Ubuntu 14.04+lxc

**LXC**

Containers are a lightweight virtualization technology. They are more akin to an enhanced chroot than to full virtualization like Qemu or VMware, both because they do not emulate hardware and because containers share the same operating system as the host. Therefore containers are better compared to Solaris zones or BSD jails. Linux-vserver and OpenVZ are two pre-existing, independently developed implementations of containers-like functionality for Linux. In fact, containers came about as a result of the work to upstream the vserver and OpenVZ functionality.

There are two user-space implementations of containers, each exploiting the same kernel features. Libvirt allows the use of containers through the LXC driver by connecting to 'lxc:///'. This can be very convenient as it supports the same usage as its other drivers. The other implementation, called simply 'LXC', is not compatible with libvirt, but is more flexible with more userspace tools. It is possible to switch between the two, though there are peculiarities which can cause confusion.

In this document we will mainly describe the lxc package. Use of libvirt-lxc is not generally recommended due to a lack of Apparmor protection for libvirt-lxc containers.

In this document, a container name will be shown as CN, C1, or C2.

* [安装](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-installation)
* [Basic usage](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-basic-usage)
* [Global configuration](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-global-conf)
* [联网](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-network)
* [LXC startup](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-startup)
* [Backing Stores](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-backinstores)
* [Templates](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-templates)
* [Autostart](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-autostart)
* [Apparmor](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-apparmor)
* [Control Groups](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-cgroups)
* [Cloning](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-cloning)
* [Lifecycle management hooks](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-hooks)
* [Consoles](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-consoles)
* [疑难解答](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-debugging)
* [LXC API](https://help.ubuntu.com/lts/serverguide/lxc.html#python-lxc)
* [安全性](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-security)
* [资源](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-resources)

**安装**

The lxc package can be installed using

sudo apt-get install lxc

This will pull in the required and recommended dependencies, as well as set up a network bridge for containers to use. If you wish to use unprivileged containers, you will need to ensure that users have sufficient allocated subuids and subgids, and will likely want to allow users to connect containers to a bridge (see [Basic unprivileged usage](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-unpriv)).

**Basic usage**

LXC can be used in two distinct ways - privileged, by running the lxc commands as the root user; or unprivileged, by running the lxc commands as a non-root user. (The starting of unprivileged containers by the root user is possible, but not described here.) Unprivileged containers are more limited, for instance being unable to create device nodes or mount block-backed filesystems. However they are less dangerous to the host, as the root userid in the container is mapped to a non-root userid on the host.

**Basic privileged usage**

To create a privileged container, you can simply to

sudo lxc-create --template download --name u1

or, abbreviated

sudo lxc-create -t download -n u1

This will interactively ask for a container root filesystem type to download - in particular the distribution, release, and architecture. To create the container non-interactively, you can specify these values on the command line:

sudo lxc-create -t download -n u1 -- --dist ubuntu --release trusty --arch amd64

or

sudo lxc-create -t download -n u1 -- -d ubuntu -r trusty -a amd64

You can now use lxc-ls to list containers, lxc-info to obtain detailed container information, lxc-start to start and lxc-stop to stop the container. lxc-attach and lxc-console allow you to enter a container, if ssh is not an option. lxc-destroy removes the container, including its rootfs. See the manual pages for more information on each command. An example session might look like:

sudo lxc-ls --fancy

sudo lxc-start --name u1 --daemon

sudo lxc-info --name u1

sudo lxc-stop --name u1

sudo lxc-destroy --name u1

**User namespaces**

Unprivileged containers allow users to create and administer containers without having any root privilege. The feature underpinning this is called user namespaces. User namespaces are hierarchical, with privileged tasks in a parent namespace being able to map its ids into child namespaces. By default every task on the host runs in the initial user namespace, where the full range of ids is mapped onto the full range. This can be seen by looking at /proc/self/uid\_map and /proc/self/gid\_map, which both will show "0 0 4294967295" when read from the initial user namespace. As of Ubuntu 14.04, when new users are created they are by default offered a range of userids. The list of assigned ids can be seen in the files /etc/subuid and /etc/subgid See their respective manpages for more information. Subuids and subgids are by convention started at id 100000 to avoid conflicting with system users.

If a user was created on an earlier release, it can be granted a range of ids using usermod, as follows:

sudo usermod -v 100000-200000 -w 100000-200000 user1

The programs newuidmap and newgidmap are setuid-root programs in the uidmap package, which are used internally by lxc to map subuids and subgids from the host into the unprivileged container. They ensure that the user only maps ids which are authorized by the host configuration.

**Basic unprivileged usage**

To create unprivileged containers, a few first steps are needed. You will need to create a default container configuration file, specifying your desired id mappings and network setup, as well as configure the host to allow the unprivileged user to hook into the host network. The example below assumes that your mapped user and group id ranges are 100000-165536.

mkdir -p ~/.config/lxc

echo "lxc.id\_map = u 0 100000 65536" > ~/.config/lxc/default.conf

echo "lxc.id\_map = g 0 100000 65536" >> ~/.config/lxc/default.conf

echo "lxc.network.type = veth" >> ~/.config/lxc/default.conf

echo "lxc.network.link = lxcbr0" >> ~/.config/lxc/default.conf

echo "$USER veth lxcbr0 2" | sudo tee -a /etc/lxc/lxc-usernet

After this, you can create unprivileged containers the same way as privileged ones, simply without using sudo.

lxc-create -t download -n u1 -- -d ubuntu -r trusty -a amd64

lxc-start -n u1 -d

lxc-attach -n u1

lxc-stop -n u1

lxc-destroy -n u1

**Nesting**

In order to run containers inside containers - referred to as nested containers - two lines must be present in the parent container configuration file:

lxc.mount.auto = cgroup

lxc.aa\_profile = lxc-container-default-with-nesting

The first will cause the cgroup manager socket to be bound into the container, so that lxc inside the container is able to administer cgroups for its nested containers. The second causes the container to run in a looser Apparmor policy which allows the container to do the mounting required for starting containers. Note that this policy, when used with a privileged container, is much less safe than the regular policy or an unprivileged container. See [Apparmor](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-apparmor) for more information.

**Global configuration**

The following configuration files are consulted by LXC. For privileged use, they are found under /etc/lxc, while for unprivileged use they are under ~/.config/lxc.

* lxc.conf may optionally specify alternate values for several lxc settings, including the lxcpath, the default configuration, cgroups to use, a cgroup creation pattern, and storage backend settings for lvm and zfs.
* default.conf specifies configuration which every newly created container should contain. This usually contains at least a network section, and, for unprivileged users, an id mapping section
* lxc-usernet.conf specifies how unprivileged users may connect their containers to the host-owned network.

lxc.conf and default.conf are exist both under /etc/lxc and $HOME/.config/lxc, while lxc-usernet.conf is only host-wide.

By default, containers are located under /var/lib/lxc for the root user, and $HOME/.local/share/lxc otherwise. The location can be specified for all lxc commands using the "-P|--lxcpath" argument.

**联网**

By default LXC creates a private network namespace for each container, which includes a layer 2 networking stack. Containers usually connect to the outside world by either having a physical NIC or a veth tunnel endpoint passed into the container. LXC creates a NATed bridge, lxcbr0, at host startup. Containers created using the default configuration will have one veth NIC with the remote end plugged into the lxcbr0 bridge. A NIC can only exist in one namespace at a time, so a physical NIC passed into the container is not usable on the host.

It is possible to create a container without a private network namespace. In this case, the container will have access to the host networking like any other application. Note that this is particularly dangerous if the container is running a distribution with upstart, like Ubuntu, since programs which talk to init, like shutdown, will talk over the abstract Unix domain socket to the host's upstart, and shut down the host.

To give containers on lxcbr0 a persistent ip address based on domain name, you can write entries to /etc/lxc/dnsmasq.conf like:

dhcp-host=lxcmail,10.0.3.100

dhcp-host=ttrss,10.0.3.101

If it is desirable for the container to be publicly accessible, there are a few ways to go about it. One is to use iptables to forward host ports to the container, for instance

iptables -t nat -A PREROUTING -p tcp -i eth0 --dport 587 -j DNAT \

--to-destination 10.0.3.100:587

Another is to bridge the host's network interfaces (see the Ubuntu Server Guide's Network Configuration chapter, [Bridging](https://help.ubuntu.com/lts/serverguide/network-configuration.html#bridging)). Then, specify the host's bridge in the container configuration file in place of lxcbr0, for instance

lxc.network.type = veth

lxc.network.link = br0

Finally, you can ask LXC to use macvlan for the container's NIC. Note that this has limitations and depending on configuration may not allow the container to talk to the host itself. Therefore the other two options are preferred and more commonly used.

There are several ways to determine the ip address for a container. First, you can use lxc-ls --fancy which will print the ip addresses for all running containers, or lxc-info -i -H -n C1 which will print C1's ip address. If dnsmasq is installed on the host, you can also add an entry to /etc/dnsmasq.conf as follows

server=/lxc/10.0.3.1

after which dnsmasq will resolve C1.lxc locally, so that you can do:

ping C1

ssh C1

For more information, see the lxc.conf manpage as well as the example network configurations under /usr/share/doc/lxc/examples/.

**LXC startup**

LXC does not have a long-running daemon. However it does have three upstart jobs.

* /etc/init/lxc-net.conf: is an optional job which only runs if /etc/default/lxc-net specifies USE\_LXC\_BRIDGE (true by default). It sets up a NATed bridge for containers to use.
* /etc/init/lxc.conf loads the lxc apparmor profiles and optionally starts any autostart containers. The autostart containers will be ignored if LXC\_AUTO (true by default) is set to true in /etc/default/lxc. See the lxc-autostart manual page for more information on autostarted containers.
* /etc/init/lxc-instance.conf: is used by /etc/init/lxc.conf to autostart a container.

**Backing Stores**

LXC supports several backing stores for container root filesystems. The default is a simple directory backing store, because it requires no prior host customization, so long as the underlying filesystem is large enough. It also requires no root privilege to create the backing store, so that it is seamless for unprivileged use. The rootfs for a privileged directory backed container is located (by default) under /var/lib/lxc/C1/rootfs, while the rootfs for an unprivileged container is under ~/.local/share/lxc/C1/rootfs. If a custom lxcpath is specified in lxc.system.com, then the container rootfs will be under $lxcpath/C1/rootfs.

A snapshot clone C2 of a directory backed container C1 becomes an overlayfs backed container, with a rootfs called overlayfs:/var/lib/lxc/C1/rootfs:/var/lib/lxc/C2/delta0. Other backing store types include loop, btrfs, LVM and zfs.

A btrfs backed container mostly looks like a directory backed container, with its root filesystem in the same location. However, the root filesystem comprises a subvolume, so that a snapshot clone is created using a subvolume snapshot.

The root filesystem for an LVM backed container can be any separate LV. The default VG name can be specified in lxc.conf. The filesystem type and size are configurable per-container using lxc-create.

The rootfs for a zfs backed container is a separate zfs filesystem, mounted under the traditional /var/lib/lxc/C1/rootfs location. The zfsroot can be specified at lxc-create, and a default can be specified in lxc.system.conf.

More information on creating containers with the various backing stores can be found in the lxc-create manual page.

**Templates**

Creating a container generally involves creating a root filesystem for the container. lxc-create delegates this work to templates, which are generally per-distribution. The lxc templates shipped with lxc can be found under /usr/share/lxc/templates, and include templates to create Ubuntu, Debian, Fedora, Oracle, centos, and gentoo containers among others.

Creating distribution images in most cases requires the ability to create device nodes, often requires tools which are not available in other distributions, and usually is quite time-consuming. Therefore lxc comes with a special download template, which downloads pre-built container images from a central lxc server. The most important use case is to allow simple creation of unprivileged containers by non-root users, who could not for instance easily run the debootstrap command.

When running lxc-create, all options which come after -- are passed to the template. In the following command, --name, --template and --bdev are passed to lxc-create, while --release is passed to the template:

lxc-create --template ubuntu --name c1 --bdev loop -- --release trusty

You can obtain help for the options supported by any particular container by passing --help and the template name to lxc-create. For instance, for help with the download template,

lxc-create --template download --help

**Autostart**

LXC supports marking containers to be started at system boot. Prior to Ubuntu 14.04, this was done using symbolic links under the directory /etc/lxc/auto. Starting with Ubuntu 14.04, it is done through the container configuration files. An entry

lxc.start.auto = 1

lxc.start.delay = 5

would mean that the container should be started at boot, and the system should wait 5 seconds before starting the next container. LXC also supports ordering and grouping of containers, as well as reboot and shutdown by autostart groups. See the manual pages for lxc-autostart and lxc.container.conf for more information.

**Apparmor**

LXC ships with a default Apparmor profile intended to protect the host from accidental misuses of privilege inside the container. For instance, the container will not be able to write to /proc/sysrq-trigger or to most /sys files.

The usr.bin.lxc-start profile is entered by running lxc-start. This profile mainly prevents lxc-start from mounting new filesystems outside of the container's root filesystem. Before executing the container's init, LXC requests a switch to the container's profile. By default, this profile is the lxc-container-default policy which is defined in /etc/apparmor.d/lxc/lxc-default. This profile prevents the container from accessing many dangerous paths, and from mounting most filesystems.

Programs in a container cannot be further confined - for instance, MySQL runs under the container profile (protecting the host) but will not be able to enter the MySQL profile (to protect the container).

lxc-execute does not enter an Apparmor profile, but the container it spawns will be confined.

**Customizing container policies**

If you find that lxc-start is failing due to a legitimate access which is being denied by its Apparmor policy, you can disable the lxc-start profile by doing:

sudo apparmor\_parser -R /etc/apparmor.d/usr.bin.lxc-start

sudo ln -s /etc/apparmor.d/usr.bin.lxc-start /etc/apparmor.d/disabled/

This will make lxc-start run unconfined, but continue to confine the container itself. If you also wish to disable confinement of the container, then in addition to disabling the usr.bin.lxc-start profile, you must add:

lxc.aa\_profile = unconfined

to the container's configuration file.

LXC ships with a few alternate policies for containers. If you wish to run containers inside containers (nesting), then you can use the lxc-container-default-with-nesting profile by adding the following line to the container configuration file

lxc.aa\_profile = lxc-container-default-with-nesting

If you wish to use libvirt inside containers, then you will need to edit that policy (which is defined in /etc/apparmor.d/lxc/lxc-default-with-nesting) to uncomment the following line

mount fstype=cgroup -> /sys/fs/cgroup/\*\*,

and re-load the policy.

Note that the nesting policy with privileged containers is far less safe than the default policy, as it allows containers to re-mount /sys and /proc in nonstandard locations, bypassing apparmor protections. Unprivileged containers do not have this drawback since the container root cannot write to root-owned proc and sys files.

Another profile shipped with lxc allows containers to mount block filesystem types like ext4. This can be useful in some cases like maas provisioning, but is deemed generally unsafe since the superblock handlers in the kernel have not been audited for safe handling of untrusted input.

If you need to run a container in a custom profile, you can create a new profile under /etc/apparmor.d/lxc/. Its name must start with lxc- in order for lxc-start to be allowed to transition to that profile. The lxc-default profile includes the re-usable abstractions file /etc/apparmor.d/abstractions/lxc/container-base. An easy way to start a new profile therefore is to do the same, then add extra permissions at the bottom of your policy.

After creating the policy, load it using:

sudo apparmor\_parser -r /etc/apparmor.d/lxc-containers

The profile will automatically be loaded after a reboot, because it is sourced by the file /etc/apparmor.d/lxc-containers. Finally, to make container CN use this new lxc-CN-profile, add the following line to its configuration file:

lxc.aa\_profile = lxc-CN-profile

**Control Groups**

Control groups (cgroups) are a kernel feature providing hierarchical task grouping and per-cgroup resource accounting and limits. They are used in containers to limit block and character device access and to freeze (suspend) containers. They can be further used to limit memory use and block i/o, guarantee minimum cpu shares, and to lock containers to specific cpus.

By default, a privileged container CN will be assigned a cgroup called /lxc/CN. In the case of name conflicts (which can occur when using custom lxcpaths) a suffix "-n", where n is an integer starting at 0, will be appended to the cgroup name.

By default, a privileged container CN will be assigned a cgroup called CN under the cgroup of the task which started the container, for instance /usr/1000.user/1.session/CN. The container root will be given group ownership of the directory (but not all files) so that it is allowed to create new child cgroups.

As of Ubuntu 14.04, LXC uses the cgroup manager (cgmanager) to administer cgroups. The cgroup manager receives D-Bus requests over the Unix socket /sys/fs/cgroup/cgmanager/sock. To fascilitate safe nested containers, the line

lxc.mount.auto = cgroup

can be added to the container configuration causing the /sys/fs/cgroup/cgmanager directory to be bind-mounted into the container. The container in turn should start the cgroup management proxy (done by default if the cgmanager package is installed in the container) which will move the /sys/fs/cgroup/cgmanager directory to /sys/fs/cgroup/cgmanager.lower, then start listening for requests to proxy on its own socket /sys/fs/cgroup/cgmanager/sock. The host cgmanager will ensure that nested containers cannot escape their assigned cgroups or make requests for which they are not authorized.

**Cloning**

For rapid provisioning, you may wish to customize a canonical container according to your needs and then make multiple copies of it. This can be done with the lxc-clone program.

Clones are either snapshots or copies of another container. A copy is a new container copied from the original, and takes as much space on the host as the original. A snapshot exploits the underlying backing store's snapshotting ability to make a copy-on-write container referencing the first. Snapshots can be created from btrfs, LVM, zfs, and directory backed containers. Each backing store has its own peculiarities - for instance, LVM containers which are not thinpool-provisioned cannot support snapshots of snapshots; zfs containers with snapshots cannot be removed until all snapshots are released; LVM containers must be more carefully planned as the underlying filesystem may not support growing; btrfs does not suffer any of these shortcomings, but suffers from reduced fsync performance causing dpkg and apt-get to be slower.

Snapshots of directory-packed containers are created using the overlay filesystem. For instance, a privileged directory-backed container C1 will have its root filesystem under /var/lib/lxc/C1/rootfs. A snapshot clone of C1 called C2 will be started with C1's rootfs mounted readonly under /var/lib/lxc/C2/delta0. Importantly, in this case C1 should not be allowed to run or be removed while C2 is running. It is advised instead to consider C1 a canonical base container, and to only use its snapshots.

Given an existing container called C1, a copy can be created using:

sudo lxc-clone -o C1 -n C2

A snapshot can be created using

sudo lxc-clone -s -o C1 -n C2

See the lxc-clone manpage for more information.

**Snapshots**

To more easily support the use of snapshot clones for iterative container development, LXC supports snapshots. When working on a container C1, before making a potentially dangerous or hard-to-revert change, you can create a snapshot

sudo lxc-snapshot -n C1

which is a snapshot-clone called 'snap0' under /var/lib/lxcsnaps or $HOME/.local/share/lxcsnaps. The next snapshot will be called 'snap1', etc. Existing snapshots can be listed using lxc-snapshot -L -n C1, and a snapshot can be restored - erasing the current C1 container - using lxc-snapshot -r snap1 -n C1. After the restore command, the snap1 snapshot continues to exist, and the previous C1 is erased and replaced with the snap1 snapshot.

Snapshots are supported for btrfs, lvm, zfs, and overlayfs containers. If lxc-snapshot is called on a directory-backed container, an error will be logged and the snapshot will be created as a copy-clone. The reason for this is that if the user creates an overlayfs snapshot of a directory-backed container and then makes changes to the directory-backed container, then the original container changes will be partially reflected in the snapshot. If snapshots of a directory backed container C1 are desired, then an overlayfs clone of C1 should be created, C1 should not be touched again, and the overlayfs clone can be edited and snapshotted at will, as such

lxc-clone -s -o C1 -n C2

lxc-start -n C2 -d # make some changes

lxc-stop -n C2

lxc-snapshot -n C2

lxc-start -n C2 # etc

**Ephemeral Containers**

While snapshots are useful for longer-term incremental development of images, ephemeral containers utilize snapshots for quick, single-use throwaway containers. Given a base container C1, you can start an ephemeral container using

lxc-start-ephemeral -o C1

The container begins as a snapshot of C1. Instructions for logging into the container will be printed to the console. After shutdown, the ephemeral container will be destroyed. See the lxc-start-ephemeral manual page for more options.

**Lifecycle management hooks**

Beginning with Ubuntu 12.10, it is possible to define hooks to be executed at specific points in a container's lifetime:

* Pre-start hooks are run in the host's namespace before the container ttys, consoles, or mounts are up. If any mounts are done in this hook, they should be cleaned up in the post-stop hook.
* Pre-mount hooks are run in the container's namespaces, but before the root filesystem has been mounted. Mounts done in this hook will be automatically cleaned up when the container shuts down.
* Mount hooks are run after the container filesystems have been mounted, but before the container has called pivot\_root to change its root filesystem.
* Start hooks are run immediately before executing the container's init. Since these are executed after pivoting into the container's filesystem, the command to be executed must be copied into the container's filesystem.
* Post-stop hooks are executed after the container has been shut down.

If any hook returns an error, the container's run will be aborted. Any post-stop hook will still be executed. Any output generated by the script will be logged at the debug priority.

Please see the lxc.container.conf manual page for the configuration file format with which to specify hooks. Some sample hooks are shipped with the lxc package to serve as an example of how to write and use such hooks.

**Consoles**

Containers have a configurable number of consoles. One always exists on the container's /dev/console. This is shown on the terminal from which you ran lxc-start, unless the -d option is specified. The output on /dev/console can be redirected to a file using the -c console-file option to lxc-start. The number of extra consoles is specified by the lxc.tty variable, and is usually set to 4. Those consoles are shown on /dev/ttyN (for 1 <= N <= 4). To log into console 3 from the host, use

sudo lxc-console -n container -t 3

or if the -t N option is not specified, an unused console will be automatically chosen. To exit the console, use the escape sequence Ctrl-a q. Note that the escape sequence does not work in the console resulting from lxc-start without the -d option.

Each container console is actually a Unix98 pty in the host's (not the guest's) pty mount, bind-mounted over the guest's /dev/ttyN and /dev/console. Therefore, if the guest unmounts those or otherwise tries to access the actual character device 4:N, it will not be serving getty to the LXC consoles. (With the default settings, the container will not be able to access that character device and getty will therefore fail.) This can easily happen when a boot script blindly mounts a new /dev.

**疑难解答**

**Logging**

If something goes wrong when starting a container, the first step should be to get full logging from LXC:

sudo lxc-start -n C1 -l trace -o debug.out

This will cause lxc to log at the most verbose level, trace, and to output log information to a file called 'debug.out'. If the file debug.out already exists, the new log information will be appended.

**Monitoring container status**

Two commands are available to monitor container state changes. lxc-monitor monitors one or more containers for any state changes. It takes a container name as usual with the -n option, but in this case the container name can be a posix regular expression to allow monitoring desirable sets of containers. lxc-monitor continues running as it prints container changes. lxc-wait waits for a specific state change and then exits. For instance,

sudo lxc-monitor -n cont[0-5]\*

would print all state changes to any containers matching the listed regular expression, whereas

sudo lxc-wait -n cont1 -s 'STOPPED|FROZEN'

will wait until container cont1 enters state STOPPED or state FROZEN and then exit.

**Attach**

As of Ubuntu 14.04, it is possible to attach to a container's namespaces. The simplest case is to simply do

sudo lxc-attach -n C1

which will start a shell attached to C1's namespaces, or, effectively inside the container. The attach functionality is very flexible, allowing attaching to a subset of the container's namespaces and security context. See the manual page for more information.

**Container init verbosity**

If LXC completes the container startup, but the container init fails to complete (for instance, no login prompt is shown), it can be useful to request additional verbosity from the init process. For an upstart container, this might be:

sudo lxc-start -n C1 /sbin/init loglevel=debug

You can also start an entirely different program in place of init, for instance

sudo lxc-start -n C1 /bin/bash

sudo lxc-start -n C1 /bin/sleep 100

sudo lxc-start -n C1 /bin/cat /proc/1/status

**LXC API**

Most of the LXC functionality can now be accessed through an API exported by liblxc for which bindings are available in several languages, including Python, lua, ruby, and go.

Below is an example using the python bindings (which are available in the python3-lxc package) which creates and starts a container, then waits until it has been shut down:

# sudo python3

Python 3.2.3 (default, Aug 28 2012, 08:26:03)

[GCC 4.7.1 20120814 (prerelease)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> import lxc

\_\_main\_\_:1: Warning: The python-lxc API isn't yet stable and may change at any p

oint in the future.

>>> c=lxc.Container("C1")

>>> c.create("ubuntu")

True

>>> c.start()

True

>>> c.wait("STOPPED")

True

**安全性**

A namespace maps ids to resources. By not providing a container any id with which to reference a resource, the resource can be protected. This is the basis of some of the security afforded to container users. For instance, IPC namespaces are completely isolated. Other namespaces, however, have various leaks which allow privilege to be inappropriately exerted from a container into another container or to the host.

By default, LXC containers are started under a Apparmor policy to restrict some actions. The details of AppArmor integration with lxc are in section [Apparmor](https://help.ubuntu.com/lts/serverguide/lxc.html#lxc-apparmor). Unprivileged containers go further by mapping root in the container to an unprivileged host userid. This prevents access to /proc and /sys files representing host resources, as well as any other files owned by root on the host.

**Exploitable system calls**

It is a core container feature that containers share a kernel with the host. Therefore if the kernel contains any exploitable system calls the container can exploit these as well. Once the container controls the kernel it can fully control any resource known to the host.

Since Ubuntu 12.10 (Quantal) a container can also be constrained by a seccomp filter. Seccomp is a new kernel feature which filters the system calls which may be used by a task and its children. While improved and simplified policy management is expected in the near future, the current policy consists of a simple whitelist of system call numbers. The policy file begins with a version number (which must be 1) on the first line and a policy type (which must be 'whitelist') on the second line. It is followed by a list of numbers, one per line.

In general to run a full distribution container a large number of system calls will be needed. However for application containers it may be possible to reduce the number of available system calls to only a few. Even for system containers running a full distribution security gains may be had, for instance by removing the 32-bit compatibility system calls in a 64-bit container. See the lxc.container.conf manual page for details of how to configure a container to use seccomp. By default, no seccomp policy is loaded.

**资源**

* The DeveloperWorks article [LXC: Linux container tools](https://www.ibm.com/developerworks/linux/library/l-lxc-containers/) was an early introduction to the use of containers.
* The [Secure Containers Cookbook](http://www.ibm.com/developerworks/linux/library/l-lxc-security/index.html) demonstrated the use of security modules to make containers more secure.
* Manual pages referenced above can be found at:
* [capabilities](http://manpages.ubuntu.com/manpages/en/man7/capabilities.7.html)
* [lxc.conf](http://manpages.ubuntu.com/manpages/en/man5/lxc.conf.5.html)
* The upstream LXC project is hosted at [linuxcontainers.org](http://linuxcontainers.org).
* LXC security issues are listed and discussed at [the LXC Security wiki page](http://wiki.ubuntu.com/LxcSecurity)
* For more on namespaces in Linux, see: S. Bhattiprolu, E. W. Biederman, S. E. Hallyn, and D. Lezcano. Virtual Servers and Check- point/Restart in Mainstream Linux. SIGOPS Operating Systems Review, 42(5), 2008.