Book Reviews

MARK LAMOURINE

Introducing Go
Caleb Doxsey
O'Reilly Media, 2016, 112 pages
ISBN: 978-1-491-94195-9

I didn't look twice when I saw this title come up as a "people also bought" on Amazon. A lot of my work recently has been developing or using applications written in Go(lang), and I've been looking for good books on Go for more than a year. For the longest time there were no professionally published books on Go, and the only resources were blogs, the official godoc Web site (https://godoc.org), and a couple of Creative Commons texts written by developers mostly found on the Golang Web site (https://golang.org/doc/). (Don't misunderstand me. These are great resources, but I like books, made of paper, in my hand.) Recently this has changed dramatically: there are half a dozen books on learning Go in my local bookstore. I'm really glad about this and have started picking up books to read.

When I first open a box of books, I take each one out and leaf through it before setting it on the pile of "to be read" books. I stopped when I got to *Introducing Go* and looked more carefully. It turns out that I'd read and reviewed the same book in 2014. Well, not quite the same book.

Caleb Doxsey was one of the first authors to publish a book on learning Go. It was first available as a Web site in collaboration with Google, then a PDF entitled "An Introduction to Programming in Go" made available by Doxsey and Google under a CC3 license in 2012. It's still there if you want to read it (http://www.golang-book.com/books/intro). For a time the book was also available as a paperback, but the conversion from e-text to paper was a mixed success.

Introducing Go is, at its core, Doxsey's original corpus but published by O'Reilly. It's still a slim volume and the contents and flow are largely lifted from the earlier work, but that's not what's important. The text has been updated and reformatted, and this is a significant improvement over the self-published version from 2012. It's not that Doxsey didn't do a good job, but O'Reilly really enforces good editing and layout: the code examples stand out and are much easier to read in the standard typeset style; the paragraphing and tables are clearer; and although the book is physically smaller, what's there is what's important and it's clear.

There is a small set of books that I call the "slim classics." I'm thinking of *The C Programming Language* (Kernighan and Ritchie), *Unix Programming Environment* (Kernighan and Pike), and a couple of others like them. They're not always the best all-around guides or references, but they distill the essence of a topic in a way that thicker tomes sometimes lose in their quest for authoritative completeness. In these days of fast-moving language development, I don't know if there's room for slim classics, but I think *Introducing Go* could be a contender.

Go in Action
William Kennedy, with Brian Ketelsen and Erik St Martin
Manning Publications Company, 2016, 241 pages
ISBN: 978-1-61729-178-1

As just mentioned, I've been on a bit of a Go(lang) book kick recently. *Go in Action* is one I looked forward to. In general I like Manning's style, and their editorial choices tend to walk the fine line between traditional dusty references and frothy Dummies-style tutorials.

The authors used the MEAP (Manning Early Access Program) process for writing and pre-release editing and commentary, in which early subscribers get to see the chapters raw as they are submitted, and the authors and editors get commentary during the writing process. In other words, they use the Internet to apply the "many eyes" principle to writing.

After reading a number of texts on Go, I see a kind of standard narrative path emerging: "Hello, world," general development and build environment, pulling packages for inclusion, then into the language constructs themselves, finally ending with a chapter on testing. This is a perfectly reasonable path to take, but after reading a number of them, I'm finding it harder to see what distinguishes one book from another.

Go in Action does have a number of aspects that set it apart. The first I noted above: Manning has developed a very clean typography and layout style, which makes following the commentary and code very easy. Line numbering each code block makes following references easy both for the authors and for the reader. Using a registration table inside the front cover of the book, Manning offers a watermarked copy of every book in PDF, EPUB, and MOBI for every paper copy purchased. Their focus on writing for both paper and electronic documents means that the two forms have parity. I can read the EPUB version and get as good an experience as when reading the hard copy.

Once you get below the general arc of these books, Kennedy et al. do a very good job of showing both how to use Go constructs and what happens when you do. The Go runtime environment is very different from anything that readers coming from scripting languages or Java might be familiar with. Each language construct has a deliberate effect and behavior in the runtime, and unlike modern scripting languages, Go is explicitly designed not to hide the underlying mechanisms. You can stick your hand in the running motor if you want to. Kennedy et al. provide text and diagrams that illustrate these behaviors. This helps new developers avoid (or in my case, recognize after the fact) the pitfalls that can lead to lost fingers.

This is not an introductory book on programming. It's likely to be too much for a reader who isn't already proficient in one or several other languages.

The code examples are available online as are updates to the e-text (assuming you've registered your copy). The authors are available by email or other means for questions or commentary.

If you're coming to Go as a student or professional developer, *Go in Action* would not be a bad introduction. I don't think you'll want to stop there, though.

Docker in Action

Jeff Nickoloff
Manning Publications, 2016, 284 pages
ISBN 978-1-63343-023-5

As with the Go language, there has been a shortage of good books on Docker. There is a relationship between the two. Docker is written in Go, and Go and Docker both have reached a level of maturity and stability where it makes sense to begin writing about them, and *Docker in Action* does a good job.

Often when people try to explain software containers they begin with Docker's shipping container metaphor. Unfortunately, this isn't really an apt metaphor. It's neither insightful or informative when applied to software containers. Then they start trying to define them as "not virtual machines" which is similarly uninformative.

Nickoloff opens *Docker in Action* with one of the best descriptions of software containers that I've seen (though he does at one point tip a hat to shipping containers). Containers are just processes with blinders on, and Nickoloff shows clearly how they relate to the OS, to VMs, and traditional processes.

I really like the progression of the narrative in this book. The author begins with simple containers doing simple jobs locally. All of the examples involve real-world tasks, using containers to replace the traditional applications. Along the way he discusses both the benefits and costs of this. Every case shows the CLI command, which invokes the container, the overt result, and then goes under the covers to show how the result was achieved in the context of containers. When readers follow the text carefully, they will know "what happened" at each of the appropriate layers. This is really important when readers put their understanding to use doing new tasks.

Another thing I really like about the book is Nickoloff's restraint when it comes to building new containers. Dockerfiles and custom images are sexy and interesting looking, but for containers to fulfill their promise, most people should be using off-the-shelf images. Nickoloff manages to get more than half way through the text before discussing container builds. Even then he treats it as a small step, merely extending existing images and moving right on to continuous integration and publishing, the true life cycle of container images. I suspect he knows that there are other texts that go into Dockerfile management in painful detail and that the "best practices" for creating new images for composition are still being developed.

I did learn quite a bit about a more open topic in container management: orchestration. Docker Inc. has been developing in-house tools for composing containers into applications. Others have been doing similar work, but Docker Inc. would really like you to use theirs. This final section introduces Docker's offerings.

Docker Compose is Docker's answer to Kubernetes. It provides a way to create containers that are meant to work in unison to form an application or service. Docker Machine (which was once known as "boot to docker") is a tuned bootable image meant solely to host the Docker runtime. It is an analog of CoreOS or Project Atomic, both minimized bootable images meant to host container runtime environments. Docker Swarm is meant to allow for scaling and distribution of containers across a multi-host environment. Again, this is comparable to Kubernetes or OpenShift.

Each of these tools gets a chapter and a little more at the end of the book. The examples and illustrations are every bit as good here as they are in the chapters covering the more mature elements of Docker. Even though I'm familiar with Kubernetes, I expect I will turn back here the next time I need to build a multi-container application quickly. Once I am more familiar with the Docker tools, I can evaluate whether they will fill the needs of the kinds of large-scale systems that Kubernetes means to build. Nickoloff has made it easy for me to explore a tool set I would otherwise not have considered given my current experience. Make of that what you will.

Of the few texts I've seen on Docker so far, this is the one I would hold out to someone who asked me where to start. Nicely done.