Issue 088: Containers

De Programmatica Ipsum

2026-01-05

Contents

Issue 088: Containers

By Adrian Kosmaczewski, January 5th, 2026

Welcome to the 88th issue of *De Programmatica Ipsum*, about *Containers*.

In this edition:

- We realize that an Internet meme encapsulated the whole truth[1] about containers all along.

[1] https://deprogrammaticaipsum.com/shipping-your-computer/

- In our Vidéothèque section[2], we watch the introduction of Docker by Solomon Hykes[3] in 2013.
- In the Library section[4], we review "The Docker Book" by James Turnbull[5].

Download this issue in DRM-free PDF[6] or EPUB[7] format, and read it on your preferred device. You can also subscribe to our RSS feed[8], featuring the full content of our articles.

We would like to thank our patrons who generously contribute every month (or have contributed in the past) to our work and help us run this magazine. Thank you so much! In alphabetical order: Adam Guest, Adrian Tineo Cabello, Benjamin Sheldon, Christopher Nascone, Colin Powell, Franz Lucien Moersdorf, Guillermo Ramos Álvarez, Jean-Paul de Vooght, Dr. Juande Santander-Vela, Patryk Matuszewski, Paul Hudson, Quico Moya, Roger Turner, Szymon Licau, and countless more leaving anonymous tips every month.

Enjoy this issue! Please share our articles on social media, or contribute[9] if you would like to support our work with a donation via Liberapay[10].

Cover photo by Frank McKenna[11] on Unsplash[12].

[2] https://deprogrammaticaipsum.com/category/videotheque/
[3] https://deprogrammaticaipsum.com/solomon-hykes/
[4] https://deprogrammaticaipsum.com/category/library/
[5] https://deprogrammaticaipsum.com/james-turnbull/
[6] https://deprogrammaticaipsum.com/pdf/issue-088-containers.pdf
[7] https://deprogrammaticaipsum.com/epub/issue-088-containers.epub
[8] https://deprogrammaticaipsum.com/index.xml
[9] https://deprogrammaticaipsum.com/contribute/
[10] https://liberapay.com/akosma/donate
[11] https://unsplash.com/@frankiefoto?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[12] https://unsplash.com/photos/assorted-color-filed-intermodal-containers-tjX_sniNzgQ?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Shipping Your Computer

By Adrian Kosmaczewski, January 5th, 2026

The analogy between intermodal containers[13] and software containers sounds cliché, but it works pretty well, which is the reason why we will keep using it in this article. Let us recap: before the standardization around intermodal containers after the Second World War, freight had to be manually loaded and unloaded from boats to trains to trucks, over and over again. Before the rise of software containers of the 2010s, software

[13] https://en.wikipedia.org/wiki/Intermodal_container

had to be manually loaded and unloaded from developer laptops to test environments to cloud systems, over and over again.

Yes, the analogy works. Even the 2021 Suez Canal obstruction[14] can likely be felt whenever Kubernetes complains of not finding a suitable node to schedule your pod. *Le sigh*.

In the final scene[15] of the 1989 buddy cop comedy "Lethal Weapon 2"[16], one of the "bad guys" is (finally!) crushed by the fall of an intermodal container, triggered by a heavily wounded Martin Riggs (played by Mel Gibson). Similarly, developers worldwide have been hit in the head by an artifact of similar virtual mass and devastating effects.

Software containers, still colloquially known as "Docker containers" and shyly introduced[17] over a decade ago, ended up being a surprisingly useful tool, and become one of those concepts everyone in the IT community agreed to be good idea more or less simultaneously. Other examples of such tacit agreements are, in no particular order, the Web[18], Agile[19] methodologies, Open Source[20], Git[21], and the IEEE 754 standard[22]. The jury is still out and debating about Worker Unions[23], but we can guess what the final verdict will be.

Interestingly, containers fit perfectly well with Web technologies, agile teams, open source, and Git repositories. Think about it. Most web applications nowadays are built in pipelines stored next to the Git repositories hosting their source code, built using open source tooling, by agile teams having their daily standup meeting every morning at 9 AM. The circle is closing.

Becoming a standard[24] meant quite a bit of lobbying and back and forth, but for the past decade or so containers have become the daily bread and butter of countless teams around the world.

---

[14] https://en.wikipedia.org/wiki/2021_Suez_Canal_obstruction
[15] https://youtu.be/rDIF3XhXTT8?t=122
[16] https://www.imdb.com/title/tt0097733
[17] https://deprogrammaticaipsum.com/solomon-hykes/
[18] https://deprogrammaticaipsum.com/issue/issue-052-the-world-wide-web/
[19] https://deprogrammaticaipsum.com/issue/issue-039-methodology/
[20] https://deprogrammaticaipsum.com/issue/issue-021-open-source/
[21] https://deprogrammaticaipsum.com/issue/issue-066-version-control/
[22] https://deprogrammaticaipsum.com/issue/issue-073-floating-point-arithmetic/
[23] https://deprogrammaticaipsum.com/issue/issue-042-trade-unions/
[24] https://opencontainers.org/

Of course not all apps could be containerized; in particular, those that did not follow the "Twelve-Factor App"[25] principles had a very hard time being put into such constrained environments, and this triggered yet another massive rewriting of the world.

And this is where an interesting thing happened: containers provoked a schism among programming languages. On one side, scripting languages; on the other, compiled ones.

The former group yielded gargantuan containers, including gigabytes of runtime libraries and dependencies tightly packed together, albeit with ridiculously fast container build times; Python[26], Ruby, JavaScript[27], and PHP[28] being the most remarkable ones in this category. On the other side, instead, we find languages that provide smaller containers but with the added cost of longer build times: Java[29] (most commonly through the use of Quarkus or Spring Boot), .NET, Rust[30], and even the venerable C++[31] language representing this category.

(Well, arguably there is a third group, occupied solely by the Go programming language[32], able to provide smaller containers in absurdly short build times. But regular readers of this magazine know that we consider Go to be a triumph of engineering, anyway.)

This split of functionality in programming languages provided a definitive answer to the question "what programming language should I use for my project?"[33]. If all else fails, choose one that will yield developer productivity *and* short build times *and* small containers for production. Finding this trifecta became the holy grail of web application development for the past decade. It also brought capacity management to the masses: Kubernetes lets you know in no ambiguous terms the amount of memory and CPU your nodes have, so you had better not schedule too many containers at once.

Even better, try to make them smaller, and get more container for the buck.

Once you reach the sweet spot of cost and size and speed (choose three), containers can be used for a myriad of purposes beyond the mere packaging of web applications: for

[25] https://12factor.net/
[26] https://deprogrammaticaipsum.com/issue/issue-035-python/
[27] https://deprogrammaticaipsum.com/innovationscript/
[28] https://deprogrammaticaipsum.com/issue/issue-083-php/
[29] https://deprogrammaticaipsum.com/issue/issue-024-java/
[30] https://deprogrammaticaipsum.com/issue/issue-047-rust/
[31] https://deprogrammaticaipsum.com/issue/issue-077-c++/
[32] https://deprogrammaticaipsum.com/issue/issue-071-go/
[33] https://deprogrammaticaipsum.com/how-to-choose-a-programming-language-for-your-book/

example, for testing[34]; as a mechanism preventing DLL Hell[35]; to share command-line tools[36] between team members and your CI/CD infrastructure; to quickly deploy[37] and run artificial intelligence models; and even for educational purposes[38].

But the most important transformation provided by containers was not just the discrimination of programming languages, but rather something even deeper:

> Over time it became clear that the benefits of containerization go beyond merely enabling higher levels of utilization. Containerization transforms the data center from being machine-oriented to being application-oriented.

(Burns, Brendan, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. "Borg, Omega, and Kubernetes." Communications of the ACM 59, no. 5 (2016): 50–57. https://doi.org/10.1145/2890784.)

Let us be crazy: we can argue that containers also provided yet another way for Unix to find revenge[39]. Because honestly, Windows[40] containers are a joke, and now you can even build and manage OCI containers from macOS[41]. Containers, since the times of `chroot` and FreeBSD jails[42], have been a Unix affair, one that Linux took to a whole new level.

But there is another contribution of container images: secret leakage. Remember when everyone was urging developers to stop committing secrets into Git repos? Well, turns out they started bundling them inside container images, and even better, pushing them to Docker Hub.

> In this paper, we analyze 337,171 images from Docker Hub and 8,076 other private registries unveiling that 8.5 % of images indeed include secrets. Specifically, we find 52,107 private keys and 3,158 leaked API secrets, both opening a large attack surface, i.e., putting authentication and confidentiality of privacy-sensitive data at stake and even allow active at-

[34] https://testcontainers.com/
[35] https://akos.ma/blog/containers-and-dll-hell/
[36] https://akos.ma/blog/reusing-apps-between-teams-and-environments-through-containers/
[37] https://podman.io/
[38] https://akos.ma/blog/containers-for-non-technical-readers-with-podman-desktop/
[39] https://asymco.com/2010/09/29/unixs-revenge/
[40] https://deprogrammaticaipsum.com/issue/issue-080-windows/
[41] https://github.com/apple/container
[42] https://en.wikipedia.org/wiki/FreeBSD_jail

> tacks.

(Dahlmanns, Markus, Constantin Sander, Robin Decker, and Klaus Wehrle. "Secrets Revealed in Container Images: An Internet-Wide Study on Occurrence and Impact." Proceedings of the ACM Asia Conference on Computer and Communications Security, July 10, 2023, 797–811. https://doi.org/10.1145/3579856.3590329.)

Insert facepalm emoji here. In my own personal experience, I have to say that I see more and more teams adopting secure container and Kubernetes practices, including the use of *ad-hoc* secret management databases:

> By default, Kubernetes stores secret data as plaintext in 'etcd'. In that case, if a malicious user gets access to 'etcd', then the malicious user can retrieve sensitive information, such as database user names, passwords, and queries. Although Kubernetes does encrypt 'etcd', the key for the encryption is stored as plaintext in the config file in the master node. For that reason, practitioners recommend using secret management tools for additional security, such as 'Vault' for encryption.

(Islam Shamim, Md. Shazibul, Farzana Ahamed Bhuiyan, and Akond Rahman. "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices." 2020 IEEE Secure Development (SecDev), IEEE, September 2020, 58–64. https://doi.org/10.1109/SecDev45635.2020.00025.)

Insert sigh of relief emoji here.

In an article[43] published 6 years ago, in our issue dedicated to the subject of DevOps[44], published right before the COVID pandemic broke, we said that:

> Applications became small Docker containers, lightweight virtual machines that can start and stop in a heartbeat. The programming language does not matter anymore. The IDE does not matter anymore. The framework does not matter anymore. The build tool does not matter anymore.

Through containers, the work of software developers reaches as far as possible into the realm of operation teams; the design of their containers becoming a form of non-trivial calisthenics, as we mentioned in another article[45]:

[43] https://deprogrammaticaipsum.com/antonomasia/

[44] https://deprogrammaticaipsum.com/issue/issue-016-devops/

[45] https://deprogrammaticaipsum.com/the-impossible-dialogue-revisited/

> Different software developers have to deal with different constraints. For example, DevOps engineers busy stuffing code inside a Docker container might not realize the amount of design they have to deal with at any step of the way: should they base their work on `alpine:latest` or `scratch`? Can they? Should they include all libraries, or can they strip a few to slash some megabytes? What programming languages would be better suited to make this container run faster and smaller?

A very-well known Internet meme[46], inspired by a scene from the 2004 movie "Finding Neverland"[47], features a sequence of images[48] in which a very young Freddie Highmore tearfully admits that "it works in my machine", to which Johnny Depp replies "then we'll ship your machine", with the last caption stating "and that is how Docker was born".

We do not need to say anything else at this point.

Cover photo by David Trinks[49] on Unsplash[50].

[46] https://knowyourmeme.com/memes/finding-neverland

[47] https://www.imdb.com/title/tt0308644/

[48] https://programmerhumor.io/docker-memes/it-works-on-my-machine-kswk

[49] https://unsplash.com/@dtrinksrph?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[50] https://unsplash.com/photos/a-large-cargo-ship-in-the-middle-of-the-ocean-0VZ1PSBleJY?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Solomon Hykes

By Adrian Kosmaczewski, January 5th, 2026

Around 2004 I was working as a .NET software developer, building custom applications for a rather large Swiss customer who shall remain nameless. As part of the engagement, we had to not only write said software application following their requirements, but also deliver it in person. Yes, in person. We had to take a snapshot of the source code of the application (stored in a Visual SourceSafe[51] repository, please do not laugh too loud), save it in a USB thumb drive, and drive to the location, where we would explain the deployment team how to build and deploy it.

Never mind that Visual Studio did actually offer the possibility of creating "MSI" in-

[51] https://en.wikipedia.org/wiki/Microsoft_Visual_SourceSafe

stallers[52] for .NET applications, bundling precisely every single dependency together with the EXEs and DLLs, and copying every artifact in the precise location they needed to be for IIS (Windows' canonical web server at the time) to find and run them. This customer did not want to hear about MSIs, and insisted on a manual delivery and a manual installation performed by their own team members.

Clearly, we were very far away from DevOps[53] at the time, let alone basic automation, or standard common sense.

Of course, no matter how long the README or how well we explained the processes in person, the teams could not get it to work, and lots of blaming and finger pointing ensued. We repeated this operation for weeks, day after day, until one day, seemingly out of a miracle, we could get the application to run in their Windows Server 2003 boxes.

Insert hallelujah emoji here.

(For the record, it was a canonical "two-tier" monolithic application[54] built on top of ASP.NET, with Visual Basic .NET, consisting of at most a couple of DLLs, talking to a rather dull SQL Server 2000 database behind the scenes. Seriously, nothing to phone home about.)

This scenario happened merely 22 years ago. Delivering software was not trivial, although I admit that the example I described above was one of the worst memories I have of this kind. But it was always complicated, requiring a separate workflow for every customer, and there was no hint of a standardized solution in the horizon.

Fast-forward 9 years, to June 7th, 2013. That day, in Paris, took place the dotScale 2013 conference, which still exists at the time of this writing, albeit named dotConferences[55]. That day, a young French-American entrepreneur called Solomon Hykes from a company called dotCloud (a name unrelated to the conference despite the naming convention) delivered a 20-minute-long presentation that would have an unprecedented impact in our industry. And the recording of that session is, precisely, this month's Vidéothèque entry: "Why we built Docker"[56]

---

[52] https://en.wikipedia.org/wiki/Windows_Installer

[53] https://deprogrammaticaipsum.com/issue/issue-016-devops/

[54] https://www.ibm.com/docs/en/db2-for-zos/12.0.0?topic=environment-architectural-characteristics-web-based-applications

[55] https://www.dotconferences.com/

[56] https://www.youtube.com/watch?v=3N3n9FzebAA

Mr. Hykes provides, in those 20 minutes, the essential idea behind containers, and the major issue they solved. Precisely, the problem I described from 2004: how to ship software from environment to environment, from workstation to workstation, from here to there, ensuring that every piece of your architecture remains in place and working.

If moving around a two-tier monolith was complicated in 2004, imagine moving around a microservices-based application, with many OLTP[57] databases plus a few OLAP[58] services, a key-value store[59] for session management, a Kafka[60] cluster for messaging, a stateless business logic engine written in some esoteric language, a UI built with some kind of ~~magic~~ JavaScript framework, connectors to external APIs, and who knows what else.

Solomon rightfully says that moving software from environment to environment is brittle and labor-intensive. I can confirm. Been there, done that. All of this was getting worse because of the explosion of cloud computing back in the 2010s.

And let us not even get into the burden of maintenance, applying security patches, *und so weiter*.

His idea, then, was to pay attention to the shipping industry, and the solution it came up with for moving stuff around: intermodal containers[61]. Essentially, a fixed, standardized box that can be moved from a boat to a train, from a train to a truck, from a truck to a warehouse, regardless of the contents of the box, in a simple sweeping movement of a crane.

Borrowing from the lexicon of software developers, Solomon rightfully states that containers allow a separation of concerns: shipping companies do not care about what is in the box, as long as the box fits the correct sizing and weighing requirements (well, and hopefully as long as the contents are not illegal, either).

In contrast, Solomon states, the software situation is pretty embarrassing, and the current canonical solution at the time of the talk, virtual machines, are just plain inefficient, with their bundling of a kernel image for every instance of every application (even though utilities like Vagrant[62] had substantially simplified their management up to that point).

---

[57] https://en.wikipedia.org/wiki/Online_transaction_processing
[58] https://en.wikipedia.org/wiki/Online_analytical_processing
[59] https://en.wikipedia.org/wiki/Key%E2%80%93value_database
[60] https://kafka.apache.org/
[61] https://en.wikipedia.org/wiki/Intermodal_container
[62] https://developer.hashicorp.com/vagrant

Speaking about operating system[63] kernels, the introduction of namespaces in the Linux kernel made containers possible, as well as the company that Solomon started (conveniently renamed later from dotCloud to Docker, Inc.[64]).

Their flagship project (in version 0.4 at the time of the talk), simply called "Docker", and built with the Go programming language, introduced three major breakthroughs: a standard format for container images; a set of tools for developers to build container images; and another set of tools for operators to run and manage those images. All in a lightweight bundle, seemingly integrated with whatever build and operation tooling already in place.

Solomon stayed at Docker Inc. until 2018[65], and these days he has been spotted starting other ventures[66], but apparently he does not do any AI stuff, so nobody is paying attention anymore.

Watch this month's Vidéothèque article, "Why we built Docker"[67], by Solomon Hykes, on YouTube, a historically relevant keynote that had a terrific impact in our industry.

After watching this, we can recommend "Kubernetes: The Documentary" (first[68] and second[69] parts). If all of this sounds interesting for your next career move, check the "Getting Started with Kubernetes"[70] video course by Nigel Poulton[71] on the ACM website. And, if you feel adventurous, get your hands dirty with the classic "Kubernetes The Hard Way"[72] by Kelsey Hightower[73]. Welcome to the club.

Cover snapshot chosen by the author.

---

[63] https://deprogrammaticaipsum.com/issue/issue-056-operating-systems/

[64] https://en.wikipedia.org/wiki/Docker,_Inc.

[65] https://www.geekwire.com/2018/docker-co-founder-solomon-hykes-leaving-company-cites-need-enterprise-focused-cto/

[66] https://devclass.com/2024/02/12/from-docker-to-dagger-solomon-hykes-on-modernisation-of-the-devops-pipeline/

[67] https://www.youtube.com/watch?v=3N3n9FzebAA

[68] https://www.youtube.com/watch?v=BE77h7dmoQU

[69] https://www.youtube.com/watch?v=318elIq37PE

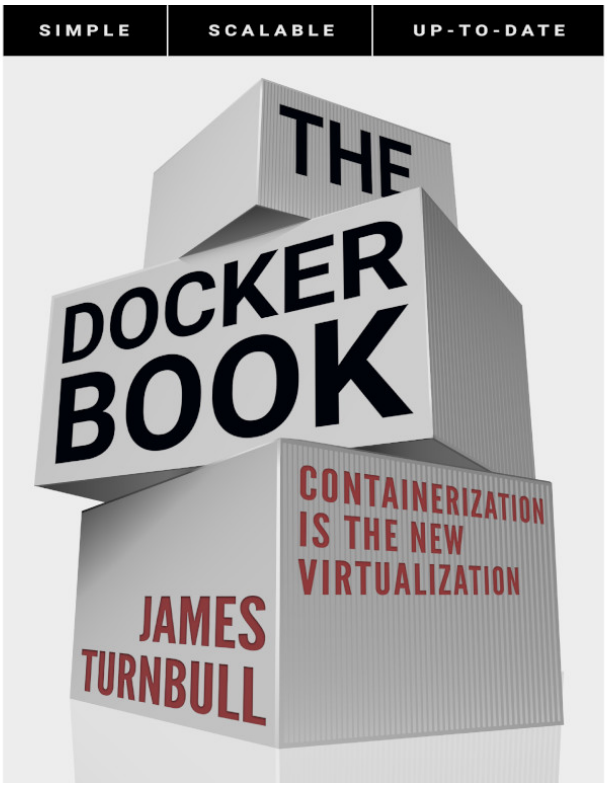
[70] https://learning.acm.org/techtalks/kubernetespoulton

[71] https://www.nigelpoulton.com/

[72] https://github.com/kelseyhightower/kubernetes-the-hard-way

[73] https://en.wikipedia.org/wiki/Kelsey_Hightower

James Turnbull



By Adrian Kosmaczewski, January 5th, 2026

It is easy to forget, in our age of AI and LLMs and slop and coding agents, that merely 10 years ago the "cloud" and "DevOps" and "containers" were all the rage. It seems like a century ago, yet it was not only during the 21st century, but it was almost yesterday. The sensation that time is flying is not anymore a privilege for the people above 40 years old.

Precisely at the time this magazine started annoying its readers with pleas for contributions[74] and tongue-in-cheek commentary, containers and Kubernetes were slowly becoming the most coveted new technology in town. It is (still) quite impressive for this author, who started his career deploying applications via FTP[75] in 1997, to see CI/CD pipelines building containers, storing them in a "registry", and then watching an Argo CD[76] system being triggered out of nowhere, running a Helm[77] chart and deploying yet another version of your application in real time, automatically, several times a day.

This is the kind of revolution that containers enabled; putting applications in production was, indeed, complex and tedious. Managing them was even more complicated. Containers enabled a layer of communication and flow between developers and sysadmins that was hitherto unprecedented. Now you can mess things up faster than ever, and you can roll them back even faster.

So, back in 2018, when this magazine saw the light, containers were synonymous with the word "Docker". For many of us in the industry, "building a Docker container" was the *mot du jour*, the prime directive, and the door to new opportunities.

Enter the Library entry of this month: "The Docker Book"[78] by James Turnbull[79], which was for half a decade (between 2014 and 2019) the official resource to learn about the ins and outs of this new technology. The author has also written "The Terraform Book"[80], "The Logstash Book"[81], "The Packer Book"[82] (can you see the pattern here), and "Monitoring with Prometheus"[83] (ah, sadly not following the same name convention), and has worked for quite a few companies in the DevOps space; you are in good hands here.

The book deals with the usual container workflows: build an image, push it to a registry, pull it back, run it, get rid of it. The reason most of your projects have a file named `Dockerfile` (or, more recently, `Containerfile`) at the root of their Git repository can be traced back to this basic procedure.

---

[74] https://deprogrammaticaipsum.com/contribute/
[75] https://en.wikipedia.org/wiki/File_Transfer_Protocol
[76] https://argoproj.github.io/cd/
[77] https://helm.sh/
[78] https://dockerbook.com/
[79] https://turnbull.press/
[80] https://terraformbook.com/
[81] https://logstashbook.com/
[82] https://packerbook.com/
[83] https://www.prometheusbook.com/

At least the latest versions of the book also dealt with the following issue: what about running lots of containers at the same time? For a while, Docker Compose and later Docker Swarm appeared as suitable solutions for the conundrum. A single mechanism allowing you to run as many containers as you need–because once you run a container, you want to run thousands of them. And the new fad called "microservice architecture" screamed for monoliths to be broken down into pieces, as small as possible, stuck inside containers, and run on an orchestrator. That was the new hype[84] at the time.

As time has shown, both projects, Docker Compose and Docker Swarm, ultimately yielded under the massive pressure of Kubernetes and the whole Cloud Native Computing Foundation landscape[85] web page, able to take down your browser given its massive size. Do not click on the previous link unless your computer has 32 GB of RAM or more.

Kubernetes became DevOps by antonomasia[86]:

> The "container orchestration wars" are over, my general, and Kubernetes won.

And through Kubernetes, ~~Docker~~ containers also won.

Docker, the company and the tool, are slowly fading away into oblivion. It is quite a telling story, that of a company creating and releasing a massive standard, yet unable to stay relevant in the market just a decade later. Even Docker Hub[87], the first container registry ever, has been slowly left behind by individuals and organizations (and not only because of its drastic terms of service changes) in favor of other solutions, of which there is no shortage nowadays: Harbor[88], Sonatype Nexus[89], GitLab, GitHub, Quay[90], AWS, Azure, or even the registry bundled inside every OpenShift cluster.

Following this trend, "The Docker Book" is no longer available for purchase or download. And rightfully so. The book belongs to a time when you had to `sudo` your way around containers, something that was, needless to say, a heresy, and a source of white hair for quite a few security boffins.

---

[84] https://deprogrammaticaipsum.com/issue/issue-001-hype/
[85] https://landscape.cncf.io/
[86] https://deprogrammaticaipsum.com/antonomasia/
[87] https://hub.docker.com/
[88] https://goharbor.io/
[89] https://help.sonatype.com/en/docker-registry.html
[90] https://www.projectquay.io/

Tools like Podman[91], its UI counterpart, Podman Desktop[92], and their terminal acolyte, Podman TUI[93], do not have such requirements. These new tools, fully open-source, implement the OCI standard[94] and give you the same containers without you having to request any privilege at all. (Disclaimer: I am employed by the very company that makes Podman, but I recommend its use regardless of this fact.)

Maybe a new version of "The Docker Book" could be made, called "The Podman Book"; that is something I've tried to do[95] in a much smaller scale. But let us be honest, a much better alternative in this respect would be "Podman in Action"[96] by Daniel Walsh.

In any case, "The Docker Book" remains a highlight of a rare moment in time, when the whole of the software and IT industries tacitly endorsed the same concept simultaneously, almost unconsciously, redefining a certain idea of productivity for the decades to come.

Cover photo from the book website.

[91] https://podman.io/
[92] https://podman-desktop.io/
[93] https://github.com/containers/podman-tui
[94] https://opencontainers.org/
[95] https://akos.ma/blog/containers-for-non-technical-readers-with-podman-desktop/
[96] https://www.manning.com/books/podman-in-action