Extracted from:

Serverless Single Page Apps

Fast, Scalable, and Available

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Serverless Single Page Apps

Fast, Scalable, and Available

Ben Rady

edited by Jacquelyn Carter

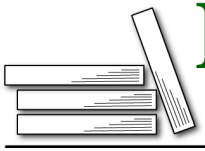
Serverless Single Page Apps

Fast, Scalable, and Available

Ben Rady

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz, Liz Welch (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-149-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2016

CHAPTER 1

Starting Simple

If you've ever thought, "There should be an app for that" and wondered who could build it for you, I have good news. We found someone. It's you.

Web applications can be powerful, efficient, and scalable, but they don't need to be complicated. Their simplicity is one of their great strengths, and you can use that strength to realize your own ideas and build your own solutions. Once you understand how all the pieces fit together, you'll be able to create the apps you want to see in the world.

This book is a practical tutorial that will demonstrate a *serverless* approach to building web applications. Using this approach, you can ignore most of the operational concerns and costs that come from running your own servers. You'll be able to focus time and attention on building the apps you want and let someone else worry about the headaches of provisioning, configuring, upgrading, and scaling servers to meet your needs as they grow. You won't achieve these gains in efficiency by adding multiple layers of web frameworks, generated code, or copy-and-paste templates. As we move through this tutorial together, you'll see how to deliver better applications by *removing* code and eliminating the middle-tier layers found in a traditional web application.

To start this tutorial quickly, we'll use a prepared workspace loaded with everything you'll need to build a complete web app. First, we'll build a *single page web app*, moving logic normally found in the server down into a web client built with JavaScript, HTML, and CSS. We'll rely primarily on web standards and dig deeply into the essential functions of single page web apps, building them up from scratch to both learn how they work and ensure the design fits the needs of our app. When our needs can't be met by web standards alone, we'll use jQuery to fill in the gaps. We'll make our single page app testable by building it incrementally, using a *Test First* approach.

To eliminate middle-tier costs and ensure our app scales into the millions of users, we'll use Amazon Web Services (AWS) to build a serverless back end. You'll see how to replace the servers, databases, and load balancers found in a traditional web application architecture with highly available and scalable cloud-based web services that are cheaper and easier to maintain. We'll look at some of the security issues you'll be faced with when building these kinds of applications, and we'll survey other technologies and tools you may want to use as your apps grow.

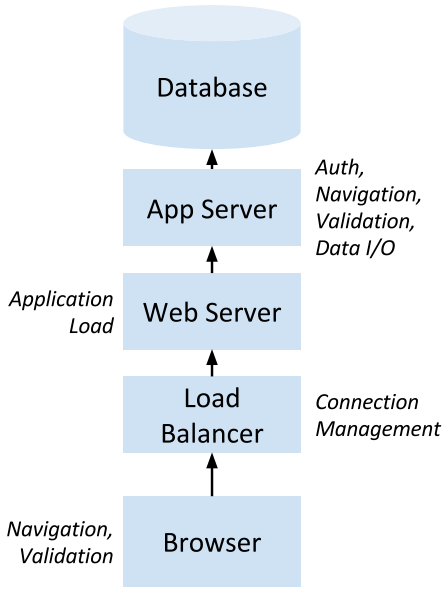
With this book, I hope to open new possibilities. Applications that previously were too expensive and time consuming to be worthwhile may become something you can finish in a day or two. As technology improves and expands your capabilities, more of your dreams will come into reach. As your understanding of these technologies grows, you'll begin to see new paths that lead you to goals you previously thought were too difficult to achieve. By the end of this journey, you'll have the skills you need to turn your ideas into reality.

Serverless Web Applications

Traditional web applications assume that the server is an essential part of the system. While sometimes fronted by a load balancer and/or dedicated web server, the *application server* does most of the heavy lifting. It performs all the essential functions of the app, including storing the user's data, issuing security credentials, and controlling navigation. The web portion of the web app is often just there to provide an interface to the back end, although some of the responsibility for controlling navigation rests there too. Many people call this traditional approach an *n-tier architecture*, where the browser, the application server, and one or more back-end services make up the tiers in the system.

With a serverless approach, you can remove all these tiers in favor of something more direct. Instead of treating the web client as the interface to the application server, you can move the application logic into the browser by building a single page web application. This means you can serve your application from a simple static web host—nothing more than a delivery mechanism for the app—while the browser acts as an application container. As you can see here, the result is a design that removes the middlemen from traditional web application architectures and allows the browser to directly connect to any services that it needs.

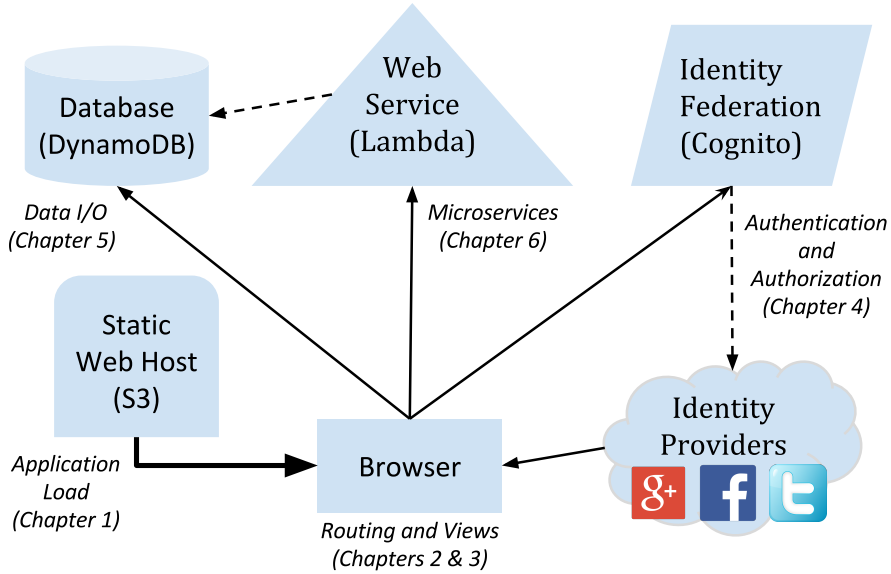
By using OAuth 2.0 identity providers such as Facebook, Google, and Twitter, you can create user identities without storing passwords. To store data, you can connect to services like Amazon DynamoDB right from the browser. Any



function that can't be performed in the browser can be handled by an Amazon Lambda microservice or other specialized web service. In addition to simplifying the architecture, making this transition to a web service back end lets you take advantage of the availability and scalability inherent in these services.

You might wonder what's changed to make this approach possible. Why is it only now that middle-tier application servers have become optional in a web application? The answer is, starting in 2015, cloud service providers such as Amazon began to provide APIs to their services to make this approach not only feasible, but a well-supported use case of their tools and infrastructure.

By building a single page web app based on web standards, rather than server-side web frameworks, we can quickly adopt these emerging technologies. For example, we don't have to tie our application's data model to any object hierarchy or data synchronization mechanism. That makes it easy to integrate with these kinds of services. Since we're starting from the foundations of the web, we don't have to fight against preconceptions of how a web application should be built, and we can create an application that is perfectly suited to the new technologies now available to us.



Benefits of a Serverless Design

If you're looking for a way to quickly build a low-cost web application, a serverless web app might be the solution. Rather than spending your time and energy on understanding all the different layers of a typical web application stack, this approach lets you focus on delivering features to your users while letting someone else worry about the headaches of runtime operations and scalability. Let's take a look at some of these benefits in depth, so that you can make an informed decision about whether this approach is right for your next project.

No More Servers

One of the most obvious benefits of a serverless design is that you don't have servers (physical or virtual) to maintain. You don't have to worry about applying security patches, monitoring CPU or memory use, rolling log files, running out of disk space, or any of the other operational issues that come up when maintaining your own servers. As with most Platform as a Service (PaaS) approaches, a serverless web application keeps you focused on your app, not worried about your infrastructure.

Easy to Scale

Another enormous benefit of this design is that it lets you rely on cloud service providers to scale your application. Rather than trying to keep data consistent between load-balanced application servers as you scale them horizontally, you can connect directly to web services that have already solved this problem. This means whether your application has just a handful of users, a few hundred, or a few hundred thousand, you can make sure it works flawlessly simply by changing a few settings in the Amazon Web Services console.

Highly Available

High availability also becomes much easier to achieve with this design. You no longer have to bring down your application server in order to upgrade it, or build out the infrastructure needed to perform a "hot" deploy. There are no service restarts to coordinate or deployment packages to copy from server to server. Best of all, you have the well-trained staff at Amazon watching your infrastructure 24/7, ready and able to respond if a problem arises.

Low Cost

The costs of these services can also be very low. Using a serverless approach and Amazon's Free Tier, you can often run your application for pennies a month. Once you exceed the free tier, costs often scale linearly (at worst) with the number of users you add. The application we'll build in this tutorial scales up to one million users for less than the cost of a cup of coffee per day.

(Micro)Service Friendly

This approach easily accommodates microservices and other service-oriented architectures. You can introduce specialty services into the system to perform custom authentication, validation, or asynchronous data processing. You can even reintroduce application servers, if you find a reason to do so, and gradually refactor your application to start using it. In contrast, if you start with a middle tier that holds all of the security credentials, it can be difficult to transition to web services that require authentication. The application server may not be able to manage identity at an application level like a serverless app can.

Less Code

With a traditional web application, operations that need to be performed in both the web client and the server, like navigation, result in duplicated code. Sometimes, this duplication is not immediately apparent, especially if the server is written in a different language. Since the application logic moves

into the client with a serverless app, it becomes much easier to ensure that there's no duplication anywhere in your app. Unifying the application logic in one place (and one language) helps resolve this problem.

This approach can also be much easier to build and troubleshoot, because there are simply fewer actors in the system. Web applications are inherently distributed; that is, they pass messages (usually in the form of requests and responses) between nodes in a network and are limited in how they can do this, as described in the CAP theorem.[1]

Some apps are more distributed than others, and the more distributed your system, the harder it can be to troubleshoot. Removing tiers from your application can make it less distributed. In the simple case, if a client needs to fetch data from a database, they connect directly to the database rather than going through a middle tier. This means fewer network nodes in the system, which means fewer places to look if things go wrong.

As you can see, you might want to build a serverless app for a lot of reasons. As we move through this tutorial, you'll be able to see firsthand why this approach can be so powerful. With these benefits in mind, let's now take a look at some of the limitations of a serverless app.

Serverless Design Limitations

While a serverless architecture has many benefits, it isn't suited to every type of application. You have to accept a number of limitations in order to get the benefits of this design, and if your application can't work with those limitations, then this isn't the right approach. Let's take a look at some of those limits before we get started building our app.

Vendor Lock-In

The first and most significant limitation of this design is that you must use web services that support an identity provider. This can limit your options when choosing cloud service providers. So when using this approach, you can become dependent on third-party services, which means vendor lock-in can be a problem. Building a system on top of a service provided by another company means that the fate of the application is now tied to the fate of the company. If the company gets acquired, goes bankrupt, or changes its business model, you might be left with an app that can't run anywhere without significant changes. Evaluating the business goals and long-term stability of a company offering these services is just as important as the technical merits.

1. https://en.wikipedia.org/wiki/CAP_theorem

Odd Logs

Any remaining operational concerns, like application logging, can take an unfamiliar form when using a serverless approach. When you route all your requests through one server, it's easy to log them all to see what users are doing. Without this centralization, logging must be done from the various web services that support the application. These kinds of logs are in a different format than most application server logs, and they contain data that you may not be familiar with. We'll take a closer look at analyzing web service logs in *Analyzing S3 Logs*, on page ?.

Different Security Model

With a serverless app, some common security risks disappear, but you'll encounter new issues that may be unfamiliar. For example, validating user data for security purposes cannot be performed safely in the browser. You have to assume that malicious users may commandeer the credentials in the browser to try to use any web service that the credentials allow them to access. When using a serverless approach, this means you cannot mix your application validation logic in the browser with validation done for security reasons. You have to do it separately.

Many web services provided by Amazon have facilities to validate requests. You'll see how to do this with DynamoDB in *Data Access and Validation*, on page ?. However, it may be difficult for some applications to enforce sufficient validity constraints using only the tools provided in the web service. For example, when you are writing directly from the browser, you cannot safely encode data written to a database to ensure it isn't attempting to perform a cross-site scripting attack, because the attacker can just add the data directly to the database without using the app.

In that case, you have (at least) two options. First, you can just assume that certain user-writable tables may contain unvalidated data, and design the rest of the system accordingly. For example, if the users can only write data that they alone can read, this is a viable option. Second, you can delegate those particular write operations to a custom web service, such as a Lambda function, to perform the validation and write in a secure fashion. We'll cover creating a custom web service in Chapter 6, *Building (Micro)Services with Lambda*, on page ?.

Different Identity Model

External identity management is a distinguishing feature of the app we'll build in this tutorial. Managing identity by using a web service has a lot of

advantages, but the mechanisms may be unfamiliar to you. Rather than storing user profiles alongside the rest of your data, these profiles will reside in other data stores that you access separately. When building a serverless app this way, some familiar techniques for dealing with user data in databases (joining in a User table with an ID, for example) may no longer apply.

Loss of Control

Additionally, routing all requests through a middle tier provides a certain level of control that can be useful in some situations. For example, denial-of-service and other attacks can sometimes be stopped at the application server. Giving up direct control over issuing identities might also be a scary prospect for you. We'll devote an entire chapter to security concerns later, in Chapter 7, *Serverless Security*, on page ?.

Big Scale: Big Money?

Lastly, you need to understand the costs of these services. While being able to scale an app automatically is powerful, making scaling easy also means we've made spending easy. You need to understand how the prices of these services are structured and how they change as you add users. We'll examine costs of our app in depth in *Costs of the Cloud*, on page ?.

Now that you understand the trade-offs of a serverless web app, we can start the tutorial and dive into how they work. As we move through the tutorial together, you may uncover other benefits or limitations that are specific to the kinds of web applications you want to build. Once you have the complete picture, you'll be able to make an informed decision about whether this approach is applicable to your next project.