Book Reviews

MARK LAMOURINE

REST API Design Rulebook

Mark Massé
O'Reilly Media Inc., 2012, 94 pages
ISBN 978-1-449-31050-9

You would be right in assuming that any book with the word "rulebook" in the title would express an opinion. Massé certainly doesn't hold back, but that seems to be a trait of REST advocates in general.

At under 100 pages, Massé's book packs quite a lot into a little space. It is really presented as a list of one-line rule statements with a matching brief explanation. Each is meant to address one of the common questions raised when designing a REST protocol.

The first three sections treat the interactions between the client and the server, detailing how each uses the HTTP protocol features to communicate and interpret the intent of the other. The fourth section describes how to add metadata that allows the self-discovery that is characteristic of REST protocols.

I was struck by how little the rules had to do with the formatting of the content. The only rules that deal directly with content are those that state that the payload must use a standard structured data format such as JSON or XML. The rest of the rules describe how to make use of the simple CRUD (Create, Read, Update, Delete) operations that HTTP offers to define the more complex interactions that a rich application protocol needs.

Massé notes that the contents of these first four sections are based largely on consensus reached over time among the developer community. In the final two sections, he discusses rules for data representation and for client-side concerns like authentication and applications with multi-origin data sources. The wording of the rules here changes from "must" to "should." Massé indicates that these are his answers to the questions that remain open, based on his experience.

This book was written in 2011, more than a decade after the publication of Roy Fielding's PhD dissertation in 2000. Since then REST has come to be the preeminent model for client-server communications, replacing proprietary binary models and earlier Web standards like SOAP and XML-RPC. While many services claim to conform to the REST conventions, a close read of this book will show that few really meet the full criteria.

When thinking about REST, people often focus on representing the payload content using a structured data format. Many forget that a major tenet of REST is that the relationships between the different data objects must be included in the query responses. Links and relationships must be discoverable by the client without the need to code assumptions into the client-side logic. Defining and presenting these relationships in the metadata of a REST response requires a lot of thought and work on the part of the server writer. Many applications that claim to be RESTful take shortcuts on the protocol design, coding the relationships into the client.

Massé correctly focuses on how to define and present these relationships. He understands that simply representing the content as structured data is the easy part. He gives very little space to how to write the code, though he does include a simple app example in the final chapter.

In the end it may not matter if developers strictly adhere to the REST guidelines, so long as the code works, but I suspect much code could be improved after a few minutes spent with the *Rulebook*.

CoreOS in Action

Matt Bailey
Manning Publications Inc., 2017, 178 pages
ISBN 978-1-61729-374-0

In the grand migration to software containers, there is a largely overlooked component that I think deserves more attention: the container host. The conventional OS distribution design is based on old assumptions about how applications work and how they will use OS underneath. Container hosts are designed with the containerized application in mind: a minimal Linux install on a read-only file system.

CoreOS began as a kind of customizable single-application host distribution. Originally, CoreOS was designed for building an image for each service as if it were an embedded system or unikernel. The build system is based on Gentoo, and the code base began as a variant of ChromeOS.

CoreOS itself didn't get much attention until the advent of Docker and the growth of containers. Creating custom images with embedded applications required skill and specialized knowledge, and there was little incentive for developers to focus on those skills. Docker changed that by creating an easy, consistent model for creating single-purpose images, with the advantage of portability and a distribution infrastructure, the container registry. Once CoreOS included the Docker runtime, it became an ideal place to create distributed container services.

Bailey packs a lot of information and many examples into a slim book. In some ways this reflects what CoreOS is good at: minimizing complexity (at least in some realms). The whole idea of container hosts is that you don't administer them in the same way that you would a conventional host: you can't install packages. Persistent storage must come from a shared resource. This doesn't mean that you don't need to manage them or that your applications will magically appear and work. For a sysadmin, using container hosts means unlearning and relearning a lot.

The examples include short bits to create the container images, to deploy CoreOS itself, and to configure the services that bind the individual hosts into a cluster. I wish Bailey had spent a little more time on the theory and internals of these services: etcd, fleet, flannel. The code fragments and the callouts that explain these services are clear and well presented, but a bit more on how they work might make these samples easier to adapt to the reader's own purposes.

Bailey asks a lot of his readers because adopting CoreOS requires thinking about applications in new ways. Only the first third of the book is given to actually installing the OS and configuring the clustering services. In the second section, Bailey shows how to build applications that will be suited to the container environment. He does address legacy applications, but leaves it implicit that they must be decomposed and migrated, not "forklifted" into containers.

In the final section, Bailey talks about aspects of using CoreOS in production. He shows a CoreOS deployment in AWS using Cloud formation to describe the configuration and topology. He closes with a brief discussion of what might be a taboo subject: a container designed to allow the sysadmin access to the tools they are used to having on a conventional host.

Container hosts are still in the shadows of the containers themselves, but I think they should be given more light. *CoreOS in Action* shines a light on the foundation. This might even be a good path for introducing containers themselves.