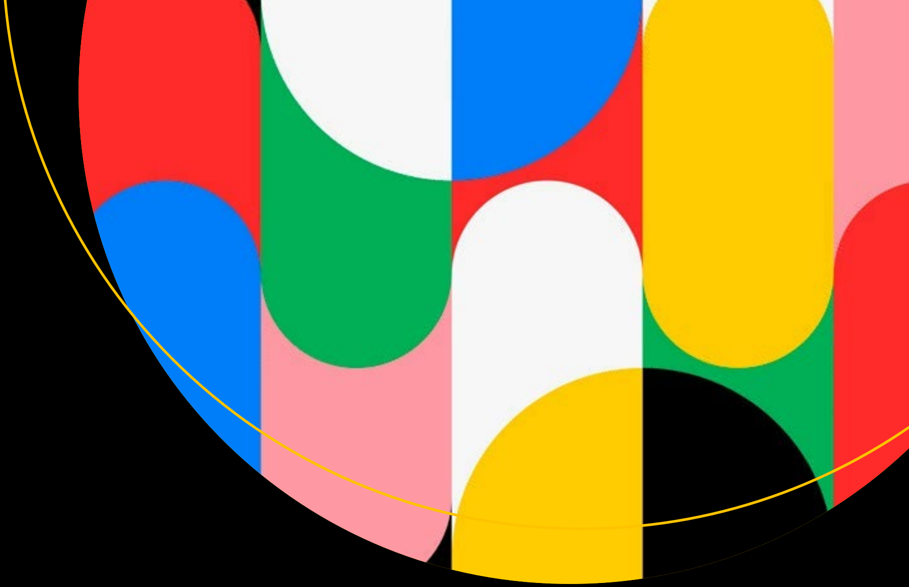

Practical Linux DevOps

Building a Linux Lab for Modern Software Development

John S. Tonello

Apress®

Practical Linux DevOps

Building a Linux Lab for Modern Software Development

John S. Tonello

Apress®

Practical Linux DevOps: Building a Linux Lab for Modern Software Development

John S. Tonello
Baldwinville, NY, USA

ISBN-13 (pbk): 978-1-4842-8317-2 ISBN-13 (electronic): 978-1-4842-8318-9
https://doi.org/10.1007/978-1-4842-8318-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: James Robinson-Prior
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on the Github repository: https://github.com/Apress/Practical-Linux-DevOps. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

John S. Tonello writes about technology, software, infrastructure as code, and DevOps and has spent more than 20 years working in and around the software industry for companies like Tenable, HashiCorp, SUSE, Chef, and Puppet. He's spent more than 25 years building Linux-based environments and regularly publishes a wide range of how-to guides and blogs about DevOps, Linux, and software-defined infrastructure.

About the Technical Reviewer

Nathan Haines is an instructor and a computer technician who has been using Linux since 1994. In addition to occasional programming projects and magazine articles, he is a member of Ubuntu, where he helps spread the word about Ubuntu and Free Software.

Introduction

The Power of Linux

When my mother gave me a cast-off x386 IBM-clone computer in the mid-1990s, I wasn't entirely sure what I was going to do with it, but I felt impelled to get it running. My biggest challenge in doing so was economic, not technical. I was too cheap to buy a licensed copy of Windows 3.1 and went hunting for an alternative. Fortunately, I came across a book titled *Linux Installation & Getting Started* by Matt Welsh, which showed me everything I needed to get started. I downloaded the Slackware Linux installation files over a 56k modem, wrote them to a stack of floppy disks, and literally gave an audible hoot when I got the system up and running.

The moment markedly changed my future, with Linux and open source software becoming a key part of my life, first as a hobby and ultimately as a career.

Other early Linux adopters have similar stories to tell, and like them, I was fascinated by the ability to freely download and install a complete operating system and do "real computing." In the ensuing years, I bought my share of thick Linux texts, installed hundreds (perhaps thousands) of Linux systems, and learned how to use and rely on open source software. That experience is encapsulated in this book.

A lot has changed in the technology world in the 30-plus years since Linus Torvalds first released Linux (and the hard-working kernel) in 1991, and much has been written about it. It's no accident that Linux and the Internet grew up together. Linux remains a critical component of the technical landscape, spawning whole industries and many well-paying jobs. It's at the core of modern life, though few recognize that fact. Today's software developers certainly have heard of it, but many have never had a chance to really explore it.

Practical Linux DevOps is written to be a go-to Linux book for IT practitioners—or those who want to be—who want to explore Linux and the technologies that make modern software happen. It provides real-world tutorials and examples centered around DevOps practices, the concept of continuously building, testing, and deploying software applications that bridge the development side (think software and security engineers) and operations (think hardware administrators). DevOps is how modern software is made, and Linux is in the midst of it all. This book seeks to teach you practical Linux concepts within the context of DevOps, giving you the knowledge you need to confidently continue your exploration.

Learning to Fish

The chapters in this book represent the culmination of my experience with Linux and open source software with new users in mind. I attempt to explain concepts in terms anyone can understand, and provide enough context to explain the *whys*, not just the *hows*. I want to leave you wanting more and with the ability to reason out how to keep learning. The adage goes, "If you give a man a fish, you feed him for a day. If you teach a man to fish, you feed him for a lifetime." The goal of *Practical Linux DevOps* is to teach you how to fish (and not fear the command line).

Where can Linux take you? With Linux running on NASA's Perseverance rover that's exploring Mars, the sky is no longer the limit. This book will help you become part of it all.

Tips for Setting Up Your Environment

Your Workstation

This book presumes you'll use a Linux workstation for all the work you do, not just the virtual machines and containers you create. As you'll read in Chapter 1, you can make good use of older hardware for a Linux workstation. The examples in this book mostly use Ubuntu (based on Debian), and there are many flavors of Ubuntu and other Linux distributions that can run on older Windows and Macintosh computers. I recommend using an existing spare machine, but if money is no object, feel free to get a new or used Intel- or AMD-based machine and install Linux on it.

Environment Settings

After installing a fresh Linux desktop, there are a few steps I always take to make the workstation environment comfortable to my way of working. For example, when you run commands as a superuser (something you'll do all the time), `sudo` requires a password. When you're running dozens or hundreds of `sudo` commands a day, this can become tiring. I solve that problem by creating a file in `/etc/sudoers.d/` that gives me superuser privileges without requiring a password.

To do this, create a new file in `/etc/sudoers.d` and add the following line, replacing `<username>` with the username you use on your Linux system:

```
$ sudo vi /etc/sudoers.d/<username>
```

```
<username> ALL=(ALL) NOPASSWD:ALL
```

After you save this file, you'll be able to run `sudo` commands without entering a password. Of course, this isn't exactly secure and shouldn't automatically be added to production systems, but when you're working on your Linux workstation, it saves a lot of time and hassle.

Terminal Look and Feel

You'll spend a lot of time in your Linux terminal, so take a moment to make it comfortable. There's no need to squint at tiny text or colors you don't like. Xterm and other modern Linux terminals allow you to easily adjust the font, font size, text colors, and background. I recommend editing the preferences to make your terminal suit your tastes.

Power Settings

It might seem like a little thing, but I hate when my workstation screen goes to sleep too often. By default, many Linux desktops go to sleep after just five minutes. You might look away for a few minutes to check headlines or focus on another machine, and you'll have to log back in. I recommend going into your system's **Power** settings to adjust the screen timeout to at least 30 minutes.

Multiple Computers, One Mouse and Keyboard

If you're running your Linux machine alongside another separate workstation, such as Windows or Macintosh, it can be quite cumbersome to have two different keyboards and mice. I resolve this by using a little tool called Barrier, a fork of Synergy, that allows you to share your mouse and keyboard with multiple systems on the same network. The machine with your physical keyboard and mouse attached becomes the Barrier server; every other machine becomes a Barrier client. You can set the geometry (left, right, up, or down) relative to your server machine and seamlessly control, copy, and paste (but not drag and drop) many different Linux, Windows, and Macintosh computers with a single keyboard and mouse.

SCP

There are times when you need to move files from one Linux machine to another. You can email them to yourself, but I've found the tool SCP to be the best way to move files. It uses SSH to copy files or folders from one system to another. There's no need to use FTP or other means:

```
$ scp /path/to/local_file username@remote_host:/path/to/
remote_file
```

You can copy whole directories using SCP by adding the `-r` (recursive) flag:

```
$ scp -r /path/to/local_folder username@remote_host:/path/to/
remote_folder
```

Set Up Passwordless SSH

When you're regularly SSHing or SCPing between machines, it can save a lot of time if you set up SSH keys between them. This enables you to use `ssh` and `scp` without having to always enter a password.

To set this up, start by creating an SSH key on your workstation:

```
$ ssh-keygen
```

Follow the prompts, but **do not** provide a password for the key. Then copy the newly generated key to one or more target systems:

```
$ ssh-copy-id username@remote_host
```

You'll be prompted to enter a password when you first use `ssh-copy-id` to copy the key, but never again after that. This simple setup makes for a more seamless integration between multiple Linux (and Macintosh) machines.

Enjoy the Ride

With that, you're ready to start digging in. Be sure to take advantage of the GitHub repo associated with this book for code examples. They'll get you moving quickly and help you avoid having to manually type out longer code snippets.

CHAPTER 1

Gather Your Hardware

To start building your Linux lab for DevOps, you first need to assemble some hardware. The idea is to be able to experiment without messing up your workstation with applications and packages that could leave it unstable or even unusable. In this chapter, you'll deploy everything for your lab on a separate machine, or machines, where you can install what you need without worrying about wrecking your daily driver.

Today, public cloud providers are a big part of DevOps, making it relatively simple to spin up instances and get to work. Unfortunately, services like AWS and Azure obscure some important aspects of the environment itself and, of course, cost money. By building your own local Linux lab, you'll get the benefits of speed and ease without having to worry about costs. Along the way, you'll also learn a lot about the full environment your servers and applications are living in, giving you a greater insight—and greater abilities—to manage it all.

This book presumes you have access to the following:

- At least one separate physical computer system
- At least one basic network switch
- Perhaps a spare monitor, keyboard, and mouse
- Ethernet and peripheral cables to put it all together

If you're like most tech-curious pack rats, you probably have some older equipment lying around that will do the job. If not, the goal is to get you up and running without spending a lot of cash.

© John S. Tonello 2022
J. S. Tonello, *Practical Linux DevOps*, https://doi.org/10.1007/978-1-4842-8318-9_1

The Basics: What You'll Need to Build Your Linux Lab

The principles of building your lab environment are simple. Create a beefy virtual machine (VM) host with as much memory and CPU resources as possible so you can create as many VMs and Linux containers (LXCs) as possible.

Here are the basics for each physical lab node, which will run the Proxmox Virtualization Environment:

- A 64-bit-capable CPU, such as an Intel i3, i5, or i7 or AMD Ryzen 3, 5, or 7
- At least 8GB of RAM
- A spinning disk or solid-state drive (SSD) that's at least 256GB
- An Ethernet port
- One VGA or HDMI video port
- One free USB port

For a more robust lab that will provide more speed, performance, and flexibility, each physical node should have

- At least 32GB of RAM
- Multiple hard drives or SSDs
- Two Ethernet ports

When you're building your lab, it's important to spend a little time thinking about your ultimate goal. Will you be running dozens of virtual machines or just a couple? How important is performance? Will you have an intricate network layout or something simple? Even if you don't know the answers to these questions now, keep them in mind as you plan to add capabilities. For example, when using Proxmox as your virtualization environment (described in detail in Chapter 2), you can't easily add machines to a cluster if you've already added VMs to the running environment. That means you need to set up two or more Linux lab hosts *before* you start deploying any virtual machines.

Using New Equipment for Your Lab

Part of the fun of building a Linux lab is finding new ways to put older gear to use, and you can certainly do that to accomplish most of the projects in this book. Sometimes, however, old hardware can be more trouble than it's worth, and starting with clean, modern systems can get you up and running quickly and with less frustration.

It wasn't so very long ago that if you wanted a computer with an eight-core CPU, 32GB of RAM, and a 1TB drive, you needed to buy a big, expensive blade server. Today, such systems are commonplace, much smaller, and far cheaper than ever before. Some good candidates for home-based Linux labs on a budget include products made by Intel, Gigabyte, ASRock, Asus, and Kingdel.

For the most part, you'll want mini PCs or tower systems for your Linux lab host servers. The former are small and compact and don't consume much power. The latter give you the most flexibility for CPU, RAM, and storage. Laptops are effective also, but the price point is probably more than you want to spend, although a laptop rig can mean fewer cables and peripherals. No need to drag out a monitor, keyboard, and mouse when you want to do initial configurations.

Something to keep in mind is that you'll access your Proxmox server remotely via a browser-based dashboard and, occasionally, a remote shell. Once it's initially configured, you'll manage the environment from your workstation. The host machine will run headless.

CPU Core Considerations

All the mini PCs built by the vendors mentioned previously come in various sizes and capabilities, including bare-bones systems without memory or hard drives preinstalled. Shop for devices that have the fastest x86_64 (Intel or AMD) CPUs and the most cores you can afford. The more cores the underlying system has, the more virtual CPUs you'll have available for your virtual machines. That's particularly important as you deploy solutions like Kubernetes or OpenStack, which require you to have several machines running simultaneously. Without enough CPUs, performance will suffer, which, in some cases, renders your Linux lab too slow to use.

You can use ARM-based CPUs as well, but recognize that the architecture still is not as robust a development environment as x86_64. Yes, many applications will run fine on ARM-based processors, but some bleeding-edge or legacy applications may not be available.

Another thing to keep in mind if you're planning to use a mini PC is its limited expandability. Some of these small boxes come with the CPU soldered to the motherboard, meaning you can't remove it or replace it. If you'd rather not go the mini PC route and prefer a tower-sized system that can, say, accommodate a removable CPU or many internal hard drives, opt instead for those systems. Just keep in mind that these systems are generally noisier and consume more power.

Of course, be sure that any system you choose supports Linux, even if it initially comes with a flavor of Windows installed. Traditional BIOS or UEFI booting frameworks are fine. With Proxmox you can deploy systems with SeaBIOS and OVMF, an open source UFI implementation.

A CPU with multiple cores—typically four—is critical for running multiple virtual machines at the same time. In the next chapter, you'll learn about the virtualization platform, but for now, be aware that the faster the CPU and the more cores each of your Linux lab machines has, the better.

Each virtual machine you create on top of your Proxmox system will consume both CPU and RAM from the host system, so you can never have too much of either. Some applications are CPU-intensive; others are RAM-intensive. The host environment will share everything among the VMs, so the more memory you have, the more systems you can create, and the more robust they'll be.

At the same time, the virtualization environment will commit only the necessary CPU and RAM to your running VMs, so even though you might assign, say, an openSUSE system 8GB of RAM, it might require only 2GB most of the time it's running. The rest of the Proxmox host's RAM is free to be used by other running systems.

Memory Considerations

As I mentioned earlier, the more memory you can afford, the better when it comes to deploying a versatile Linux lab. Generally speaking, mini computers like those listed previously may have a hard limit of 32GB. That's nothing to sneeze at, but an affordable tower machine will likely give you the ability to easily double or triple that amount.

If you come across a good deal on a mini computer with just 4GB of RAM, don't buy it with such little memory. Plan to add more. Since most motherboards—large or mini—want memory chips installed in pairs, you might be limited to adding just another 4GB chip, which would give you only 8GB total in most mini PCs, which typically have only two RAM slots. It's better to order the device with at least one 8GB RAM chip so you can expand it later.

Remember, it can be tricky to buy the correct memory for a computer, and some computers require RAM chips that are far more expensive than others. Check the technical specs before buying.

Storage Considerations

The price of most storage devices continues to drop, and you shouldn't have any trouble finding a 500GB SSD for about the price of this book. SSDs and the newer NVMe solid-state drives are fast and come in many different sizes. Plan on at least half a terabyte, but if you can afford more, buy it. Proxmox itself doesn't require much disk space at all. The disks you use will be filled by your virtual machines.

If you're planning to mix and match new drives with old ones, that'll work fine. Just be sure to have a physical machine that can fit them all and has enough motherboard connectors to accommodate them. One scenario is to install Proxmox itself (the server runs a version of Debian Linux) on a smaller drive and reserve additional drives for your VMs.

Some mini devices come with NVMe connectors and no SATA connectors. You can get PCIe adapters that plug into a device's USB 3.0 port to add external storage, which can be an inexpensive option. Without SATA interfaces, you won't be able to add an SSD as a secondary internal drive, and in those situations, you'll want to install the biggest single drive you can afford. Most towers, on the other hand, have motherboards with six or more SATA drive connectors, ideal for your older spinning SATA drives and most SSDs. NVMe drives are a different matter. They require a special slot. Do your homework before shelling out.

You can make do with spinning hard disks, but recognize that they'll be slower than SSDs when it comes to reading and writing data. For example, an operating system running on a spinning disk will take longer to boot than one booting from an SSD. Depending on your use cases, an SSD's higher input/output (I/O) speed can make a big difference and make life less tedious, but they're not critical for your lab hosts.

If you have several older drives available and your lab host nodes have the room, install as many as you can. You'll be able to take advantage of those drives, even if they're relatively small. I've built nodes with a laughably small 60GB drive that enabled me to run several VMs without any trouble. That's because the virtual machines you create all use *thin provisioning*, which means they don't consume any disk space until they need it. If you install Ubuntu on a 32GB virtual disk, it initially uses only 5GB, but the virtual machine won't touch the other 27GB you committed until it needs it.

A Linux lab based on a virtualization platform like Proxmox or VMware gives you the ability to overcommit all the physical system's resources without requiring you to do any tricky math to make everything work.

Using Old PCs and Laptops for Your Linux Lab

Although it's true that new computers and networking gear can save you some time and frustration, there's a lot to be said for older hardware you might already have.

The first Linux lab I built was on an older tower desktop machine with an Intel i3 quad-core processor with 8GB of RAM and a 1TB spinning drive. It had two Ethernet ports and served my needs for years. In fact, I still have it and fire it up from time to time. You may have similar older desktops or laptops like mine gathering dust that can serve as nodes for your lab cluster.

The most important consideration for using older gear is the system's underlying architecture. Though you can still install Linux distributions on 32-bit systems, they don't make good Linux lab hosts because virtualization is limited or unavailable. For virtualization, you need the multi-threading capabilities of 64-bit systems. How can you tell the difference? Check the vendor's website for the original technical specs on the machines you have. That'll give you a good start.

An old laptop might be another good choice, particularly if portability is important to you. You can take your lab on the road if necessary (or at least to work).

If the system you're planning to use is running Windows, open the file manager (or any folder) and right-click the *This PC* icon and choose *Properties*. You'll see the system configuration there. If you have an Intel-based Mac, you can click the apple in the top-left corner and choose *System Properties* to see what it has under the hood. If there's no OS installed on the system, you can boot a live Linux USB and follow the instructions in Chapter 2 to check the system.

It's tempting to want to build a Linux lab environment right on your main workstation or perhaps set up a dual-boot configuration so you can switch back and forth. That's a workable option, particularly if you're digging into containerized environments with Docker and the like. However, I don't recommend it as the single resource for your DevOps lab. Later chapters will take you through container basics so you'll get a chance to work with microservices, but the idea here is to have a fully independent environment to work on *and* access from your regular workstation. Even if you're someone who always keeps their workstation up and running, this isn't quite enough for a good lab. Instead, find at least one separate machine to use.

Raspberry Pis and IoT Devices

Internet of Things (IoT) devices are becoming an important part of the hardware landscape, and you can definitely incorporate such devices into your lab environment. One of the best and easiest ways to do that is to get a few Raspberry Pis.

Though there are many, many single-board devices from which to choose, including the Pine64 and Orange Pi, Raspberry Pi devices are a good choice because they support a wide variety of Linux OSes (including the Debian-based Raspbian default), they're inexpensive, and they have a vast community of developers. This last point is important when it comes time to deploy applications and services because chances are good that someone else has tried what you want to do, and the Internet is full of guides and information. No other single-board device has as much readily available content for you to take advantage of when you're stuck.

When buying Raspberry Pis for your Linux lab, be sure to get a model that's at least version 3. RPi4 models are even better. These have faster CPUs, more RAM, and onboard WiFi and Bluetooth. Check with the vendor for the best storage options, armed with the knowledge that not all SD cards are suitable (durable and fast enough) to run a Raspberry Pi.

If you buy your Raspberry Pis as standalone devices, and not as part of a kit, be sure you get enough mini-USB and USB-C cables to power them. The Raspberry Pi 4 uses USB-C for power and has micro HDMI ports, so you'll need an adapter to connect to monitors with full-size HDMI ports. Most of the time, you'll run your Raspberry Pis headless—with no monitor or keyboard—but initial configuration often requires these interfaces.

Something else that's handy with RPis is a USB power hub. These enable you to plug five or more mini-USB cables into a single device that uses just one wall outlet. This is a much better option than adding power strips to accommodate half a dozen wall warts.

Raspberry Pis are small, so you can fit a lot of them in very little space. I like to buy inexpensive racks to hold four or more in a neat stack.

Once considered a mere plaything, the Raspberry Pi is now anything but. They can power 4k monitors, and with the additional USB storage, you can use them as media streaming devices. In your lab environment, they are a great choice for deploying applications and containers, giving you the option to expand your lab for very little money.

In my own lab, I've deployed a four-RPi cluster to host containerized applications and even a full-fledged Ceph storage cluster. Of course, the performance for storage isn't something you'd use in production, but for applications that require multiple devices, they provide an excellent, inexpensive learning platform.

The capabilities of modern RPis make them even more useful for lab environments, and it's possible to build an entire Linux environment using just them and your workstation. I won't go into a lot of detail of how to accomplish this, but it's something to keep in mind if you're limited on resources and cash. For the purposes of this book, consider them excellent supplements, but not primary resources.

Building Your Network

In order for your Linux lab to be truly valuable for all the DevOps work you want to do, it must connect to a network so you can install packages, remotely access it via the shell, and have it serve up resources like web pages and DNS information. That requires at least a single Ethernet interface on the host machine itself.

Fortunately, you have a lot of choices and can get great performance without having to spend much money. Well-known vendors, such as NETGEAR, Dell, D-Link, Linksys, and TP-Link, make suitable 1GB networking devices for your lab hosts, if they're not built in already.

If you plan to use an older system, you can make do with a 100MB Ethernet port, but a 1GB connection is better. Everything in your lab environment will perform better with faster networking, and network installations can really fly with the additional bandwidth. Of course, to take advantage of a 1GB network interface on any Linux lab host, you'll need a network switch capable of handling 1GB speeds. Many low-cost options are available.

Running two separate networks is ideal for your lab environment, and for lab purposes that requires two separate network interfaces on your physical systems, including your separate workstation. Ideally, these should be physical Ethernet ports or USB 3 dongles, but one physical port and WiFi capability can be better than just a single port. Two separate network interfaces enable you to isolate your network traffic and help keep your lab secure. If you're relying solely on your home network that everyone in your house uses for Netflix, having a separate lab network allows you—and your family—to avoid slowdowns.

The principle here is to create one network for all your lab traffic and one network for accessing the Internet. In a home environment, you typically accomplish the latter via the router provided by your ISP. The former would be a private network using a small gigabit Ethernet switch or router.

WiFi is robust enough to support most of the applications and deployments described in this book, but getting it to work can be tricky, and it's not always as robust as a wired network interface. If you're okay with spending a little more time and having a few more hiccups, try it.

Managed vs. Unmanaged Switches

If you're just starting out, unmanaged switches are easy to use and cheap to buy. They provide everything you need to get a simple network up and running. Just plug in some Ethernet cables, connect them to your workstation and lab hosts, and you're off and running. They move network traffic well and are pretty foolproof.