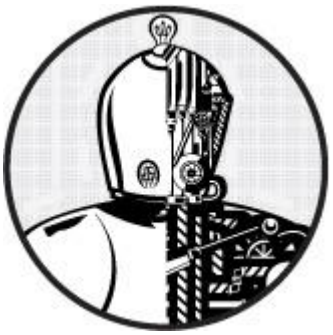
# Chapter 17. Building on the Basics



The chapters in this book have covered the fundamental components of a Linux system, from low-level kernel and process organization, to networking, to some of the tools used to build software. With all of that behind you, what can you do now? Quite a lot, as it turns out! Because Linux supports nearly every kind of non-proprietary programming environment, it's only natural that a plethora of applications is available. Let's look at a few application areas where Linux excels and see how what you've learned in this book relates.

## 17.1 Web Servers and Applications

Linux is a popular operating system for web servers, and the reigning monarch of Linux application servers is the Apache HTTP Server (usually referred to as just "Apache"). Another web server that you'll often hear about is Tomcat (also an Apache project), which provides support for Java-based applications.

By themselves, web servers don't do much—they can serve files, but that's about it. The end goal of most web servers such as Apache is to provide an underlying platform to serve web applications. For example, Wikipedia is built on the MediaWiki package, which you can use to set up your own wiki. Content management systems like Wordpress and Drupal let you build your own blogs and media sites. All of these applications are built on programming languages that run especially well on Linux. For example, MediaWiki, Wordpress, and Drupal are all written in PHP.

The building blocks that make up web applications are highly modular, so it's easy to add your own extensions and create applications with frameworks such as Django, Flask, and Rails, which offer facilities for common web infrastructure and features, such as templates, multiple users, and database support.

A well-functioning web server depends on a solid operating system foundation. In particular, the material in Chapter 8 through Chapter 10 is particularly important. Your network configuration must be flawless, but perhaps more importantly, you must understand resource management. Adequately-sized, efficient memory and disk are critical, especially if you plan to use a database in your application.

## 17.2 Databases

Databases are specialized services for storing and retrieving data, and many different database servers and systems run on Linux. Two primary features of databases make them attractive: They offer easy, uniform ways to manage individual pieces and groups of data, and superior access performance.

Databases make it easier for applications to examine and alter data, especially when compared with parsing and changing text files. For example, the */etc/passwd* and */etc/shadow* files on a Linux system can become difficult to maintain over a network of machines. Instead, you can set up a database that offers user information LDAP (Lightweight Directory Access Protocol) to feed this information into the Linux authentication system. The configuration on the Linux client side is easy; all you need to do is edit the */etc/ nsswitch.conf* file and add a little extra configuration.

The primary reason that databases generally offer superior performance when retrieving data is that they use indexing to keep track of data locations. For example, say you have a set of data representing a directory containing first and last names and telephone numbers. You can use a database to place an index on any of these attributes, like the last name. Then, when looking up a person by last name, the database simply consults the index for the last name rather than searching the entire directory.

### 17.2.1 Types of Databases

Databases come in two basic forms: relational and non-relational. *Relational databases* (also called *Relational Database Management Systems*, or *RDBMS*), such as MySQL, PostgreSQL, Oracle, and MariaDB, are general-purpose databases that excel in tying different sets of data together. For example, say you have two sets of data, one with postal (ZIP) codes and names, and another with the postal codes and their corresponding states. A relational database would allow you to very quickly retrieve all of the names located in a particular state. You normally talk to relational databases using a programming language called SQL (Structured Query Language).

*Non-relational databases*, sometimes known as *NoSQL* databases, tend to solve particular problems that relational databases don't easily handle. For example, document-store databases, such as MongoDB, attempt to make storing and indexing entire documents easier. Key-value databases, such as redis, tend to focus on performance. NoSQL databases don't have a common query language like SQL for access. Instead, you'll talk to them using a variety of interfaces and commands.

The disk and memory performance issues discussed in Chapter 8 are extremely important in most database implementations because there's a trade-off between how much you can store in RAM (which is fast) versus on disk. Most larger database systems also involve significant networking because they're distributed over many servers. The most common such network setup is called *replication*, where one database is basically copied to a number of database servers to increase the number of clients that connect to the servers.

## 17.3 Virtualization

In most large organizations, it's inefficient to dedicate hardware to specific server tasks because installing an operating system tailored to one task on one server means that you're limited to that task until you reinstall it. Virtual machine technology makes it possible to simultaneously install one or more operating systems (often called *guests*) on a single piece of hardware, and then activate and deactivate the systems at will. You can even move and copy the virtual machines to other machines.

There are many virtualization systems for Linux, such as the kernel's KVM (kernel virtual machine) and Xen. Virtual machines are especially handy for web servers and database servers. Although it's possible to set up a single Apache server to serve several websites, this comes at a cost of flexibility and maintainability. If those sites are all run by different users, you have to manage the servers and the users together. Instead, it's usually preferable to set up virtual machines on one physical server with their own supporting users, so that they don't interfere with each other and you can alter and move them at will.

The software that operates virtual machines is called a *hypervisor*. The hypervisor manipulates many pieces of the lower levels of a Linux system that you've seen in this book with the result that, if you install a Linux guest on a virtual machine, it should behave just like any other installed Linux system.

## 17.4 Distributed and On-Demand Computing

To ease local resource management, you can build sophisticated tools on top of virtual machine technology. The term *cloud computing* is a catch-all term that's often used as label for this area. More specifically, *infrastructure as a service (IaaS )* refers to systems that allow you to provision and control basic computing resources such as CPU, memory, storage, and networking on a remote server. The OpenStack project is one

such API and platform that includes IaaS.

Moving up past the raw infrastructure, you can also provision platform resources such as the operating system, database servers, and web servers. Systems that offer resources on this level are said to be *platform as a service (PaaS)*.

Linux is central to many of these computing services, as it's often the underlying operating system behind all of it. Nearly all of the elements that you've seen in this book, starting with the kernel, are reflected throughout these systems.

## 17.5 Embedded Systems

An *embedded system* is anything designed to serve a specific purpose, such as a music player, video streamer, or thermostat. Compare this to a desktop or server system that can handle many different kinds of tasks (but may not do one specific thing very well).

You can think of embedded systems as almost the opposite of distributed computing; rather than expanding the scale of the operating system, an embedded system usually (but not always) shrinks it, often into a small device. Android is perhaps the most widespread embedded version of Linux in use today.

Embedded systems often combine specialized hardware with software. For example, you can set up a PC to do anything a wireless router can by adding enough network hardware and correctly configuring a Linux installation. But it's usually preferable to buy a smaller, dedicated device consisting of the necessary hardware and eliminate any hardware that isn't necessary. For example, a router needs more network ports than most desktops but doesn't need video or sound hardware. And once you have custom hardware, you must tailor the system's software, such as the operating system internals and user interface. OpenWRT, mentioned in Chapter 9, is one such customized Linux distribution.

Interest in embedded systems is increasing as more capable small hardware is introduced, particularly system-on-a-chip (SoC) designs that can cram a processor, memory, and peripheral interfaces into a small space. For example, the Raspberry Pi and BeagleBone single-board computers are based around such a design, with several Linux variants to choose from as an operating system. These devices have easily accessible output and sensor input that connects to language interfaces such as Python, making them popular for prototyping and small gadgets.

Embedded versions of Linux vary in how many features from the server/ desktop version can be carried over. Small, very limited devices must strip out everything except the bare minimum because of lack of space, which often means that even the shell and core utilities come in the form of a single BusyBox executable. These systems tend to exhibit the most differences between a full-featured Linux installation, and you'll often see older software on them, such as System V init.

You'll normally develop software for embedded devices using a regular desktop machine. More powerful devices, such as the Raspberry Pi, have the luxury of more storage and the power to run newer and more complete software, so you can even natively run many development tools on them.

Regardless of the differences, though, embedded devices still share the Linux genes described in this book: You'll see a kernel, a bunch of devices, network interfaces, and an init alongside a bunch of user processes. Embedded kernels tend to be close (or identical) to regular kernel releases, simply with many features disabled. As you work your way up through user space, though, the differences become more pronounced.

## 17.6 Final Remarks

Whatever your goals for gaining a better understanding of Linux systems, I hope that you've found this book to be helpful. My goal has been to instill you with confidence when you need to get inside your system to