linux-hardened and custom kernels

Users wishing to run *unprivileged* containers on **linux-hardened (https://archlinux.org/packages/?name=linux-hardened)** or their custom kernel need to complete several additional setup steps.

Firstly, a kernel is required that has support for user namespaces (a kernel with `CONFIG_USER_NS`). All Arch Linux kernels have support for `CONFIG_USER_NS`. However, due to more general security concerns, the **linux-hardened (https://archlinux.org/packages/?name=linux-hardened)** kernel does ship with user namespaces enabled only for the *root* user. There are two options to create *unprivileged* containers there:

- Start the unprivileged containers only as *root*. Also give the **sysctl** setting `user.max_user_namespaces` a positive value to suit your environment if its current value is `0` (this fixes `Failed to clone process in new user namespace` errors seen in `lxc info --show-log container_name`).
- Under **linux-hardened (https://archlinux.org/packages/?name=linux-hardened)** & `lxd 5.0.0` you may need to set `/etc/subuid & /etc/subgid` to use a range of `root:1000000:65536`. You may also need to start your **first**

container as privileged. This fixes error `newuidmap failed to write mapping "newuidmap: uid range [0-1000000000) -> [1000000-1001000000) not allowed"`

- Enable the *sysctl* setting `kernel.unprivileged_userns_clone` to allow normal users to run unprivileged containers. This can be done for the current session by running `sysctl kernel.unprivileged_userns_clone=1` as root and can be made permanent with **sysctl.d(5) (https://man.archlinux.org/man/sysctl.d.5)**.

2.2 Host network configuration

LXCs support different virtual network types and devices (see **lxc.container.conf(5) (https://man.archlinux.org/man/lxc.container.conf.5)**). A bridge device on the host is required for most types of virtual networking, which is illustrated in this section.

There are several main setups to consider:

1. A host bridge
2. A NAT bridge

The host bridge requires the host's network manager to manage a shared bridge interface. The host and any lxc will be assigned an IP address in the same network (for example 192.168.1.x). This might be more simplistic in cases where the goal is to containerize some network-exposed service like a webserver, or VPN server. The user can think of the lxc as just another PC on the physical LAN, and forward the needed ports in the router accordingly. The added simplicity can also be thought of as an added threat vector, again, if WAN traffic is being forwarded to the lxc, having it running on a separate range presents a smaller threat surface.

The NAT bridge does not require the host's network manager to manage the bridge. **lxc (https://archlinux.org/packages/?name=lxc)** ships with `lxc-net` which creates a NAT bridge called `lxcbr0`. The NAT bridge is a standalone bridge with a private network that is not bridged to the host's ethernet device or to a physical network. It exists as a private subnet in the host.

2.2.1 Using a host bridge

See **Network bridge**.

2.2.2 Using a NAT bridge

Install **dnsmasq (https://archlinux.org/packages/?name=dnsmasq)**, which is a dependency for `lxc-net`. Before you start the bridge, first create a configuration file for it:

/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 bridge's IP range does not interfere with the 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
```

Optionally create a configuration file to manually define the IP address of any containers:

/etc/lxc/dnsmasq.conf

```
dhcp-host=playtime,10.0.3.100
```

Now **start** and **enable** `lxc-net.service` to create the bridge interface.

2.2.2.1 Firewall considerations

Depending on which firewall the host machine is running, it might be necessary to allow inbound packets from `lxcbr0` to the host, and outbound packets from `lxcbr0` to traverse through the host to other networks. To test this, try to bring up a container configured to use DHCP for its IP assignment and see if `lxc-net` is able to assign an IP address to the container (check with . If no IP is assigned, the host's policies will need to be adjusted.

Users of **ufw (https://archlinux.org/packages/?name=ufw)** can simply run the **following two lines (https://discuss.linuxcontainers.org/t/lxd-bridge-doesnt-work-with-ipv4-and-ufw-with-nftables/10034/17)** to enable this:

```
# ufw allow in on lxcbr0
# ufw route allow in on lxcbr0
```

Alternatively, users of **nftables (https://archlinux.org/packages/?name=nftables)** can modify `/etc/nftables.conf` (and reload it with `nft -f /etc/nftables.conf`; check if the config syntax is correct with `nft -cf /etc/nftables.conf`) to allow the container to have internet access (replace `"eth0"` with the device on your system that has internet access; list existing devices with `ip link`):

/etc/nftables.conf

```
table inet filter {
  chain input {
    ...
    iifname "lxcbr0" accept comment "Allow lxc containers"

    pkttype host limit rate 5/second counter reject with icmpx type admin-prohibi
ted
    counter
  }
  chain forward {
    ...
    iifname "lxcbr0" oifname "eth0" accept comment "Allow forwarding from lxcbr0
to eth0"
    iifname "eth0" oifname "lxcbr0" accept comment "Allow forwarding from eth0 to
lxcbr0"
  }
}
```

Additionally, since the container is running on the 10.0.3.x subnet, external access to services such as ssh, httpd, etc. will need to be actively forwarded to the lxc. In principle, the firewall on the host needs to forward incoming traffic on the expected port on the container.

2.2.2.1.1 Example iptables rule

The goal of this rule is to allow ssh traffic to the lxc:

```
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 2221 -j DNAT --to-destinat
ion 10.0.3.100:22
```

This rule forwards tcp traffic originating on port 2221 to the IP address of the lxc on port 22.

Note: Make sure to allow traffic on 2221/tcp on the host and to allow 22/tcp traffic on the lxc.

To ssh into the container from another PC on the LAN, one needs to ssh on port 2221 to the host. The host will then forward that traffic to the container.

```
$ ssh -p 2221 host.lan
```

2.2.2.1.2 Example ufw rule

If using **ufw (https://archlinux.org/packages/?name=ufw)**, append the following at the bottom of `/etc/ufw/before.rules` to make this persistent:

```
/etc/ufw/before.rules
```

```
*nat
:PREROUTING ACCEPT [0:0]
-A PREROUTING -i eth0 -p tcp --dport 2221 -j DNAT --to-destination 10.0.3.100:22
COMMIT
```

2.2.2.2 Running containers as non-root user

To create and start containers as a non-root user, extra configuration must be applied.

Create the usernet file under `/etc/lxc/lxc-usernet`. According to **lxc-usernet(5) (https://man.archlinux.org/man/lxc-usernet.5)**, the entry per line is:

```
user type bridge number
```

Configure the file with the user needing to create containers. The bridge will be the same as defined in `/etc/default/lxc-net`.

A copy of the `/etc/lxc/default.conf` is needed in the non-root user's home directory, e.g. `~/.config/lxc/default.conf` (create the directory if needed).

Running containers as a non-root user requires `+x` permissions on `~/.local/share/`. Make that change with **chmod** before starting a container.

2.3 Container creation

Containers are built using `lxc-create`. With the release of lxc-3.0.0-1, upstream has deprecated locally stored templates.

To build an Arch container, invoke like this:

```
# lxc-create -n playtime -t download -- --dist archlinux --release current --arch
amd64
```

For other distributions, invoke like this and select options from the supported distributions displayed in the list:

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

Tip:

- Users may optionally install **haveged (https://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.
- 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.

Note: Users wanting the legacy templates can find them in **lxc-templates (https://aur.archlinux.org/packages/lxc-templates/)**[AUR] or alternatively, users can build their own templates with **distrobuilder (https://archlinux.org/packages/?name=distrobuilder)**.

2.4 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.

2.4.1 Basic configuration 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 configuration with networking supplied by `lxc-net.service`:

```
/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)

# Distribution configuration
lxc.include = /usr/share/lxc/config/common.conf
lxc.arch = x86_64

# Container specific configuration
lxc.rootfs.path = dir:/var/lib/lxc/playtime/rootfs
lxc.uts.name = playtime

# Network configuration
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
```

```
lxc.net.0.flags = up
lxc.net.0.hwaddr = ee:ec:fa:e9:56:7d
```

2.4.2 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 containerized and one wants to share pacman packages between the host and container. Another example could be shared directories. The syntax is:

```
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.

2.4.3 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 still experiencing a *permission denied* error in the LXC guest, call `xhost +` in the host to allow the guest to connect to the host's display server. Take note of the security concerns of opening up the display server by doing this. In addition, add the following line **before** the above bind mount lines.

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

2.4.4 VPN considerations

To run a containerized **OpenVPN** or **WireGuard**, see **Linux Containers/Using VPNs**.

3 Managing containers

3.1 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
```

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
```

That works nearly the same as lxc-console, but it causes starts with a 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).

3.2 Advanced usage

3.2.1 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.

> **Note:** overlayfs for unprivileged containers is not supported in the current mainline Arch Linux kernel due to security considerations.

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 configuration 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-service-snapshots (https://github.com/graysky2/lxc-service-snapshots)**.

3.3 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**, users will need to shift them manually!

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.

4 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
```

5 Troubleshooting

5.1 Root login fails

If presented with following error upon trying to login using lxc-console:

```
login: root
Login incorrect
```

And the container's **journal** shows:

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

Delete `/etc/securetty` **[1] (https://unix.stackexchange.com/questions/41840/effect-of-entries-in-etc-securetty/41939#41939)** and `/usr/share/factory/etc/securetty` on the **container** file system. Optionally add them to **NoExtract** in `/etc/pacman.conf` to prevent them from getting reinstalled. See **FS#45903 (https://bugs.archlinux.org/task/45903)** for details.

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
```

5.2 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 the IP through a preferred method **inside** the container, see also **Network configuration#Network management**.

5.3 Error: unknown command

The error may happen when a basic command (*ls*, *cat*, etc.) on an attached container is typed hen a different Linux distribution is containerized relative to the host system (e.g. Debian container in Arch Linux host system). Upon attaching, use the argument `--clear-env` :

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

5.4 Error: Failed at step KEYRING spawning...

Services in an unprivileged container may fail with the following message

```
some.service: Failed to change ownership of session keyring: Permission denied
some.service: Failed to set up kernel keyring: Permission denied
some.service: Failed at step KEYRING spawning ....: Permission denied
```

Create a file `/etc/lxc/unpriv.seccomp` containing

```
/etc/lxc/unpriv.seccomp
```

```
2
blacklist
[all]
keyctl errno 38
```

Then add the following line to the container configuration **after** lxc.idmap

```
lxc.seccomp.profile = /etc/lxc/unpriv.seccomp
```

6 Known issues

6.1 lxc-execute fails due to missing lxc.init.static

`lxc-execute` fails with the error message `Unable to open lxc.init.static`. See **FS#63814 (https://bugs.archlinux.org/task/63814)** for details.

Starting containers using `lxc-start` works fine.

7 See also

- **Official Website (https://linuxcontainers.org/lxc/introduction/)**

- Getting Started Guide (https://linuxcontainers.org/lxc/getting-started/)
- Documentation (https://linuxcontainers.org/lxc/documentation/)
- Forum (https://discuss.linuxcontainers.org/)
- Official Github Repository (https://github.com/lxc/lxc)
