Chapter W23 Virtualization Methodologies

Objectives

\* To give background information on operating system virtualization

\* To provide a description of LXC/LXD virtualization method

\* To explicitly detail LXC/LXD installation on both Debian-Family and CentOS 7 Linux Systems

\* To illustrate LXC/LXD basic usage

\* To give command references for LXC/LXD

\* To list LXC/LXD best practices, and provide examples of advanced usage

\* To give a set of complete worked examples of LXC/LXD usage

\* To describe and illustrate VirtualBox with different types of installation examples

\* How to install VirtualBox on Linux Mint

\* How to install Ubuntu as a guest on a Linux Mint host

\* To show how to secure an FTP server in a VirtualBox guest

\* To cover the commands and primitives:

lxc, lxd

W23.1 Introduction to Virtualization Methodologies and Background

Question: What does the word “virtual” mean?

Answer: As if

The three governing functions that the Linux kernel performs in order to maintain the system in a steady state are Virtualization, Concurrency, and Persistence. At a certain level of abstraction, the kernel itself, and its global resources, are virtualized by systemd, as shown in Chapter 20. In this chapter, namespaces, LXD, and VirtualBOX also do forms of “virtualizing”.

Historically speaking, to perform the kernel functions of Virtualization and Concurrency, the multiprogramming paradigm was established. Multiprogramming is a computer system model where the computer hardware and software would be shared by several programs (and users) running and working on the system simultaneously. This led to the need for autonomy and sharing among programs. The needs of multiprogramming are closely tied to the concept of “virtual memory” in computers as well. In turn, this virtualization of memory is directly related to the virtualization of the entire operating system itself, as we describe it in this chapter.

Computer hardware virtualization is the simulation, to various degrees, of hardware platforms, parts of them, or only the functionality required to run one or more operating systems (OSs). It abstracts and effectively “hides” the physical characteristics of the hardware from the users. Traditionally, the software that controlled virtualized machines was known as the hypervisor. Currently, the hypervisor is often called a Virtual Machine Monitor (VMM). For example, in VirtualBox, the Virtual Machine (VM) VirtualBox manager is the graphical front-end for a VMM.

Platform virtualization is accomplished on any given hardware by host software (the hypervisor), which creates a simulated computer environment, a VM, for its guest software. The guest software can be as small as a single user application or as large as a complete OS. The guest software executes as if it were running directly on the physical hardware.

Virtualization comes with some performance disadvantages, both in resources required to run the hypervisor and in reduced performance on the VM guest compared with running applications on a non-virtualized host physical machine. A VM can be more easily controlled and inspected from outside than a physical one, and its configuration is more flexible. This is very useful in kernel development and for

teaching OS courses. A new VM can be implemented as needed without the need for an up-front hardware purchase. A VM can easily be moved from one physical machine to another as needed. An unrecoverable fault inside a VM guest does not harm the host system, so there is no risk of crashing the host OS.

Examples of virtualization implementations are as follows:

\* Running one or more applications that are not supported by the host OS: A VM run-

ning the required guest OS could allow the desired applications to be run, without altering the host OS.

\* Evaluating an alternate OS: The new OS could be run within a VM, without altering the host OS.

\* Server virtualization: Multiple virtual servers in containers could be run on a single physical server, to utilize more fully the hardware resources of the physical server.

\* Duplicating specific environments: A VM could, depending on the virtualization software used, be duplicated and installed on multiple hosts, or restored to a previously backed-up system state.

\* Creating a protected environment: If a guest OS running on a VM becomes damaged in a way that is difficult to repair, such as may occur when testing, the VM can be discarded without harm to the host system, and a clean copy used next time.

Primary actual contemporary virtualization techniques are as follows:

\* Full virtualization: In full virtualization, the VM simulates enough hardware to allow a complete “guest” OS, one designed for the same processor instruction set architecture (ISA) to be run in isolation. Examples for Linux systems running on X86 ISAs include VirtualBox, Parallels Workstation, Oracle VM, Virtual Server, Hyper-V, VMware Workstation, and VMware.

\* Hardware-assisted virtualization: In hardware-assisted virtualization, the hardware

provides architectural support that facilitates building a VMM and allows guest OSs

to be run in isolation. Examples of virtualization platforms adapted to such hardware include KVM, VMware Workstation, VMware Fusion, Hyper-V, Xen, Oracle VM server for SPARC, and VirtualBox.

\* Partial virtualization: In partial virtualization, including address space virtualization, the VM simulates multiple instances of much of an underlying hardware environment, particularly address spaces.

\* Paravirtualization: In paravirtualization, the VM does not necessarily simulate hard-

ware, but instead (or in addition) offers a special application programmer’s interface

(API) that can only be used by modifying the “guest” OS. For this to be possible, the

“guest” OS’s source code must be available.

\* OS-level virtualization: In OS-level virtualization, a physical server is virtualized at the

OS level, enabling multiple isolated and secure virtualized servers to run on a single

physical server. The “guest” OS environments share the same running instance of

the OS as the host system. Thus, the same OS kernel is also used to implement the

“guest” environments, and applications running in a given “guest” environment view

it as a stand-alone system. In Linux, examples include LXC and LXD, and their derivative management

system, Docker. Similar proprietary derived techniques are used by AWS EC2, Google Cloud, and iCloud .

In this chapter, we provide examples of virtualization methodologies using our representative Linux systems from the Debian-family, and CentOS releases, as shown in the printed book. Additionally, here in this chapter, we provide three fully-worked practical examples of LXC/LXD containerization using Ubuntu 18.04 Desktop and Server Editions.

We illustrate full virtualization with VirtualBox examples, and OS-level/ hardware-assisted virtualization examples with LXC/LXD . These techniques are very contemporary, popular, and important facilities for creating a virtual environment within which a Linux OS can work. What differentiates these facilities is that, in LXC/LXD, all virtual environments are running under the same kernel (implementing OS-level virtualization). In VirtualBox, any number of different kernels can be running simultaneously on one machine ( implementing full virtualization).

Practically speaking, an important application of these implementations, as already stated, is to provide a measure of system security, in addition to what is described in Chapter 17, Section 9 in the printed book. But that is not the only reason an ordinary user or system administrator would deploy the virtualization methods we demonstrate.

A user might want to take advantage of some of the facilities that an additional OS

offers, above and beyond what is available as the main OS into which the computer boots.

Instead of shutting down the main OS, and then booting into the additional OS, both can

be run simultaneously using the virtualization method shown here. So if you run both OSs simultaneously, you can use both OS applications and facilities on the same physical machine. Of course, there are trade-offs in doing this, mainly in terms of performance speed and disk usage.

Also advantageous is the deployment of VMs to allow you to “test drive” a particular OS without devoting an entire hardware platform to it. This can also be achieved by running a “live” version of it from a DVD or a USB thumb drive, but the performance speed and persistence of data using these techniques is somewhat limited. In this chapter, we show examples of installing guest VMs on Linux,

to allow you to “test drive” systems in a more fully functional way.

W23.2 Linux Containers With LXC/LXD

Question: Why would an ordinary Linux user, who has some flavor of Linux installed on her home desktop computer, need or want to use Linux containers?

Answer: Perhaps she wants to safely test one or many software applications (or even entire operating systems!), and doesn’t want them, through misbehavior, or for other reasons, destroying or significantly changing in any way her installed host system. Or perhaps she wants to securely run a web server on her machine, and via port forwarding on her modem/router, publish pages and a website, on the Internet.

Both of these answers are very easily accommodated through the use of LXC/LXD containers.

In-Chapter ExerciseW23.1

Why would you personally want to add a container system, and one or more containers, to your computer system? What “flavor”of container would you want to add, and why? What advantages would that give even an ordinary user on a single desktop or laptop computer? As you proceed through this chapter, compare adding a container system that we have shown in this section to VirtualBox. Relative to those types of virtualization, how do LXC/LXD containers compare, across all of the installation, maintenance, disk storage footprint, etc. profiles for your particular use case(s)?

The LXC/LXD container model uses a system programming feature, known as “namespaces”, to implement isolated process environments within which LXD containers operate. We briefly mentioned namespaces in Chapter 17, when we dealt with the forms and sites of Linux security. This programming feature, and the namespaces API modules clone(), unshare() and setns(), provide another distinct form of virtualization in terms of the virtualization/concurrency of the Linux kernel.

The term “namespace”, sometimes called a “name scope”, is popularly defined as-

“an abstract container or environment created to hold a logical grouping of unique identifiers or symbols (i.e. names). An identifier defined in a namespace is associated only with that namespace.”

The concept and use of the term “namespace”, as it is used in relation to the system programming API for Linux container virtualization, is analogous to several other applications of the same term in computer science and programming. We give these analogies here to help make the higher-level abstraction more clear to the ordinary user.

One obvious similarity of the application of this term is to locally scoped versus globally scoped variables. We show a brief example of this in Chapter W19 at the book website, with Python variables. A namespace is like a locally scoped Python variable. In non-OOP Python programming, an identifier, or “name” is equated to some expression, or object, that gives that name a value. So the “namespace” of a locally-scoped variable is the module that fully contains, limits, or encapsulates this equation of identifier to object.

Another analogy can be drawn to the use of the term in Python OOP program constructs. For example, OOP itself has the fundamental features of multiple representation, encapsulation, sub-typing, inheritance, and open recursion, which hide the objects, their classes, and methods in namespace environments. The representation of an object is hidden from view outside of the object’s definition: only the object’s own methods can directly inspect or manipulate its fields. \*\*

Other OOP-capable languages, such as XML, also use the term to describe isolated environments. XML collects elements and attributes in a definition, in a style sheet for example, to limit the scope of an identifier.

There are currently six types of Linux namespace: mount,UTS, IPC, PID, network, and user, each of which provides a context within which a process can “virtualize” the global system resources, such as file system mounting, Interprocess Communication, Process Identification, networking, and isolated user spaces. A particular user namespace can, for example, also overlap with any of the other contexts as well.

Relative to namespaces, it is “as if” a user’s process or processes were the only program(s) running on the hardware. We encourage you to read the man pages for namespaces and user\_namespaces on your system, not only to gain some insight into the namespaces higher-level abstraction, but also to appreciate the complexity of its lower-level details. In our exposition of LXC/LXD, we do not make any references to the internals of namespaces, or to how LXC/LXD works in conjunction with these details. In this chapter we have smaller fish to fry, and take a bottoms-up approach to virtualization.

The current premier operating system-level virtualization software in Linux is LXD (pronounced “lex-dee”). It is a combination of an older command line “client” utility named LXC, and a newer controlling container management daemon named LXD .Working with LXD involves using a combination of LXC and LXD commands, as we show throughout this section.

We first cover the installation and basic usage of LXC/LXD, and then give some extended examples of creating, running, and managing LXC/LXD containers.

In particular, we do these some of these tasks by creating our containers with a ZFS file system on the Debian-family system, Linux Mint 18.2. The details of working with ZFS are fully detailed in Chapter W22 at the book website.

In-Chapter Exercise W23.2

If you have already completed all materials in Chapter W22 on ZFS, what advantages do you see in using ZFS as the storage file system for LXD containers?

\* Wikipedia, namespaces.

\*\* Pierce, Types and Programming Languages, 2002

W23.2.1 Introduction to LXC/LXD

LXD (pronounced lex-dee) is a lightweight container hypervisor. It also nicely distinguishes LXD from LXC.

LXC (lex-see) is the originary, eralier version program which actually creates and administers the containers on the targeted system. LXC also provides an Application Programming Interface (API) to allow higher level managers, such as LXD, to administer containers.

The LXC API deals with a 'container'. LXD also has an API, which deals with 'remotes', remote to the target system, and which serve images and containers. Thus LXD extends the LXC functions over a network, and allows important management tasks, such as container migration image publishing.

LXD uses LXC for almost all of its container management tasks.

Caveat: At the time of the writing of this book, the Debian-family of Linux systems (most prominently Linux Mint 18.2 Sonya) were the only Linux systems that we could reliably and easily install LXD on. Many of the examples below reflect this, particularly the LXD/ZFS-based ones. On CentOS 7.4, we could only reliably and easily install LXC. If that changes, we will certainly modify this document as it exists at the website for this book.

LXD is the latest version of LXC, a Linux container system that uses operating system-level virtualization. It incorporates newer, and more useful features for the creation, provisioning, management, and deployment of containers on host systems.

For our purposes, “provisioning” means configuring a container to have a set of features, which may be easily duplicated, or cloned, into as many containers as necessary. An example of that, which we show below, would be downloading and installing an Ubuntu operating system image in the container. The container and the image it is built from by LXC/LXD are synonymous.

It is important to differentiate the host system from the container(s) running on the host. Different operating system images can be downloaded and installed in containers, even though the host is running a particular operating system; in our case, our host is running either Linux Mint by default, or CentOS 7.

It is possible to download and install “unofficial” images of other operating systems, such as Ubuntu, Debian, openSUSE, CentOS 7, etc..

These images, and the version available for each, is highly dependent upon the support provided by the maintainers of the host operating system you are using. For example, at the time of the writing of this book, for a CentOS 7.4 host that has LXC installed, the default template available for Ubuntu was Ubuntu 12.04 LTS. That template was a significantly earlier version of Ubuntu than was currently available for download and installation on bare metal.

W23.2.1.1 Getting Help on LXC and LXD

The best documentation you can get for LXC and LXD is right on your system, in the man pages for lxc and lxd. We encourage you to refer to these man pages for further explanations, and a summary overview of the options and the sub-commands that can be used with each.

There are man pages on the system as well for the sub-commands. For example, to see a man page for the execute sub-command of the lxc command, type man lxc-execute. Following is a table of those sub-commands, which gives a brief description of what each of them do:

Table W23.1 LXC/LXD “Sub-Commands” Listing and What They Do

|  |  |
| --- | --- |
| attach | Start a process inside a running container |
| autostart | Start/stop/kill auto-started containers |
| checkconfig | Check the current kernel for lxc support |
| checkpoint | Checkpoint a container |
| config | Manage configuration |
| console | Launch a console in the specified container |
| copy | Copy containers within or in between lxd instances |
| create | Creates a new container |
| delete | Delete containers or container snapshots |
| destroy | Destroys a container |
| device | Manage devices of running containers |
| execute | Execute the specified command in a container |
| file | Manage files on a container |
| freeze | Freeze the containers processes |
| help | Presents details on how to use LXD |
| image | Manipulate container images |
| info | List information on LXD servers and containers |
| launch | Launch a container from a particular image |
| list | Lists the available resources |
| ls | Lists the containers that exist on the system |
| monitor | Monitor container state |
| move | Move containers within or in between lxd instances |
| profile | Manage configuration profiles |
| publish | Publish containers as images |
| remote | Manage remote LXD servers. |
| restart | Changes state of one or more containers to restart |
| restore | Set the current state of a resource back to a snapshot |
| snapshot | Create a read-only snapshot of a container |
| start | Changes state of one or more containers to start |
| stop | Changes state of one or more containers to stop |
| top | Monitor container statistics |
| unfreeze | Thaw all container’s processes |
| unshare | Run a task in a new set of namespaces |
| user-nic | Create and attach a nic to another network namespace |
| usernet | Unpriveleged user network administration file |
| usernsexec | Run a task as root in a new user namespace |
| version | Prints the version number of this client tool |
| wait | Wait for a Specific container state |

In-Chapter Exercise W23.3

Use the man command on your system to review all of the man pages for lxc and lxd, that contain the sub-commands listed in Table W23.1. Then, make a preliminary list of which sub-commands you think are the most important and critical to the operation of LXC/LXD.

W23.2.2 LXD and LXC Installation and Basic Commands for Debian-Family Linux Systems

This section details some basic commands, such as how to download and install LXD and LXC, how to create and start a new container, and how to get information about images at your default repository and on your machine. We illustrate these basic commands and procedures for Debian-family systems (in particular Linux Mint 18.2), in this sub-section.

See Section W23.2.7 for download, installation, and basic commands executed on Redhat-family CentOS 7.4.

Debian-Family

An essential procedure to follow, as shown in the sequencing of the first three commands you would do, is:

1. Download LXD

2. Initialize LXD

3. Launch container(s)

1. a. To download and install the “machinery” of LXD for Linux Mint 18.2 and Ubuntu 16.04, or the LXD package itself from your default repository, use the following command:

$ **sudo apt install -y -o 'apt::install-recommends=true' lxd lxd-client lxd-tools**

1.b. For Debian 9 “Stretch”, which requires the snap package to install LXD, use the following commands:

$ **lsb\_release -a**

$ **sudo apt update**

$ **sudo apt install lxc -y**

$ **sudo apt install snapd**

$ **sudo snap install lxd**

$ **sudo systemctl reboot**

After reboot-

$ **lxc version**

2.18

$

2. After downloading and installing the LXD package, you should initialize its configuration with the following command:

$ **sudo lxd init**

This command allows you to establish things like the file system for subsequent container storage, the type of network interface you will have for containers, and other critical configuration choices and options. We illustrate an example of this in Example W23.1 in Section W23.2.6.

3. To download, install, and start a particular image and container on your host system, such as an Ubuntu 16.04 image and container, use the following command:

$ **sudo lxc launch ubuntu:16.04 containerx**

This downloads and installs the Ubuntu 16.04 image, named containerx. It will initially be provisioned as a complete operating system, with the configuration choices and options you made in the previous command.

A full list of the Ubuntu images can be seen by using the following command:

$ **sudo lxc image list ubuntu:**

To see all of the “unofficial” images available in the repository:

$ **sudo lxc image list image:**

An abbreviated excerpt from the long output listing of this command is as follows:

+---------------------------------+--------------+--------+-----------------------------------------+-

| centos/7 (3 more) | 475338b18e86 | yes | Centos 7 amd64 (20160921\_02:16) | x86\_64 | 62.91MB | Sep 21, 2016 at 12:00am (UTC) |

To launch a centos7 image, shown in the repository from the previous command, use the following command:

$ **sudo lxc launch images:centos/7/amd64 centos7**

This command downloads the centos7 image, installs it, and names it centos7.

To create a container, that is download and install its image, but not start it immediately, replace “lxc launch” by “lxc init”, as in the following command:

$ **sudo lxc init ubuntu:**

To list all the containers currently installed on your machine, use the following command:

$ **sudo lxc list**

To add an option to the list command that abbreviates the column output so it is more legible, use the following:

$ **sudo lxc list --fast**

To get more complete and legible information about a particular container named “containerx”, use the following command:

$ **sudo lxc info containerx**

W23.2.3 Container Management Commands

Now that you have LXC/LXD installed, and a container started, the following are the most important commands that a beginner needs to know to actually manage a container.

To start a container, named “containerx”, use the following command:

$ **sudo lxc start containerx**

To stop a container named “containery”, or to force it to stop, use the following commands:

$ **sudo lxc stop containery**

$ **sudo lxc stop containery --force**

To restart a container named “containerz”, and force it to restart, use the following command:

$ **sudo lxc restart containerz**

**$ sudo lxc restart containerz --force**

To delete a container named “containerq”, and force it to be deleted if it is still running, use the following:

$ **sudo lxc delete containerq**

$ **sudo lxc delete container --force**

W23.2.4 Internal Container Management Commands Executed from the Host

The most useful and practical thing that you can do to manage containers is to be able to work inside of the container operating system environment itself. This allows you to do tasks such as user file and container operating system maintenance, install software in the container, create users and groups, monitor container performance, etc..

There are a few equivalent ways to do this. One of them is to start a shell, such as bash, inside the container and operate in that shell. Another is to ssh into the container, as we show in Section W23.2.6. There are also LXC commands available on the host to execute Linux commands in a container, put or get files into/from a running container. As we show in Section W23.2.6, the container’s file system is mounted on the host, and is accessible from the host operating system as well. We show some of these techniques in this section.

To start a bash shell inside a container name “containert”, use the following command:

$ **sudo lxc exec containert bash**

To execute a Linux command inside a running containert from the host, you can use the following syntax:

$ **sudo lxc exec containert -- ls -la /**

Notice the syntax that partitions the command sent to containert, the – and the /.

To get filexyz from a container, named “containera”, that is in the container directory /home/bob, and “pull” it to the host into a specific destination, use the following command:

$ **sudo lxc file pull containera/home/bob/filexyz /home/mansoor/**

To put a file into containera from the host, use the following command:

$ **sudo lxc file push /home/bob/.ssh/id\_rsa.pub containera/home/bob/.ssh/authorized\_keys/**

W23.2.5 Container Backups, Snapshots, and Cloning

Certainly one of the most important tasks for the system administrator, or for that matter, the individual user that may be working and managing her own home computer, is file system backup. For LXD, backup strategies follow two distinct paths. You must not only backup the containers, and their file systems, but also the “machinery” of LXD itself on the host computer you are running it on.

The backup of containers, and the data that supports each individually, is done with the snapshot command. Restoring a container from it’s backup snapshots is done with the restore command. It is important to note here that LXD snapshots are more inclusive ,and particularly in the examples we show in Section W23.2.6, are not the same thing as using zfs snapshot and rollback on a container file systems that is using ZFS.

The backup and restoration of LXD “machinery” is something we address in a problem at the end of this chapter.

To take a snapshot of a container, named containera, use the following command:

$ **sudo lxc snapshot containera**

The name of the snapshot will be snap\_number, where snap\_number is an incrementing integer number, starting at zero(0).

To create a snapshot of containera with a distinct name that you choose, use the following command:

$ **sudo lxc snapshot containera containera\_snapshot\_2**

The number of snapshots that have been taken on a container is shown in the lxc list command, but a listing of snapshots with their names is only shown in the the lxc info command. To list the names of snapshots taken on containera, for example, use the following command:

$ **sudo lxc info containera**

To rename a snapshot, use the lxc move command as follows:

$ **sudo lxc move containera/containera\_snapshot\_2 containera/containera\_snapshot\_3**

The lxc move command can also be used to rename a stopped container, retaining its IP address according to our methodology of bridged adapter network address assignment, using the following syntax:

$ **sudo lxc move <old name> <new name>**

To restore a snapshot, use the following command, and be aware that the snapshot, in this case named containera\_snapshot\_2, is restored into the container that it is a snapshot of: in this case, containera.

That brings up an interesting similarity between LXD snapshots, and ZFS dataset snapshot and rollback: Once you restore, or rollback, you have lost the version of the container ( or in the case of ZFS, the dataset) as it perhaps evolved since the snapshot was taken. Contrast and compare this to using the git command, where you can “rollback” to a previous commit, and then roll forward again to the latest commit along any branch.

$ **sudo lxc restore containera containera\_snapshot\_3**

To escape the snapshot/rollback dilemma presented by the previous command, you can create a new container which is identical to another container’s snapshot.

$ **sudo lxc copy containera/containera\_snapshot\_ 3 containerb**

So following from all the commands presented in this section, a list of the containers and snapshots you have is as follows:

1. snap0 ( the first automatically-named snapshot),

2. containera (not the originary container, but the one overwritten by containera\_snapshot\_3),

3. containera\_snapshot\_3 (the renamed snapshot ,which was containera\_snapshot\_2), and 4. containerb.

To delete a snapshot, do the following:

$ **sudo lxc delete containera/containera\_snapshot\_3**

In container management, it is often necessary to use one container as a master “template” for provisioning and creating several other containers. To make a copy of a container and then effectively “clone” it into a new container, use the following command:

$ **sudo lxc copy <source container> <destination container>**

W23.2.6 Extended LXD Installation and Container Management Examples

In this section, we show specific extended examples of the installation, execution and creation of LXD containers, and ZFS and Bridge-Utils on Linux Mint. These examples should be done consecutively, as they build upon each other to achieve the goals of this section. A very useful technique we employ is using ZFS in conjunction with LXD to configure a ZFS zpool on a vdev that is a file, and then using the file system created on that zpool for LXD container storage on the host. Furthermore, in the primary example, we implement a bridged networking setup, so individual LXD containers provisioned by LXD can be automatically assigned an IP address via the DHCP server on your network.

These examples execute tasks that achieve a few system management objectives for us, as follows:

1. Integrates our ZFS system management paradigm and knowledge with a contemporary operating system-level virtualization model.

2. Use an “experimental” way of working with ZFS, using a file for a zpool, rather than a physical disk (or an existing disk partition, which is not recommended for ZFS). Experimental in the sense that you can simulate an actual hard disk vdev with a file used as a vdev, as shown in the ZFS chapter available at the Github site for this book. We also pose a problem at the end of the chapter that allows you to use an actual physical disk as a vdev for ZFS/LXD installation.

3. Expose the LXD containers with their own “public facing” network addresses, rather than on a private subnet such as 10.0.0X. This is different from using the traditional LXC methods of using the iptables command to expose containers as ports on the host computer. We find that this “public-facing” IP address method is simpler and more useful for a user trying to learn about LXD containers and the networking of these containers on their own home network, than the traditional way of networking the containers.

4. Gets your hands dirty with LXD.

The examples in this section make these assumptions:

\*You are working on a Linux Mint host that is installed on “bare metal”, i.e. not running as a VirtualBox guest under some other operating system, or as a guest under some other hypervisor virtualization software.

\* You have enough space on your system disk to accommodate not only the installation of ZFS, LXD, and bridge-utils packages, but also for a file that will be your ZFS/LXD filesystem, similar to the preliminary examples we show in the ZFS chapter available at the Github site for this book.

Since we will be using a file to simulate a disk in our primary Example W23.1, and all subsequent examples, if you wish to use an actual (and available) physical hard disk instead, see Problem X at the end of the chapter for alternative instructions on how to complete Example W23.1.

\* You have superuser privileges on your computer system.

\* In the primary example we download and install an Ubuntu 16.04 image, and create our first container with it. The fundamental reason for doing this is, since Linux Mint is derived from Ubuntu, you can work with Ubuntu in the container without having to relearn any new commands or facilities, such as working with another package management system to download and install software in the container. As seen above, there are several unofficial images which can be used to create containers, and we encourage you to experiment with these by posing a problem as such at the end of the chapter.

Example W23.1 Installing LXD and Running Containers With a ZFS Filesystem

Objective: Installation of LXD, execution and creation of an LXD container, and utilization of ZFS and Bridge-Utils for that container.

Pre-Requisites: Knowledge of basic Linux commands from the core Chapters 1-14 from the printed book. completion of Chapter 17 from the printed book. Optional completion of the W22 ZFS chapter available at the website for this book.

Background: This primary example allows you install, create, and start an LXD container. It also allows you to experiment with ZFS and bridged network adapters. It also assumes that you will be working in your home account on your computer system.

Procedures: Do the following steps, in the order shown, to fulfill the requirements of this example-

1.This step is optional if you have already downloaded and installed zfsutils-Linux from the ZFS chapter available at the Github site for this book.

Download and install zfs with the following command:

$ **sudo apt-get install zfsutils-Linux**

[sudo] password for bob: **yyyxxx**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

libnvpair1Linux libuutil1Linux libzfs2Linux libzpool2Linux zfs-doc

Suggested packages:nfs-kernel-server zfs-initramfs

Recommended packages:

zfs-zed

The following NEW packages will be installed:

libnvpair1Linux libuutil1Linux libzfs2Linux libzpool2Linux zfs-doc

zfsutils-Linux

0 upgraded, 6 newly installed, 0 to remove and 35 not upgraded.

Need to get 867 kB of archives.

After this operation, 2,770 kB of additional disk space will be used.

Do you want to continue? [Y/n] **Y**

Output truncated …

Processing triggers for ureadahead (0.100.0-19) ...

Processing triggers for systemd (229-4ubuntu4) …

2. If you have an actual expendable hard disk, or USB thumb drive, that you can add to your computer on the SATA or USB bus to complete this example with, then skip ahead to the instructions for Problem 2. at the end of this chapter.

Otherwise, to implement the “experimental” technique of working with ZFS shown in the ZFS chapter available at the Github site for this book, create a file to be used as a ZFS vdev, with the following command:

$ **truncate -s 1G disk\_file**

3. Create a zpool using that file as a vdev with the following command:

$ **sudo zpool create lxd\_pool /home/bob/disk\_file**

4. Download and install LXD:

$ **sudo apt install -y -o 'apt::install-recommends=true' lxd lxd-client lxd-tools**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

apparmor libapparmor-perl liblxc1 lxc-common lxcfs squashfs-tools uidmap

Suggested packages:

apparmor-profiles apparmor-profiles-extra apparmor-docs apparmor-utils criu

lxd-tools

Recommended packages:

lxd-client

The following NEW packages will be installed:

apparmor libapparmor-perl liblxc1 lxc-common lxcfs lxd squashfs-tools uidmap

0 upgraded, 8 newly installed, 0 to remove and 35 not upgraded.

Need to get 4,498 kB of archives.

After this operation, 22.0 MB of additional disk space will be used.

Do you want to continue? [Y/n] **Y**

Output truncated…

Processing triggers for systemd (229-4ubuntu4) ...

Processing triggers for libc-bin (2.23-0ubuntu3) …

$

5. Download and install the bridge\_utils package, which will enable containers to have a “public-facing” IP address:

$ **sudo apt-get install bridge-utils**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

bridge-utils

0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.

Need to get 28.6 kB of archives.

After this operation, 102 kB of additional disk space will be used.

Get:1 http://archive.ubuntu.com/ubuntu xenial/main amd64 bridge-utils amd64 1.5-9ubuntu1 [28.6 kB]

Fetched 28.6 kB in 0s (36.5 kB/s)

Selecting previously unselected package bridge-utils.

(Reading database ... 230047 files and directories currently installed.)

Preparing to unpack .../bridge-utils\_1.5-9ubuntu1\_amd64.deb ...

Unpacking bridge-utils (1.5-9ubuntu1) ...

Processing triggers for man-db (2.7.5-1)

$

6. Use the ip command to find the default network interface (nic) name, as follows:

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state

UNKNOWN group default qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: en01: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state

UP group default qlen 1000

link/ether 00:01:c0:1a:3f:0e brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic enp1s0

valid\_lft 604598sec preferred\_lft 604598sec

inet6 fe80::d591:4bb9:1f10:47f0/64 scope link

valid\_lft forever preferred\_lft forever

On our system, the default network adapter was named eno1, as shown in the output.

7. To utilize a bridged adapter for the LXD containers, use your favorite text editor to edit the the file /etc/network/interfaces, so that it appears as follows-

# The primary network interface

auto br0

iface br0 inet dhcp

bridge\_ports eno1

iface eno1 inet manual

Save and quit the editor.

8. Restart the network interfaces, so the bridged adapter will be deployed, using the following command:

$ **sudo ifdown eno1 && sudo ifup eno1 && sudo ifup br0**

ifdown: interface eno1 not configured

Waiting for br0 to get ready (MAXWAIT is 32 seconds).

Internet Systems Consortium DHCP Client 4.3.3

Copyright 2004-2015 Internet Systems Consortium.

All rights reserved.

For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/br0/b8:ac:6f:9a:80:dc

Sending on LPF/br0/b8:ac:6f:9a:80:dc

Sending on Socket/fallback

DHCPDISCOVER on br0 to 255.255.255.255 port 67 interval 3 (xid=0x34444f5f)

DHCPREQUEST of 192.168.0.8 on br0 to 255.255.255.255 port 67 (xid=0x5f4f4434)

DHCPOFFER of 192.168.0.8 from 192.168.0.1

DHCPACK of 192.168.0.8 from 192.168.0.1

bound to 192.168.0.8 -- renewal in 239724 seconds.

$

9. Verify the change in network interfaces using the ip command as follows:

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state

UNKNOWN group default qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eno1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq

master br0 state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ffhat it looks like this:

# interfaces(5) file used by ifup(8) and ifdown(8)

auto lo

iface lo inet loopback

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue

state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::e611:5bff:fe12:c277/64 scope link

valid\_lft forever preferred\_lft forever

$

10. This is the most critical step. Initialize your container system by using the LXD daemon with the following command:

$ **sudo lxd init**

Name of the storage backend to use (dir or zfs) [default=zfs]: **zfs**

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

Name of the existing ZFS pool or dataset: **lxd\_pool**

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

Do you want to configure the LXD bridge (yes/no) [default=yes]? **Yes**

*At this point we finalize the LXD configuration by entering the following responses into a set of three windows that appear on-screen.*

a. In the first screen, we don’t want LXD to create a new bridge for us, so enter **no** here.

b. In the second screen, enter **yes** in order to supply the name of our bridge

c. Type-in the bridge name as **br0** and select OK.

Warning: Stopping lxd.service, but it can still be activated by:

lxd.socket

LXD has been successfully configured.

$

What we have achieved in the first 10 steps is

\* Prepared for our LXD installation by downloading and installing ZFS ( so that LXD containers will use a ZFS filesystem for storage on the host),

\* Created a zpool for LXD use,

\* Downloaded and installed the bridge-utils package (which gives our containers “public-facing” IP addresses)

\* Downloaded and installed LXD, so that it is configured with ZFS and a bridged adapter.

11. To verify the zpools on the system, use the following command:

$ **sudo zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

lxd\_pool 1008M 64K 1008M - 0% 0% 1.00x ONLINE -

$

12. As of the time of the writing of this book, the critical package lxd-client was not installed along with LXD. So install that package now with the following command:

$ **sudo apt install lxd-client**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

lxd-client

0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.

Need to get 1,841 kB of archives.

After this operation, 8,214 kB of additional disk space will be used.

Get:1 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 lxd-client amd64 2.0.4-0ubuntu1~ubuntu16.04.1 [1,841 kB]

Fetched 1,841 kB in 14s (123 kB/s)

Selecting previously unselected package lxd-client.

(Reading database ... 230073 files and directories currently installed.)

Preparing to unpack .../lxd-client\_2.0.4-0ubuntu1~ubuntu16.04.1\_amd64.deb ...

Unpacking lxd-client (2.0.4-0ubuntu1~ubuntu16.04.1) ...

Processing triggers for man-db (2.7.5-1) ...

Setting up lxd-client (2.0.4-0ubuntu1~ubuntu16.04.1) …

$

13. Now we can get LXD configuration information using the command line interface known as lxc. Type the following command:

$ **sudo lxc info**

Generating a client certificate. This may take a minute...

If this is your first time using LXD, you should also run: sudo lxd init

To start your first container, try: lxc launch ubuntu:16.04

apiextensions: []

apistatus: stable

apiversion: "1.0"

auth: trusted

environment:

addresses: []

architectures:

- x86\_64

- i686

certificate: |

-----BEGIN CERTIFICATE-----

MIIFyjCCA7KgAwIBAgIRAKv57M5fjnfmrNA5PcHdpygwDQYJKoZIhvcNAQELBQAw

Output truncated…

-----END CERTIFICATE-----

certificatefingerprint: 9c51a6d95c7ec08520da8a093516758871e2449c04609b8f6ca9c40b952a9433

driver: lxc

driverversion: 2.0.4

kernel: Linux

kernelarchitecture: x86\_64

kernelversion: 4.4.0-21-generic

server: lxd

serverpid: 18736

serverversion: 2.0.4

storage: zfs

storageversion: "5"

config:

storage.zfs\_pool\_name: lxd\_pool

public: false

$

What we can see in the output of the above command is that LXD is using ZFS for storage of container information, and the name of the zpool for that storage is lxd\_pool. It is also interesting that a helpful message is delivered as output, telling you how to install your first image and container!

14. At this point, it would also be prudent to examine the LXD default profile, using the following command:

$ **sudo lxc profile show default**

name: default

config: {}

description: Default LXD profile

devices:

eth0:

name: eth0

nictype: bridged

parent: br0

type: nic

We can see that the bridged adapter br0 is being used by containers that we have as yet to download or install from images.

15. Create and start the first new container, which we give the name container1, from an Ubuntu image:

$ **sudo lxc launch ubuntu:xenial container1**

Creating container1

Retrieving image: 100%

Starting container1

$

16. To examine the state of the container container1, type the following command:

$ **sudo lxc list**

+------------+---------+---------------------+------+------------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+------------+---------+---------------------+------+------------+-----------+

| container1 | RUNNING | 192.168.0.34 (eth0) | | PERSISTENT | 0 |

+------------+---------+---------------------+------+------------+-----------+

17. We can now open a bash shell into the container container1, and then exit the container, with the following commands:

$ **sudo lxc exec container1 bash**

root@container1:~#

root@container1:~# **exit**

$

18. Check the status of datasets on the host-

$ **sudo zfs list**

NAME USED AVAIL REFER MOUNTPOINT

lxd\_pool 675M 301M 19K /lxd\_pool

lxd\_pool/containers 7.47M 301M 19K /lxd\_pool/containers

lxd\_pool/containers/container1 7.45M 301M 667M /var/lib/lxd/containers/container1.zfs

lxd\_pool/images 667M 301M 19K /lxd\_pool/images

lxd\_pool/images/d0c…3b 667M 301M 667M /var/lib/lxd/images/d0c…3b.zfse

$

The important information seen in the above command output is that the mount point on our system for the ZFS file system of container1 is /var/lib/containers/container1.zfs.

This information allows us to access all of the files in container1, from the host, along the path /var/lib/containers/container1.zfs/rootfs .

Additionaly, we see that container1 takes up 7.45M of storage in the “experimental” file we used as a disk for LXD ZFS.

Example W23.2 Enabling an ssh server in an LXD container

Objective: Installation of an ssh server inside of an LXD container

Pre-Requisites: Completion of Example W23.1

Background: This example is a follow-up to Example W23.1, and allows you ssh into an LXD container.

Procedures: Do the following steps, in the order shown, to fulfill the requirements of this example-

1. For the purposes of this example, you will add a new user with the same name as the user on the host, to the container named container1, and then exit to the host using the following commands:

$ **sudo lxc exec container1 bash**

[sudo] password for bob: **yyyxxx**

root@container1:~# **adduser bob**

Adding user `bob' ...

Adding new group `bob' (1001) ...

Adding new user `bob' (1001) with group `bob' ...

Creating home directory `/home/bob' ...

Copying files from `/etc/skel' ...

Enter new Linux password:

Retype new Linux password:

passwd: password updated successfully

Changing the user information for bob

Enter the new value, or press <Enter> for the default

Full Name []: **<Enter>**

Room Number []: **<Enter>**

Work Phone []: **<Enter>**

Home Phone []: **<Enter>**

Other []: **<Enter>**

Is the information correct? [Y/n] **Y**

root@container1:~# exit

$

2. Now back on the host, check the automatically-generated IP address of the container named container1:

$ **sudo lxc list**

+------------+---------+---------------------+------+------------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+------------+---------+---------------------+------+------------+-----------+

| container1 | RUNNING | 192.168.0.34 (eth0) | | PERSISTENT | 0 |

+------------+---------+---------------------+------+------------+-----------+

The IP address 192.168.0.34, which was automatically assigned to the container by the DHCP server on the network, will be used later to ssh to this container.

3. Check to see if sshd is installed on the host-

$ **sudo systemctl status sshd**

● sshd.service

Loaded: not-found (Reason: No such file or directory)

Active: inactive (dead)

From this output, sshd is not installed.

4. Install it on the host with the following command:

$ **sudo apt-get install openssh-server**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

openssh-sftp-server

Suggested packages:

rssh molly-guard monkeysphere

Recommended packages:

ncurses-term ssh-import-id

The following NEW packages will be installed:

openssh-server openssh-sftp-server

0 upgraded, 2 newly installed, 0 to remove and 35 not upgraded.

Need to get 377 kB of archives.

After this operation, 1,021 kB of additional disk space will be used.

Do you want to continue? [Y/n] **Y**

Output truncated…

Processing triggers for systemd (229-4ubuntu4) ...

Processing triggers for ufw (0.35-0ubuntu2) …

$

5. Edit the file /etc/ssh/sshd\_config with your favorite text editor, and delete the # before the line that reads-

PasswordAuthentication yes

This enables PasswordAuthentication on the host.

6. Get a bash shell in the container, change to Bob’s home directory, and see what files are in it. Then create a .ssh sub-directory in it (if necessary), and also an authorized\_keys sub-directory under Bob’s .ssh directory:

$ **sudo lxc exec container1 bash**

[sudo] password for bob: **yyyxxx**

root@container1:~#

root@container1:~# **cd /home/bob**

root@container1:/home/bob# **ls -la**

total 8

drwxr-xr-x 2 bob bob 5 Sep 20 00:16 .

drwxr-xr-x 4 root root 4 Sep 20 00:16 ..

-rw-r--r-- 1 bob bob 220 Sep 20 00:16 .bash\_logout

-rw-r--r-- 1 bob bob 3771 Sep 20 00:16 .bashrc

-rw-r--r-- 1 bob bob 655 Sep 20 00:16 .profile

root@container1:/home/bob#

Notice there is no .ssh directory.

7. Create an .ssh directory, and a sub-directory underneath it named authorized\_keys, using the following commands:

root@container1:/home/bob# **mkdir .ssh**

root@container1:/home/bob# **cd .ssh**

root@container1:/home/bob/.ssh# **mkdir authorized\_keys**

root@container1:/home/bob/.ssh#

8. Enable password authentication in the container by setting PassworAuthentication to yes, as you did in step 5. above. To do this, use your favorite text editor to edit the /etc/ssh/sshd\_config file.

9. Check the status of the sshd server in the container, then exit the container-

root@container1:/home/bob/.ssh# **systemctl status sshd**

● ssh.service - OpenBSD Secure Shell server

Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)

Active: active (running) since Mon 2016-09-19 21:29:33 UTC; 4h 9min ago

Main PID: 317 (sshd)

Tasks: 1

Memory: 1.3M

CPU: 38ms

CGroup: /system.slice/ssh.service

└─317 /usr/sbin/sshd -D

Sep 19 21:29:33 container1 systemd[1]: Starting OpenBSD Secure Shell server...

Sep 19 21:29:33 container1 sshd[317]: Server listening on 0.0.0.0 port 22.

Sep 19 21:29:33 container1 sshd[317]: Server listening on :: port 22.

Sep 19 21:29:33 container1 systemd[1]: Started OpenBSD Secure Shell server.

root@container1:/home/bob/.ssh# **exit**

$

10. Back on the host, start the ssh service-

$ **sudo systemctl start sshd**

[sudo] password for bob: **yyyxxx**

$

11. Make a directory under /home/bob named .ssh, and change to that directory-

$ **mkdir .ssh**

$ **cd .ssh**

~/.ssh$

12. Generate an id\_rsa.pub key in this directory with the ssh-keygen -t rsa command, as follows:

~/.ssh $ **ssh-keygen -t rsa**

Generating public/private rsa key pair.

Enter file in which to save the key (/home/bob/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/bob/.ssh/id\_rsa.

Your public key has been saved in /home/bob/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:A0gyCwz+Brt1PW3EKDSB0y04j22t7IsIy2yvAwK5gUQ bob@bob-PowerEdge-T110

The key's randomart image is:

+---[RSA 2048]----+

|\*Eo =+o |

|.+ X.+..o |

|o+. B.+. o |

|= +. +ooo |

|.+ +o..oSo |

|+ + .o o. |

|+. . |

|++. .. |

|o=+o .. |

+----[SHA256]-----+

13. Use the following LXD command to copy that keyfile to the directory /home/bob/.ssh/authorized\_keys on container1:

~/.ssh$ **sudo lxc file push /home/bob/.ssh/id\_rsa.pub container1/home/bob/.ssh/authorized\_keys/**

14. Restart the sshd service on the host with the following command:

~/.ssh $ **sudo systemctl restart sshd**

15. Get a bash shell on the container, restart the sshd service there, and then exit to the host, using the following commands:

~/.ssh $ **sudo lxc exec container1 bash**

root@container1:~# **systemctl restart sshd**

root@container1:~# **exit**

~/.ssh $

16. Back on the host, ssh to the container, whose IP address you found in step 2., using the following command:

~/.ssh $ **ssh bob@192.168.0.34**

The authenticity of host '192.168.0.34 (192.168.0.34)' can't be established.

ECDSA key fingerprint is SHA256:w/smq3qewadZ0dBjgfXJGfvm0SPy6bxc6mmFXNCKf8c.

Are you sure you want to continue connecting (yes/no)? **Yes**

Warning: Permanently added '192.168.0.34' (ECDSA) to the list of known hosts.

bob@192.168.0.34's password: **yyyxxx**

The programs included with the Ubuntu system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by

applicable law.

bob@container1:~$

In-Chapter Exercise W23.4

Execute all of the steps of Example W23.2 on your own system to provision and start a new LXD container so that you can ssh to it from another machine on your LAN, or from the Internet.

Example W23.3 Enabling the nginx web server in an LXD container

Objective: Installation of the web server nginx in an LXD container

Pre-Requisites: Completion of Examples W23.1 and W23.2

Background: This example allows you to install a web server named nginx in container1, which you can then use to display web pages stored in the container, by using a web browser.

Procedures: Do the following steps, in the order shown, to fulfill the requirements of this example-

1. Open a bash shell on the container named container1, and check the status of the nginx service:

$ **sudo lxc exec container1 bash**

[sudo] password for bob: **yyyxxx**

root@container1:~# **systemctl status nginx**

● nginx.service

Loaded: not-found (Reason: No such file or directory)

Active: inactive (dead)

root@container1:~#

From the above output, nginx is not installed.

2. To get the latest nginx package, first update the Ubuntu package repository, and then install the nginx service, using the following commands:

root@container1:~# **sudo apt-get update**

Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [94.5 kB]

Hit:2 http://archive.ubuntu.com/ubuntu xenial InRelease

Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [95.7 kB]

Get:4 http://security.ubuntu.com/ubuntu xenial-security/main Sources [39.3 kB]

Get:5 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [9,340 B]

Output truncated…

Fetched 10.7 MB in 39s (272 kB/s)

Reading package lists... Done

root@container1:~# **sudo apt-get install nginx**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

fontconfig-config fonts-dejavu-core libfontconfig1 libfreetype6 libgd3

libjbig0 libjpeg-turbo8 libjpeg8 libtiff5 libvpx3 libxpm4 libxslt1.1

nginx-common nginx-core

Suggested packages:

libgd-tools fcgiwrap nginx-doc ssl-cert

The following NEW packages will be installed:

fontconfig-config fonts-dejavu-core libfontconfig1 libfreetype6 libgd3

libjbig0 libjpeg-turbo8 libjpeg8 libtiff5 libvpx3 libxpm4 libxslt1.1 nginx

nginx-common nginx-core

0 upgraded, 15 newly installed, 0 to remove and 8 not upgraded.

Need to get 3,311 kB of archives.

After this operation, 10.7 MB of additional disk space will be used.

Do you want to continue? [Y/n] **Y**

Output truncated…

Processing triggers for ureadahead (0.100.0-19) ...

Processing triggers for ufw (0.35-0ubuntu2) ...

root@container1:~#

3. A firewall application name Uncomplicated Firewall (ufw) exists in the container. Check which applications are available to ufw, and check the status of ufw to see if it is protecting those applications:

root@container1:~# **sudo ufw app list**

Available applications:

Nginx Full

Nginx HTTP

Nginx HTTPS

OpenSSH

root@container1:~# **sudo ufw status**

Status: inactive

root@container1:~#

The above output shows that ufw is not active. If on your system, it is active, and firewall rules are in effect, see Chapter W26 at the book website for instructions on disabling ufw, and/or modifying its rules.

4. Check the status of the nginx service:

root@container1:~# **systemctl status nginx**

● nginx.service - A high performance web server and a reverse proxy server

Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: en

Active: active (running) since Tue 2016-09-20 05:07:52 UTC; 4min 30s ago

Main PID: 1165 (nginx)

CGroup: /system.slice/nginx.service

├─1165 nginx: master process /usr/sbin/nginx -g daemon on; master\_pro

├─1166 nginx: worker process

└─1167 nginx: worker process

Sep 20 05:07:52 container1 systemd[1]: Starting A high performance web server an

Sep 20 05:07:52 container1 systemd[1]: nginx.service: Failed to read PID from fi

Sep 20 05:07:52 container1 systemd[1]: Started A high performance web server and

root@container1:~#

What we see from the output of this command is that the installation of the service using apt-get not only installed it, but started it as well, and that the service is active and running.

5. You can now test the nginx web server by pointing your web browser to the URL 192.168.0.34. You will see the nginx welcome page displayed.

In-Chapter Exercise W23.5

Execute all of the steps of Example W23.3 on your own computer system to provision and start a new LXD container, so that the web server nginx is installed in it. Then do an initial test of the default nginx web page, either from a browser on another machine on your LAN, or from the Internet.

Example W23.4 Installing Webmin in an LXD Container:

Objectives: To allow administration of an LXD container from a web browser interface

Pre-Requisites: Completion of Examples W23.1 – W23.3, having a running LXD container as created in those examples. The assumption that we make here is that you are working on a Debian-famile Linux Mint system.

Background: An easy, reliable, and fast way of administrating LXD containers, one at a time and from inside the container, is to use a web browser interface known as Webmin. The advantage this has over using LXD/LXC commands from the host is that it can be done remotely, since our containers and LXD have been set up to have “public-facing” IP addresses that are exposed on your LAN or the Internet. Another advantage is that Webmin offers a GUI interface, with a very rich set of administrative options that you can perform on the container operating system, and on the files in the container.

*Please note that Steps 2 though 4 of this example can also be found in Appendix A, Section W26.1.2 of the printed book. The instructions found there are applicable to both Debian-family and CentOS systems.*

Requirements: Do the following steps, in the order presented below, to complete the requirements.

1. On the host, open a bash shell on the container you want to install Webmin in:

$ **sudo lxc exec container1 bash**

[sudo] password for bob: **yyyxxx**

2. On the container command line, get the Webmin package from its repository, using the following command:

root@container1:~# **wget http://prdownloads.sourceforge.net/webadmin/webmin\_1.810\_all.deb \**

**--2016-09-24 15:06:42-- http://prdownloads.sourceforge.net/webadmin/webmin\_1.810\_all.deb**

Resolving prdownloads.sourceforge.net (prdownloads.sourceforge.net)... 216.34.181.59

Connecting to prdownloads.sourceforge.net (prdownloads.sourceforge.net)|216.34.181.59|:80... connected.

Output truncated…

Saving to: ‘webmin\_1.810\_all.deb’

webmin\_1.810\_all.de 100%[===================>] 14.51M 573KB/s in 29s

2016-09-24 15:07:17 (512 KB/s) - ‘webmin\_1.810\_all.deb’ saved [15220142/15220142]

root@container1:~#

3. Get some additional modules that are dependencies for installation of Debian systems, such as Linux Mint, using the following command:

root@container1:~# **apt-get install perl libnet-ssleay-perl openssl \**

**libauthen-pam-perl libpam-runtime libio-pty-perl apt-show-versions python**

Reading package lists... Done

Building dependency tree

Reading state information... Done

After this operation, 18.3 MB of additional disk space will be used.

Output truncated…

Do you want to continue? [Y/n]**Y**

Output truncated…

Setting up libauthen-pam-perl (0.16-3build2) ...

Setting up libio-pty-perl (1:1.08-1.1build1) ...

Setting up libnet-ssleay-perl (1.72-1build1) …

root@container1:~#

4. Install the Webmin package into the container with the following command:

root@container1:~# **dpkg --install webmin\_1.810\_all.deb**

Selecting previously unselected package webmin.

(Reading database ... 26659 files and directories currently installed.)

Preparing to unpack webmin\_1.810\_all.deb ...

Unpacking webmin (1.810) ...

Setting up webmin (1.810) ...

Webmin install complete. You can now login to https://container1:10000/

as root with your root password, or as any user who can use sudo

to run commands as root.

Processing triggers for systemd (229-4ubuntu7) ...

Processing triggers for ureadahead (0.100.0-19) ...

root@container1:~#

5. Check the IP address of this container using the ip command, as seen from inside the container:

root@container1:~# **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state

UNKNOWN group default qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc

noqueue state UP group default qlen 1000

link/ether 00:16:3e:29:57:6b brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet 192.168.0.34/24 brd 192.168.0.255 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::216:3eff:fe29:576b/64 scope link

valid\_lft forever preferred\_lft forever

root@container1:~#

We see on our system that eth0 has an IP address on our LAN of 192.168.0.34 , which we will use to navigate to our container from the network.

6. Leave the container, and return to the host with the following command:

root@container1:~# **exit**

$

7. To use the Webmin interface, open your favorite web browser either on the host machine, or anywhere that you can access the IP address of the container from, and type the following into the URL locator field. Be sure to substitute the IP address for your container, that you found in step 5. Also, a number of connection error messages appear, as shown below:

https://192.168.0.34:10000

Your connection is not secure

The owner of 192.168.0.34 has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more…

Report errors like this to help Mozilla identify and block malicious sites

192.168.0.34:10000 uses an invalid security certificate.

The certificate is not trusted because it is self-signed.

The certificate is not valid for the name 192.168.0.34.

Error code: SEC\_ERROR\_UNKNOWN\_ISSUER

8. Click on the button Add Exception.

9. Click on the button Accept Certificate.

10. The Webmin login screen appears in your web browser. Login as root, with the root password.

The Webmin display appears on-screen, allowing you to do many common system administration tasks inside the container, from anwhere on your LAN or the Internet.

In-Chapter Exercise W23.6

Execute all of the steps of Example W23.4 to provision and start a new LXD container that you then install Webmin in. Test the Webmin interface, either from another machine on your LAN, or from the Internet, using a capable web browser.

Example W23.5 Setting Up LXD with a Public-Facing IP Address Assigned Via DHCP

Objectives: To initialize LXD in a newly-installed Ubuntu 18.04 Server edition, so that new LXD containers on the server have public-facing DHCP-supplied IP addresses.

Prerequisites: Completion of Examples W23.1 through W23.4, having the ISO DVD or USB media for Ubuntu 18.04 Server edition(non-alternate install), being able to use a spare physical hard disk that you want to use to install a fresh, bare metal version of Ubuntu 18.04 Server edition onto.

Also, the computer you work with here must be able to have an IP address assigned to it automatically from a DHCP server on your LAN or intranet.

Background: By default, the network connections that LXD containers come with out of the box are to private network addresses. It is very useful from a security perspective, as seen in previous Examples, to have new containers automatically assigned “public-facing” IP addresses on a LAN or intranet, so that outside traffic can be directed to these containers easily and safely. The technique we use here uses the more “modern” netplan facility to manage the network connections through systemd’s networkd renderer.

Procedures: Do the following steps, in the order presented, to fulfill the objectives of this Example:

1. Install the bridge-utils to allow you to create a “bridged” network connection between your computers Network Interface Card (NIC) and the LXD containers.

$ **sudo apt-get install bridge-utils**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

bridge-utils

Output truncated…

2. Check the IP address of your computer system. Notice here that our system was automatically assigned an IP address of 192.168.0.6 by the DHCP server.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic enp2s0

valid\_lft 604327sec preferred\_lft 604327sec

inet6 fe80::e611:5bff:fe12:c277/64 scope link

valid\_lft forever preferred\_lft forever

$

3. Edit the netplan configuration file, and establish the means of assigning the bridged connection. We use nano here, but you can use your favorite text editor to modify the default configuration file to suit your IP addresses as seen in Step 2.

$ **sudo** **nano /etc/netplan/01-netcfg.yaml**

# This file is generated from information provided by

# the datasource. Changes to it will not persist across an instance.

# To disable cloud-init's network configuration capabilities, write a file

# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:

# network: {config: disabled}

network:

version: 2

renderer: networkd

ethernets:

enp2s0:

dhcp4: false

bridges:

br0:

interfaces: [enp2s0]

dhcp4: false

addresses: [192.168.0.6/24]

gateway4: 192.168.0.1

nameservers:

addresses: [1.1.1.1,8.8.8.8]

parameters:

forward-delay: 0

Then save and exit.

$

4. Initialize the network connection with the new configuration file.

$ **sudo netplan apply**

5. Recheck the IP addresses. Notice that the bridge br0 has now taken the duly assigned IP address.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether fe:d5:a9:1f:17:0b brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::fcd5:a9ff:fe1f:170b/64 scope link

valid\_lft forever preferred\_lft forever

$

6. Initialize lxd, with the parameters shown. Notice that on the Ubuntu 18.04 Server edition, LXD was already installed. Also, **<Enter>** signifies pressing the Enter key on the keyboard.

$ **sudo lxd init**

Would you like to use LXD clustering? (yes/no) [default=no]: **<Enter>**

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

Name of the new storage pool [default=default]: **<Enter>**

Name of the storage backend to use (btrfs, dir,lvm) [default=btrfs]: **dir <Enter>**

Would you like to connect to a MAAS server? (yes/no) [default=no]: **<Enter>**

Would you like to create a new network bridge? (yes/no) [default=yes]: **no <Enter>**

Would you like to configure LXD to use an existing bridge or host interface? (yes/no) [default=no]: **yes <Enter>**

Name of the existing bridge or host interface: **br0 <Enter>**

Is this interface connected to your MAAS server? (yes/no) [default=yes]: **no <Enter>**

Would you like LXD to be available over the network? (yes/no) [default=no]: **<Enter>**

Would you like stale cached images to be updated automatically? (yes/no) [default=yes] **<Enter>**

Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]: **<Enter>**

$

7. Launch an LXD container, based upon the Ubuntu 18.04 template, named “container2”.

$ **sudo lxc launch ubuntu:18.04 container**2

If this is your first time running LXD on this machine, you should also run: lxd init

Creating container2

Output truncated…

Starting container2

$

8. Check to see what LXD containers are running, and most importantly, what their IP addresses are!

$ **sudo lxc list**

+------------+---------+---------------------+------+------------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+------------+---------+---------------------+------+------------+-----------+

| container2 | RUNNING | 192.168.0.47 (eth0) | | PERSISTENT | 0 |

+------------+---------+---------------------+------+------------+-----------+

9. Recheck the IP addresses of your network connections.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether fe:d5:a9:1f:17:0b brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::fcd5:a9ff:fe1f:170b/64 scope link

valid\_lft forever preferred\_lft forever

5: veth6ET1TU@if4: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000

link/ether fe:2a:81:02:d6:00 brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet6 fe80::fc2a:81ff:fe02:d600/64 scope link

valid\_lft forever preferred\_lft forever

$

Conclusion: This Example initialized LXD in a newly-installed Ubuntu 18.04 Server edition, so that new LXD containers have public-facing DHCP-supplied IP addresses. Of particular importance was the use of netplan and the systemd networkd facility to manage network connections.

Example W23.6 Setting Up LXD with a Public-Facing IP Address Assigned Via DHCP

Objectives: To initialize LXD in a newly-installed Ubuntu 18.04 Desktop edition, so that new LXD containers have public-facing DHCP-supplied IP addresses.

Prerequisites: Completion of Examples W23.1 through W23.4, having the ISO DVD or USB media for Ubuntu 18.04 Desktop edition, being able to use a spare physical hard disk that you want to use to install a fresh, bare metal version of Ubuntu 18.04 Desktop edition onto.

Also, the computer you work with here must be able to have an IP address assigned to it automatically from a DHCP server on your LAN or intranet.

Background: By default, the network connections that LXD containers come with out of the box are to private network addresses. It is very useful from a security perspective, as seen in previous Examples, to have new containers automatically assigned “public-facing” IP addresses on a LAN or intranet, so that outside traffic can be directed to these containers easily and safely. The technique we use here uses the legacy /etc*/*network/interfaces file and ifdown facility to manage the network connections through the Network Manager, which are all installed by default on the Desktop edition of Ubuntu 18.04.

Procedures: Do the following steps, in the order presented, to fulfill the objectives of this Example:

1. Install the bridge-utils to allow you to create a “bridged” network connection between your computers Network Interface Card (NIC) and the LXD containers.

$ **sudo apt-get install bridge-utils**

[sudo] password for bob:

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

bridge-utils

0 upgraded, 1 newly installed, 0 to remove and 19 not upgraded.

Output truncated…

$

2. Check the IP address of your computer system. Notice here that our system was automatically assigned an IP address of 192.168.0.6 by the DHCP server.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic noprefixroute enp2s0

valid\_lft 603153sec preferred\_lft 603153sec

inet6 fe80::5806:f15a:9ca4:a354/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

$

3. Edit the /etc/network/interfaces configuration file, and establish the means of assigning the bridged connection. We use nano here, but you can use your favorite text editor to modify the default configuration file to suit your IP addresses as seen in Step 2.

$ **sudo nano /etc*/*network/interfaces**

# interfaces(5) file used by ifup(8) and ifdown(8)

# The loopback network interface

auto lo

iface lo inet loopback

# The primary network interface

auto br0

iface br0 inet static

address 192.168.0.6

netmask 255.255.255.0

network 192.168.0.0

broadcast 192.168.0.255

gateway 192.168.0.1

# dns-\* options are implemented by the resolvconf package, if installed

dns-nameservers 8.8.8.8 8.8.4.4 #google-dns

dns-search localdomain.local #optional-line

# bridge options

bridge\_ports enp2s0

iface enp2s0 inet manual

Then save and exit.

$

4. Initialize the network connection with the new configuration file.

$ **sudo ifdown enp2s0 && sudo ifup enp2s0 && sudo ifup br0**

ifdown: interface enp2s0 not configured

Waiting for br0 to get ready (MAXWAIT is 32 seconds).

$

5. Recheck the IP addresses. Notice that the bridge br0 has now taken the duly assigned IP address.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic noprefixroute enp2s0

valid\_lft 603883sec preferred\_lft 603883sec

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::e611:5bff:fe12:c277/64 scope link

valid\_lft forever preferred\_lft forever

$

6. Install LXD, and initialize it with the parameters shown. Notice that on Ubuntu 18.04 Desktop edition, LXD was not installed by default. Also, **<Enter>** signifies pressing the Enter key on the keyboard.

$ **sudo apt-get install lxd**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

ebtables liblxc-common liblxc1 lxcfs lxd-client uidmap xdelta3

Suggested packages:

criu lxd-tools

The following NEW packages will be installed:

ebtables liblxc-common liblxc1 lxcfs lxd lxd-client uidmap xdelta3

0 upgraded, 8 newly installed, 0 to remove and 19 not upgraded.

Output truncated…

$

$ **sudo lxd init**

Would you like to use LXD clustering? (yes/no) [default=no]: **<Enter>**

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

Name of the new storage pool [default=default]: **<Enter>**

Would you like to connect to a MAAS server? (yes/no) [default=no]: **<Enter>**

Would you like to create a new network bridge? (yes/no) [default=yes]: **no <Enter>**

Would you like to configure LXD to use an existing bridge or host interface? (yes/no) [default=no]: **yes <Enter>**

Name of the existing bridge or host interface: **br0 <Enter>**

Is this interface connected to your MAAS server? (yes/no) [default=yes]: **no <Enter>**

Would you like LXD to be available over the network? (yes/no) [default=no]: **<Enter>**

Would you like stale cached images to be updated automatically? (yes/no) [default=yes] **<Enter>**

Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]: **<Enter>**

$

7. Launch an LXD container, based upon the Ubuntu 18.04 template, named “container3”.

$ s**udo lxc launch ubuntu:18.04 container3**

If this is your first time running LXD on this machine, you should also run: lxd init

To start your first container, try: lxc launch ubuntu:16.04

Creating container3

Output truncated…

Starting container3

$

8. Check to see what LXD containers are running, and most importantly, what their IP addresses are!

$ **sudo lxc list**

+------------+---------+---------------------+------+------------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+------------+---------+---------------------+------+------------+-----------+

| container3 | RUNNING | 192.168.0.48 (eth0) | | PERSISTENT | 0 |

+------------+---------+---------------------+------+------------+-----------+

$

9. Recheck the IP addresses of your network connections.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic noprefixroute enp2s0

valid\_lft 601930sec preferred\_lft 601930sec

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::e611:5bff:fe12:c277/64 scope link

valid\_lft forever preferred\_lft forever

5: vethDVFWMY@if4: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000

link/ether fe:b4:31:b8:06:3a brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet6 fe80::fcb4:31ff:feb8:63a/64 scope link

valid\_lft forever preferred\_lft forever

$

Conclusion: We initialized LXD in a newly-installed Ubuntu 18.04 Desktop Edition, so that new LXD containers have public-facing DHCP-supplied IP addresses.

Example W23.7 Using a ZFS “Loopback” Device as a Backing Store in LXD Containers

Objectives: To initialize LXD in a newly-installed Ubuntu 18.04 Desktop edition, so that new LXD containers use ZFS as the storage system. Additionally, each of the LXD containers are configured to have a “public-facing” IP address assigned by the DHCP server on your LAN or intranet.

Pre-Requisites: This example can be done independently of Examples W23.1 through W23.6., although some of the Requirement steps are repeats of things done in those previous Examples. It is highly advisable to have completed Chapter W22 on ZFS before attempting this Example, but not essentially necessary.

Background: Using ZFS as the storage back end for LXD containers is highly recommended. This Example show how to achieve that using a ZFS “loopback” device automatically created by LXD during initialization.

Requirements: Do the following steps, in the order presented, to fulfill the objectives of this Example:

0. Install ZFS.

$ **sudo apt-get install zfsutils-linux**

Reading package lists... Done

Building dependency tree

Reading state information... Done

[...]

The following NEW packages will be installed:

libnvpair1linux libuutil1linux libzfs2linux libzpool2linux zfs-doc zfs-zed

zfsutils-linux

0 upgraded, 7 newly installed, 0 to remove and 16 not upgraded.

Output truncated...

1. Install the bridge-utils to allow you to create a “bridged” network connection between your computers Network Interface Card (NIC) and the LXD containers.

$ **sudo apt-get install bridge-utils**

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

bridge-utils

0 upgraded, 1 newly installed, 0 to remove and 19 not upgraded.

Output truncated…

$

2. Check the IP address of your computer system. Notice here that our system was automatically assigned an IP address of 192.168.0.6 by the DHCP server.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eno1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether e4:11:5b:12:c2:77 brd ff:ff:ff:ff:ff:ff

inet 192.168.0.6/24 brd 192.168.0.255 scope global dynamic noprefixroute enp2s0

valid\_lft 603153sec preferred\_lft 603153sec

inet6 fe80::5806:f15a:9ca4:a354/64 scope link noprefixroute

valid\_lft forever preferred\_lft forever

$

3. Edit the /etc/network/interfaces configuration file, and establish the means of assigning the bridged connection. We use nano here, but you can use your favorite text editor to modify the default configuration file to suit your IP addresses as seen in Step 2.

$ **sudo nano /etc*/*network/interfaces**

# ifupdown has been replaced by netplan(5) on this system. See

# /etc/netplan for current configuration.

# To re-enable ifupdown on this system, you can run:

# sudo apt install ifupdown

# The primary network interface

auto br0

iface br0 inet dhcp

bridge\_ports eno1

iface eno1 inet manual

4. Initialize the network connection with the new configuration file.

$ **sudo ifdown eno1 && sudo ifup eno1 && sudo ifup br0**

ifdown: interface eno1 not configured

Waiting for br0 to get ready (MAXWAIT is 32 seconds).

$

5. Optional- If LXD is not installed, install it with the following command-

$ **sudo apt-get install lxd**

Reading package lists... Done

Building dependency tree

Reading state information... Done

Suggested packages:

criu lxd-tools

The following NEW packages will be installed:

lxd

0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.

Need to get 5,056 kB of archives.

After this operation, 19.9 MB of additional disk space will be used.

Output truncated...

To go through the initial LXD configuration, run: lxd init

Processing triggers for libc-bin (2.27-3ubuntu1) ...

Processing triggers for systemd (237-3ubuntu10) ...

Processing triggers for man-db (2.8.3-2) ...

Processing triggers for ureadahead (0.100.0-20) ...

6. Check the IP address of the system.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eno1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether b8:ac:6f:9a:80:dc brd ff:ff:ff:ff:ff:ff

inet6 fe80::baac:6fff:fe9a:80dc/64 scope link

valid\_lft forever preferred\_lft forever

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether b8:ac:6f:9a:80:dc brd ff:ff:ff:ff:ff:ff

inet 192.168.0.8/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::baac:6fff:fe9a:80dc/64 scope link

valid\_lft forever preferred\_lft forever

7. *This is the most critical step.* Initialize LXD so that the backing storage used by the containers will be a ZFS loopback device.

$ **sudo lxd init**

Would you like to use LXD clustering? (yes/no) [default=no]: **<Enter>**

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

Name of the new storage pool [default=default]: **<Enter>**

Name of the storage backend to use (btrfs, dir, lvm, zfs) [default=zfs]: **<Enter>**

Create a new ZFS pool? (yes/no) [default=yes]: **<Enter>**

Would you like to use an existing block device? (yes/no) [default=no]: **<Enter>**

Size in GB of the new loop device (1GB minimum) [default=100GB]: **<Enter>**

Would you like to connect to a MAAS server? (yes/no) [default=no]: **<Enter>**

Would you like to create a new network bridge? (yes/no) [default=yes]: **no <Enter>**

Would you like to configure LXD to use an existing bridge or host interface? (yes/no) [default=no]: **yes <Enter>**

Name of the existing bridge or host interface: **br0 <Enter>**

Is this interface connected to your MAAS server? (yes/no) [default=yes]: **no <Enter>**

Would you like LXD to be available over the network? (yes/no) [default=no]: **<Enter>**

Would you like stale cached images to be updated automatically? (yes/no) [default=yes] **<Enter>**

Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]: **<Enter>**

8. Launch a Ubuntu 18.04 container named “container3”.

$ **sudo lxc launch ubuntu:18.04 container3**

Creating container3

Starting container3

9. Check the IP addresses again.

$ **ip addr show**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eno1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq master br0 state UP group default qlen 1000

link/ether b8:ac:6f:9a:80:dc brd ff:ff:ff:ff:ff:ff

inet6 fe80::baac:6fff:fe9a:80dc/64 scope link

valid\_lft forever preferred\_lft forever

3: br0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether b8:ac:6f:9a:80:dc brd ff:ff:ff:ff:ff:ff

inet 192.168.0.8/24 brd 192.168.0.255 scope global br0

valid\_lft forever preferred\_lft forever

inet6 fe80::baac:6fff:fe9a:80dc/64 scope link

valid\_lft forever preferred\_lft forever

7: vethIX7UQR@if6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue master br0 state UP group default qlen 1000

link/ether fe:f4:23:c6:42:85 brd ff:ff:ff:ff:ff:ff link-netnsid 0

inet6 fe80::fcf4:23ff:fec6:4285/64 scope link

valid\_lft forever preferred\_lft forever

10. Check the zpools on the system.

$ **sudo zpool list**

NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALTROOT

default 99.5G 338M 99.2G - 0% 0% 1.00x ONLINE -

11. List the LXD containers.

$ **sudo lxc list**

+------------+---------+---------------------+------+------------+-----------+

| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |

+------------+---------+---------------------+------+------------+-----------+

| container3 | RUNNING | 192.168.0.48 (eth0) | | PERSISTENT | 0 |

+------------+---------+---------------------+------+------------+-----------+

12. Check the ZFS datasets on the system.

$ **sudo zfs list**

NAME USED AVAIL REFER MOUNTPOINT

default 338M 96.1G 24K none

default/containers 6.99M 96.1G 24K none

default/containers/container3 6.96M 96.1G 331M /var/lib/lxd/storage-pools/default/containers/container3

default/custom 24K 96.1G 24K none

default/deleted 24K 96.1G 24K none

default/images 331M 96.1G 24K none

default/images/b36ec... 331M 96.1G 331M none

default/snapshots 24K 96.1G 24K none

$

Conclusion: We initialized LXD in a newly-installed Ubuntu 18.04 Desktop edition, so that new LXD containers use a ZFS loopback device as the storage system. Additionally, each of the LXD containers was configured to have a “public-facing” IP address assigned by the DHCP server on your LAN or intranet. These two objectives can be easily extended to actual physical devices as vdevs, and those vdevs can be created as various levels of RAIDZ if desired. Or via the use of mdadm, as shown in Chapter W26, the system disk itself can be mirrored so that the LXD containers residing on the physical hard drive that is the system disk can be made redundant. Once the containers have a public-facing IP address, any number of applications, such as the nginx web server, can be installed in them and take advantage of this means of accessing the LAN, intranet, or Internet.

W23.2.7 LXC Installation and Basic Commands for CentOS 7.4

Objectives: To install the LXC container virtualization package on a CentOS7.4 system.

Prerequisites:

A “minimal” install of CentOS 7.4. Superuser privilege on the system.

Requirements: Do the following steps, in the order presented below, to achieve the objective of this example. All commands executed in this example are done as superuser.

Installing LXC

1. LXC virtualization is in the Epel repositories. If you have not done so already, execute the following command in a terminal to install the Epel repositories:

# **yum install epel-release**

2. Before continuing with LXC installation process, assure that Perl language interpreter, and debootstrap packages are installed by issuing the below commands.

# **yum install debootstrap perl libvirt**

3. Install LXC virtualization solution with the following command.

# **yum install lxc lxc-templates**

4. After LXC has been installed, use systemctl to start the services, and check that LXC and libvirt are running, by using the following commands:

# **systemctl status lxc.service**

# **systemctl start lxc.service**

# **systemctl start libvirtd**

# **systemctl status lxc.service**

Also check the LXC Kernel Virtualization Configuration, using the following command:

# lxc-checkconfig

Kernel configuration not found at /proc/config.gz; searching...

Kernel configuration found at /boot/config-3.10.0-229.el7.x86\_64

--- Namespaces ---

Namespaces: enabled

Utsname namespace: enabled

Ipc namespace: enabled

Output truncated...

Creating and Managing LXC Containers

5. To list available LXC templates containers already installed on your system, execute the following command. We show the entire output on our CentOS 7 system, so you can see the range of templates available.

# ls -alh /usr/share/lxc/templates/

total 344K

drwxr-xr-x. 2 root root 4.0K Dec 5 00:44 .

drwxr-xr-x. 6 root root 106 Dec 5 00:44 ..

-rwxr-xr-x. 1 root root 11K Oct 20 13:21 lxc-alpine

-rwxr-xr-x. 1 root root 14K Oct 20 13:21 lxc-altLinux

-rwxr-xr-x. 1 root root 11K Oct 20 13:21 lxc-archLinux

-rwxr-xr-x. 1 root root 9.5K Oct 20 13:21 lxc-busybox

-rwxr-xr-x. 1 root root 30K Oct 20 13:21 lxc-centos

-rwxr-xr-x. 1 root root 11K Oct 20 13:21 lxc-cirros

-rwxr-xr-x. 1 root root 18K Oct 20 13:21 lxc-debian

-rwxr-xr-x. 1 root root 18K Oct 20 13:21 lxc-download

-rwxr-xr-x. 1 root root 49K Oct 20 13:21 lxc-fedora

-rwxr-xr-x. 1 root root 28K Oct 20 13:21 lxc-gentoo

-rwxr-xr-x. 1 root root 14K Oct 20 13:21 lxc-openmandriva

-rwxr-xr-x. 1 root root 14K Oct 20 13:21 lxc-opensuse

-rwxr-xr-x. 1 root root 35K Oct 20 13:21 lxc-oracle

-rwxr-xr-x. 1 root root 12K Oct 20 13:21 lxc-plamo

-rwxr-xr-x. 1 root root 6.7K Oct 20 13:21 lxc-sshd

-rwxr-xr-x. 1 root root 24K Oct 20 13:21 lxc-ubuntu

-rwxr-xr-x. 1 root root 12K Oct 20 13:21 lxc-ubuntu-cloud

6. The general syntactic format of the command to create a new container is as follows:.

lxc-create -n container\_name -t container\_template

Create a new container named myubuntu based on an ubuntu template that is found in the LXC repositories, as seen in the previous output..

# **lxc-create -n myubuntu -t ubuntu**

Checking cache download in /var/cache/lxc/precise/rootfs-amd64 ...

Installing packages in template: ssh,vim,language-pack-en

Downloading ubuntu precise minimal ...

Output truncated…

##

# The default user is 'ubuntu' with password 'ubuntu'!

# Use the 'sudo' command to run tasks as root in the container.

##

#

7. When the above command finishes, a message will display your default root account password. As seen above, in our case the username was ubuntu, and the password was set at ubuntu as well. Optionallay at this point, you can change this password once you start and login to the container console, for security reasons.

8. Now, you can use lxc-ls to list your containers and lxc-info to obtain information about a running/stopped container. lxc-ls is part of the lxc-extra package. So to install it, use the following command:

# **yum install lxc-extra**

In order to start the newly created container in background ( we make it run as a daemon, because we specify the -d option) issue the following command:

# **lxc-start -n myubuntu -d**

9. After the container has been started you can list running containers using the lxc-ls --active command and get detailed information about the running container.

# **lxc-ls –active**

myubuntu

10. The easiest and quickest way to get “into” the container is to open a console on it, using the lxc-console command. Login with the user root and the password generated by default by lxc supervisor.

Once logged in the container you can run several commands in order to verify the distribution by displaying the /etc/issue.net file content, change the root password by issuing passwd command or view details about network interfaces with ip addr show command.

# lxc-console -n myubuntu

Ubuntu 12.04.5 LTS myubuntu tty1

myubuntu login: **ubuntu**

Password: **ubuntu**

Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.10.0-693.5.2.el7.x86\_64 x86\_64)

\* Documentation: https://help.ubuntu.com/

This Ubuntu 12.04 LTS system is past its End of Life, and is no longer

receiving security updates. To protect the integrity of this system, it’s

critical that you enable Extended Security Maintenance updates:

\* https://www.ubuntu.com/esm

The programs included with the Ubuntu system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by

applicable law.

Would you like to enter a security context? [N] **<Enter>**

ubuntu@myubuntu:~$# cat /etc/issue.net

Ubuntu 12.04.5 LTS

ubuntu@myubuntu:~$# ip addr show

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP qlen 1000

link/ether 00:16:3e:58:76:9e brd ff:ff:ff:ff:ff:ff

inet 192.168.122.57/24 brd 192.168.122.255 scope global eth0

valid\_lft forever preferred\_lft forever

inet6 fe80::216:3eff:fe58:769e/64 scope link

valid\_lft forever preferred\_lft forever

ubuntu@myubuntu:~$# passwd root

Enter new Linux password: **qqq**

Retype new Linux password: **qqq**

passwd: password updated successfully

11. To detach from the container console and go back to your host console, leaving the container in active state, hit <Ctrl>+<A> then <Q> on the keyboard.

To stop the a running container issue the following command.

# **lxc-stop -n myubuntu**

12. Containers can be deleted from your host with the lxc-destroy command issued against a container name.

# **lxc-destroy -n myubuntu**

Conclusion: This Example allowed you to install the LXC container virtualization package on a CentOS7.4 system, and start an Ubuntu container on that system.

W23.3 VirtualBox

VirtualBox is a software virtualization tool that can be used to install other OSs as guests

on a host computer system that is running Linux Mint. We first discuss what advantages the VirtualBox method of virtualization provides, and then define some basic terms used in it. Finally, using an example-based format, we detail the installation and basic usage of VirtualBox on Linux Mint.

Why use VirtualBox virtualization?

The advantages of the kind of virtual environment that VirtualBox gives you are:

You can run multiple OSs simultaneously. VirtualBox allows you to run more than

one OS at a time. You can switch between OS environments by simply moving your

mouse into the window that contains one of the operating systems!

From the system security point of view, you can isolate a particular application, such

as a Web or FTP site inside a VirtualBox guest, and anything that intrudes on

its operation from the Internet only infects the guest system.

VirtualBox is similar in this respect to using LXD, as shown in Section W23.2.

Once installed, a VM and its virtual hard disks can be considered a “sandbox” that

can be arbitrarily frozen, unfrozen, copied, backed up, and moved between hosts.

With “snapshots,” you can save a particular state of a VM and revert back to that

state, if necessary.

When dealing with VirtualBox, the following basic terminology is important:

Host OS: This is the operating system of the physical computer on which VirtualBox

was installed. In our next examples, the hosts can be either Linux Mint or Windows.

We show VirtualBox on Linux and Windows host computers because there are many

advantages to using these systems as hosts, and deploying other operating systems on them

as guests. We speak about some of these advantages in detail.

Guest OS: This is the operating system that is running inside the VM. VirtualBox can

run any Intel or AMD x86 OS, but only officially supports and is optimized for a

limited number of operating systems. As we show here, it is pos-

sible to use Linux and Windows computers as hosts to deploy

guest operating systems on them.

VM: This is special environment that VirtualBox creates for your guest OS while it

is running. You run your guest OS via the use of a VM. A VirtualBox VM appears as

a window on your computer’s desktop.

If you install VirtualBox as a host on a Linux or other computer, you

can install a wide variety of operating systems as a VM on that host computer very easily.

In fact, we show examples of just such a situation.

Two ways of controlling the settings of the VM are via the VirtualBox manager window,

and via the VBoxManage command line program. A VM is essentially what you can

manage and work with via either GUI or command-line in these two controlling facilities.

Guest additions: This refers to special software packages which are shipped with

VirtualBox but designed to be installed inside a VM to improve performance of the

guest OS and to add extra features. A good example of this is the setting for a bidi-

rectional system paste buffer, which allows you to copy text in a guest window, and

paste it into a host window, or vice versa. We show how to do this in example form.

We found that on a Linux Mint host, we needed to completely create a new

VM and start it before we could enable the guest additions for that VM. This is done

after the VM has started by making the VirtualBox menu choice “Devices > Insert

Guest Additions CD”. VirtualBox will then place/mount a “virtual” CD on the desktop

of the VM, and if you click on it to launch the CD, you can install the contents of the

virtual CD.

W23.3.1 Installing and Running VirtualBox on a Linux Mint 18.2 Host OS

As of the latest Linux Mint available at the time of the writing of this book (Linux Mint 18.2 Sonya), VirtualBox does not come pre-installed. But VirtualBox can be quickly and easily installed using the Linux Mint Software Manager. Refer to Appendix A, Section W23.3.1 for details of how to install new software on your Debian-family or CentOS Linux system. On our Linux Mint system, we installed the VirtualBox (base) and VirtualBox (qt-based user interface) via the Software Manager. There were already some of the VirtualBox modules pre-installed.

To then run VirtualBox, from the Linux Mint Menu, make the choice Accessories > VirtualBox.

Additionally, we also put an icon for VirtaulBox on our desktop so that we could run it at any time by double-clicking on the icon.

W23.3.2 Installing a VM Guest

With VirtualBox installed on your host system, there is a very patented and universal way

of creating VMs and then installing other OSs in those VMs. The following example shows

the steps necessary to install Ubuntu 16.04 in a guest VM, running on a Linux Mint host sys-

tem. The technique shown is easily extended to allow you to install any supported OS as

a guest VM in any supported host environment. We show examples of this scenario in

Section W23.3.5.

Example W23.8 Creating an Ubuntu 16.04 Virtual Machine Guest on a Linux Mint Host

Objectives: Using VirtualBox that has been installed on a Linux Mint host to create a VM guest .

Introduction: The advantages of running two different OS kernels simultaneously on the same hardware are evident when you can utilize the stronger features of one or more of them in virtual environments to provide secure services over a LAN or the Internet. One of the lessons to be learned from this example is that resources such as memory and disk space are shared between host and guest(s), thereby affecting the performance of each.

We chose this particular flavor of VM guest because it is the “parent” of Linux Mint, and you can use a vast majority of the same commands and operations in Ubuntu 16.04 as you can in Linux Mint. Also, if you could install Linux Mint on your computer system, the chances are that you can also install Ubuntu 16.04 on the hardware. That might not be so if you choose to install another flavor of Linux as a VM guest.

Prerequisites:

1. You must have VirtualBox installed on your Linux Mint system, and be able to run it.

2. You must have downloaded and burned to DVD the installation media, in the form of a DVD or an International Organization for Standardization (ISO) file for Ubuntu 16.04, or any other supported OS you want to substitute for it in the VM guest we create in this example.

Or you can use the ISO DVD medium for any other distribution version of Ubuntu that is current at the time you are doing this Example.

Procedures: Do the following steps, in the order presented here, to meet the objectives of this example.

1. Once installed on Linux Mint, run VirtualBox. When VirtualBox launches for the first time, the screen shown in Figure W23.1 appears.

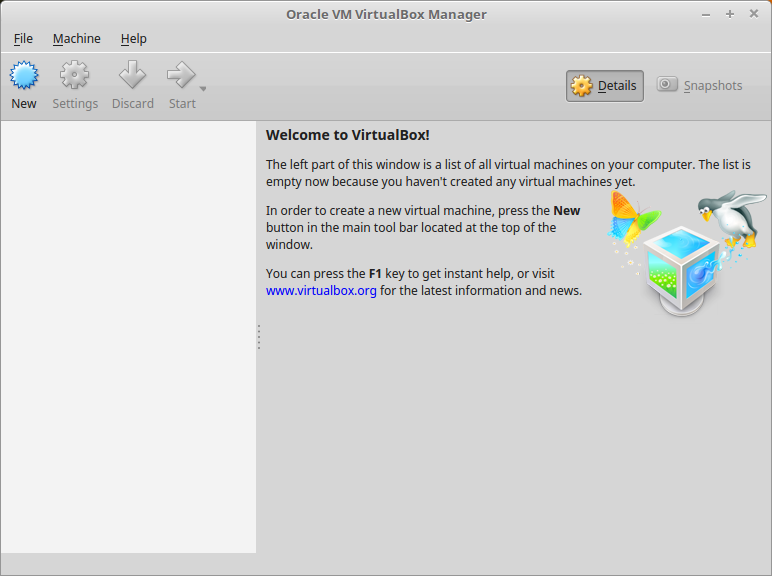


FIGURE W23.1 VirtualBox VM manager screen.

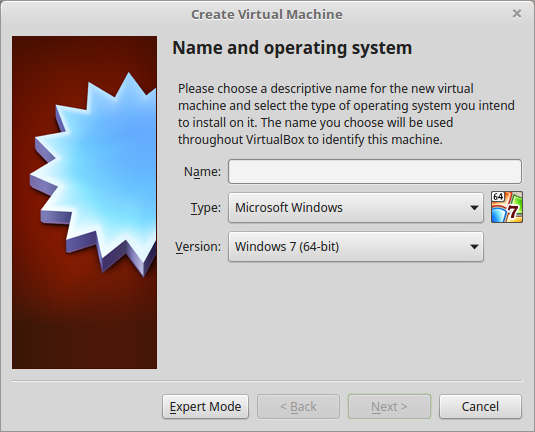


FIGURE W23.2 Creating Virtual Machine Wizard.

In any of the VirtualBox manager or wizard screens, you can always click the “< Back” button to go back to the previous step, or click the “Cancel” button to terminate the creation of a new guest VM.

2. To create a new guest VM, click the “New” button to start the “Create Virtual Machine” wizard, as seen in Figure W23.2.

3. Since our guest VM will be an installation of Ubuntu, type in Ubuntu in the Name field of the Create Virtual Machine window. In the Type: and Version: fields, Linux and Ubuntu (64-bit) have been pre-selected. Click the “Next >” button to continue.

The base memory size must be changed to at least 768 MB. If your system has a lot of RAM, use more. In our case, we used 2000 MB. Any number within the green area below the slider is considered a value that should not slow down your computer too much. Click the “Next >” button to continue.

4. This next screen is used to create the virtual hard drive, the amount of disk space on the host OS hardware that will be available to the VM guest. If this is your first VM, keep the default of “Create a virtual hard drive now” and click “Create” to continue.

5. On the next screen, choose the pre-selected “VDI (VirtualBox Disk Image)” and click the “Next” button.

6. You can now choose whether you want “Dynamically allocated” or “Fixed size” storage. The first option uses disk space as needed until it reaches the maximum size that you will set in the next screen. The second option creates a disk the same size as that specified amount of disk space, whether it is used or not. Choose the first option if you are concerned about disk space; otherwise, choose the second option as it allows VirtualBox to run slightly faster. Click the “Next >” button to continue.

7. You can now choose to set the size (or upper limit) of the VM. If you plan to install Ubuntu into the VM, increase the size to at least 8 GB or you will receive an error during the Ubuntu installation. We typed-in 100.0 GB for the disk size, and went with the default Ubuntu folder location. Whatever size you set, make sure that your computer has enough free disk space to support it. If you desire, use the folder icon to browse to a directory on disk with sufficient space to hold your VM.

8. Once you make your selection and press “Next”, you will see a summary of your choices. You can use the “Back” button to return to a previous screen if you wish to change any values. Otherwise, click “Create” to close the wizard. Your VM should now show up in the left-hand box of the VirtualBox VM Manager screen, as seen in example in Figure W23.3.

9. Configuring the storage device: On our installation, this step was automatically configured when we wanted to install Ubuntu from a DVD in a drive already mounted on the system at Serial Advanced Technology Attachment (SATA) port 1.

You may want to configure it to use another installation medium, depending on what that is. To do this, make menu choice in the Oracle VM VirtualBox Manager window Settings > Storage.

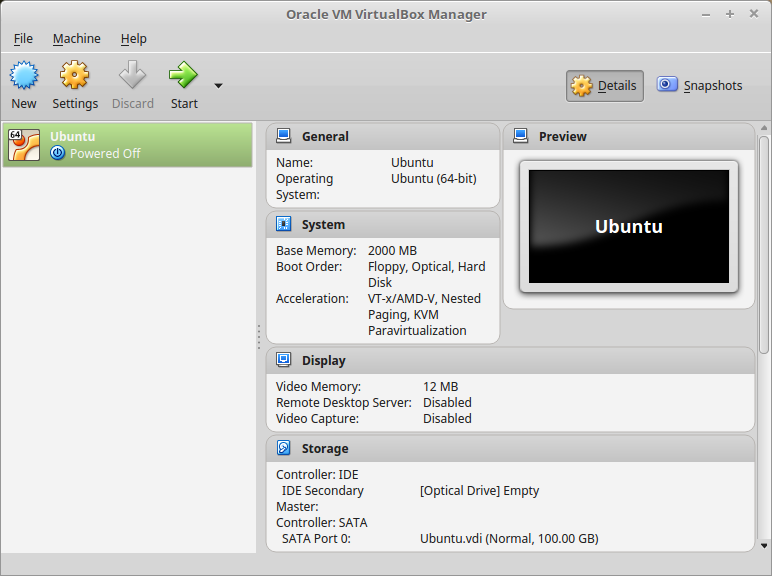


FIGURE W23.3 VirtualBox VM Manager Screen with Ubuntu VM Shown

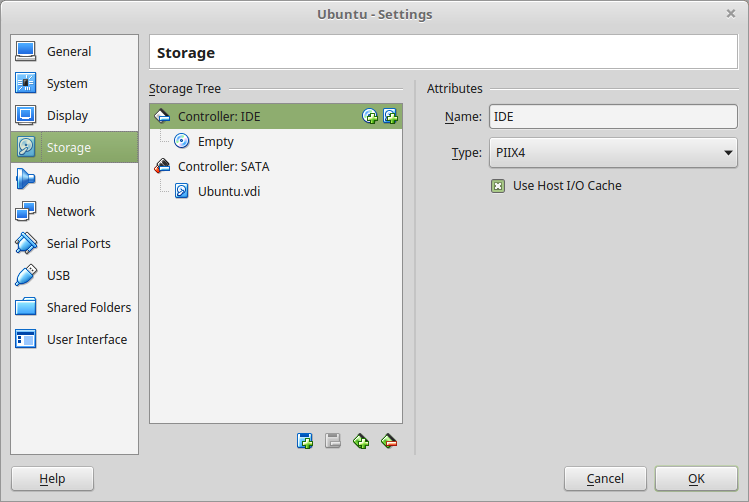


FIGURE W23.4 Storage settings of the VM.

frame of the VirtualBox VM Manager window. This allows you to access the storage screen seen in Figure W23.4. Double-click the word “Empty”, which represents your DVD reader. If you wish to access the Ubuntu installer from your DVD reader, double-check that the slot is pointing to the correct location (e.g., Controller: IDE) and use the drop-down menu to change it if the location is incorrect. Click the “CD/DVD Device” drop-down menu to change it from “Empty” to the correct host drive value.

If you prefer to use an ISO file that is stored on your hard disk, click the DVD icon “-> Choose a virtual CD/DVD disk file” to open a browser menu where you can navigate to the location of the ISO file. Highlight the desired ISO file and click “Open”. The name of the ISO file will now appear in the “Storage Tree” section.

10. Insert an Ubuntu 16.04 DVD (or the ISO DVD medium for any other distribution version of Ubuntu that is current at the time you are doing this Example) into your DVD drive, and let it spin up. You have to make the Cancel choice when Linux Mint asks yu what to do with this media. You are now ready to install Ubuntu into your VM. Simply highlight the VM and click on the green “Start” button. A window will open, indicating that the VM is starting. If you have a correct and useable DVD inserted, you should hear it spin and it should start to boot into the installation program. In the Select start-up disk window that appears, your DVD should be highlighted. Choose Start. If it does not, or if you are using an ISO file stored on the hard disk, press <F12> to select the boot device when you see the message to do so, then press C to boot.

11. Go through the installation steps for Ubuntu 16.04, which are self-explanatory. If you are installing another Flavor of Linux, use the installation instructions for that flavor. When the Ubuntu installation is completed, reboot the Ubuntu guest (remembering to remove the DVD medium before the reboot takes effect!). You may have to completely close down the VirtualBox VM guest instance, and then make the Start choice from the VirtualBox Manager window to do an initial boot into the system.

12. Work inside of your Ubuntu 16.04 VM!

13. To close down both the VirtualBox Ubuntu 16.04 guest and the VirtualBox program itself, first make the Ubuntu pull-down menu choice (as seen in the extreme upper-right of the Ubuntu window) “System >Shutdown...”. Then, after Ubuntu has completely shut down, from the VirtualBox Manager window, make the pull-down menu choice “File>Exit”.

Conclusions: Making the proper configuration choices when installing a guest operating into a VM affects the performance of that OS significantly. You must be aware of memory and disk capacities on your host machine to optimize the performance of both host and guest.

W23.3.4 Securing a vsftpd Server in a VirtualBox Guest

The following example shows how to take advantage of having a VM running on your host

machine. In it, we secure a vsftpd server by isolating it in a VirtualBox VM guest. The host

in the example is a Linux Mint system with VirtualBox already installed, and an Ubuntu guest OS

acting as the secured FTP server on a LAN.

This does not preclude you using any supported OS as the host, and installing any other supported OS as the guest on that host. We have chosen to show this methodology with these particular example systems to illustrate the process with two Linux systems. The particulars of accomplishing the same thing with other OSs are left up to you.

Example W23.9 Installing and Setting up a Service in a VirtualBox VM Guest

Objectives: To secure an vsftpd server on Ubuntu 16.04 running as a VirtualBox VM guest on

a Linux Mint host machine.

Introduction: This method, and the details of the following example, highlight the important use of virtualization-system security. By running a vsftpd server on a guest VM that is isolated from the host operating system, and is perhaps running a completely different kernel than the host system, you create a more secure host.

What we will accomplish in this example is as follows:

\* Assign an IP address, within a range of addresses, to the VM guest.

\* Check what the actual address assigned to our VM guest is.

\* Ensure that the VM guest is attached to a bridged network adapter on the host.

What that means is that our host NIC can now handle two IP addresses: the host IP address and the VM guest IP address.

\* Update and upgrade the packages on the Ubuntu 16.04 VM guest.

\* Install the vsftpd server in the Ubuntu 16.04 VM guest.

\* Test the FTP server on the VM guest via a LAN.

Prerequisites:

1. You must have completed Example W23.8.

2. You must make sure you have VirtualBox installed on your Linux Mint system, and know how to launch it.

3. You must have installed Ubuntu 16.04 as a VM guest, as shown in Example W23.8.

4. Your host machine must be connected to a LAN and the Internet.

*Please note that the Step 3. instruction for installing vsftpd can also be found in Appendix A, Section 17.2.3.4, in the printed book.*

Requirements: Do the following steps, in the order presented, to meet the requirements of this example:

1. To ensure that the guest VM IP address is in the subnetwork of the host, type the following in a Linux Mint host console window as superuser when the Ubuntu 16.04 VM guest is not running:

# **VBoxManage modifyvm "VMname" –natnet1 "192.168/16"**

where:

"VMname" is the name you gave the guest VM when you installed it.

This ensures that when the Ubuntu 16.04 guest VM is run, the IP address of the NIC will

be in the range of 192.168.0.0 to 192.168.254.254. In this example, the guest VM is

assigned an IP address of 192.168.0.28, and the host an IP address of 192.168.0.13.

2. What this critical step allows you to do is make the NIC that the Ubuntu 16.04 guest uses to communicate over the network available in a “bridged mode”, so that a DHCP server on your LAN assigns this Ubuntu 16.04 VM guest a public-facing IP address automatically.

Run VirtualBox, and make sure the Ubuntu 16.04 VM is highlighted. From the Oracle VirtualBox VM Manager menu at the top of the window, make the Settings menu choice. Click on the Network setting. Change the Attached to: choice to Bridged Adapter. Then make the OK choice in the Ubuntu-Settings window.

3. Start the Ubuntu 16.04 guest VM as shown in Example 24.8. In Ubuntu 16.04, hold down the Ctrl-Alt-T keys at the same time to get a terminal going. Then use the following commands in the terminal window to update the package manager, upgrade the installed packages, and download and install the vsftpd server :

$ **sudo apt update**

Output truncated...

$ **sudo apt upgrade**

Output truncated…

*Make sure you respond Y to the prompt asking you to continue! This command step usually takes some time, depending on the speed of your Internet connection, and also most importantly on how many packages must be downloaded and upgraded.*

$ **sudo apt-get install vsftpd**

[sudo] password for bob: **yyyxxx**

Output truncated…

Processing triggers for ureadahead (0.100.0-19) ...

Processing triggers for systemd (229-4ubuntu4) ...

$

4. Use the systemctl command to check the status of the vsftpd service, using the following command:

$ **sudo systemctl status vsftpd.service**

● vsftpd.service - vsftpd FTP server

Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)

Active: active (running) since Sat 2016-07-02 12:10:11 PDT; 2min 15s ago

Main PID: 8780 (vsftpd)

CGroup: /system.slice/vsftpd.service

└─8780 /usr/sbin/vsftpd /etc/vsftpd.conf

Jul 02 12:10:10 bob-VirtualBox systemd[1]: Starting vsftpd FTP server...

Jul 02 12:10:11 bob-VirtualBox systemd[1]: Started vsftpd FTP server.

$

Notice from the above output that the command from step 3. not only downloaded, but also installed and started, the vsftpd service.

5. Find out the IP address of the Ubuntu 16.04 guest VM by typing ip addr show on the terminal command line. For our machine, the IP adress of the machine using NIC enp0s3 was 192.168.0.30.

From another host machine on your network, use the ftp command to connect to the vsftpd server on your machine. Substitute the IP address of the machine you want to ftp into for the IP 192.168.0.30 shown in this command:

# **ftp 192.168.0.30**

Connected to 192.168.0.30.

220 (vsFTPd 3.0.3)

Name (192.168.0.30:bob): **bob**

331 Please specify the password.

Password: **yyyxxx**

230 Login successful.

Remote system type is Linux.

Using binary mode to transfer files.

6. Get a directory listing of the files on the machine your are connected to:

ftp> **ls**

*Here is a directory listing on the machine you connected to-*

200 PORT command successful. Consider using PASV.

150 Here comes the directory listing.

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Desktop

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Documents

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Downloads

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Music

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Pictures

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Public

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Templates

drwxr-xr-x 2 1000 1000 4096 Nov 23 11:57 Videos

-rw-r--r-- 1 1000 1000 8980 Nov 23 11:37 examples.desktop

226 Directory send OK.

ftp>

7. Terminate the connection with the following ftp command:

ftp> **exit**

221 Goodbye.

#

8. In order to stop the vsftpd service on the Ubuntu 16.04 Guest VM, use the following command:

$ **sudo systemctl stop vsftpd.service**

$

9. Check the status of the vsftpd service on the Ubuntu 16.04 Guest VM with the following command:

$ **sudo systemctl status vsftpd.service**

● vsftpd.service - vsftpd FTP server

Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)

Active: inactive (dead) since Sat 2016-07-02 13:09:59 PDT; 12s ago

Process: 9053 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)

Main PID: 8780 (code=killed, signal=TERM)

Jul 02 13:09:59 bob systemd[1]: Stopping vsftpd FTP server...

Jul 02 13:09:59 bob systemd[1]: Stopped vsftpd FTP server.

$

10. To restart the vsftpd service on the Ubuntu 16.04 Guest VM, use the following command:

$ **sudo systemctl restart vsftpd.service**

11. To make sure a service on the Ubuntu 16.04 Guest VM starts automatically at boot time, use the following command:

$ **sudo systemctl enable vsftpd.service**

Synchronizing state of vsftpd.service with SysV init with /lib/systemd/systemd-sysv-install...

Executing /lib/systemd/systemd-sysv-install enable vsftpd

$

12. To disable the vsftpd service on the Ubuntu 16.04 Guest VM from starting at boot, use the following command:

$ **sudo systemctl disable vsftpd.service**

Synchronizing state of vsftpd.service with SysV init with /lib/systemd/systemd-sysv-install...

Executing /lib/systemd/systemd-sysv-install disable vsftpd

insserv: warning: current start runlevel(s) (empty) of script `vsftpd' overrides LSB defaults (2 3 4 5).

insserv: warning: current stop runlevel(s) (0 1 2 3 4 5 6) of script `vsftpd' overrides LSB defaults (0 1 6).

$

Conclusion: This example allowed you to download, install, and start a system service known as vsftpd on a Ubuntu 16.04 guest VM. You were then able to connect to and use the Ubuntu machine as an ftp server. Additionally, we showed some basic systemd service management commands applied to the vsftpd service. More information about using systemd for service management may be found in Chapter W23.

In-Chapter exercise W23.7

Repeat Example W23.8 for an SSH service, to allow you to ssh from another system on your

network to the Ubuntu 16.04 (or later version of the Ubuntu distribution that you have available at the time you are doing this Exercise) VM guest running on a Linux Mint host.

Summary

In this chapter, we gave background information on operating system virtualization, and some of its characteristics and examples.

We provided a description of the LXD virtualization method, and explicitly detailed LXD installation on Linux Mint. We then illustrated LXD basic usage, and gave a command references for LXD. We listed LXD best practices, and provide complete worked examples of advanced usage of LXD, most notable how to install a container with a ZFS backing store, how to make an LXD container “web-facing”, and how to install a web server named nginx in an LXD container.

We then went on to describe and illustrate VirtualBox with different types of installation examples, particularly how to install VirtualBox on Linux Mint. We showed how to install an Ubuntu virtual machine as a VitualBox guest on a Linux Mint host. We also showed how to secure an FTP server in a VirtualBox guest.

Questions and Problems

W23.1 Following the sequence of commands exactly as shown in Section W23.2, create the snapshot structure of a running LXD container of your choice. Assign names of your choice to all snapshots and containers created. Then answer the following questions:

a. What is the IP address of the newly created “containerb” (or whatever you have chosen to name it) , in relation to the container that it was created from?

b. How do the commands in this section affect the IP address of newly created containers, and why?

W23.2 As an alternative method of initializing LXD, with the objective of setting up the ZFS storage file system for containers on a whole disk rather than on a file (as shown in Example W23.1), do the following:

a. Install an available and expendable second formatted hard disk, or USB thumb drive of sufficient capacity, on your computer. This disk can be mounted internally on the SATA bus, or externally on the USB bus. See Chapter 19 for instructions on how to recognize these two types of disk when they appear mounted in /dev.

b. Omit Example W23.1, step 2, and modify step 3. so that the zpool named lxd\_pool is created on the second hard disk, rather than on a file on the primary system disk. On our system, this was done with the command **zpool create lxd\_pool /dev/sdb1**

c. Proceed through the rest of Example W23.1, and the remaining Examples in the chapter.

W23.3 You first initialize LXD as in Example W23.1, but now you want to change some of its configuration parameters. Outline the steps you would need to take to reconfigure the LXD initial installation on your machine. Give detailed instructions for a particular configuration change, with specific command examples, that show how you would change the LXD that configuration parameter.

W23.4 When you have completed all of the steps of Example W23.2 on your own system, try using various secure methods of copying files between the host computer and the container that you enabled the sshd in. For example, use scp to copy files from host to container, and vice versa. Also use the **rsync** command to copy files back and forth between host and container, as shown in Chapter 19.

W23.5 When you have completed all of the steps in Example W23.3 on your own system, modify the appropriate nginx files so that you can display a custom web page using the nginx server in the container. Consult nginx documentation to find the default location of .html files you would use for the custom web page.

W23.6 When you have completed all of the steps in Example W23.4 on your own system, use the various system administration menu choice presented by Webmin to do some of the common system maintenance tasks we showed in Chapter 19. These might include creation of new users and groups, execution of operating system commands such as manipulation of systemd-controlled services, etc..

W23.7 Choose an “unofficial” but available image to install in an LXD container, such as Centos7. The host system for the container can be whatever Linux system you are working on, and that supports installation and management of LXC/LXD. Choose this image based upon your familiarity with that “flavor” of Linux. Then go through the steps necessary to install that system into a container. Try doing examples W23.2 through W23.4 in this “unofficial” image container.

W23.8 Install vsftpd in the virtual server developed in Section W23.4, and then enable ftp login to it from the Internet via Filezilla.

W23.9 Of the virtualization methodologies shown in this chapter, which is the most useful for you, given the host OS you have on your computer? For example, what kinds of application packages would you run in an LXD container? Why would you be running them in a container?

Advanced “Project-Class” Material

Project 1

Using the methods of this chapter, Chapter W124, Section 5.7 on git, and Chapter W26, Section 9.3 on Access Control Lists (ACLs), which are all available at the Github site for this book:

a. Create a new LXD container that has access via ssh to and from the Internet.

b. In that container, create a project directory for a git repository that a user base of remote users can push to and pull from.

c. Use the appropriate ACL commands to set the ACLs on that project directory so that local and remote users can access it, and can interact with the git repository in that project directory.

d. Design the directory structure of the repository for the intended local and remote user base.

e Create the git repository in the project directory according to your design.

f. Test the git repository, both locally and from the Internet, to confirm that local allowed users and the remote user base can work with the repository to push to and pull from it.

g. Test that unallowed local users or Internet traffic cannot interact with the repository.

Project 2

In VirtualBox, create a clone, a snapshot, and a copy of an installed VM on your host Linux system. Which of these techniques produces a true “backup” of the VM? Once you have produced the copy, how do you restore it?

Which ever approach you select as the true backup technique, its restoration is the critical component of backing up a system and its data.

How does this true backup technique compare to the script files we have written for backing up user directories and files, cloning the system disk and data disk(s) with Clonezilla, and ZFS RAID mirroring? Which do you think would perform the backups in the most reliable and expedient manner, given the use case(s) you have for your Linux system? Which do you think would work best in a commercial application, and why?

Project 3

On your Linux host system, install and use ZFS as described in Example W23.1, to create an LXD container. Then, devise a complete strategy for backing up that container on the host. Be specific about commands and operations that you would employ to accomplish this objective, and provide, in report form, a fully developed scenario with examples. In what directories are the container files, and their controlling LXC/LXD data structures and databases kept? How would you use real-time, on the fly backups to ensure that the container state is backed up? Be explicit in answering these two questions in your report.

Devise a strategy for backing up the LXD “machinery”, and the container ZFS files, on your system. Whether or not you have initialized and configured LXD to use a second hard disk for ZFS storage, detail the steps you would take to back up the relevant files in /var/lib/lxd ( the “machinery” of LXD), and the totality of ZFS files in /var/lib/lxd/containers/ for all of your containers.

Could this entire strategy be accomplished with mdadm? How?