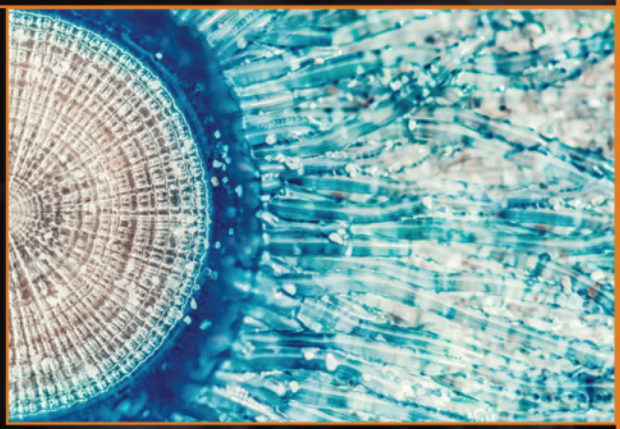
The Addison Wesley Signature Series

A Vaughn Vernon Signature Book

Serverless as a Game Changer

How to Get the Most Out of the Cloud

Joseph Emison



"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

Summary: Why or Why Not Serverless?

Adopting Serverless infrastructures often sparks debate within most organizations due to the shift of numerous in-house tasks to external vendors. Individuals whose professional identities have been rooted in skills that are no longer essential to the organization tend to actively find justifiable reasons to resist the transition to Serverless. This chapter discussed several typical concerns associated with Serverless, such as loss of control, vendor lock-in, performance issues, and security risks, and dispels them as generally unwarranted grounds for resistance. Moreover, this chapter underscores various success stories that contradict the apprehension, ambiguity, and skepticism that those opposing Serverless tend to instigate.

References

[1] "Amazon cloud chief jabs Oracle: 'Customers are sick of it'." www.cnbc.com/2017/04/19/amazon-aws-chief-andy-jassy-on-oracle-customers-are-sick-of-it.html

[2] "Microsoft Audits." https://microsoftaudits.com/

[3] "Don't get locked up into avoiding lock-in." https://martinfowler.com/articles/oss-lockin.html

[4] Debate over Cloudinary vs. running image manipulation locally. https://news.ycombinator.com/item?id=14612332

[5] "Can I bulk download all of my Cloudinary resources?" https://support.cloudinary.com/hc/en-us/articles/203068641-Can-I-bulk-download-all-of-my-Cloudinary-resources-

[6] "Migrate sensitive payments data." https://stripe.com/docs/security/data-migrations

[7] "New - Provisioned Concurrency for Lambda Functions." https://aws.amazon.com/blogs/aws/new-provisioned-concurrency-for-lambda-functions/

[8] "What is Algolia's product compliance?" www.algolia.com/doc/faq/security-privacy/product-compliance/

[9] "Security, Privacy & Compliance." https://auth0.com/security

[10] "A guide to PCI compliance." https://stripe.com/guides/pci-compliance

[11] "Serverless IoT @iRobot." www.infoq.com/presentations/serverless-iot-irobot/

[12] https://twitter.com/ben11kehoe/status/1475556704473399297

[13] "Accelerating with Serverless!" https://medium.com/lego-engineering/accele-rating-with-serverless-625da076964b

[14] "Why Lego Went Cloud and Serverless to Handle Traffic Spikes." www.informationweek.com/cloud/why-lego-went-cloud-and-serverless-to-handle-traffic-spikes/d/d-id/1339742

[15] "How The COVID Tracking Project Scaled From 0 to 2M API Requests in 3 Months." www.netlify.com/blog/2020/07/06/how-the-covid-tracking-project-scaled-from-0-to-2m-api-requests-in-3-months/

[16] "AWS Serverless Customer Success." https://aws.amazon.com/serverless/customers

Chapter 5

Introducing Branch

Earlier chapters laid out the details and theory behind the benefits of Serverless architectures over traditional ones, but theory alone can be useless. Branch, a rapidly growing insurance company based in the United States, has been following the Serverless architecture practices described in this book. It offers a pertinent example for understanding the practical benefits of going Serverless.

Serverless from the Start

I am the chief technology officer and cofounder of Branch,[1] the only personal lines insurance company in the United States that enables the instant purchase of home and car insurance. The Serverless principles within this book were at the heart of Branch from the beginning. Before anyone at Branch wrote a single line of code, we wrote down our primary principle and corollary.

Branch Primary Development Principle

When we write code, we optimize for maintainability.

We build systems that can be maintained by the average developer, as opposed to needing ninjas, geniuses, or rockstars, who often choose to optimize for their own interests over systems that stand the test of time.

Corollary

The easiest systems to maintain are those that we do not build, run, or maintain, but those that we pay others to build, run, and maintain. The most maintainable code is no code.

We choose to optimize for maintainability because a core part of our strategy requires being able to develop software at a high rate of output for many years. Companies that optimize for getting a minimum viable product (MVP) out the door can quickly end up with significant technical debt and difficulties in shipping new features even a few years after the MVP. We believe that any organization that plans on building software over time and at scale will achieve more by optimizing for maintainability over other goals.

The Problem to Solve

Branch was built to solve a problem that its founders saw in personal lines insurance in the United States. Insurers spend around 25% of the money they bring in on acquiring new customers, even with economies of scale and even as the cost of paying claims has continued to rise. In this era of software, automation, the Internet, and smartphones, the potential cost of home and car insurance should have been decreasing, but it wasn't. The reason insurers continue to spend more money on acquiring customers is that insurance is difficult to buy; people find it painful to shop around. Accordingly, the first or second place someone turns for insurance often wins, even if another, lesser-known option is cheaper and better.

Branch was built to sell insurance at "insurance moments," when someone is buying or refinancing a home or a car, by leveraging its technology and insurance product expertise. By building sophisticated, modern insurance products that can be purchased in seconds with minimal data entry and then integrating them with partners who are handling the underlying home or car transaction, Branch cuts out many of the costs that are traditionally allocated to acquiring customers.

Key Differentiators/What to Build

Insurance is a complicated business that requires a significant number of processes, people, and technology to run. Every insurance company built in the United States *except for Branch* launched with a single line of insurance at its beginning because of how different the requirements are for each. To sell its first policies, Branch needed to

be able to build insurance products (up to tens of thousands of pages for each product in each state, most of which are printed-out Excel tables detailing insurance pricing algorithms). It also needed to underwrite those products, handle regulatory compliance, file mandated reports, pass technical and accounting audits, and staff the phones with licensed sales and service staff—and that doesn't include any technology budget.

Using Branch's Serverless philosophy, the main question to answer was, "What is it that Branch has to build and cannot buy?" Branch started by trying not to write any custom code at all. Branch built its initial prototype with a no-code platform, Bubble.[2] Branch pushed Bubble to its limits, finding that it was possible to use a managed service, ClarionDoor,[3] for calculating rates and generating required forms. Branch also experimented with Software-as-a-Service (SaaS) offerings for insurance companies that can run all operations for more traditional insurers. Ultimately, however, Branch's plan to control the user experience at a very fine-grained level, combined with the desire to make it possible to buy home and car insurance in seconds in a single transaction, forced Branch to develop its purchase experience in-house.

In addition, Branch could not find a way to provide the user experience it wanted beyond the initial purchase for its members on the web and in mobile apps without building those itself, so they were added to the build list. But everything else could be bought, so Branch set out to do just that.

What to Buy

Branch was able to buy many pieces that other startups, founded around the same time, decided to build: an underwriting and support ticketing system (ZenDesk), omnichannel communication platforms (ZenDesk for members, Five9 for prospects), a rating engine (ClarionDoor), a forms generator (ClarionDoor), a claims management system (Snapsheet), an accounting system (QuickBooks Online), a customer data platform (Segment), a custom email campaign tool (Customer.io), and a search engine (Algolia), among others. Executives at Branch didn't necessarily see these services as perfect—or even meeting all the features that they wanted to have. But they did recognize the value of getting up and running quickly and being able to use them to understand better how and when (if ever) Branch would need to move to different systems.

Using so much SaaS at Branch has involved so many wonderful aspects. Every department has been fully in charge of the configuration and use of the software that matters the most to it. An entire book could be written about empowering departments to own their key vendor relationships, even if those vendors are SaaS vendors; that is not the focus of this book, but it is very much in alignment with the principles laid out in the Amazon memo in Chapter 3, "Serverless Architectures," and with the Serverless mindset.

To the extent that Branch has a central information technology organization, it is focused on information security and enabling Branchers to successfully use technology. Central information technology at Branch does not slow down adoption or prevent the full use of products, provided that they meet information security guidelines. Departments interact directly with the software providers, so they get direct expert advice and configuration. Additionally, many leaders in companies over the past decades have realized that they were wrong in at least some of their initial beliefs about how they would want to use the software and what their key needs would be. Instead of spending a lot of development dollars building the wrong thing, Branch has been able to use the work of other developers (and their customer success teams) to identify the best possible ways to succeed. Finally, buying over building has made it much easier for Branch to make changes when it has needed to—for example, when it switched from QuickBooks Online to NetSuite because of the increasing complexity of its accounting needs.

Minimize the Innovation Tokens

Dan McKinley, previously a principal engineer at Etsy, wrote an influential and powerful article about how to build software systems entitled "Choose Boring Technology."[4] McKinley's thesis is that it's better to work with languages, tools, frameworks, and patterns that have been tried and tested throughout the years. If an organization wants to leverage some new technologies, it should do so sparingly. McKinley calls the use of a new technology "spending innovation tokens" and says that an organization can't use many tokens before the innovation will go broke. After all, using new technologies can incur a compounding impact as an organization struggles to understand how the technologies interact with each other.

It might be surprising to find a book about embracing a different way of building software—Serverlessly—citing a philosophy around using proven technologies, but Branch has been sparing in the innovation tokens it has spent. What McKinley doesn't address in his essay is that, with so many different proven technologies today, it is very difficult to find people who have experience with all of them. Thus, there is a need to discover proven, "boring" technologies. One might read McKinley's article as a defense of "use only what *you* know," but that's not correct. You almost certainly don't know *most* of the proven technologies that you could leverage, but once you identify them, you should be able to rely upon mostly proven technologies.

In July 2019, when Branch sold its first insurance policies, almost all the managed services and SaaS it relied upon had been around for more than 5 years, and each had significant market share and marquee customers in its category. Perhaps the only innovation token Branch spent was on AWS AppSync, which Branch selected before it became generally available in April 2018. Branch chose AppSync because