Creating a full LXD server On Rocky Linux (English version)

A book from the Documentation Team

Version : 2025/05/21

Rocky Documentation Team

Table of contents

1. Licence

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

The documents and their sources are freely downloadable from:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Our media sources are hosted at github.com. You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using mkdocs. You will find instructions for generating your document here.

How can I contribute to the documentation project?

You'll find all the information you need to join us on our git project home page.

We wish you all a pleasant reading and hope you enjoy the content.

2. Creating a full LXD server

Info

This procedure should work for Rocky Linux 8.x or 9.x. If you are looking for a modern implementation of this project from the former lead developers of LXD, but available only for Rocky Linux 9.x, check out the Incus Server book.

What happened with the LXD project

Over a year ago now, the following announcement came out on the lxc-users mailing list:

Canonical, the creator and main contributor of the LXD project has decided that after over eight years as part of the Linux Containers community, the project would now be better served directly under Canonical's own set of projects.

One of the deciding factors was the resignations of some lead developers for LXD, who then went on to fork LXD into Incus, announcing the fork in August 2023. A release version (0.1) came out in October 2023, and the developers have since rapidly built on that version with step releases through 0.7 (on March 2024). On the heels of 0.7 came the long-term support version, 6.0 LTS, on April 4, 2024, and now 6.4 LTS (as of September 2024).

Throughout the process, Canonical was thought to continue maintaining links to the container images provided by Linux Containers. Still, because of a licensing change, it became impossible for Linux Containers to continue offering the container images within LXD. While Linux Containers can no longer provide container images to LXD, the LXD project has managed to build some containers, including containers for Rocky Linux.

This document uses LXD rather than Incus.

2.1 Introduction

LXD is best described on the official website, but think of it as a container system that provides the benefits of virtual servers in a container.

It is very powerful, and with the proper hardware and setup, it is possible to create many server instances on a single piece of hardware. If you pair that with a snapshot server, you also have a set of containers you can spin up almost immediately if your primary server goes down.

(You should think of this as something other than a traditional backup. You still need a regular backup system of some sort, rsnapshot, for example.)

The learning curve for LXD can be steep, but this book will attempt to give you a wealth of knowledge at your fingertips to help you deploy and use LXD on Rocky Linux.

For those wanting to use LXD as a lab environment on their notebooks or workstations, see Appendix A: Workstation Setup.

2.2 Prerequisites and assumptions

- One Rocky Linux server, nicely configured. Consider a separate hard disk for ZFS disk space (you have to do so if you are using ZFS) in a production environment. And yes, the assumption here is a bare metal server, not a VPS (Virtual Private Server).
- This is an advanced topic, but it is not too difficult to understand. If you follow these instructions from the beginning, you should be successful. That said, knowing a few basic things about container management will go a long way.
- Comfort at the command line on your machine(s) and fluent in a command line editor. (Using *vi* throughout these examples, but you can substitute in your favorite editor.)
- You must be your unprivileged user for most of these processes. For the early setup steps, you will need to be the root user or be able to `sudo` to become so. Throughout these chapters, we assume your unprivileged user to be "lxdadmin". You'll need to create this user account later in the process.
- For ZFS, please ensure that UEFI secure boot is NOT enabled. Otherwise, you will have to sign the ZFS module to get it to load.
- Using Rocky Linux-based containers for the most part

2.3 Synopsis

- **Chapter 1: Install and Configuration** deals with installing the primary server. Generally, the right way to do LXD in production is to have a primary and snapshot server.
- **Chapter 2: ZFS Setup** deals with setting up and configuring ZFS. ZFS is an open-source logical volume manager and file system created by Sun Microsystems, originally for its Solaris operating system.
- **Chapter 3: LXD Initialization and User Setup** deals with the base initialization and options, and also the set up of your unprivileged user that you will use throughout most of the rest of the process
- **Chapter 4: Firewall Setup** Has `firewalld` setup options
- **Chapter 5: Setting Up and Managing Images** describes the process for installing operating system images to a container and configuring them

- **Chapter 6: Profiles** deals with adding profiles and applying them to containers and mainly covers macvlan and its importance for IP addressing on your LAN or WAN
- **Chapter 7: Container Configuration Options** briefly covers some of the basic configuration options for containers and offers some benefits and side effects for modifying configuration options
- **Chapter 8: Container Snapshots** details the snapshot process for containers on the primary server
- **Chapter 9: The Snapshot Server** covers the setup and configuration of the snapshot server and how to create the symbiotic relationship between the primary and snapshot server
- **Chapter 10: Automating Snapshots** covers the automation of snapshot creation and populating the snapshot server with snapshots
- **Appendix A: Workstation Setup** is not technically part of the production server documents but offers a solution for people who want to build a lab of LXD containers on their personal notebook or workstation.

2.4 Conclusions

You can use these chapters to set up an enterprise-level primary and snapshot LXD server effectively. In the process, you will learn a great deal about LXD. Just be aware that there is much more to learn, and treat these documents as a starting point.

The most significant advantage of LXD is that it is economical to use on a server, allows you to spin up operating system installs quickly, and allows for many stand-alone application servers running on a single piece of hardware, leveraging that hardware for maximum use.

3. Chapter 1: Install and configuration

Throughout this chapter you will need to be the root user or you will need to be able to *sudo* to root.

3.1 Install EPEL and OpenZFS repositories

LXD requires the EPEL (Extra Packages for Enterprise Linux) repository, which is easy to install using:

```
dnf install epel-release
```

When installed, verify there are no updates:

```
dnf upgrade
```

If there were any kernel updates during the upgrade process, reboot the server.

3.1.1 OpenZFS repository for 8 and 9

Install the OpenZFS repository with:

```
dnf install https://zfsonlinux.org/epel/zfs-release-2-2$(rpm --eval "%
{dist}").noarch.rpm
```

3.2 Install `snapd`, `dkms`, `vim`, and `kernel-devel`

LXD installation requires a snap package on Rocky Linux. For this reason, you need to install `snapd` (and a few other useful programs) with:

```
dnf install snapd dkms vim kernel-devel
```

Now enable and start snapd:

```
systemctl enable snapd
```

Then run:

```
systemctl start snapd
```

Reboot the server before continuing here.

3.3 Install LXD

Installing LXD requires the use of the snap command. At this point, you are just installing it, you are not doing the set up:

```
snap install lxd
```

3.4 Install OpenZFS

```
dnf install zfs
```

3.5 Environment set up

Most server kernel settings are not sufficient to run a large number of containers. If you assume from the beginning that you will use your server in production, you need to make these changes up front to avoid errors such as "Too many open files" from occurring.

Luckily, tweaking the settings for LXD is not hard with a few file modifications and a reboot.

3.5.1 Modifying `limits.conf`

The first file you need to change is the `limits.conf` file. This file is self-documented. Examine the explanations in the comment in the file to understand what this file does. To make your modifications enter:

```
vi /etc/security/limits.conf
```

This entire file consists of comments, and at the bottom, shows the current default settings. In the blank space above the end of file marker (#End of file) you need to add our custom settings. The end of the file will look like this when completed:

```
# Modifications made for LXD

*               soft    nofile          1048576
*               hard    nofile          1048576
root            soft    nofile          1048576
root            hard    nofile          1048576
*               soft    memlock         unlimited
*               hard    memlock         unlimited
```

Save your changes and exit (⇧ Shift + : + w + q + ! for *vi*).

3.5.2 Modifying sysctl.conf with `90-lxd.override.conf`

With *systemd*, you can make changes to your system's overall configuration and kernel options *without* modifying the main configuration file. Instead, put your settings in a separate file that will override the particular settings you need.

To make these kernel changes, you are going to create a file called `90-lxd-override.conf` in `/etc/sysctl.d`. To do this type:

```
vi /etc/sysctl.d/90-lxd-override.conf
```

> ⚠ **RL 9 and MAX value of `net.core.bpf_jit_limit`**
>
> Because of recent kernel security updates, the max value of `net.core.bpf_jit_limit` appears to be 1000000000. Please adjust this value in the self-documenting file below if you are running Rocky Linux 9.x. If you set it above this limit **OR** if you fail to set it at all, it will default to the system default of 264241152, which may not be enough if you run a large number of containers.

Place the following content in that file. Note that if you are wondering what you are doing here, the file content is self-documenting:

```
## The following changes have been made for LXD ##

# fs.inotify.max_queued_events specifies an upper limit on the number of events
that can be queued to the corresponding inotify instance
 - (default is 16384)

fs.inotify.max_queued_events = 1048576
```

```
# fs.inotify.max_user_instances This specifies an upper limit on the number of
inotify instances that can be created per real user ID -
(default value is 128)

fs.inotify.max_user_instances = 1048576

# fs.inotify.max_user_watches specifies an upper limit on the number of watches
that can be created per real user ID - (default is 8192)

fs.inotify.max_user_watches = 1048576

# vm.max_map_count contains the maximum number of memory map areas a process
may have. Memory map areas are used as a side-effect of cal
ling malloc, directly by mmap and mprotect, and also when loading shared
libraries - (default is 65530)

vm.max_map_count = 262144

# kernel.dmesg_restrict denies container access to the messages in the kernel
ring buffer. Please note that this also will deny access t
o non-root users on the host system - (default is 0)

kernel.dmesg_restrict = 1

# This is the maximum number of entries in ARP table (IPv4). You should
increase this if you create over 1024 containers.

net.ipv4.neigh.default.gc_thresh3 = 8192

# This is the maximum number of entries in ARP table (IPv6). You should
increase this if you plan to create over 1024 containers.Not nee
ded if not using IPv6, but...

net.ipv6.neigh.default.gc_thresh3 = 8192

# This is a limit on the size of eBPF JIT allocations which is usually set to
PAGE_SIZE * 40000. Set this to 1000000000 if you are running Rocky Linux 9.x

net.core.bpf_jit_limit = 3000000000

# This is the maximum number of keys a non-root user can use, should be higher
than the number of containers

kernel.keys.maxkeys = 2000

# This is the maximum size of the keyring non-root users can use

kernel.keys.maxbytes = 2000000
```

```
# This is the maximum number of concurrent async I/O operations. You might need
to increase it further if you have a lot of workloads th
at use the AIO subsystem (e.g. MySQL)

fs.aio-max-nr = 524288
```

Save your changes and exit.

At this point reboot the server.

3.5.3 Checking *sysctl.conf* values

After the reboot, log back in as the root user to the server. You need to check that our override file has actually completed the job.

This is not hard to do. There's no need to verify every setting unless you want to, but checking a few will verify that the settings have changed. Do this with the `sysctl` command:

```
sysctl net.core.bpf_jit_limit
```

Which will show you:

```
net.core.bpf_jit_limit = 3000000000
```

Do the same with a few other settings in the override file to verify the changes.

4. Chapter 2: ZFS setup

Throughout this chapter you will need to be the root user or able to `sudo` to become root.

If you have already installed ZFS, this section will walk you through ZFS setup.

4.1 Enabling ZFS and setting up the pool

First, enter this command:

```
/sbin/modprobe zfs
```

If there are no errors, it will return to the prompt and echo nothing. If you get an error, stop now and begin troubleshooting. Again, ensure that secure boot is off. That will be the most likely culprit.

Next you need to examine the disks on our system, find out where the operating system is, and what is available to use for the ZFS pool. You will do this with `lsblk`:

```
lsblk
```

Which will return something like this (your system will be different!):

```
AME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0  32.3M  1 loop /var/lib/snapd/snap/snapd/11588
loop1    7:1    0  55.5M  1 loop /var/lib/snapd/snap/core18/1997
loop2    7:2    0  68.8M  1 loop /var/lib/snapd/snap/lxd/20037
sda      8:0    0 119.2G  0 disk
├─sda1   8:1    0   600M  0 part /boot/efi
├─sda2   8:2    0     1G  0 part /boot
├─sda3   8:3    0  11.9G  0 part [SWAP]
├─sda4   8:4    0     2G  0 part /home
└─sda5   8:5    0 103.7G  0 part /
sdb      8:16   0 119.2G  0 disk
├─sdb1   8:17   0 119.2G  0 part
└─sdb9   8:25   0     8M  0 part
sdc      8:32   0 149.1G  0 disk
└─sdc1   8:33   0 149.1G  0 part
```

In this listing, you can see that */dev/sda* is in use by the operating system. You are going to use */dev/sdb* for our zpool. Note that if you have many available hard drives, you may want to consider using raidz (a software raid specifically for ZFS).

That falls outside the scope of this document, but definitely is a consideration for production. It offers better performance and redundancy. For now, create your pool on the single device you have identified:

```
zpool create storage /dev/sdb
```

What this says is to create a pool called "storage" that is ZFS on the device */dev/sdb*.

After creating the pool, reboot the server again.

5. Chapter 3: LXD initialization and user setup

Throughout this chapter you will need to be root or able to `sudo` to become root. In addition, the assumption is that you have setup a ZFS storage pool described in Chapter 2. You can use a different storage pool if you have chosen not to use ZFS, but you will need to make adjustments to the initialization questions and answers.

5.1 LXD initialization

Your server environment is all set up. You are ready to initialize LXD. This is an automated script that asks a series of questions to get your LXD instance up and running:

```
lxd init
```

Here are the questions and our answers for the script, with a little explanation where warranted:

```
Would you like to use LXD clustering? (yes/no) [default=no]:
```

If interested in clustering, do some additional research on that here

```
Do you want to configure a new storage pool? (yes/no) [default=yes]:
```

This seems counter-intuitive. You have already created your ZFS pool, but it will become clear in a later question. Accept the default.

```
Name of the new storage pool [default=default]: storage
```

Leaving this "default" is an option, but for clarity, using the same name you gave our ZFS pool is better.

```
Name of the storage backend to use (btrfs, dir, lvm, zfs, ceph) [default=zfs]:
```

You want to accept the default.

```
Create a new ZFS pool? (yes/no) [default=yes]: no
```

Here is where the resolution of the earlier question about creating a storage pool comes into play.

```
Name of the existing ZFS pool or dataset: storage
Would you like to connect to a MAAS server? (yes/no) [default=no]:
```

Metal As A Service (MAAS) is outside the scope of this document.

```
Would you like to create a new local network bridge? (yes/no) [default=yes]:
What should the new bridge be called? [default=lxdbr0]:
What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]:
What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]: none
```

If you want to use IPv6 on your LXD containers, you can turn on this option. That is up to you.

```
Would you like the LXD server to be available over the network? (yes/no)
[default=no]: yes
```

This is necessary to snapshot the server.

```
Address to bind LXD to (not including port) [default=all]:
Port to bind LXD to [default=8443]:
Trust password for new clients:
Again:
```

This trust password is how you will connect to the snapshot server or back from the snapshot server. Set this with something that makes sense in your environment. Save this entry to a secure location, such as a password manager.

```
Would you like stale cached images to be updated automatically? (yes/no)
[default=yes]
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

5.2 Setting up user privileges

Before you continue on, you need to create your "lxdadmin" user and ensure that it has the privileges it needs. You need the "lxdadmin" user to be able to `sudo` to root and you need it to be a member of the lxd group. To add the user and ensure it is a member of both groups do:

```
useradd -G wheel,lxd lxdadmin
```

Set the password:

```
passwd lxdadmin
```

As with the other passwords, save this to a secure location.

6. Chapter 4: firewall setup

Throughout this chapter you will need to be root or able to `sudo` to become root.

As with any server, you need to ensure that it is secure from the outside world and on your LAN. Your example server only has a LAN interface, but it is totally possible to have two interfaces, one each facing your LAN and WAN networks.

> **A note regarding Rocky Linux 9.x and `iptables`**
>
> Starting with Rocky Linux 9.0, `iptables` and all of the associated utilities are officially deprecated. This means that in future versions of the operating system, they will disappear altogether. A previous version of this document contained instructions for `iptables` set up, but it has now been removed.
>
> For all current versions of Rocky Linux, using `firewalld` is recommended.

6.1 Firewall set up - `firewalld`

For *firewalld* rules, you need to use this basic procedure or be familiar with those concepts. Our assumptions are: LAN network of 192.168.1.0/24 and a bridge named lxdbr0. To be clear, you might have many interfaces on your LXD server, with one perhaps facing your WAN. You are also going to create a zone for the bridged and local networks. This is just for zone clarity's sake. The other zone names do not really apply. This procedure assumes that you already know the basics of *firewalld*.

```
firewall-cmd --new-zone=bridge --permanent
```

You need to reload the firewall after adding a zone:

```
firewall-cmd --reload
```

You want to allow all traffic from the bridge. Just add the interface, and change the target from "default" to "ACCEPT":

> **Warning**
>
> Changing the target of a `firewalld` zone *must* be done with the `--permanent` option, so we might as well just enter that flag in our other commands as well and forgo the `--runtime-to-permanent` option.

Note

If you need to create a zone that you want to allow all access to the interface or source, but do not want to have to specify any protocols or services, then you *must* change the target from "default" to "ACCEPT". The same is true of "DROP" and "REJECT" for a particular IP block that you have custom zones for. To be clear, the "drop" zone will take care of that for you as long as you are not using a custom zone.

```
firewall-cmd --zone=bridge --add-interface=lxdbr0 --permanent
firewall-cmd --zone=bridge --set-target=ACCEPT --permanent
```

Assuming no errors and everything is still working just do a reload:

```
firewall-cmd --reload
```

If you list out your rules now with `firewall-cmd --zone=bridge --list-all` you will see:

```
bridge (active)
  target: ACCEPT
  icmp-block-inversion: no
  interfaces: lxdbr0
  sources:
  services:
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Note that you also want to allow your local interface. Again, the included zones are not appropriately named for this. Create a zone and use the source IP range for the local interface to ensure you have access:

```
firewall-cmd --new-zone=local --permanent
firewall-cmd --reload
```

Add the source IPs for the local interface, and change the target to "ACCEPT":

```
firewall-cmd --zone=local --add-source=127.0.0.1/8 --permanent
firewall-cmd --zone=local --set-target=ACCEPT --permanent
firewall-cmd --reload
```

Go ahead and list out the "local" zone to ensure your rules are there with `firewall-cmd --zone=local --list all` which will show:

```
local (active)
  target: ACCEPT
  icmp-block-inversion: no
  interfaces:
  sources: 127.0.0.1/8
  services:
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

You want to allow SSH from our trusted network. We will use the source IPs here, and the built-in "trusted" zone. The target for this zone is already "ACCEPT" by default.

```
firewall-cmd --zone=trusted --add-source=192.168.1.0/24
```

Add the service to the zone:

```
firewall-cmd --zone=trusted --add-service=ssh
```

If everything is working, move your rules to permanent and reload the rules:

```
firewall-cmd --runtime-to-permanent
firewall-cmd --reload
```

Listing out your "trusted" zone will show:

```
trusted (active)
  target: ACCEPT
```

```
  icmp-block-inversion: no
  interfaces:
  sources: 192.168.1.0/24
  services: ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

By default, the "public" zone is in the enabled state and has SSH allowed. For security, you do not want SSH allowed on the "public" zone. Ensure that your zones are correct and that the access you are getting to the server is by one of the LAN IPs (in the case of our example). You might lock yourself out of the server if you do not verify this before continuing. When you are sure you have access from the correct interface, remove SSH from the "public" zone:

```
firewall-cmd --zone=public --remove-service=ssh
```

Test access and ensure you are not locked out. If not, move your rules to permanent, reload, and list out zone "public" to ensure the removal of SSH:

```
firewall-cmd --runtime-to-permanent
firewall-cmd --reload
firewall-cmd --zone=public --list-all
```

There may be other interfaces on your server to consider. You can use built-in zones where appropriate, but if the names do not appear logical, you can definitely add zones. Just remember that if you have no services or protocols that you need to allow or reject specifically, you will need to change the zone target. If it works to use interfaces, as with the bridge, you can do that. If you need more granular access to services, uses source IPs instead.

7. Chapter 5: Setting up and managing images

Throughout this chapter you will need to run commands as your unprivileged user ("lxdadmin" if you have been following this book from the beginning).

7.1 List available images

You probably can not wait to get started with a container. There are many container operating system possibilities. To get a feel for how many possibilities, enter this command:

```
lxc image list images: | more
```

Enter the space bar to page through the list. This list of containers and virtual machines continues to grow.

The **last** thing you want to do is to page through looking for a container image to install, particularly if you know the image that you want to create. Change the command to show only Rocky Linux install options:

```
lxc image list images: | grep rocky
```

This brings up a much more manageable list:

```
| rockylinux/8 (3 more)                    | 0ed2f148f7c6 | yes    |
Rockylinux 8 amd64 (20220805_02:06)          | x86_64       | CONTAINER
| 128.68MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/8 (3 more)                    | 6411a033fdf1 | yes    |
Rockylinux 8 amd64 (20220805_02:06)          | x86_64       | VIRTUAL-MACHINE
| 643.15MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/8/arm64 (1 more)              | e677777306cf | yes    |
Rockylinux 8 arm64 (20220805_02:29)          | aarch64      | CONTAINER
| 124.06MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/8/cloud (1 more)              | 3d2fe303afd3 | yes    |
Rockylinux 8 amd64 (20220805_02:06)          | x86_64       | CONTAINER
| 147.04MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/8/cloud (1 more)              | 7b37619bf333 | yes    |
Rockylinux 8 amd64 (20220805_02:06)          | x86_64       | VIRTUAL-MACHINE
| 659.58MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/8/cloud/arm64                 | 21c930b2ce7d | yes    |
```

```
Rockylinux 8 arm64 (20220805_02:06)          | aarch64      | CONTAINER
| 143.17MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9 (3 more)                     | 61b0171b7eca | yes    |
Rockylinux 9 amd64 (20220805_02:07)          | x86_64       | VIRTUAL-MACHINE
| 526.38MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9 (3 more)                     | e7738a0e2923 | yes    |
Rockylinux 9 amd64 (20220805_02:07)          | x86_64       | CONTAINER
| 107.80MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9/arm64 (1 more)               | 917b92a54032 | yes    |
Rockylinux 9 arm64 (20220805_02:06)          | aarch64      | CONTAINER
| 103.81MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9/cloud (1 more)               | 16d3f18f2abb | yes    |
Rockylinux 9 amd64 (20220805_02:06)          | x86_64       | CONTAINER
| 123.52MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9/cloud (1 more)               | 605eadf1c512 | yes    |
Rockylinux 9 amd64 (20220805_02:06)          | x86_64       | VIRTUAL-MACHINE
| 547.39MB  | Aug 5, 2022 at 12:00am (UTC)  |
| rockylinux/9/cloud/arm64                  | db3ce70718e3 | yes    |
Rockylinux 9 arm64 (20220805_02:06)          | aarch64      | CONTAINER
| 119.27MB  | Aug 5, 2022 at 12:00am (UTC)  |
```

7.2 Installing, renaming, and listing images

For the first container, you are going to use "rockylinux/8". To install it, you *might* use:

```
lxc launch images:rockylinux/8 rockylinux-test-8
```

That will create a Rocky Linux-based container named "rockylinux-test-8". You can rename a container after creating it, but you first need to stop the container, which starts automatically when created.

To start the container manually, use:

```
lxc start rockylinux-test-8
```

To Rename the image (we are not going to do this here, but this is how to do it) first stop the container:

```
lxc stop rockylinux-test-8
```

Use the `move` command to change the container's name:

```
lxc move rockylinux-test-8 rockylinux-8
```

If you followed this instruction anyway, stop the container and move it back to the original name to continue to follow along.

For the purposes of this guide, go ahead and install two more images for now:

```
lxc launch images:rockylinux/9 rockylinux-test-9
```

and

```
lxc launch images:ubuntu/22.04 ubuntu-test
```

Examine what you have by listing your images:

```
lxc list
```

which will return this:

```
+-------------------+---------+----------------------+------+-----------
+-----------+
|       NAME        |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 10.146.84.179 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 10.146.84.180 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 10.146.84.181 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
```

8. Chapter 6: Profiles

Throughout this chapter you will need to run commands as your unprivileged user ("lxdadmin" if you've been following from the beginning in this book).

You get a default profile when you install LXD, and this profile cannot be removed or modified. That said, you can use the default profile to create new profiles to use with your containers.

If you examine your container listing, you will notice that the IP address in each case is from the bridged interface. In a production environment, you may want to use something else. This might be a DHCP assigned address from your LAN interface or even a statically assigned address from your WAN.

If you configure your LXD server with two interfaces and assign each an IP on your WAN and LAN, it is possible to assign your container's IP addresses based on the interface the container needs to be facing.

As of version 9.0 of Rocky Linux (and really any bug for bug copy of Red Hat Enterprise Linux) the method for assigning IP addresses statically or dynamically with the profiles does not work.

There are ways to get around this, but it is annoying. This appears to have something to do with changes made to Network Manager that affect macvlan. macvlan allows you to create many interfaces with different Layer 2 addresses.

For now, just be aware that this has drawbacks when choosing container images based on RHEL.

8.1 Creating A macvlan profile and assigning it

To create our macvlan profile, use this command:

```
lxc profile create macvlan
```

If you were on a multi-interface machine and wanted more than one macvlan template based on the network you wanted to reach, you might use "lanmacvlan"

or "wanmacvlan" or any other name that you wanted to use to identify the profile. Using "macvlan" in our profile create statement is totally up to you.

You want to change the macvlan interface, but before you do, you need to know what the parent interface is for our LXD server. This will be the interface that has a LAN (in this case) assigned IP. To find what interface that is, use:

```
ip addr
```

Look for the interface with the LAN IP assignment in the 192.168.1.0/24 network:

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 40:16:7e:a9:94:85 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.106/24 brd 192.168.1.255 scope global dynamic noprefixroute
enp3s0
       valid_lft 4040sec preferred_lft 4040sec
    inet6 fe80::a308:acfb:fcb3:878f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

In this case, the interface is "enp3s0".

Next change the profile:

```
lxc profile device add macvlan eth0 nic nictype=macvlan parent=enp3s0
```

This command adds all of the necessary parameters to the macvlan profile required for use.

Examine what this command created, by using the command:

```
lxc profile show macvlan
```

Which will give you output similar to this:

```
config: {}
description: ""
devices:
  eth0:
    nictype: macvlan
    parent: enp3s0
```

```
    type: nic
name: macvlan
used_by: []
```

You can use profiles for many other things, but assigning a static IP to a container, or by using your own DHCP server are common needs.

To assign the macvlan profile to rockylinux-test-8 you need to do the following:

```
lxc profile assign rockylinux-test-8 default,macvlan
```

Do the same thing for rockylinux-test-9:

```
lxc profile assign rockylinux-test-9 default,macvlan
```

This says, you want the default profile, and to apply the macvlan profile too.

8.2 Rocky Linux macvlan

In RHEL distributions and clones, Network Manager has been in a constant state of change. Because of this, the way the `macvlan` profile works does not work (at least in comparison to other distributions), and requires a little additional work to assign IP addresses from DHCP or statically.

Remember that none of this has anything to do with Rocky Linux particularly, but with the upstream package implementation.

If you want to run Rocky Linux containers and use macvlan to assign an IP address from your LAN or WAN networks, the process is different based on the container version of the operating system (8.x or 9.x).

8.2.1 Rocky Linux 9.x macvlan - the DHCP fix

First, let us illustrate what happens when stopping and restarting the two containers after assigning the macvlan profile.

Having the profile assigned, however, does not change the default configuration, which is DHCP by default.

To test this, do the following:

```
lxc restart rocky-test-8
lxc restart rocky-test-9
```

List your containers again and note that the rockylinux-test-9 does not have an IP address anymore:

```
lxc list
```

```
+-------------------+---------+----------------------+------+-----------
+-----------+
|       NAME        |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING |                      |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 10.146.84.181 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
```

As you can see, our Rocky Linux 8.x container received the IP address from the LAN interface, whereas the Rocky Linux 9.x container did not.

To further demonstrate the problem here, you need to run `dhclient` on the Rocky Linux 9.0 container. This will show us that the macvlan profile, *is* in fact applied:

```
lxc exec rockylinux-test-9 dhclient
```

Another container listing now shows the following:

```
+-------------------+---------+----------------------+------+-----------
+-----------+
|       NAME        |  STATE  |         IPV4         | IPV6 |   TYPE    |
```

```
SNAPSHOTS |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 192.168.1.113 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 10.146.84.181 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
```

That should have happened with a stop and start of the container, but it does not. Assuming that you want to use a DHCP assigned IP address every time, you can fix this with a simple crontab entry. To do this, we need to gain shell access to the container by entering:

```
lxc exec rockylinux-test-9 bash
```

Next, lets determine the path to `dhclient`. To do this, because this container is from a minimal image, you will need to first install `which`:

```
dnf install which
```

then run:

```
which dhclient
```

which will return:

```
/usr/sbin/dhclient
```

Next, change root's crontab:

```
crontab -e
```

Add this line:

```
@reboot    /usr/sbin/dhclient
```

The crontab command entered uses *vi* . To save your changes and exit use ⇧ Shift + : + w + q.

Exit the container and restart rockylinux-test-9:

```
lxc restart rockylinux-test-9
```

Another listing will reveal that the container has the DHCP address assigned:

```
+-------------------+---------+----------------------+------+-----------
+-----------+
|       NAME        |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 192.168.1.113 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 10.146.84.181 (eth0) |      | CONTAINER |
0         |
+-------------------+---------+----------------------+------+-----------
+-----------+
```

8.2.2 Rocky Linux 9.x macvlan - The static IP fix

To statically assign an IP address, things get even more convoluted. Since `network-scripts` is now deprecated in Rocky Linux 9.x, the only way to do this is through static assignment, and because of the way the containers use the network, you are not going to be able to set the route with a normal `ip route` statement. The problem turns out to be that the interface assigned when applying the macvlan profile (eth0 in this case), is not manageable with Network Manager. The fix is to rename the network interface on the container after restart and assign the static IP.

You can do this with a script and run (again) within root's crontab. Do this with the `ip` command.

To do this, you need to gain shell access to the container again:

```
lxc exec rockylinux-test-9 bash
```

Next, you are going to create a bash script in `/usr/local/sbin` called "static":

```
vi /usr/local/sbin/static
```

The contents of this script are not difficult:

```
#!/usr/bin/env bash

/usr/sbin/ip link set dev eth0 name net0
/usr/sbin/ip addr add 192.168.1.151/24 dev net0
/usr/sbin/ip link set dev net0 up
/usr/sbin/ip route add default via 192.168.1.1
```

What are we doing here?

- you rename eth0 to a new name that we can manage ("net0")
- you assign the new static IP that we have allocated for our container (192.168.1.151)
- you bring up the new "net0" interface
- you need to add the default route for our interface

Make our script executable with:

```
chmod +x /usr/local/sbin/static
```

Add this to root's crontab for the container with the @reboot time:

```
@reboot     /usr/local/sbin/static
```

Finally, exit the container and restart it:

```
lxc restart rockylinux-test-9
```

Wait a few seconds and list out the containers again:

```
lxc list
```

You should see success:

```
+------------------+---------+----------------------+------+-----------
+-----------+
|       NAME       |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 192.168.1.151 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 10.146.84.181 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
```

8.3 Ubuntu macvlan

Luckily, In Ubuntu's implementation of Network Manager, the macvlan stack is NOT broken. It is much easier to deploy!

Just like with your rockylinux-test-9 container, you need to assign the profile to our container:

```
lxc profile assign ubuntu-test default,macvlan
```

To find out if DHCP assigns an address to the container stop and start the container again:

```
lxc restart ubuntu-test
```

List the containers again:

```
+------------------+---------+----------------------+------+-----------
+-----------+
|       NAME       |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 192.168.1.151 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 192.168.1.132 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
```

Success!

Configuring the Static IP is just a little different, but not at all hard. You need to change the .yaml file associated with the container's connection (`10-lxc.yaml`). For this static IP, you will use 192.168.1.201:

```
vi /etc/netplan/10-lxc.yaml
```

Change what is there to the following:

```
network:
  version: 2
  ethernets:
    eth0:
      dhcp4: false
      addresses: [192.168.1.201/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8,8.8.4.4]
```

Save your changes and exit the container.

Restart the container:

```
lxc restart ubuntu-test
```

When you list your containers again, you will see your static IP:

```
+------------------+---------+----------------------+------+-----------
+-----------+
|       NAME       |  STATE  |         IPV4         | IPV6 |   TYPE    |
SNAPSHOTS |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-8 | RUNNING | 192.168.1.114 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| rockylinux-test-9 | RUNNING | 192.168.1.151 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
| ubuntu-test       | RUNNING | 192.168.1.201 (eth0) |      | CONTAINER |
0         |
+------------------+---------+----------------------+------+-----------
+-----------+
```

Success!

In the examples used in this chapter, a hard container to configure was intentionally chosen, and two less difficult ones. There are many more versions of Linux available in the image listing. If you have a favorite, try installing it, assigning the macvlan template, and setting IPs.

9. Chapter 7: Container configuration options

Throughout this chapter you will need to run commands as your unprivileged user ("lxdadmin" if you've been following from the beginning of this book).

There are a wealth of options for configuring the container after installation. Before seeing those, however, let us examine the `info` command for a container. In this example, you will use the ubuntu-test container:

```
lxc info ubuntu-test
```

This will show the following:

```
Name: ubuntu-test
Location: none
Remote: unix://
Architecture: x86_64
Created: 2021/04/26 15:14 UTC
Status: Running
Type: container
Profiles: default, macvlan
Pid: 584710
Ips:
  eth0:    inet    192.168.1.201    enp3s0
  eth0:    inet6    fe80::216:3eff:fe10:6d6d    enp3s0
  lo:    inet    127.0.0.1
  lo:    inet6    ::1
Resources:
  Processes: 13
  Disk usage:
    root: 85.30MB
  CPU usage:
    CPU usage (in seconds): 1
  Memory usage:
    Memory (current): 99.16MB
    Memory (peak): 110.90MB
  Network usage:
    eth0:
      Bytes received: 53.56kB
      Bytes sent: 2.66kB
      Packets received: 876
      Packets sent: 36
    lo:
      Bytes received: 0B
```

```
        Bytes sent: 0B
        Packets received: 0
        Packets sent: 0
```

There is much good information there, from the profiles applied, to the memory in use, disk space in use, and more.

9.1 A word about configuration and some options

By default, LXD will assign the required system memory, disk space, CPU cores, and other resources, to the container. But what if you want to be more specific? That is totally possible.

There are trade-offs to doing this, though. For instance, if you assign system memory and the container does not use it all, you have kept it from another container that might actually need it. The reverse, though, can happen. If a wants to use more than its share of memory, it can keep other containers from getting enough, thereby pinching their performance.

Just remember that every action you make to configure a container *can* have negative effects somewhere else.

Rather than run through all of the options for configuration, use the tab auto-complete to see the options available:

```
lxc config set ubuntu-test
```

and Tab ⇥.

This shows you all of the options for configuring a container. If you have questions about what one of the configuration options does, head to the official documentation for LXD and do a search for the configuration parameter, or Google the entire string, such as `lxc config set limits.memory` and examine the results of the search.

Here we examine a few of the most used configuration options. For example, if you want to set the max amount of memory that a container can use:

```
lxc config set ubuntu-test limits.memory 2GB
```

That says that if the memory is available to use, for example there is 2GB of memory available, then the container can actually use more than 2GB if it is available. It is a soft limit, for example.

```
lxc config set ubuntu-test limits.memory.enforce 2GB
```

That says that the container can never use more than 2GB of memory, whether it is currently available or not. In this case it is a hard limit.

```
lxc config set ubuntu-test limits.cpu 2
```

That says to limit the number of CPU cores that the container can use to 2.

> **Note**
>
> When this document was rewritten for Rocky Linux 9.0, the ZFS repository for 9 was not available. For this reason all of our test containers were built using "dir" in the init process. That is why the example below shows a "dir" instead of "zfs" storage pool.

Remember when you set up our storage pool in the ZFS chapter? You named the pool "storage," but you could have named it anything. If you want to examine this, you can use this command, which works equally well for any of the other pool types too (as shown for dir):

```
lxc storage show storage
```

This shows the following:

```
config:
  source: /var/snap/lxd/common/lxd/storage-pools/storage
description: ""
name: storage
driver: dir
used_by:
- /1.0/instances/rockylinux-test-8
- /1.0/instances/rockylinux-test-9
- /1.0/instances/ubuntu-test
- /1.0/profiles/default
status: Created
locations:
- none
```

This shows that all of our containers use our dir storage pool. When using ZFS, you can also set a disk quota on a container. Here is what that command looks like, setting a 2GB disk quota on the ubuntu-test container:

```
lxc config device override ubuntu-test root size=2GB
```

As stated earlier, use configuration options sparingly, unless you have got a container that wants to use way more than its share of resources. LXD, for the most part, will manage the environment well on its own.

Many more options exist that might be of interest to some people. Doing your own research will help you to find out if any of those are of value in your environment.

10. Chapter 8: container snapshots

Throughout this chapter you will need to run commands as your unprivileged user ("lxdadmin" if you've been following along from the beginning of this book).

Container snapshots, along with a snapshot server (more on that later), are probably the most important aspect of running a production LXD server. Snapshots ensure quick recovery. It is a good idea to use them as a fail safe when updating the primary software that runs on a particular container. If something happens during the update that breaks that application, you just restore the snapshot and you are back up and running with only a few seconds worth of downtime.

The author used LXD containers for PowerDNS public facing servers, and the process of updating those applications became less worrisome, thanks to taking snapshots before every update.

You can even snapshot a container when it is running.

10.1 The snapshot process

Start by getting a snapshot of the ubuntu-test container by using this command:

```
lxc snapshot ubuntu-test ubuntu-test-1
```

Here, you are calling the snapshot "ubuntu-test-1", but you can call it anything. To ensure that you have the snapshot, do an `lxc info` of the container:

```
lxc info ubuntu-test
```

You have looked at an info screen already. If you scroll to the bottom, you now see:

```
Snapshots:
  ubuntu-test-1 (taken at 2021/04/29 15:57 UTC) (stateless)
```

Success! Our snapshot is in place.

Get into the ubuntu-test container:

```
lxc exec ubuntu-test bash
```

Create an empty file with the *touch* command:

```
touch this_file.txt
```

Exit the container.

Before restoring the container how it was prior to creating the file, the safest way to restore a container, particularly if there have been many changes, is to stop it first:

```
lxc stop ubuntu-test
```

Restore it:

```
lxc restore ubuntu-test ubuntu-test-1
```

Start the container again:

```
lxc start ubuntu-test
```

If you get back into the container again and look, our "this_file.txt" that you created is now gone.

When you do not need a snapshot anymore you can delete it:

```
lxc delete ubuntu-test/ubuntu-test-1
```

Warning

You should always delete snapshots with the container running. Why? Well the *lxc delete* command also works to delete the entire container. If we had accidentally hit enter after "ubuntu-test" in the command above, AND, if the container was stopped, the container would be deleted. No warning is given, it simply does what you ask.

If the container is running, however, you will get this message:

```
Error: The instance is currently running, stop it first or pass --force
```

So always delete snapshots with the container running.

In the chapters that follow you will:

- set up the process of creating snapshots automatically
- set up expiration of a snapshot so that it goes away after a certain length of time
- set up auto refreshing of snapshots to the snapshot server

11. Chapter 9: snapshot server

This chapter uses a combination of the privileged (root) user, and the unprivileged (lxdadmin) user, based on the tasks you are executing.

As noted at the beginning, the snapshot server for LXD must be a mirror of the production server in every way possible. The reason is that you might need to take it to production if the hardware fails on your primary server, and having backups, and a quick way to restart production containers, keeps those system administrator panic telephone calls and text messages to a minimum. THAT is ALWAYS good!

The process of building the snapshot server is exactly like the production server. To fully emulate our production server set up, do all of **Chapters 1-4** again on the snapshot server, and when completed, return to this spot.

You are back!! Congratulations, this must mean that you have successfully completed the basic installation for the snapshot server.

11.1 Setting up the primary and snapshot server relationship

Some housekeeping is necessary before continuing. First, if you are running in a production environment, you probably have access to a DNS server that you can use for setting up IP to name resolution.

In our lab, we do not have that luxury. Perhaps you've got the same scenario running. For this reason, you are going to add both servers IP addresses and names to the /etc/hosts file on the primary and the snapshot server. You'll need to do this as your root (or *sudo*) user.

In our lab, the primary LXD server is running on 192.168.1.106 and the snapshot LXD server is running on 192.168.1.141. SSH into each server and add the following to the /etc/hosts file:

```
192.168.1.106 lxd-primary
192.168.1.141 lxd-snapshot
```

Next, you need to allow all traffic between the two servers. To do this, you are going to change the `firewalld` rules. First, on the lxd-primary server, add this line:

```
firewall-cmd zone=trusted add-source=192.168.1.141 --permanent
```

and on the snapshot server, add this rule:

```
firewall-cmd zone=trusted add-source=192.168.1.106 --permanent
```

then reload:

```
firewall-cmd reload
```

Next, as our unprivileged (lxdadmin) user, you need to set the trust relationship between the two machines. This is done by running the following on lxd-primary:

```
lxc remote add lxd-snapshot
```

This displays the certificate to accept. Accept it, and it will prompt for your password. This is the "trust password" that you set up when doing the LXD initialization step. Hopefully, you are securely keeping track of all of these passwords. When you enter the password, you will receive this:

```
Client certificate stored at server:  lxd-snapshot
```

It does not hurt to have this in reverse also. For example, set the trust relationship on the lxd-snapshot server too. That way, if needed, the lxd-snapshot server can send snapshots back to the lxd-primary server. Repeat the steps and substitute in "lxd-primary" for "lxd-snapshot."

11.1.1 Migrating your first snapshot

Before you can migrate your first snapshot, you need to have any profiles created on lxd-snapshot that you have created on the lxd-primary. In our case, this is the "macvlan" profile.

You will need to create this for lxd-snapshot. Go back to Chapter 6 and create the "macvlan" profile on lxd-snapshot if you need to. If your two servers have the same parent interface names ("enp3s0" for example) then you can copy the "macvlan" profile over to lxd-snapshot without recreating it:

```
lxc profile copy macvlan lxd-snapshot
```

With all of the relationships and profiles set up, the next step is to actually send a snapshot from lxd-primary over to lxd-snapshot. If you have been following along exactly, you have probably deleted all of your snapshots. Create another snapshot:

```
lxc snapshot rockylinux-test-9 rockylinux-test-9-snap1
```

If you run the "info" command for `lxc`, you can see the snapshot at the bottom of our listing:

```
lxc info rockylinux-test-9
```

Which will show something like this at the bottom:

```
rockylinux-test-9-snap1 at 2021/05/13 16:34 UTC) (stateless)
```

OK, fingers crossed! Let us try to migrate our snapshot:

```
lxc copy rockylinux-test-9/rockylinux-test-9-snap1 lxd-snapshot:rockylinux-
test-9
```

This command says, within the container rockylinux-test-9, you want to send the snapshot, rockylinux-test-9-snap1 over to lxd-snapshot and name it rockylinux-test-9.

After a short time, the copy will be complete. Want to find out for sure? Do an `lxc list` on the lxd-snapshot server. Which should return the following:

```
+------------------+---------+------+------+-----------+-----------+
|    NAME          |  STATE  | IPV4 | IPV6 |   TYPE    | SNAPSHOTS |
+------------------+---------+------+------+-----------+-----------+
```

```
| rockylinux-test-9 | STOPPED |      |      | CONTAINER | 0         |
+-------------------+---------+------+------+-----------+-----------+
```

Success! Try starting it. Because we are starting it on the lxd-snapshot server, you need to stop it first on the lxd-primary server to avoid an IP address conflict:

```
lxc stop rockylinux-test-9
```

And on the lxd-snapshot server:

```
lxc start rockylinux-test-9
```

Assuming all of this works without error, stop the container on lxd-snapshot and start it again on lxd-primary.

11.2 Setting boot.autostart to off for containers

The snapshots copied to lxd-snapshot will be down when they migrate, but if you have a power event or need to reboot the snapshot server because of updates or something, you will end up with a problem. Those containers will try to start on the snapshot server creating a potential IP address conflict.

To eliminate this, you need to set the migrated containers so that they will not start on reboot of the server. For our newly copied rockylinux-test-9 container, you will do this with:

```
lxc config set rockylinux-test-9 boot.autostart 0
```

Do this for each snapshot on the lxd-snapshot server. The "0" in the command will make sure that `boot.autostart` is off.

11.3 Automating the snapshot process

It is great that you can create snapshots when you need to, and sometimes you *do* need to manually create a snapshot. You might even want to manually copy it over to lxd-snapshot. But for all the other times, particularly for many containers running on your lxd-primary server, the **last** thing you want to do is spend an afternoon deleting snapshots on the snapshot server, creating new snapshots and

sending them over to the snapshot server. For the bulk of your operations, you will want to automate the process.

The first thing you need to do is schedule a process to automate snapshot creation on lxd-primary. You will do this for each container on the lxd-primary server. When completed, it will take care of this going forward. You do this with the following syntax. Note the similarities to a crontab entry for the timestamp:

```
lxc config set [container_name] snapshots.schedule "50 20 * * *"
```

What this is saying is, do a snapshot of the container name every day at 8:50 PM.

To apply this to our rockylinux-test-9 container:

```
lxc config set rockylinux-test-9 snapshots.schedule "50 20 * * *"
```

You also want to set up the name of the snapshot to be meaningful by our date. LXD uses UTC everywhere, so our best bet to keep track of things, is to set the snapshot name with a date and time stamp that is in a more understandable format:

```
lxc config set rockylinux-test-9 snapshots.pattern "rockylinux-
test-9{{ creation_date|date:'2006-01-02_15-04-05' }}"
```

GREAT, but you certainly do not want a new snapshot every day without getting rid of an old one, right? You would fill up the drive with snapshots. To fix this you run:

```
lxc config set rockylinux-test-9 snapshots.expiry 1d
```

12. Chapter 10: automating snapshots

Throughout this chapter you will need to be root or able to `sudo` to become root.

Automating the snapshot process makes things a whole lot easier.

12.1 Automating the snapshot copy process

Perform this process on lxd-primary. First thing you need to do is create a script that will run by a cron in /usr/local/sbin called "refresh-containers" :

```
sudo vi /usr/local/sbin/refreshcontainers.sh
```

The script is pretty minimal:

```
#!/bin/bash
# This script is for doing an lxc copy --refresh against each container,
copying
# and updating them to the snapshot server.

for x in $(/var/lib/snapd/snap/bin/lxc ls -c n --format csv)
        do echo "Refreshing $x"
        /var/lib/snapd/snap/bin/lxc copy --refresh $x lxd-snapshot:$x
        done
```

Make it executable:

```
sudo chmod +x /usr/local/sbin/refreshcontainers.sh
```

Change the ownership of this script to your lxdadmin user and group:

```
sudo chown lxdadmin.lxdadmin /usr/local/sbin/refreshcontainers.sh
```

Set up the crontab for the lxdadmin user to run this script, in this case at 10 PM:

```
crontab -e
```

Your entry will look like this:

```
00 22 * * * /usr/local/sbin/refreshcontainers.sh > /home/lxdadmin/refreshlog
2>&1
```

Save your changes and exit.

This will create a log in lxdadmin's home directory called "refreshlog" which will give you knowledge of whether your process worked or not. Very important!

The automated procedure will fail sometimes. This generally happens when a particular container fails to refresh. You can manually re-run the refresh with the following command (assuming rockylinux-test-9 here, is our container):

```
lxc copy --refresh rockylinux-test-9 lxd-snapshot:rockylinux-test-9
```

13. Appendix A - workstation setup

While not a part of the chapters for an LXD Server, this procedure will help those who want to have a lab environment, or semi-permanent OS and application, running on a Rocky Linux workstation or notebook.

13.1 Prerequisites

- comfortable at the command line
- able to use a command line editor, such as `vi` or `nano` fluently
- willing to install `snapd` to install LXD
- a need for a stable testing environment used every day or as needed
- able to become root or have `sudo` privileges

13.2 Installation

From the command line, install the EPEL repository:

```
sudo dnf install epel-release
```

When installation finishes, do an upgrade:

```
sudo dnf upgrade
```

Install `snapd`

```
sudo dnf install snapd
```

Enable the `snapd` service

```
sudo systemctl enable snapd
```

Reboot your notebook or workstation

Install the snap for LXD:

```
sudo snap install lxd
```

13.3 LXD initialization

If you have looked through the production server chapters, this is nearly the same as the production server init procedure.

```
sudo lxd init
```

This will start a question and answer dialog.

Here are the questions and our answers for the script, with a little explanation where warranted:

```
Would you like to use LXD clustering? (yes/no) [default=no]:
```

If you have interest in clustering, do some additional research on that at Linux containers here.

```
Do you want to configure a new storage pool? (yes/no) [default=yes]:
Name of the new storage pool [default=default]: storage
```

Optionally, you can accept the default.

```
Name of the storage backend to use (btrfs, dir, lvm, ceph) [default=btrfs]: dir
```

Note that `dir` is somewhat slower than `btrfs`. If you have the foresight to leave a disk empty, you can use that device (example: /dev/sdb) for the `btrfs` device and then select `btrfs`, but only if your host computer has an operating system that supports `btrfs`. Rocky Linux and any RHEL clone will not support `btrfs` - not yet, anyway. `dir` will work fine for a lab environment.

```
Would you like to connect to a MAAS server? (yes/no) [default=no]:
```

Metal As A Service (MAAS) is outside the scope of this document.

```
Would you like to create a new local network bridge? (yes/no) [default=yes]:
What should the new bridge be called? [default=lxdbr0]:
```

```
What IPv4 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]:
What IPv6 address should be used? (CIDR subnet notation, “auto” or “none”)
[default=auto]: none
```

If you want to use IPv6 on your LXD containers, you can turn on this option. That is up to you.

```
Would you like the LXD server to be available over the network? (yes/no)
[default=no]: yes
```

This is necessary to snapshot the workstation. Answer "yes" here.

```
Address to bind LXD to (not including port) [default=all]:
Port to bind LXD to [default=8443]:
Trust password for new clients:
Again:
```

This trust password is how you will connect to the snapshot server or back from the snapshot server. Set this with something that makes sense in your environment. Save this entry to a secure location, such as a password manager.

```
Would you like stale cached images to be updated automatically? (yes/no)
[default=yes]
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]:
```

13.4 User privileges

The next thing you need to do is to add your user to the lxd group. Again, you will need to use `sudo` or be root for this:

```
sudo usermod -a -G lxd [username]
```

where [username] is your user on the system.

At this point, you have made a bunch of changes. Before you go any further, reboot your machine.

13.5 Verifying the install

To ensure that `lxd` started and that your user has privileges, from the shell prompt do:

```
lxc list
```

Note you have not used `sudo` here. Your user has the ability to enter these commands. You will see something like this:

```
+------------+---------+----------------------+------+-----------+-----------+
|    NAME    |  STATE  |         IPV4         | IPV6 |   TYPE    | SNAPSHOTS |
+------------+---------+----------------------+------+-----------+-----------+
```

If you do, you are looking good!

13.6 The rest of it

From this point, you can use the chapters from our "LXD Production Server" to continue on. There are some things on a workstation setup though that you need to pay less attention to. Here are the recommended chapters to get you going:

- Chapter 5 - Setting Up And Managing Images
- Chapter 6 - Profiles
- Chapter 8 - Container Snapshots

13.7 More reading

- LXD Beginners Guide which will get you started using LXD productively.
- Official LXD Overview and Documentation

13.8 Conclusion

LXD is a powerful tool that you can use on workstations or servers for increased productivity. On a workstation, it is great for lab testing, but can also keep semi-permanent instances of operating systems and applications available in their own private space.