Generic Pipelines Using Docker

The DevOps Guide to Building Reusable, Platform Agnostic CI/CD Frameworks

—

Brandon Atkinson
Dallas Edwards

Apress®

Generic Pipelines Using Docker

The DevOps Guide to Building Reusable, Platform Agnostic CI/CD Frameworks

Brandon Atkinson
Dallas Edwards

Apress®

Generic Pipelines Using Docker

Brandon Atkinson
North Chesterfield, VA, USA

Dallas Edwards
Midlothian, VA, USA

ISBN-13 (pbk): 978-1-4842-3654-3
ISBN-13 (electronic): 978-1-4842-3655-0
https://doi.org/10.1007/978-1-4842-3655-0

Library of Congress Control Number: 2018962365

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Joan Murray
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484236543. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Authors

Brandon Atkinson is a software engineer with more than 14 years of industry experience encompassing analysis, design, development, and implementation of enterprise-level solutions. His passion is building scalable teams and enterprise architecture that can transform businesses and alleviate pain points. He has extensive experience in various technologies/methodologies, including Azure, AWS, .NET, DevOps, Cloud, JavaScript, Angular, Node.js, and more. Brandon lives in Richmond, Virginia, USA with his wife and two daughters.

Dallas Edwards has more than ten years of experience as a software engineer. He thrives on creating solutions that are pragmatic, scale easily, and that are easy to test and maintain. His experience encompasses a wide range of expertise, including software development, iOS application development, and DevOps. Dallas lives in Richmond, Virginia, USA with his wife and is an avid scuba diver.

About the Technical Reviewer

Alex Fabian is a professional software engineer with experience building enterprise-scale Java systems, APIs, and web applications. He has hands-on experience implementing Docker-based deployment pipelines for microservices and has spent considerable time integrating with various offerings from Amazon Web Services. Alex has had the opportunity to work on large- *and* small-scale projects for Fortune 500 companies, technology startups, government contractors, and small organizations. He holds certifications from Amazon Web Services and a B.S. in Computer Science from The University of Virginia.

Acknowledgments

Brandon Atkinson

Writing a book is very hard work, and very time consuming. I could not have done this without the support of my family, especially my amazing wife Jennie and my wonderful kids. I also want to thank Dallas for coming on this journey with me. I did my best to take over your life; you're about to get it back! I also want to thank Chris Bowers for setting me on this DevOps path, and providing a lot of the vision that made this pattern come true.

Dallas Edwards

There's no way this could've happened without the constant support and encouragement of my family. I am so grateful for the role each of you has played in my life.

I'd like to thank Brandon for giving me this opportunity. I can't wait to see what you drag me into next!

Finally, to my wife Cierra: thank you so much for all of your advice and sticking with me through all the late nights. I already proved it.

Introduction

DevOps (a clipped compound of "development" and "operations") is a software engineering culture and practice that aims at unifying software development (Dev) and software operation (Ops). The main characteristic of the DevOps movement is to strongly advocate automation and monitoring at all steps of software construction, from integration, testing, and releasing to deployment and infrastructure management.

That definition pretty much sums up what most people think of when they hear the term DevOps. Not because it perfectly describes what it is, however; it leaves you more confused than before you asked the question "What is DevOps?". DevOps comes in so many forms it can be hard to nail down exactly what it is. I've worked in a lot of different shops in my time and DevOps is always something different at each stop.

For most it involves developers writing code that is checked into source control, which immediately kicks off a build pipeline that deploys your application. That pipeline will perform various steps that may include stages like building, testing, and deploying. From my experience, if you're just doing these simple steps through automation, you're way ahead of some. However, for many this is not enough to have your organization be considered as fully embracing DevOps.

You may also want to include things like automated infrastructure creation, security scans, static code analysis, and more. In an ideal situation everything you do in a software shop is stored as code in source control. This includes your application code, infrastructure scripts, database scripts, networking setup, etc. With a push of a button everything your application needs to run can be created on the fly. For a lot shops, this is what it means to truly embrace DevOps.

Some organizations are more mature than others when it comes to DevOps practices. In my experience you're well on your way to maturity if you're doing the following items:

- You deploy your application via an automated pipeline.
- Once built and tested, application code binaries can be promoted via an automated pipeline.

I realize a lot of people will look at that list and exclaim "WHAT?!". There are only two items, and your organization may be way beyond those. However, most are not and would see immense improvements by just doing these two things. If you're one of the folks who thinks this list is crazy small, then count yourself lucky. I know many people who would kill for just these two items.

Automated pipelines in my opinion are the lifeblood of good DevOps practices. They provide so many benefits to both the team of developers as well as the business that relies on their code. A well-crafted pipeline gives you a repeatable process for building, testing, and deploying your application. It can be used to create artifacts that, once built, are simply promoted, ensuring that what makes it to Production has been tested and vetted.

Pipelines can also be a pain point for organizations as well. You may have multiple applications, each written in different languages, and each with their own finicky way of being built. It can be a nightmare at times jumping between technologies, debugging the various stages, and keeping the lights on. Luckily modern technology is helping us get around some of these issues. Technology like Docker has given us an opportunity to standardize our platforms. By utilizing Docker in your pipeline, you can give your developers some peace of mind that if they can build it, so can you.

Combine this with cloud technology like Amazon ECS or Azure Container Service and now we can extend that piece of mind all the way

to deployment. If it runs locally in your Docker daemon, it will run in the cloud. This even applies if you're running an enterprise cloud platform like Nutanix or Dell; if the destination is a container orchestration service you're golden. Now, consider that .NET Core is open source and can run on Linux, and you've pretty much covered all your bases. It's a great time to be in technology!

This book aims to show how you can use all this technology to simplify your pipelines and make them truly generic. Imagine a single pipeline that deploys all your code regardless of the tech stack it is written in. It's not a dream; we'll show you how.

Who This Book is For

This book was written with the DevOps professional in mind who may be struggling with writing and maintaining multiple pipelines. However, it's also for anyone in technology who is interested in learning about building generic pipelines.

You don't have to be a DevOps master to get a lot from this book; however, you will benefit more with experience in the following areas:

- *Docker*: Being familiar with Docker from creating Dockerfiles, building and running images, etc. We won't get very deep into Docker, but a working knowledge is key.
- *Bash Scripts*: Most of the examples in this book are written in Bash. You should have at least some minimal experience with scripts.
- *CI/CD Platform Experience*: You'll be much better off if you already have experience with a platform like Circle CI or Jenkins. However, throughout the book we'll walk you through working with these platforms.

As you can see, having some DevOps and pipeline experience will certainly benefit you as you read this book, but it's not a hard requirement. If you're a developer of any type, you should have no issues jumping right into the examples in this book. As stated, the future of DevOps and pipelines is everything is code. If you're comfortable writing code, you'll be fine.

Why We Wrote This Book

This book is the culmination of a lot of hard work our team has done over the last year. We were tasked with creating a generic, reusable pipeline that any development team could utilize regardless of the language they were writing in. Our organization was going through a transformation from monolithic applications to microservices, and the old way of delivering code wasn't going to cut it.

After we decided on our plan of attack and began implementing, we realized we had something special on our hands. Talking with other groups around us, no one was tackling CI/CD pipelines in quite the same way. Most were still building a pipeline per application. Some were taking a step further and building tech stack–specific pipelines that could work with teams using the same language. However, no one was going the extra step and building a single pipeline for all.

Now that we've been iterating on our code for a while we've seen some amazing results. Our development teams can quickly produce microservices and on-board them to the pipeline in a matter of minutes. By utilizing Docker locally on the developer's machines and in the pipeline, we can ensure that the experience is extremely similar. Using Docker in the pipeline gives us full control over the process regardless of what the underlying platform has installed.

We've seen this pattern take hold in our group and how it's transformed the way we deliver code. This book is our effort to share what

we've learned and hopefully pass along some of that knowledge. If you come away from this book with even a few things you find useful, we'll take that as a success. If you end up taking this pattern and applying it in your organization—even better. We've seen its successes and believe in it 100%!

What We Won't Cover in This Book

This book is meant to show you how to build generic pipelines utilizing Docker. We won't be covering any CI/CD principles (except by reference), or talking about the benefits of DevOps and how it can transform your organization. We're going to assume you are either already on-board with these concepts or at least know about them in some way.

There are many well-written books out there that extoll the benefits of proper continuous integration and delivery. We don't need to hash all that out here, nor could we do a better job than those before us. We won't be covering Agile principles or how to write better software. We won't argue which CI/CD platform is better or which language produces the best code.

While you're in this book we're going to focus on why monolithic applications and their pipelines are bad. How microservices and smaller pipelines are good. We'll follow that up with why generic reusable pipelines are even better. Then we'll wrap all that up with a lot of examples. So, if you want or need a deep dive into the benefits of DevOps and proper CI/CD practices, search out another book. If you want to build leaner more efficient pipelines, read on.

What You'll Need

Before moving on we'll need to cover the items you'll need to work with the examples in this book. The first few chapters deal with why you want to look at generic pipelines, and later chapters present implementing an example. So, you will not immediately need all of these. If you like, you can

read ahead and deal with these later. If you prefer to be prepared ahead of time, here is what you'll need:

- Mac or Windows 10
- Docker for Mac/Windows
- IntelliJ IDEA (or another IDE)
- CI/CD Platform
- Docker Hub Account
- GitHub Account

Let's dig into each of these and walk through installing, signing up, and configuring each.

Mac or Windows 10

At first glance you may be thinking "Why include a section on this? Of course we have one of these." You would not be entirely wrong for thinking this, but there is a good reason. If you're on a Mac you're all set, you are set up well for what's about to come. If you're on a Windows machine, we need to talk.

We highly recommend using Windows 10 if your only option is Windows. The main reason for this is the Docker experience. In Windows 10, Docker is a first-class citizen and the experience closely matches that on a Mac. In addition, Windows 10 now supports the Linux subsystem, which allows you to use Bash natively.

We also realize not everyone can be on the latest hardware and operating systems. You should be able to complete all the examples in this book with any operating system that you can install Docker on. However, this means you may have to jump through more hoops to get everything working correctly. As we move forward, we will assume you are either on a Mac or Windows 10.