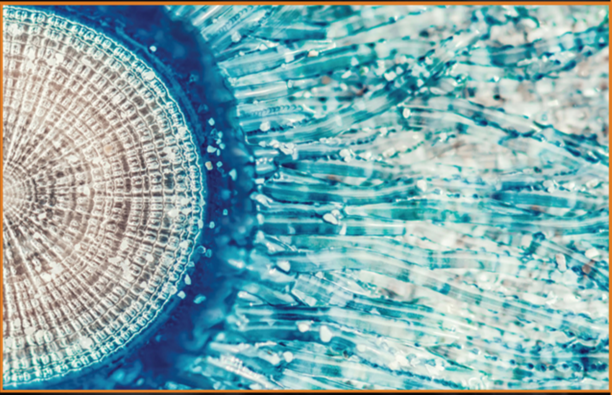
The Addison-Wesley Signature Series

A Vaughn Vernon Signature Book

Serverless as a Game Changer

How to Get the Most Out of the Cloud

Joseph Emison



FREE SAMPLE CHAPTER

"Joe Emison is the apostle of leverage in technical decision-making: betting on managed services for scale and speed while putting developers in positions to contribute maximum business value. Leaders bold enough to follow Joe's path in this brilliant book will reap outsized rewards."

—Forrest Brazeal, Head of Developer Media at Google Cloud

"Joe's been telling the world for years that modern software architecture isn't just about getting rid of your servers, but deleting most of your server code. It's a message that bucks conventional wisdom, and the "best practices" of the kubernetes-industrial complex. But here's the weird thing—he might just be right."

—Mike Roberts, Partner and Co-founder, Symphonia

"This book backs up theory with the real-world practices that Joe has been successfully implementing as a business strategy for years. It is a handbook for modern development teams that want to deliver value faster, more securely, and with less technical debt."

—Jeremy Daly, CEO at Ampt and AWS Serverless Hero

"*Serverless as a Game Changer* is an indispensable book, guiding startups and enterprises to better business outcomes. With its technical expertise, it unveils the power of Serverless applications, focusing on organizational needs and risk mitigation. If you aim to embrace cutting-edge tech and build Serverless solutions, this is a must-read."

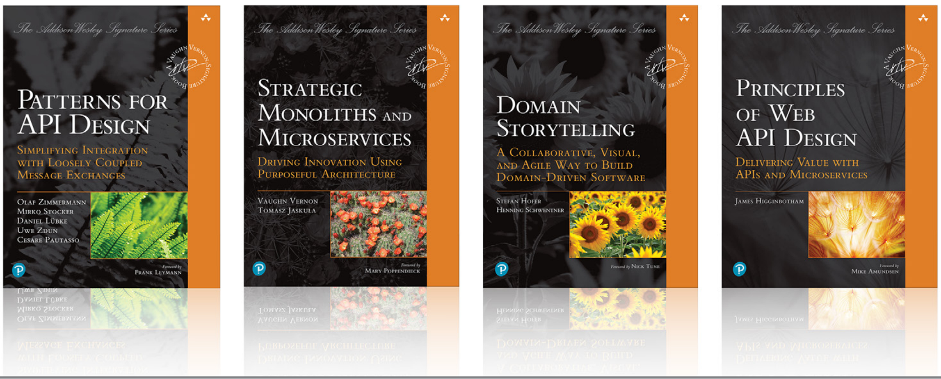
—Farrah Campbell, Head of Modern Compute Community, AWS

"A must-read for executives and technologists who want to go faster and capture more business value for less. Serverless will change the way you build businesses with technology, and this book will show you how."

—Yan Cui, AWS Serverless Hero

This page intentionally left blank

Serverless as a Game Changer


The Addison-Wesley Signature Series
Patterns for API Design
Simplifying Integration with Loosely Coupled Message Exchanges
Olaf Zimmermann
Mirko Stocker
Daniel Lübke
Uwe Zdun
Cesare Pautasso
Frank Leymann
The Addison-Wesley Signature Series
Strategic Monoliths and Microservices
Driving Innovation Using Purposeful Architecture
Vaughn Vernon
Tomasz Jaskula
Mary Poppendieck
The Addison-Wesley Signature Series
Domain Storytelling
A Collaborative, Visual, and Agile Way to Build Domain-Driven Software
Stefan Hofer
Henning Schwentner
Nick Tune
The Addison-Wesley Signature Series
Principles of Web API Design
Delivering Value with APIs and Microservices
James Higginbotham
Mike Amundsen

Serverless as a Game Changer

How to Get the Most Out of the Cloud

Joseph Emison

Library of Congress Control Number: 2023944699

ISBN-13: 978-0-13-739262-9
ISBN-10: 0-13-739262-1

$PrintCode

Trademarks

Contents

This page intentionally left blank

Foreword

When I was introduced to Joe Emison (nearly three years ago at the time of writing this foreword) I was very impressed by his knowledge of serverless. It's not that Joe—a serial entrepreneur CTO—knows more about the entire topic than other cloud experts, although he probably does in a lot of cases. What impressed me the most is his understanding of how to use cloud serverless and managed services to truly improve the software's production behaviors and dependability, all while saving a lot of money as a result. And when I say, "saving a lot of money," I mean a staggering amount. I've heard of startups going under due to unexpected cloud costs, such as the surprise invoice for US $100,000+ for one month. At the time that I met Joe, his startup, Branch, was spending a whopping US $800 per month on everything cloud and serverless. Try to beat that. Since then, he's kept costs way down and company value way up by delivering good software with customer value that always stays running. The bottom line is that Joe is solving business problems first by using technology intelligently. He's been successful to the extent that his company became a unicorn in 2022, just four years after starting up. Without a doubt, Joe knows business and he knows the business of software. And his track record started long before Branch. I explain more below about Joe's stellar, success-rendering approaches to cloud-native systems.

My Signature Series is designed and curated to guide readers toward advances in software development maturity and greater success with business-centric practices. The series emphasizes organic refinement with a variety of approaches—reactive, object, as well as functional architecture and programming; domain modeling; right-sized services; patterns; and APIs—and covers best uses of the associated underlying technologies.

From here I am focusing now on only two words: organic refinement.

The first word, *organic*, stood out to me recently when a friend and colleague used it to describe software architecture. I have heard and used the word *organic* in connection with software development, but I didn't think about that word as carefully as I did then when I personally consumed the two used together: *organic architecture*.

Think about the word *organic*, and even the word *organism*. For the most part these are used when referring to living things, but they are also used to describe inanimate things that feature some characteristics that resemble life forms. *Organic* originates in Greek. Its etymology is with reference to a functioning organ of the body. If you read the etymology of *organ*, it has a broader use, and in fact organic followed suit: body organs; to implement; describes a tool for making or doing; a musical instrument.

We can readily think of numerous organic objects—living organisms—from the very large to the microscopic single-celled life forms. With the second use of organism, though, examples may not as readily pop into our mind. One example is an organization, which includes the prefix of both *organic* and *organism*. In this use of *organism*, I'm describing something that is structured with bidirectional dependencies. An organization is an organism because it has organized parts. This kind of organism cannot survive without the parts, and the parts cannot survive without the organism.

Taking that perspective, we can continue applying this thinking to nonliving things because they exhibit characteristics of living organisms. Consider the atom. Every single atom is a system unto itself, and all living things are composed of atoms. Yet, atoms are inorganic and do not reproduce. Even so, it's not difficult to think of atoms as living things in the sense that they are endlessly moving, functioning. Atoms even bond with other atoms. When this occurs, each atom is not only a single system unto itself but becomes a subsystem along with other atoms as subsystems, with their combined behaviors yielding a greater whole system.

So then, all kinds of concepts regarding software are quite organic in that nonliving things are still "characterized" by aspects of living organisms. When we discuss software model concepts using concrete scenarios, or draw an architecture diagram, or write a unit test and its corresponding domain model unit, software starts to come alive. It isn't static, because we continue to discuss how to make it better, subjecting it to refinement, where one scenario leads to another, and that has an impact on the architecture and the domain model. As we continue to iterate, the increasing value in refinements leads to incremental growth of the organism. As time progresses so does the software. We wrangle with and tackle complexity through useful abstractions, and the software grows and changes shapes, all with the explicit purpose of making work better for real living organisms at global scales.

Sadly, software organics tend to grow poorly more often than they grow well. Even if they start out life in good health they tend to get diseases, become deformed, grow unnatural appendages, atrophy, and deteriorate. Worse still is that these symptoms are caused by efforts to refine the software that go wrong instead of making things better. The worst part is that with every failed refinement, everything that goes wrong with these complexly ill bodies doesn't cause their death. Oh, if they could just die! Instead, we have to kill them, and killing them requires nerves, skills, and the intestinal fortitude of a dragon slayer. No, not one, but dozens of vigorous dragon slayers. Actually, make that dozens of dragon slayers who have really big brains.

That's where this series comes into play. I am curating a series designed to help you mature and reach greater success with a variety of approaches—reactive, object, and functional architecture and programming; domain modeling; right-sized services; patterns; and APIs. And along with that, the series covers best uses of the associated underlying technologies. It's not accomplished at one fell swoop. It requires organic refinement with purpose and skill. I and the other authors are here to help. To that end, we've delivered our very best to achieve our goal.

Considering my goals, I couldn't pass up the opportunity to include Joe's book in my Signature Series. "It's not my uptime" is Joe's fundamental thinking on using serverless. It's not a cliché. It's smart computing.

I was impressed that Joe uses multiple cloud providers for what they do best. That might sound obvious, but I don't mean just AWS, Google Cloud, and Azure. Would most CTOs consider using Netlify, and later switching to Vercel with little rework, to serve Web content and application functionality? I doubt it, and in my experience, no they wouldn't. Yet, Joe has proven how powerful, practical, and cost effective that is. How many chief architects would insist on using a fully managed cloud search service rather than tweaking, contributing to, and self-managing ElasticSearch? In my experience, not many. Every single time that Joe could use a cloud-managed service rather than building a service or use a self-managed software product, he chooses managed services. He weaved it into company culture. What CTO would have the patience to hire and train JavaScript frontend developers to build serverless applications? Joe does, and his people learn quickly. Which architect or developer would choose to build a small-footprint monolith that's deployed as a set of serverless functions? Joe has delivered around 12 of those so far, rather than hundreds or thousands of tiny functions. That's not all. I've only scratched the surface. Joe is an example in business computing and his book is a master class on *Serverless as a Game Changer*. So read on and persuade others in technology leadership roles to do the same.

—Vaughn Vernon, series editor

This page intentionally left blank

Acknowledgments

I would like to thank my wife, Abbie, and children, Seamus, Grace, and Amelia, who have been a consistent source of support. It wouldn't be possible or enjoyable to write a book without you.

Special thanks to Vaughn Vernon, whose invitation to put my thoughts on how to build Serverlessly several years ago pushed me into action.

Holly Tachovsky and Steve Lekas, thank you both for being amazing co-founders who encouraged me to build software and run tech organizations in the way that I saw would be best, as opposed to judging me by any past/other standards you had encountered.

Thank you to all the reviewers of early drafts of this book: Ben Kehoe and Mike Roberts, thank you for your subject-matter expertise and encouragement to make the book stronger and more applicable to specific leaders and executives. Joel Goldberg, thank you for your stakeholder expertise and excellent questions about how to apply the advice you read. And Geraldine Powell (my mother!), thank you for your help in making the book much clearer for everyone.

I would like to thank everyone at Pearson, and especially Haze Humbert, for your kindness and support along the way.

And finally, I would like to thank my parents, who supported me (financially and emotionally) to pursue software development at an early age with Turbo Pascal 5.0 in 1994, and to my uncle, Jon Masters, who bought me *Turbo Pascal by Example* by Greg Perry, which brought me over that huge initial hurdle every software developer must overcome.

About the Author

Joe Emison is the co-founder and CTO of Branch, a personal lines insurance company. Before Branch, Joe built five other companies as a technical co-founder, across many industries, including consumer electronics, local government, big data analysis, and commercial real estate. Joe graduated with degrees in English and Mathematics from Williams College and has a law degree from Yale Law School.

Figure Credits

Figure 2.2: Quora, Inc
Figures 5.1a, B.3: Amazon Web Services, Inc
Figure 6.2, C.3: Microsoft Corporation
Figures A.2, B.1, C.2: Google LLC
Figure B.2: Okta, Inc
Figure E.1: Algolia
Figure E.2: Cloudinary
Figure E.3: Twilio Inc
Figure E.4: Peaberry Software, Inc
Figure E.5: Lob
Figure E.6: Smarty
Figure E.7: Expected Behavior, LLC
Figure E.8: Prismic
Figure E.9: Flatfile
Figure E.10: QuotaGuard
Figure E.11: Basis Theory, Inc
Figure E.12: Upstash, Inc

Chapter 1

Introduction

The gap between the best software development teams and average software development teams is enormous. The best teams deliver delightful, scalable, stable software to users quickly and regularly. Average teams deliver acceptable software that works most of the time, with months or even years between significant improvements. Many teams even struggle to deliver anything but small, iterative changes, with even those taking surprising amounts of time. Software development and cloud services and tools make developing and deploying software increasingly cheaper, faster, and better—but very few organizations take advantage of these innovations. This book lays out the principles, strategies, and tactics that can propel your organization or teams to world-class output by harnessing a Serverless mindset.

Historically, to deliver web-scale applications with wonderful customer experience, organizations have had to hire expensive, superior talent in many disciplines: networking, infrastructure, systems administration, software architecture, back-end software development, API design, and front-end application design. The demand for these talented employees has always exceeded the number of them, and most companies have simply been unable to acquire talent that would enable them to build products such as a Google or Netflix.

That world has changed, and building web-scale, world-class applications is now possible without needing to hire any “ninja” or “rockstar” developers. Serverless applications leverage the immense talent within cloud providers and managed services to solve the hard problems of building and scaling complex infrastructure and software systems. The average developer today is more capable of delivering the next Gmail or TikTok than the most talented, expensive team of developers 10 years ago. Organizations with solid, attainable development teams can leapfrog their competitors by leveraging a Serverless approach and building applications differently, to deliver like the top-notch software teams in the world.

Perhaps most important, building Serverlessly enables organizations to concentrate on what makes them unique and to focus investments on differentiated projects. Organizations can stop spending enormous amounts of budget on projects that no one except the software development team can even understand. Customers do not care that an organization is running a world-class Kubernetes cluster with hundreds of thousands of lines of customization; customers want their experience to be world class. Just as cloud adoption over the past 15 years has enabled every organization to build web-scale applications, Serverless enables every organization to deliver software as efficiently and effectively as the best software teams in the world.

The winners will be the organizations that change their games by recognizing how to build Serverlessly first. These companies will be able to accelerate the delivery of new products, services, and features for their customers because they will not be focusing on the commodity plumbing that underpins world-class software. They will leverage managed services with the best software developers in the world for services that are non-differentiating for their business, to allow them to focus more on building highly specific interfaces and logic for their customers. Ultimately, they will outpace, outmaneuver, and create unmatchable cost structures over any competitors that are slow to adapt to this new way of working.

How Many Employees Does It Take to Make and Run a Scalable Social Network?

In the recent history of software development, one of the hardest services to build and launch has been a scalable social network. Social networks need to be fast, always be online, and scale to ever-increasing traffic as they become more successful. One of the first popular social networks, Friendster, famously lost its lead because of its inability to scale[1] and was subsequently eclipsed by Myspace and Facebook, both of which had hundreds of employees supporting their builds.

Fast-forward to 2014, and a curiously simple social network called Yo launched. Members could sign up quickly, choose people or organizations to follow, and send only one message: "Yo!" In the first few days after its public launch on both iOS and Android, Yo scaled successfully to more than a million users; more than 100 million "Yo!" messages were delivered before it shut down in 2016.[2]

Yo was built by one developer (initially in about 8 hours) and never needed systems administrators, operations teams, or even full-time developers for its initial ramp existence.[3] Somewhat ironically, all the Yo code was being run and scaled by Facebook, which had acquired one of the first Backend as a Service companies, Parse. Yo leveraged Facebook's engineering and operational expertise in its Parse division, paying only $100 a month to handle 40 requests per second with 1 million unique push-notification recipients.[4, 5]

You can see this same leverage of managed services by Instagram, which had 13 employees at the time Facebook acquired it for $1 billion.[6] Of the 13, only 3 were full-time software developers; Instagram had no dedicated systems administrators or operations staff.

A relatively short time after Friendster was crushed by its inability to handle web traffic with hundreds of employees, two startups with only a handful of software developers conquered the same challenges with ease (see Figure 1.1).[7, 8] They did so by outsourcing much of the required expertise to newly created managed-service providers, who handled the "running and scaling infrastructure" expertise.[9]

An easy answer to the question posed by the title of this section, then, is that the number of employees needed to make and run a scalable social network approaches zero over time (absent moderation). Many components that both Instagram and Yo had to build are now much easier to handle, around a decade later. Many organizations spend so much time and effort building infrastructure, systems, and code that they could rent at a fraction of the price, with no dedicated personnel and with substantially better performance and uptime. It is amazing that organizations continue to compete on the playing field against Amazon, Google, or Microsoft when they could simply incorporate Amazon or Google's great players into their own teams by going Serverless.

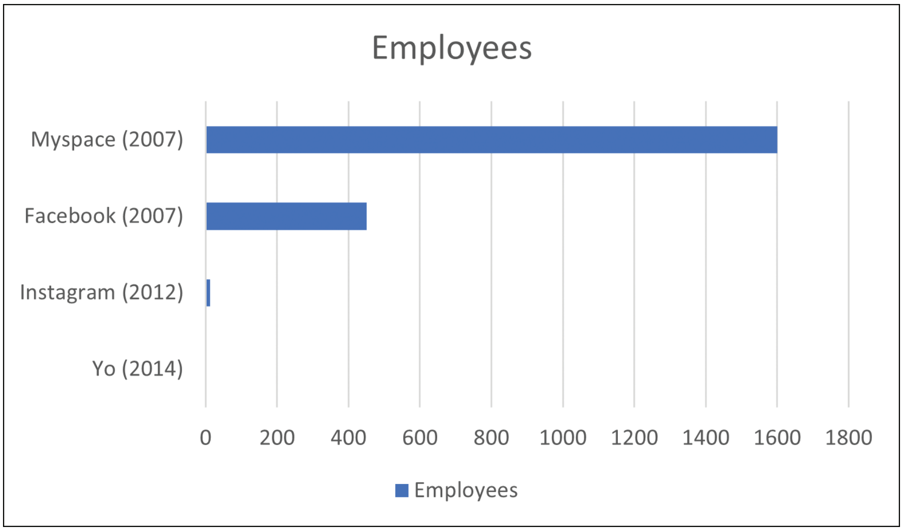


Figure 1.1 *Number of Employees Needed to Make and Run a Scalable Social Network over Time*

Leveraging Technology as It Improves

Software Development Has Been Improving Constantly…

Technology moves quickly. Moore's Law, that the number of transistors that can fit on a chip doubles every two years, has an equivalent in the software development world: The best way to build a piece of software from scratch gets a little bit easier every year. Often these small incremental changes are missed.

Everyone sees and recognizes that big tectonic shifts such as the public cloud have changed the game. Organizations that were slow to adapt have fallen behind, whereas others that recognized this shift and adapted have been able to win in the marketplace. But these big shifts are only part of the story. Hundreds of new services launched every year improve software development and provide meaningful benefits to organizations willing to incorporate them into their software.

Tasks that were extremely challenging 10 years ago, such as reliably resizing images, performing a fuzzy text search with tens of gigabytes of data, or handling hundreds of millions of monthly visitors from around the globe in milliseconds, are now often as simple to achieve as plugging a lamp into a wall. This is because technological improvements are not built in isolation; they are built on top of other technological innovations.

Thus, if one innovation per year can double output, then after 10 years, output could be increased by $2^{10} = 1{,}024$ times, not $2 \times 10 = 20$ times. This rate of innovation far exceeds what any one organization can build, so the benefit of learning how to harness it for an organization far exceeds the value of being the most innovative organization, building the maximum that the organization can build.

…But Isn't Being Adopted Effectively

If innovation in technology—specifically, in how to build software more efficiently and effectively—is happening and making development so much easier, why are these changes not being widely adopted?

Sadly, there is no continuing education in information technology. Most managers, architects, and senior staff in IT departments also have no set expectations that they should be regularly changing how they do their job, as new technology comes out. It is torture for someone to learn how to do something at the start of their career and then, after several successful years, throw out that knowledge and start again. The benefits of change must be intuitively obvious because so much personal and organizational inertia surrounds the industry. Even then, real change often requires significant changes in culture, mindset, architecture, coding practices, and technology leadership.

Additionally, even if teams identify significant changes they want to make, they still are called upon to deliver new features and functionality within their existing technology stacks. An organization that identifies changes it wants to make in how it builds and runs software must have buy-in and coordination at all levels in order to make significant changes.

The most successful people do two jobs: the job they were hired to do and the job of figuring out how to do that primary job better. This is a required mentality for everyone working on, in, and around the best software development teams. It is especially difficult in software development because of the rapid rate of change. Constantly thinking about how to do a job better, even if it requires learning brand new skills and doing jobs differently, is the only way to take full advantage of technology—and is the right way to approach reading this book.

This Book Is For…

Executives in Business and Technology

I have written this book for both business-minded technologists and technology-minded businesspeople. Increasingly, if you are a businessperson, you must be technology minded because, as is often quoted, "software is eating the world."[10] More companies are increasingly becoming software companies. Whether you are a business-minded technologist or a technology-minded businessperson, this book teaches you how to build and deploy software better, faster, cheaper, and more effectively than most companies in the world.

If you are a senior business leader, this book helps you bring technological change into your organization by showing you how a Serverless approach drives better business outcomes. If you are a technical team lead, this book helps you think like an executive leader and focus more on delivering value than optimizing for tech, to set you apart from your peers. You will increase your focus on what matters to your organization, release new features and functionality more quickly, and spend less money, time, and effort doing it. You will learn how to do all this while also learning how to mitigate the risks that are inherent when you rely on third parties to be successful.

Enterprises

I have run technology at companies of all sizes, and Serverless is just as beneficial for enterprises as brand-new startups. Enterprises will likely get the most benefit of adopting a Serverless mindset over their similarly sized competitors than startups because most enterprises suffer from more technical debt and an internal discontent with software development velocity. The journey within an enterprise to build fully Serverless applications is a long one that involves as much cultural as technical transformation. Part III of this book, "Getting to Serverless," focuses on the key steps to drive Serverless adoption within an enterprise.

Startups and Smaller Businesses

Serverless is a game changer for startups and smaller businesses because it enables them to run lean budgets and use small numbers of developers to deliver software and services that look like they come from teams orders of magnitude larger. Branch, which I founded (and which is discussed in more detail in Part II, "Real-World Serverless"), has operated with approximately 1/30th (that is, around 97% less) the developer payroll of an insurance startup that launched fewer products in fewer states than Branch. Part III is mainly targeted at larger and more established organizations, but it is also useful for startups because the cultural and technical transitions are much the same.

This Book Is Not About…

Service-Oriented Architectures

Serverless applications generally have service-oriented architectures (SOAs), with some custom code calling managed services and other custom code acting as a service (as opposed to using a lot of repeated code in different, similar features).[11] Using unnecessarily repeated code or repeated bespoke functionality within Serverless applications is difficult because leveraging managed services is native and simple within Serverless architectures.

However, this book does not spend significant time talking about SOAs in the context of building Serverless applications. Much of the discussion surrounding SOAs is predicated on older, non-Serverless methods of building software. Once an organization adopts a Serverless mindset, developers are highly unlikely to stray from SOA patterns because of how natural the patterns are in Serverless.

Monoliths and Microservices

The monolith vs. microservice debate has raged for more than a decade. That age is increasingly apparent when viewed in light of Serverless development. Serverless architectures tend to muddy the definitional waters between the two strategies; many Serverless applications, including those at Branch, have characteristics of both.

For example, one factor that drove a lot of initial microservice development was concern about the uptime and scalability of databases. Netflix famously implemented microservices to increase the resilience of other systems, giving stateful microservices their own databases to prevent other database failures from impacting their uptime.[12] However, many Serverless applications use managed, resilient, global databases such as AWS DynamoDB that have incredible historical uptime and low latency. When using DynamoDB, the traditional microservice motivations for separation of concerns around the datastore no longer exist.

Another operational motivation for separation of concerns within microservices has been preventing cascading failures due to traffic overload on a particular service. However, if service code is running on a service such as AWS Lambda, Google Cloud Functions, or Azure Functions, those cloud providers have proven that they can prevent any kind of cascading failure because of the capacity and architecture they have built for running code.

Nevertheless, Serverless applications often feel more like microservice applications because of how they use services and how they tend to separate business concerns. In my experience, most Serverless applications are too broken up into separate functions (more on this in Chapter 3, "Serverless Architectures"). Branch's architecture at the time of publication consists of around 100 Lambda functions and 5 React applications, but all in a monorepo designed to be deployed monolithically. Definitionally, that's not a monolith or a microservices architecture, and I don't think the distinction is helpful when building Serverlessly.

No-Code/Low-Code Platforms

This book is built on the assumption that the organization has software developers who will use software development languages, frameworks, and environments. My experiences with no-code and low-code platforms have led me to believe that the only ones that really work are those that use standard languages and function as a higher-level framework that can be "ejected" (for example, Expo for React Native, or AWS Amplify for web applications)[13] so that the application can continue development in a more traditional development and/or hosting environment.

I believe that no- and low-code platforms have their place. We used one to build our initial interactive demo for Branch, which was a much faster and easier route than building it in code (and also less likely to lead us to technical debt). But ultimately, there is no comparison between what is possible with no- and low-code platforms versus fully fledged languages and frameworks. Building a customer-facing, business-critical application in a no- or low-code platform is risky because of how inherently limited the platforms are. Therefore, this book does not address any tools or services that I consider no- or low-code (that is, proprietary language and/or IDE combined with proprietary hosting and no ability to eject).

Structure of the Book

The first part of this book explains the Serverless mindset. The Serverless mindset is a way of developing, deploying, and running software in which an organization retains the responsibility for building differentiated experiences for its customers and uses managed services for everything else. The key to the Serverless mindset is separating assets versus liabilities and differentiated versus undifferentiated capabilities. The book walks through several detailed technical examples of Serverless architectures. Many technologists take a narrow view of Serverless; these examples help explain the broader picture and value that Serverless delivers. The last chapter in Part I, "The Serverless Mindset," addresses common objections to Serverless architectures and features several real-world case studies of Serverless in practice.

Part II compares Branch, the insurance company that I started in 2018, with the fictional Insureco, a representative enterprise insurance company. The goal of this part of the book is to give specific details on exactly how Branch handles all the intricacies of being a regulated, for-profit financial services company while being Serverless. This part also illustrates the massive benefits delivered over what might be considered a "best practices" enterprise today.

Part III tackles this question: "Okay, I buy all of this, but how can we get to Serverless from where we are now?" It considers the best metrics to use to align everyone across the business on how to measure success. It explains why the only viable method for making changes to existing systems is iterative replacement. Finally, it talks about continuing education and job retraining, which are critical parts of outsourcing functions that you no longer should have in-house.

Part III concludes with Chapter 11, "Your Serverless Journey," which brings together everything that was previously discussed. It lays out a high-level set of observations on how generally to think about building serverless applications, as well as a tactical outline for how you can approach building serverless applications.

Finally, Part IV, "Appendixes," and the companion website to this book feature a comprehensive listing of managed services, organized by category. Most people building software have no idea how much software developer time can be saved by leveraging managed services. Managed services can handle many functions of applications, from the core of an application, to authentication, search, image manipulation, customer communication, and even software development functions such as code review and testing. The directory provided herein gives examples and details that will help technologists and business leaders alike understand which services they can leverage today.

References

[1] Hoff, Todd. “Friendster Lost Lead Because of a Failure to Scale.” http://highscalability.com/blog/2007/11/13/friendster-lost-lead-because-of-a-failure-to-scale.html

[2] “MobileBeat 2014: How Did a Stupidly Simple App Get Such Stupidly Huge Growth?” (2014). www.youtube.com/watch?v=ZA-Hnd1j_II

[3] Shontell, Alyson. “The Inside Story of Yo: How a ‘Stupid’ App Attracted Millions of Dollars and Rocketed to the Top of the App Store.” (June 21, 2014) www.businessinsider.com/the-inside-story-of-yo-there-isnt-actually-1-million-in-the-bank-2014-6

[4] Parse Plans and Pricing. (2014) http://web.archive.org/web/20140714210913/https://parse.com/plans

[5] Amazon Web Services Case Study: Yo. https://aws.amazon.com/solutions/case-studies/yo/

[6] “Instagram Is Celebrating Its 10th Birthday. A Decade After Launch, Here Is Where Its Original 13 Employees Have Ended Up.” www.businessinsider.com/instagram-first-13-employees-full-list-2020-4

[7] Facebook Number of Employees. https://statinvestor.com/data/22188/facebook-number-of-employees/

[8] Yo (app). https://en.wikipedia.org/wiki/Yo_(app)

[9] Cutler, Kim-Mai, and Josh Constine. “Facebook Buys Parse to Offer Mobile Development Tools as Its First Paid B2B Service.” (2003) https://techcrunch.com/2013/04/25/facebook-parse/

[10] Andreesen, Marc. “Software Is Eating the World.” (2011) www.wsj.com/articles/SB10001424053111903480904576512250915629460

[11] Amazon: Service-Oriented Architectures. https://aws.amazon.com/what-is/service-oriented-architecture/

[12] “A Design Analysis of Cloud-based Microservices at Netflix.” https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f

[13] “Being Free from ‘expo’ in React Native Apps.” https://medium.com/reactbrasil/being-free-from-expo-in-react-native-apps-310034a3729

This page intentionally left blank

Index