Oracle Linux 7

Working With LXC

F32445-03
September 2022

Oracle Linux 7 Working With LXC,

F32445-03

Contents

Preface

Oracle® Linux 7: Working With LXC describes how to use Linux Containers to isolate applications and entire operating system images from the other processes that are running on a host system. The version of LXC described here is 1.0.7 or later, which has some significant enhancements over previous versions.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees,

customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Linux Containers

This chapter provides an overview of Linux Containers (LXC) and the supported Oracle Linux container versions.

Information about Linux Containers are also available in the following sources:

- Oracle Cloud Native Environment documentation.
- Oracle Linux: Oracle Container Runtime for Docker User's Guide that describes how to use the Docker Engine to create application containers.

For more information, see https://wiki.archlinux.org/index.php/Linux_Containers and the LXC manual pages.

About Linux Containers

> **Note:**
>
> Prior to UEK R3, LXC was a Technology Preview feature that was made available for testing and evaluation purposes, but was not recommended for production systems. LXC is a supported feature with UEK R3 and UEK R4.

The LXC feature is a lightweight virtualization mechanism that does not require you to set up a virtual machine on an emulation of physical hardware. This feature takes the cgroups resource management facilities as its basis and adds POSIX file capabilities to implement process and network isolation. You can run a single application within a container (an *application container*) whose name space is isolated from the other processes on the system in a similar manner to a `chroot` jail. However, the main use of LXC is to allow you to run a complete copy of the Linux operating system in a container (a *system container*) without the overhead of running a level-2 hypervisor such as VirtualBox. In fact, the container is sharing the kernel with the host system, so its processes and file system are completely visible from the host. When you are logged into the container, you only see its file system and process space. Because the kernel is shared, you are limited to the modules and drivers that it has loaded.

Typical use cases for Linux Containers are:

- Running Oracle Linux 5, Oracle Linux 6, and Oracle Linux 7 containers in parallel. You can run an Oracle Linux 5 container on an Oracle Linux 7 system with the UEK R3 or UEKR4 kernel, even though UEK R3 and UEK R4 are not supported for Oracle Linux 5. You can also run an i386 container on an x86_64 kernel. For more information, see Supported Oracle Linux Container Versions.
- Running applications that are supported only by Oracle Linux 5 in an Oracle Linux 5 container on an Oracle Linux 7 host. However, incompatibilities might exist in the modules and drivers that are available.

- Running many copies of application configurations on the same system. An example configuration would be a LAMP stack, which combines Linux, Apache HTTP server, MySQL, and Perl, PHP, or Python scripts to provide specialised web services.
- Creating sandbox environments for development and testing.
- Providing user environments whose resources can be tightly controlled, but which do not require the hardware resources of full virtualization solutions.
- Creating containers where each container appears to have its own IP address. For example you can use the `lxc-sshd` template script to create isolated environments for untrusted users. Each container runs an `sshd` daemon to handle logins. By bridging a container's Virtual Ethernet interface to the host's network interface, each container can appear to have its own IP address on a LAN.

When you use the `lxc-start` command to start a system container, by default the copy of `/sbin/init` (for an Oracle Linux 6 or earlier container) or `/usr/lib/systemd/systemd` (for an Oracle Linux 7 container) in the container is started to spawn other processes in the container's process space. Any system calls or device access are handled by the kernel running on the host. If you need to run different kernel versions or different operating systems from the host, use a full virtualization solution such as Oracle VM or Oracle VM VirtualBox instead of Linux Containers.

For a container to run correctly, ensure that the following steps are completed:

- Disable any `init` or `systemd` scripts that load modules to access hardware directly.
- Disable `udev` and instead create static device nodes in `/dev` for any hardware that needs to be accessible from within the container.
- Configure the network interface so that it is bridged to the network interface of the host system.

LXC provides a number of template scripts in `/usr/share/lxc/templates` that perform much of the required configuration of system containers for you. However, it is likely that you will need to modify the script to allow the container to work correctly as the scripts cannot anticipate the idiosyncrasies of your system's configuration. You use the `lxc-create` command to create a system container by invoking a template script. For example, the `lxc-busybox` template script creates a lightweight BusyBox system container.

The example system container in this chapter uses the template script for Oracle Linux (`lxc-oracle`). The container is created on a btrfs file system (`/container`) to take advantage of its snapshot feature. A btrfs file system allows you to create a subvolume that contains the root file system (`rootfs`) of a container, and to quickly create new containers by cloning this subvolume.

You can use control groups to limit the system resources that are available to applications such as web servers or databases that are running in the container.

Application containers are not created by using template scripts. Instead, an application container mounts all or part of the host's root file system to provide access to the binaries and libraries that the application requires. You use the `lxc-execute` command to invoke `/usr/sbin/init.lxc` (a cut-down version of `/sbin/init`) in the container. `init.lxc` mounts any required directories such as `/proc`, `/dev/shm`, and `/dev/mqueue`, executes the specified application program, and then waits for it to finish executing. When the application exits, the container instance ceases to exist.

Supported Oracle Linux Container Versions

All versions of Oracle Linux 7, running `kernel-uek-3.8.13-35.3.1` or later, support the following container versions:

- Oracle Linux 5.9 or later
- Oracle Linux 6.5 or later
- Oracle Linux 7.0 or later

Note that subsequent versions of Oracle Linux 7 and UEK are tested to support the listed container versions. Exceptions, if any, are listed in the release notes for the version of Oracle Linux 7 affected.

2
Administering Linux Containers

This chapter describes different ways of configuring and customizing Linux Containers according to your organization's needs and preferences.

> **Note:**
>
> Throughout this documentation, you must run all the commands that are provided on the host system unless indicated otherwise.

Configuring Operating System Containers

The procedures in the following sections describe how to set up Linux Containers that contain a copy of the root file system installed from packages on the Oracle Linux yum server.

Installing and Configuring the Software

1. Install the `btrfs-progs` package.

   ```
   sudo yum install btrfs-progs
   ```

2. Install the `lxc` and `wget` packages.

   ```
   sudo yum install lxc wget
   ```

   This command installs all of the required packages. The LXC template scripts are installed in `/usr/share/lxc/templates`. LXC uses `wget` to download packages from the Oracle Linux yum server.

3. Start the LXC network management service, `lxc-net`, and configure the service to start at boot time.

   ```
   sudo systemctl start lxc-net.service
   sudo systemctl enable lxc-net.service
   ```

   LXC includes its own network management service to support network bridging for containers.

4. If you are going to compile applications that require the LXC header files and libraries, install the `lxc-devel` package.

   ```
   sudo yum install lxc-devel
   ```

Setting up the File System for the Containers

> **Note:**
>
> The LXC template scripts assume that containers are created in `/container`. You must edit the script if your system's configuration differs from this assumption.

1. Create a btrfs file system on a suitably sized device such as `/dev/sdb`.

   ```
   sudo mkfs.btrfs /dev/sdb
   ```

2. Mount the `/container` file system. The `/container` directory is created automatically when you install LXC.

   ```
   sudo mount /dev/sdb /container
   ```

3. Add an entry for `/container` to the `/etc/fstab` file.

   ```
   /dev/sdb      /container    btrfs    defaults    0 0
   ```

For more information, see About the Btrfs File System in Oracle Linux 7: Managing File Systems.

Creating and Starting a Container

> **Note:**
>
> The procedure in this section uses the LXC template script for Oracle Linux (`lxc-oracle`), which is located in `/usr/share/lxc/templates`.
>
> An Oracle Linux container requires a minimum of 400 MB of disk space.

1. Create an Oracle Linux 6 container named `ol6ctr1` using the `lxc-oracle` template script.

   ```
   sudo lxc-create -n ol6ctr1 -B btrfs -t oracle -- --release=6.latest

   Host is OracleEverything 7.0
   Create configuration file /container/ol6ctr1/config
   Yum installing release 6.latest for x86_64
     .
     .
     .
     yum-metadata-parser.x86_64
   0:1.1.2-16.el6
     zlib.x86_64
   0:1.2.3-29.el6

   Complete!
   Rebuilding rpm database
   Patching container rootfs /container/ol6ctr1/rootfs for Oracle Linux 6.5
   ```

```
Configuring container for Oracle Linux 6.5
Added container user:oracle password:oracle
Added container user:root password:root
Container : /container/ol6ctr1/rootfs
Config    : /container/ol6ctr1/config
Network   : eth0 (veth) on lxcbr0
```

> **Note:**
>
> For LXC version 1.0 and later, you must specify the `-B btrfs` option if you want to use the snapshot features of btrfs. For more information, see the `lxc-create(1)` manual page.

The `lxc-create` command runs the template script `lxc-oracle` to create the container in `/container/ol6ctr1` with the btrfs subvolume `/container/ol6ctr1/rootfs` as its root file system. The command then uses `yum` to install the latest available update of Oracle Linux 6 from the Oracle Linux yum server. It also writes the container's configuration settings to the file `/container/ol6ctr1/config` and its `fstab` file to `/container/ol6ctr1/fstab`. The default log file for the container is `/container/ol6ctr1/ol6ctr1.log`.

You can specify the following template options after the `--` option to `lxc-create`:

`-a | --arch=i386|x86_64`
Specifies the architecture. The default value is the architecture of the host.

`--baseurl=`*`pkg_repo`*
Specify the file URI of a package repository. You must also use the `--arch` and `--release` options to specify the architecture and the release, for example:

```
mount -o loop OracleLinux-R7-GA-Everything-x86_64-dvd.iso /mnt
lxc-create -n ol70beta -B btrfs -t oracle -- -R 7.0 -a x86_64 --
baseurl=file:///mnt/Server
```

`-P | --patch=`*`path`*
Patch the `rootfs` at the specified path.

`--privileged[=rt]`
Allows you to adjust the values of certain kernel parameters under the `/proc` hierarchy.

The container uses a privilege configuration file, which mounts `/proc` read-only with some exceptions. See Configuring Kernel Parameter Settings for Oracle Linux Containers.

This option also enables the `CAP_SYS_NICE` capability, which allows you to set negative `nice` values (that is, more favored for scheduling) for processes from within the container.

If you specify the `=rt` (real-time) modifier, you can configure the `lxc.cgroup.cpu.rt_runtime_us` setting in the container's configuration file or when you start the container. This setting specifies the maximum continuous period in microseconds for which the container has access to CPU resources from the base period set by the system-wide value of `cpu.rt_period_us`. Otherwise, a container uses

the system-wide value of `cpu.rt_runtime_us`, which might cause it to consume too many CPU resources. In addition, this modifier ensures that rebooting a container terminates all of its processes and boots it to a clean state.

`-R | --release=`*`major.minor`*

Specifies the major release number and minor update number of the Oracle release to install. The value of *major* can be set to 4, 5, 6, or 7. If you specify `latest` for *minor*, the latest available release packages for the major release are installed. If the host is running Oracle Linux, the default release is the same as the release installed on the host. Otherwise, the default release is the latest update of Oracle Linux 6.

`-r | --rpms=`*`rpm_name`*

Install the specified RPM in the container.

`-t | --templatefs=`*`rootfs`*

Specifies the path to the root file system of an existing system, container, or Oracle VM template that you want to copy. Do not specify this option with any other template option. See Creating Additional Containers.

`-u | --url=`*`repo_URL`*

Specifies a yum repository other than Oracle Public Yum. For example, you might want to perform the installation from a local yum server. The repository file in configured in `/etc/yum.repos.d` in the container's root file system. The default URL is `https://yum.oracle.com`.

2. If you want to create additional copies of the container in its initial state, create a snapshot of the container's root file system, for example:

```
sudo btrfs subvolume snapshot /container/ol6ctr1/rootfs /container/ol6ctr1/
rootfs_snap
```

See About the Btrfs File System in Oracle® Linux 7: Managing File Systems and Creating Additional Containers.

3. Start the container `ol6ctr1` as a daemon that writes its diagnostic output to a log file other than the default log file.

```
sudo lxc-start -n ol6ctr1 -d -o /container/ol6ctr1_debug.log -l DEBUG
```

> **Note:**
>
> If you omit the `-d` option, the container's console opens in the current shell.
>
> The following logging levels are available: `FATAL`, `CRIT`, `WARN`, `ERROR`, `NOTICE`, `INFO`, and `DEBUG`. You can set a logging level for all `lxc-*` commands.

If you run the `ps -ef --forest` command on the host system and the process tree below the `lxc-start` process shows that the `/usr/sbin/sshd` and `/sbin/mingetty` processes have started in the container, you can log in to the container from the host. See Logging in to Containers.

About the lxc-oracle Template Script

> **Note:**
>
> If you amend a template script, you alter the configuration files of all containers that you subsequently create from that script. If you amend the `config` file for a container, you alter the configuration of that container and all containers that you subsequently clone from it.

The `lxc-oracle` template script defines system settings and resources that are assigned to a running container, including the following:

- Default passwords for the `oracle` and `root` users, which are set to `oracle` and `root` respectively
- Host name (`lxc.utsname`), which is set to the name of the container
- Number of available terminals (`lxc.tty`), which is set to 4
- Location of the container's root file system on the host (`lxc.rootfs`)
- Location of the `fstab` mount configuration file (`lxc.mount`)
- All of the system capabilities that are not available to the container (`lxc.cap.drop`)
- Local network interface configuration (`lxc.network`)
- All of the allowed cgroup devices (`lxc.cgroup.devices.allow`)

The template script sets the virtual network type (`lxc.network.type`) and bridge (`lxc.network.link`) to `veth` and `lxcbr0`. If you want to use a macvlan bridge or Virtual Ethernet Port Aggregator that allows external systems to access your container via the network, you must modify the container's configuration file. See About Veth and Macvlan and Modifying a Container to Use Macvlan.

To enhance security, you can uncomment `lxc.cap.drop` capabilities to prevent `root` in the container from performing certain actions. For example, dropping the `sys_admin` capability prevents `root` from remounting the container's `fstab` entries as writable. However, dropping `sys_admin` also prevents the container from mounting any file system and disables the `hostname` command. By default, the template script drops the following capabilities: `mac_admin`, `mac_override`, `setfcap`, `setpcap`, `sys_module`, `sys_nice`, `sys_pacct`, `sys_rawio`, and `sys_time`.

For more information, see the `capabilities(7)` and `lxc.conf(5)` manual pages.

When you create a container, the template script writes the container's configuration settings and mount configuration to `/container/`*`name`*`/config` and `/container/`*`name`*`/fstab`, and sets up the container's root file system under `/container/`*`name`*`/rootfs`.

Unless you specify to clone an existing root file system, the template script installs the following packages under `rootfs` (by default, from the Oracle Linux Yum Server at `https://yum.oracle.com`):

- `chkconfig`: `chkconfig` utility for maintaining the `/etc/rc*.d` hierarchy.

- `dhclient`: DHCP client daemon (`dhclient`) and `dhclient-script`.
- `initscripts`: `/etc/inittab` file and `/etc/init.d` scripts.
- `openssh-server`: Open source SSH server daemon, `/usr/sbin/sshd`.
- `oraclelinux-release`: Oracle Linux release and information files.
- `passwd`: `passwd` utility for setting or changing passwords using PAM.
- `policycoreutils`: SELinux policy core utilities.
- `rootfiles`: Basic files required by the `root` user.
- `rsyslog`: Enhanced system logging and kernel message trapping daemons.
- `vim-minimal`: Minimal version of the VIM editor.
- `yum`: `yum` utility for installing, updating and managing RPM packages.

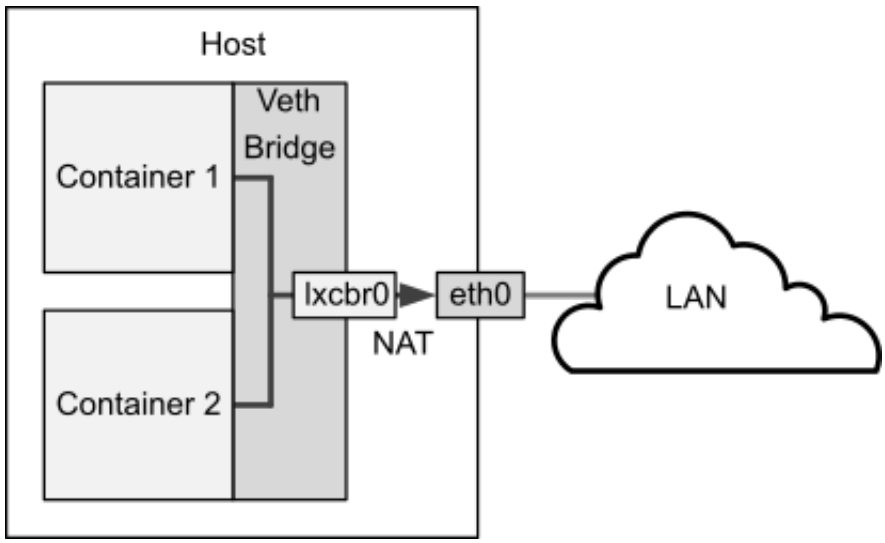
The template script edits the system configuration files under `rootfs` to set up networking in the container and to disable unnecessary services including volume management (LVM), device management (`udev`), the hardware clock, `readahead`, and the Plymouth boot system.

About Veth and Macvlan

By default, the `lxc-oracle` template script sets up networking by setting up a veth bridge. In this mode, a container obtains its IP address from the `dnsmasq` server that the `lxc-net` service runs on the private virtual bridge network (`lxcbr0`) between the container and the host. The host allows a container to connect to the rest of the network by using NAT rules in `iptables`, but these rules do not allow incoming connections to the container. Both the host and other containers on the veth bridge have network access to the container via the bridge.

Figure 2-1 illustrates a host system with two containers that are connected via the veth bridge `lxcbr0`.

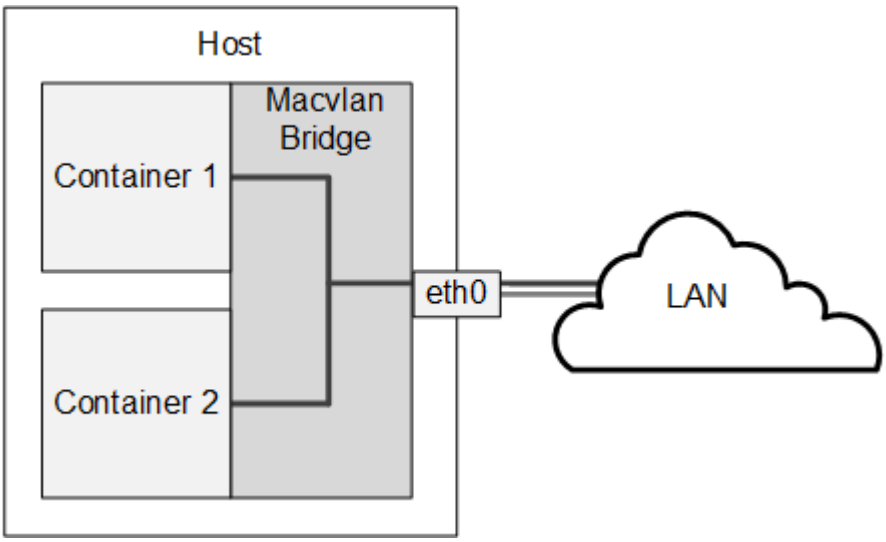
Figure 2-1 Network Configuration of Containers Using a Veth Bridge



If you want to allow network connections from outside the host to be able to connect to the container, the container needs to have an IP address on the same network as the host. One way to achieve this configuration is to use a macvlan bridge to create an independent logical network for the container. This network is effectively an extension of the local network that is connected the host's network interface. External systems can access the container as though it were an independent system on the network, and the container has network access to other containers that are configured on the bridge and to external systems. The container can also obtain its IP address from an external DHCP server on your local network. However, unlike a veth bridge, the host system does not have network access to the container.

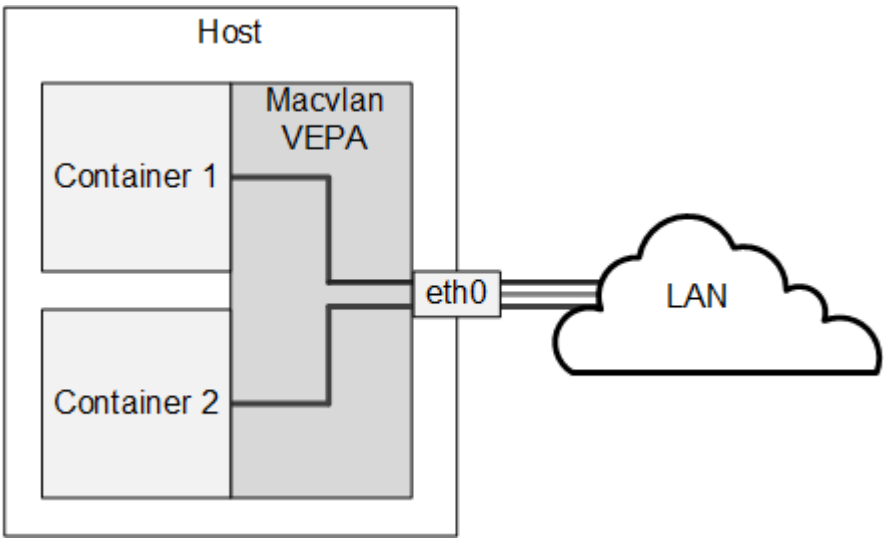
Figure 2-2 illustrates a host system with two containers that are connected via a macvlan bridge.

Figure 2-2 Network Configuration of Containers Using a Macvlan Bridge



If you do not want containers to be able to see each other on the network, you can configure the Virtual Ethernet Port Aggregator (VEPA) mode of macvlan. Figure 2-3 illustrates a host system with two containers that are separately connected to a network by a macvlan VEPA. In effect, each container is connected directly to the network, but neither container can access the other container nor the host via the network.

Figure 2-3 Network Configuration of Containers Using a Macvlan VEPA



For information about configuring macvlan, see Modifying a Container to Use Macvlan and the `lxc.conf(5)` manual page.

Modifying a Container to Use Macvlan

To modify a container so that it uses the bridge or VEPA mode of macvlan, edit `/container/`*`name`*`/config` and replace the following lines:

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

with these lines for bridge mode:

```
lxc.network.type = macvlan
lxc.network.macvlan.mode = bridge
lxc.network.flags = up
lxc.network.link = eth0
```

or these lines for VEPA mode:

```
lxc.network.type = macvlan
lxc.network.macvlan.mode = vepa
lxc.network.flags = up
lxc.network.link = eth0
```

In these sample configurations, the setting for `lxc.network.link` assumes that you want the container's network interface to be visible on the network that is accessible via the host's `eth0` interface.

Modifying a Container to Use a Static IP Address

By default, a container connected by macvlan relies on the DHCP server on your local network to obtain its IP address. If you want the container to act as a server, you would

usually configure it with a static IP address. You can configure DHCP to serve a static IP address for a container or you can define the address in the container's `config` file.

To configure a static IP address that a container does not obtain using DHCP:

1. Edit `/container/name/rootfs/etc/sysconfig/network-scripts/ifcfg-iface` , where *iface* is the name of the network interface, and change the following line:

   ```
   BOOTPROTO=dhcp
   ```

   to read:

   ```
   BOOTPROTO=none
   ```

2. Add the following line to the `/container/name/config` file:

   ```
   lxc.network.ipv4 = xxx.xxx.xxx.xxx/prefix_length
   ```

   In the previous example, *xxx.xxx.xxx.xxx/prefix_length* is the IP address of the container in CIDR format, for example: `192.168.56.100/24`.

> **Note:**
>
> The address must not already be in use on the network or potentially be assignable by a DHCP server to another system.
>
> You might also need to configure the firewall on the host to allow access to a network service that is provided by a container.

Logging in to Containers

You can use the `lxc-console` command to log in to a running container.

```
sudo lxc-console -n name [-t tty_number]
```

If you do not specify a tty number, you log in to the first available terminal.

For example, to log in to a terminal on `ol6ctr1`, type:

```
sudo lxc-console -n ol6ctr1
```

To exit an `lxc-console` session, type `Ctrl-A` followed by `Q`.

Alternatively, you can use `ssh` to log in to a container if you install the `lxc-0.9.0-2.0.5` package (or later version of this package).

> **Note:**
>
> To be able to log in using `lxc-console`, the container must be running an `/sbin/mingetty` process for the terminal. Similarly, using `ssh` requires that the container is running the SSH daemon (`/usr/sbin/sshd`).

Creating Additional Containers

To clone an existing container, use the `lxc-clone` command\, as shown in this example:

```
sudo lxc-clone -o ol6ctr1 -n ol6ctr2
```

Alternatively, you can use the `lxc-create` command to create a container by copying the root file system from an existing system, container, or Oracle VM template. Specify the path of the root file system as the argument to the `--templatefs` template option.

```
sudo lxc-create -n ol6ctr3 -B btrfs -t oracle -- --templatefs=/container/ol6ctr1/
rootfs_snap
```

This example copies the new container's `rootfs` from a snapshot of the `rootfs` that belongs to container `ol6ctr1`. The additional container is created in `/container/ol6ctr3` and a new `rootfs` snapshot is created in `/container/ol6ctr3/rootfs`.

> **Note:**
>
> For LXC version 1.0 and later, you must specify the `-B btrfs` option if you want to use the snapshot features of btrfs. For more information, see the `lxc-create(1)` manual page.
>
> To change the host name of the container, edit the `HOSTNAME` settings in `/container/`*`name`*`/rootfs/etc/sysconfig/network` and `/container/`*`name`*`/rootfs/etc/sysconfig/network-scripts/ifcfg-`*`iface`* , where *iface* is the name of the network interface, such as `eth0`.

Monitoring and Shutting Down Containers

To display the containers that are configured, use the `lxc-ls` command.

```
sudo lxc-ls

ol6ctr1
ol6ctr2
```

To display the containers that are running on the host system, specify the `--active` option.

```
sudo lxc-ls --active

ol6ctr1
```

To display the state of a container, use the `lxc-info` command on the host.

```
sudo lxc-info -n ol6ctr1

Name:           ol6ctr1
State:          RUNNING
PID:            5662
```

```
IP:             192.168.122.188
CPU use:        1.63 seconds
BlkIO use:      18.95 MiB
Memory use:     11.53 MiB
KMem use:       0 bytes
Link:           vethJHU5OA
 TX bytes:      1.42 KiB
 RX bytes:      6.29 KiB
 Total bytes:   7.71 KiB
```

A container can be in one of the following states: `ABORTING`, `RUNNING`, `STARTING`, `STOPPED`, or `STOPPING`. Although `lxc-info` might show your container to be in the `RUNNING` state, you cannot log in to it unless the `/usr/sbin/sshd` or `/sbin/mingetty` processes have started running in the container. You must allow time for the `init` or `systemd` process in the container to first start networking and the various other services that you have configured.

To view the state of the processes in the container from the host, either run `ps -ef --forest` and look for the process tree below the `lxc-start` process or use the `lxc-attach` command to run the `ps` command in the container.

```
sudo ps -ef --forest

UID   PID   PPID  C STIME TTY    TIME     CMD
...
root  3171     1  0 09:57 ?      00:00:00 lxc-start -n ol6ctr1 -d
root  3182  3171  0 09:57 ?      00:00:00  \_ /sbin/init
root  3441  3182  0 09:57 ?      00:00:00      \_ /sbin/dhclient -H ol6ctr1 ...
root  3464  3182  0 09:57 ?      00:00:00      \_ /sbin/rsyslogd ...
root  3493  3182  0 09:57 ?      00:00:00      \_ /usr/sbin/sshd
root  3500  3182  0 09:57 pts/5  00:00:00      \_ /sbin/mingetty ... /dev/console
root  3504  3182  0 09:57 pts/1  00:00:00      \_ /sbin/mingetty ... /dev/tty1
root  3506  3182  0 09:57 pts/2  00:00:00      \_ /sbin/mingetty ... /dev/tty2
root  3508  3182  0 09:57 pts/3  00:00:00      \_ /sbin/mingetty ... /dev/tty3
root  3510  3182  0 09:57 pts/4  00:00:00      \_ /sbin/mingetty ... /dev/tty4
...

sudo lxc-attach -n ol6ctr1 -- /bin/ps aux

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  19284  1516 ?        Ss   04:57   0:00 /sbin/init
root       202  0.0  0.0   9172   588 ?        Ss   04:57   0:00 /sbin/dhclient
root       225  0.0  0.1 245096  1332 ?        Ssl  04:57   0:00 /sbin/rsyslogd
root       252  0.0  0.1  66660  1192 ?        Ss   04:57   0:00 /usr/sbin/sshd
root       259  0.0  0.0   4116   568 lxc/console Ss+ 04:57   0:00 /sbin/mingett
root       263  0.0  0.0   4116   572 lxc/tty1 Ss+  04:57   0:00 /sbin/mingetty
root       265  0.0  0.0   4116   568 lxc/tty2 Ss+  04:57   0:00 /sbin/mingetty
root       267  0.0  0.0   4116   572 lxc/tty3 Ss+  04:57   0:00 /sbin/mingetty
root       269  0.0  0.0   4116   568 lxc/tty4 Ss+  04:57   0:00 /sbin/mingetty
root       283  0.0  0.1 110240  1144 ?        R+   04:59   0:00 /bin/ps aux
```

> **Tip:**
>
> If a container appears not to be starting correctly, examining its process tree from the host will often reveal where the problem might lie.

If you were logged into the container such as `ol6ctrl`, the output from the `ps -ef` command would look similar to the following example:

```
sudo ps -ef

UID   PID  PPID  C STIME TTY          TIME CMD
root    1     0  0 11:54 ?        00:00:00 /sbin/init
root  193     1  0 11:54 ?        00:00:00 /sbin/dhclient -H ol6ctr1 ...
root  216     1  0 11:54 ?        00:00:00 /sbin/rsyslogd -i ...
root  258     1  0 11:54 ?        00:00:00 /usr/sbin/sshd
root  265     1  0 11:54 lxc/console 00:00:00 /sbin/mingetty ... /dev/console
root  271     1  0 11:54 lxc/tty2 00:00:00 /sbin/mingetty ... /dev/tty2
root  273     1  0 11:54 lxc/tty3 00:00:00 /sbin/mingetty ... /dev/tty3
root  275     1  0 11:54 lxc/tty4 00:00:00 /sbin/mingetty ... /dev/tty4
root  297     1  0 11:57 ?        00:00:00 login -- root
root  301   297  0 12:08 lxc/tty1 00:00:00 -bash
root  312   301  0 12:08 lxc/tty1 00:00:00 ps -ef
```

Note that the process numbers differ from those of the same processes on the host, and that they all descend from process 1, `/sbin/init`, in the container.

To suspend or resume the execution of a container, use the `lxc-freeze` and `lxc-unfreeze` commands on the host system.

```
sudo lxc-freeze -n ol6ctr1
sudo lxc-unfreeze -n ol6ctr1
```

From the host, you can use the `lxc-stop` command with the `--nokill` option to shut down the container in an orderly manner.

```
sudo lxc-stop --nokill -n ol6ctr1
```

Alternatively, you can run a command such as `halt` while logged in to the `ol6ctrl` container.

```
sudo halt

Broadcast message from root@ol6ctr1
    (/dev/tty2) at 22:52 ...

The system is going down for halt NOW!
lxc-console: Input/output error - failed to read
```

As shown in the example, you are returned to the shell prompt on the host.

To shut down a container by terminating its processes immediately, use `lxc-stop` with the `-k` option.

```
sudo lxc-stop -k -n ol6ctr1
```

If you are debugging the operation of a container, this is the quickest method as you would usually destroy the container and create a new version after modifying the template script.

To monitor the state of a container, use the `lxc-monitor` command.

```
sudo lxc-monitor -n ol6ctr1

'ol6ctr1' changed state to [STARTING]
'ol6ctr1' changed state to [RUNNING]
'ol6ctr1' changed state to [STOPPING]
'ol6ctr1' changed state to [STOPPED]
```

To wait for a container to change to a specified state, use the `lxc-wait` command.

```
sudo lxc-wait -n $CTR -s ABORTING && lxc-wait -n $CTR -s STOPPED && echo
"Container $CTR terminated with an error."
```

Starting a Command Inside a Running Container

> **Note:**
>
> The `lxc-attach` command is supported by UEK R3 with the `lxc-0.9.0-2.0.4` package or later.

From the host system, you can use `lxc-attach` to execute an arbitrary command inside a container that is already running, for example:

```
sudo lxc-attach -n ol6ctr1 -- /bin/ps aux

USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1  19284  1516 ?        Ss   04:57   0:00 /sbin/init
root        202  0.0  0.0   9172   588 ?        Ss   04:57   0:00 /sbin/dhclient
root        225  0.0  0.1 245096  1332 ?        Ssl  04:57   0:00 /sbin/rsyslogd
root        252  0.0  0.1  66660  1192 ?        Ss   04:57   0:00 /usr/sbin/sshd
root        259  0.0  0.0   4116   568 lxc/console Ss+ 04:57   0:00 /sbin/mingett
root        263  0.0  0.0   4116   572 lxc/tty1 Ss+  04:57   0:00 /sbin/mingetty
root        265  0.0  0.0   4116   568 lxc/tty2 Ss+  04:57   0:00 /sbin/mingetty
root        267  0.0  0.0   4116   572 lxc/tty3 Ss+  04:57   0:00 /sbin/mingetty
root        269  0.0  0.0   4116   568 lxc/tty4 Ss+  04:57   0:00 /sbin/mingetty
root        283  0.0  0.1 110240  1144 ?        R+   04:59   0:00 /bin/ps aux
```

For more information, see the `lxc-attach(1)` manual page.

Controlling Container Resources

Linux containers use cgroups in their implementation, and you can use the `lxc-cgroup` command to control the access that a container has to system resources relative to other containers. For example, to display the CPU cores to which a container can run on, enter the following command:

> **Note:**
>
> Run the commands in this section from the host system.

```
sudo lxc-cgroup -n ol6ctr1 cpuset.cpus

0-7
```

To restrict a container to cores 0 and 1, you would enter a command such as the following:

```
sudo lxc-cgroup -n ol6ctr1 cpuset.cpus 0,1
```

To change a container's share of CPU time and block I/O access, you would enter:

```
sudo lxc-cgroup -n ol6ctr2 cpu.shares 256
sudo lxc-cgroup -n ol6ctr2 blkio.weight 500
```

Limit a container to 256 MB of memory when the system detects memory contention or low memory; otherwise, set a hard limit of 512 MB:

```
sudo lxc-cgroup -n ol6ctr2 memory.soft_limit_in_bytes 268435456
sudo lxc-cgroup -n ol6ctr2 memory.limit_in_bytes 53687091
```

To make the changes to a container's configuration permanent, add the settings to the file `/container/name/config`, for example:

```
# Permanently tweaked resource settings
lxc.cgroup.cpu.shares=256
lxc.cgroup.blkio.weight=500
```

For more information about the resources that can be controlled, see the Linux Kernel documentation at https://www.kernel.org/doc/html/latest/.

Configuring ulimit Settings for an Oracle Linux Container

A container's `ulimit` setting honors the values of `ulimit` settings such as `memlock` and `nofile` in the container's version of `/etc/security/limits.conf/` provided that these values are lower than or equal to the values on the host system.

The values of `memlock` and `nofile` determine the maximum amount of address space in kilobytes that can be locked into memory by a user process and the maximum number of file descriptors that a user process can have open at the same time.

If you require a higher `ulimit` value for a container, increase the value of the settings in `/etc/security/limits.conf` on the host, for example:

```
#<domain>      <type>  <item>         <value>
*               soft    memlock       1048576
*               hard    memlock       2097152
*               soft    nofile        5120
*               hard    nofile        10240
```

A process can use the `ulimit` built-in shell command or the `setrlimit()` system call to raise the current limit for a shell above the soft limit. However, the new value cannot exceed the hard limit unless the process is owned by `root`.

You can use `ulimit` to set or display the current soft and hard values on the host or from inside the container.

```
echo "host: nofile = $(ulimit -n)"

host: nofile = 1024

echo "host: nofile = $(ulimit -H -n)"

host: nofile = 4096

sudo ulimit -n 2048
echo "host: nofile = $(ulimit -n)"

host: nofile = 2048

sudo lxc-attach -n ol6ctr1 -- echo "container: nofile = $(ulimit -n)"
```

```
container: nofile = 1024
```

> **Note:**
>
> Log out and log in again or, if possible, reboot the host before starting the container in a shell that uses the new soft and hard values for `ulimit`.

Configuring Kernel Parameter Settings for Oracle Linux Containers

If you specify the `--privileged` option with the `lxc-oracle` template script, you can adjust the values of certain kernel parameters for a container under the `/proc` hierarchy.

The container mounts `/proc` read-only with the following exceptions, which are writable:

- `/proc/sys/kernel/msgmax`
- `/proc/sys/kernel/msgmnb`
- `/proc/sys/kernel/msgmni`
- `/proc/sys/kernel/sem`
- `/proc/sys/kernel/shmall`
- `/proc/sys/kernel/shmmax`
- `/proc/sys/kernel/shmmni`
- `/proc/sys/net/ipv4/conf/default/accept_source_route`
- `/proc/sys/net/ipv4/conf/default/rp_filter`
- `/proc/sys/net/ipv4/ip_forward`

Each of these parameters can have a different value than that configured for the host system and for other containers running on the host system. The default value is derived from the template when you create the container. Oracle recommends that you change a setting only if an application requires a value other than the default value.

> **Note:**
>
> Prior to UEK R3 QU6, the following host-only parameters were not visible within the container due to kernel limitations:
>
> - `/proc/sys/net/core/rmem_default`
> - `/proc/sys/net/core/rmem_max`
> - `/proc/sys/net/core/wmem_default`
> - `/proc/sys/net/core/wmem_max`
> - `/proc/sys/net/ipv4/ip_local_port_range`
> - `/proc/sys/net/ipv4/tcp_syncookies`
>
> With UEK R3 QU6 and later, these parameters are read-only within the container to allow Oracle Database and other applications to be installed. You can change the values of these parameters only from the host. Any changes that you make to host-only parameters apply to all containers on the host.

Deleting Containers

To delete a container and its snapshot, use the `lxc-destroy` command on the host system as shown in the following example.

```
sudo lxc-destroy -n ol6ctr2

Delete subvolume '/container/ol6ctr2/rootfs'
```

This command also deletes the `rootfs` subvolume.

Running Application Containers

You can use the `lxc-execute` command to create a temporary application container in which you can run a command that is effectively isolated from the rest of the system. For example, the following command creates an application container named `guest` that runs `sleep` for 100 seconds.

```
sudo lxc-execute -n guest -- sleep 100
```

While the container is active, you can monitor it by running commands such as `lxc-ls --active` and `lxc-info -n guest` from another window.

```
sudo lxc-ls --active

sudo lxc-ls --active

guest

sudo lxc-info -n guest

Name:           guest
State:          RUNNING
```

```
PID:            11220
CPU use:        0.02 seconds
BlkIO use:      0 bytes
Memory use:     544.00 KiB
KMem use:       0 bytes
```

If you need to customize an application container, you can use a configuration file. For example, you might want to change the container's network configuration or the system directories that it mounts.

The following example shows settings from a sample configuration file where the `rootfs` is mostly not shared except for mount entries to ensure that `init.lxc` and certain library and binary directory paths are available.

```
lxc.utsname = guest
lxc.tty = 1
lxc.pts = 1
lxc.rootfs = /tmp/guest/rootfs
lxc.mount.entry=/usr/lib usr/lib none ro,bind 0 0
lxc.mount.entry=/usr/lib64 usr/lib64 none ro,bind 0 0
lxc.mount.entry=/usr/bin usr/bin none ro,bind 0 0
lxc.mount.entry=/usr/sbin usr/sbin none ro,bind 0 0
lxc.cgroup.cpuset.cpus=1
```

The mount entry for `/usr/sbin` is required so that the container can access `/usr/sbin/init.lxc` on the host system.

In practice, you should limit the host system directories that an application container mounts to only those directories that the container needs to run the application.

> **Note:**
>
> To avoid potential conflict with system containers, do not use the `/container` directory for application containers.

You must also configure the required directories and symbolic links under the `rootfs` directory.

```
sudo TMPDIR=/tmp/guest/rootfs
sudo mkdir -p $TMPDIR/usr/lib $TMPDIR/usr/lib64 $TMPDIR/usr/bin $TMPDIR/usr/sbin
\$TMPDIR/dev/pts $TMPDIR/dev/shm $TMPDIR/proc
sudo ln -s $TMPDIR/usr/lib $TMPDIR/lib
sudo ln -s $TMPDIR/usr/lib64 $TMPDIR/lib64
sudo ln -s $TMPDIR/usr/bin $TMPDIR/bin
sudo ln -s $TMPDIR/usr/sbin $TMPDIR/sbin
```

In this example, the directories include `/dev/pts`, `/dev/shm`, and `/proc` in addition to the mount point entries defined in the configuration file.

You can then use the `-f` option to specify the configuration file (`config`) to `lxc-execute`:

```
sudo lxc-execute -n guest -f config /usr/bin/bash
```

At the bash prompt, type the following series of commands:

```
ps -ef
```

```
UID    PID  PPID  C STIME TTY          TIME CMD
0      1       0  0 14:17 ?        00:00:00 /usr/sbin/init.lxc -- /usr/bin/bash
0      4       1  0 14:17 ?        00:00:00 /usr/bin/bash
0      5       4  0 14:17 ?        00:00:00 ps -ef

mount

/dev/sda3 on / type btrfs (rw,relatime,seclabel,space_cache)
/dev/sda3 on /usr/lib type btrfs (ro,relatime,seclabel,space_cache)
/dev/sda3 on /usr/lib64 type btrfs (ro,relatime,seclabel,space_cache)
/dev/sda3 on /usr/bin type btrfs (ro,relatime,seclabel,space_cache)
/dev/sda3 on /usr/sbin type btrfs (ro,relatime,seclabel,space_cache)
devpts on /dev/pts type devpts (rw,relatime,seclabel,gid=5,mode=620,ptmxmode=666)
proc on /proc type proc (rw,relatime)
shmfs on /dev/shm type tmpfs (rw,relatime,seclabel)
mqueue on /dev/mqueue type mqueue (rw,relatime,seclabel)

ls -l /

total 16
lrwxrwxrwx.   1 0 0  7 May 21 14:03 bin -> usr/bin
drwxr-xr-x.   1 0 0 52 May 21 14:27 dev
lrwxrwxrwx.   1 0 0  7 May 21 14:03 lib -> usr/lib
lrwxrwxrwx.   1 0 0  9 May 21 14:27 lib64 -> usr/lib64
dr-xr-xr-x. 230 0 0  0 May 21 14:27 proc
lrwxrwxrwx.   1 0 0  8 May 21 14:03 sbin -> usr/sbin
drwxr-xr-x.   1 0 0 30 May 21 12:58 usr

touch /bin/foo

touch: cannot touch '/bin/foo': Read-only file system

echo $?

1
```

In this example, running the `ps` command reveals that `bash` runs as a child of `init.lxc`. `mount` shows the individual directories that the container mounts read-only, such as `/usr/lib`, and `ls -l /` displays the symbolic links that you set up in `rootfs`.

Attempting to write to the read-only `/bin` file system results in an error. If you were to run the same `lxc-execute` command without specifying the configuration file, it would make the entire root file system of the host available to the container in read/write mode.

As for system containers, you can set `cgroup` entries in the configuration file and use the `lxc-cgroup` command to control the system resources to which an application container has access.

> **Note:**
>
> `lxc-execute` is intended to run application containers that share the host's root file system, and not to run system containers that you create using `lxc-create`. Use `lxc-start` to run system containers.

For more information, see the `lxc-execute(1)` and `lxc.conf(5)` manual pages.