Book Reviews

RIK FARROW AND MARK LAMOURINE

The Site Reliability Workbook: Practical Ways to Implement SRE

Niall Murphy, David Rensin, Betsy Beyer, Kent Kawahara, and Stephen Thorne
O'Reilly Media, 2018, 512 pages
ISBN: 978-1-492-02950-2

Reviewed by Rik Farrow

When I think of a workbook, I expect something that contains exercises and complements an existing book or course. *The Site Reliability Workbook* fits the second part of that description. The authors intended that *TSRW* expand upon the best-selling *Site Reliability Engineering*, in part because of all the questions raised by readers of the first book.

Today, you can find all of the *SRE* book online, and as *TSRW* relies on that book, there are frequent references to chapters in the earlier book, all as bit.ly-shortened URLs. While that's useful, there are often summaries to the material, and I found that all I needed were the summaries to recall enough for the current material to make sense.

And instead of exercises, you get examples, case studies, and more in-depth descriptions. Right away I could see how useful this was in making the principles described in *SRE* concrete. There is even a chapter on Non-Abstract Large System Design, with tangible examples of what the authors, including Salim Virji, were teaching during LISA tutorials, a step-by-step approach to designing a reliable service for monitoring AdWords.

There was criticism that *SRE*, both the practice and the book, were something only Google, and a handful of companies like it, could take good advantage of. *TSRW* attempts to dispel those objections, largely by including authors outside of Google for many of the sections.

You will often find that books written by many authors have an uneven writing style. *TSRW* doesn't read that way at all: the writing remains clear, consistent, and easy-to-read throughout.

As to the argument that *SRE* is only for large organizations, I found myself thinking many times as I read *TSRW*, "If only I had known that 35 years ago." In the chapter about On-Call, I read about many practices that would have made my life easier in my first Bay Area job and prevented burnout. I also encountered some things I had tried to do, with partial success, in that long ago era. In other words, even if you don't consider yourself an SRE, there are definitely things you can learn from this book.

Managing Kubernetes: Operating Kubernetes Clusters in the Real World

Brendan Burns and Craig Tracey
O'Reilly Media, 2018, 188 pages
ISBN: 978-1-492-03391-2

Reviewed by Mark Lamourine

Often it seems that sysadmins are forgotten when people are writing documentation. It is common to see books for service users and for API developers. When it comes to managing services, it feels like the first response is to try to write some kind of GUI to smooth over the sharp bits and pretend they don't exist. This leaves the sysadmin needing to understand, manage, and diagnose complex systems with little guidance but their own wits and experience.

Managing Kubernetes won't solve every sysadmin problem, but it does go a long way toward illuminating the dark interior of one of the hottest buzzword services of the last few years.

Brendan Burns is one of the three original authors of Kubernetes and is still one of the top three contributors. With Craig Tracey, he provides the clearest description I've seen of the moving parts that, together, make a Kubernetes cluster.

Kubernetes is a distributed software container management service. That's quite a mouthful. If you're not already familiar with software containers, you should really start somewhere else. The most well-known container runtime system is Docker. There are others, but Docker is the BASIC programming language of containers. You'll be back quickly, because standalone containers have limited value. They come into their own when you start combining single-purpose containers into complex applications. How you combine them and then deploy them to make working services is what Kubernetes is all about.

Kubernetes is itself a (mostly) containerized service, built up of a number of cooperating service components. The hosts that participate in the clusters are called *nodes*. All nodes must have a container runtime environment such as Docker already installed and running.

Some nodes, called *head nodes*, are special. These run the management components and provide the brains of the cluster. The remainder of the nodes, called *worker nodes*, run components that control local containers and provide network communications. All of these coordinate by communicating with an API service that is distributed across the head nodes.

The arc of the book is a little different from most. Burns and Tracey don't have the reader attempt an installation until almost halfway through, in Chapter 6. Ordinary users would want to get started creating containers as soon as possible, but the sysadmin's purpose is to understand what is happening underneath when normal users start their work. The authors devote the first half of the book to describing the structure that installation will create.

In the second half of the book, the authors walk the reader through common operational processes. Many of these are concerned with providing and controlling access to the cluster. Users interact with the cluster by making requests to the API server. The next three chapters detail how user requests are validated and accepted.

The authors provide one of the better explanations I've read of the distinction between *authentication*, *authorization*, and what they call *admission*, which I might have called *policy*. In each case, they provide examples of the REST data structures that implement the communication protocol. The examples demonstrate the rationale and the structure, but none of them are meant to be comprehensive. The authors know that the Kubernetes project documentation [1] provides detailed specifications, though I do wish they had provided the appropriate links in-line with the text.

The final three chapters cover additional operational concerns: networking, monitoring, and disaster recovery. Again, the discussion is meant to give the reader a starting point for understanding what is possible and where to learn more. It is not a run-book but, rather, is concerned with architecture and taxonomy. It provides references to resources that the reader can use to learn and plan for a deployment.

Rather than being an operator's manual or a comprehensive reference, *Managing Kubernetes* describes the purpose and basic configuration of each component and gives the reader a sense of the structure and dynamics of Kubernetes as a whole. I have noted in other places that it is often very useful to understand any technology at least one layer, and preferably two, beneath the level where you mean to work. For both operators and architects of Kubernetes services, *Managing Kubernetes* will provide the peek beneath the covers.

Reference

[1] Kubernetes REST API specification: https://kubernetes.io/docs/concepts/overview/kubernetes-api/.

Learn Git in a Month of Lunches

Rick Umali
Manning Publications, 2015, 352 pages
ISBN: 978-1-617292415

Reviewed by Mark Lamourine

Many authors can't seem to decide whether they want to write a reference or a tutorial, often making their book less than ideal for either the beginner or the experienced reader. Rick Umali doesn't make this mistake. He knows he's writing a book for beginners, and *Learn Git in a Month of Lunches* is ideal for his audience.

Git is well suited to this kind of learning. It is a tool that is used by software developers to organize and manage their work. It allows them to share their work in a way that makes conflict avoidable or at least manageable. It has one purpose and a well-defined set of operations to accomplish that purpose. Tools like this are often learned fitfully, by experience, looking up the single solution to a single problem then going back to work. Umali has offered a straightforward and complete path for learning to use the most important capabilities of Git and the grounding to explore and learn more.

Umali, to his credit, dodges several common problems that arise from trying to present material in a narrative format. He avoids creating a contrived straw-man project. Instead, each chapter focuses on just one task or subcommand, and he discusses the most common aspects of that task. He does interlace examples for the three common platforms, Windows, Mac, and Linux, but each example is clearly labeled and distinguished by graphical conventions.

He also starts at the true learner's beginning (after installation) by creating an empty local repository. While most work in the real world will involve a remote repository, Umali leaves that for Chapter 12, well past the halfway point in the book. That first half is dedicated to getting comfortable with Git and just managing files in a repository. I was reminded firmly that all of the common operations, committing, cloning, branching, merging, and viewing logs are local operations. In every case the pattern for a file reference is first a local path that can then be extended to a URL by adding a standard prefix.

That said, the next four chapters cover the details of working with remote repositories; push, sync, rebase, and a chapter on branching conventions and collaborative workflows.

He wraps up with chapters on third-party Git software, working with GitHub, and configuration and tuning.

The "month of lunches" format limits the size of each chapter. This is a good thing. Umali crafts each one so that it is complete and self-contained. He encourages readers to spend a bounded time reading and then to go away and think and practice on their own. No chapter is longer than 20 pages. The longest ones are those with a lot of graphics. They either showcase the GUI interface or are concerned with the theory of revision control and so use lots of drawings to show the workflow for the reader. I'm not a good judge of GUI tools, but the base level introduction Umali offers is comparable to the CLI capabilities, and for those who like graphical tools it should serve well.

This is a beginner's book, but I will pass it on with compliments. I did pick up a number of tips and ideas that will stick with me.

Gamestorming: A Playbook for Innovators, Rulebreakers, and Changemakers

Dave Gray, Sunni Brown, and James Macanufo
O'Reilly Media, 2010, 288 pages
ISBN 978-0596804176

Reviewed by Mark Lamourine

Brainstorming is a term in common use. To me it means going somewhere different (even if only in my head), preferably with a couple of my most trusted co-workers, presenting a problem I have in its broadest terms and then throwing around ideas without judgment or ego until something grabs all of our attention. Then we play with a couple of the "best" ideas until we better understand the problem, the challenges that remain, and, most importantly, what we want to try next. This is a very unstructured concept, and other people will have a different vision of what brainstorming is.

Gamestorming is a book that offers a lot of different ways to structure that communal thinking process.

The main idea of *Gamestorming* is to use the framework of a "game" to direct and focus the thinking and sharing process in a way that suits the particular goals of the session. A game, according to Gray, Brown, and Macanufo, is defined primarily by a play space, a set of rules, and a goal. With this loose but clear definition, they set out to give the reader a sense of how game play in a working context can lead both to the results that might elude more conventional planning sessions and to the relevant tools to get those results.

Chapters 2 and 3 present that toolbox. A moderator's job in these kinds of planning meetings is to create an environment that will promote participation and cooperation. There are any number of ways the plan can be derailed. Chapter 2 enumerates 10 "essentials" that are the material needs for a good session. In Chapter 3 the authors lay out the skills and tactics that a moderator should have in order to be able to guide the participants and avoid ratholes and pitfalls.

The body of the book is four chapters that are a catalog of core games, those for opening, closing, and for exploring an idea space. The authors make a clear distinction between games meant to start a session and generate lots of wild ideas and those that are meant to refine and then focus on one concept and come to a close. In longer planning sessions the games might be chained together, or they can be played in separate sessions over a longer period of time if needed.

You may have visions of whiteboards and flip charts and multicolored sticky notes, and you wouldn't be wrong. Most of us won't use *Gamestorming* in day-to-day life as a software developer or sysadmin. The subtitle of the book, "A Playbook for Innovators, Rulebreakers, and Changemakers," feels a bit grandiose to me. Many of the games are fairly common in dramatic training, especially those aimed at creating group coherence. I suspect very little here would be surprising to professional moderators or facilitators.

But we're not that kind of professionals. I think, used judiciously, the ideas here could be helpful to those of us who find ourselves in that position despite our inclinations (or our best efforts). Sometimes it might be a good thing to shake us out of the stale format of our regular planning meetings, standups, or retrospectives. In that case, *Gamestorming* would be a good resource for getting ourselves into the mindset of a facilitator. For the hour or so it takes, perhaps a game is a good way to engage a whole team on a common problem and uncover a solution no one had thought of or felt invited to voice.