Q1 2022

FRACTALSOFTWARE.COM

The Fractal Guide to Choosing a Vertical SaaS Tech Stack

fractal

Introduction

The job of a CTO is multifaceted and involves a combination of both managerial and engineering skills. They must identify and hire talented programmers, structure their engineering department, create a timeline for their product roadmap, manage the quality of their code, and ensure that the engineering choices they make today won't negatively impact their ability to expand and refine their product in the future.

One of the earliest decisions a CTO makes is selecting a tech stack, which will be an important factor in who they hire and how they build. For many engineers transitioning to a CTO role, choosing a tech stack is simple: They use the technologies they're most familiar with. This can be a useful heuristic, but choosing a tech stack is not a decision that should be made lightly. CTOs should carefully consider the tradeoffs of selecting a given technology and have a clear understanding of how this choice may affect their company.

This guide is designed to help new CTOs weigh their options for key elements in their tech stack including frameworks, databases, server providers, and component libraries. It draws upon insights from vertical SaaS CTOs at companies ranging from pre-Series A startups to late-stage unicorns. While Fractal provides recommendations for each category in the stack, it's ultimately up to the CTO to decide which technologies best suit the needs of their company, engineering team, and customers.

Five Principles for a Robust Vertical Tech Stack

Before we dive into the primary features of a vertical SaaS tech stack, it is worth considering the five principles that should inform how you think about your tech stack. These are not inviolable rules. Instead, they are a framework for thinking about how technologies can affect your engineering organization. They can be used to simplify the process of selecting technologies for each layer of your tech stack.

1. Keep it simple

If you are transitioning into a CTO role after working as an engineer in a consumer-facing software company you are accustomed to building for scale. Consumer products aim for virality, which requires engineering decisions that are robust enough to handle hypergrowth.

For vertical B2B SaaS startups, simplicity means choosing infrastructure and technologies appropriate for the scale of the business, and only adding complexity as the need arises. Microservices, caches, asynchronous message queues, and distributed databases are all powerful tools to scale an application, but they often come at a cost of developer velocity and code complexity. While vertical SaaS businesses grow quickly, their request loads aren't nearly at the level of a consumer-facing application. As such, a clear need should be present before you use tools that can create drag on velocity.

Early in a company's life, a lot of the work is around discovery: the founders and early employees are learning about their target customer's business workflows and how to model that in code. This often requires rewriting a lot of the software as that understanding is refined, and having fewer moving parts makes everything easier. It allows for more engineering time to be spent on product and feature development, and less on maintaining and supporting infrastructure.

Another benefit of simplifying your tech stack is rapid engineer onboarding. This is critical during the earliest days of your company when you're racing to launch your MVP before your runway evaporates. The less time engineers need to get up to speed on your tech, the more time they can spend building your product. It also helps you avoid the need for specialist engineers. In the beginning of your company, engineers who can work across the stack will deliver the most value for your limited employee budget.

Keep in mind that a simple tech stack will allow for more complexity in the future. If you launch with complex tech, it can be difficult to change course down the line if or when you need different solutions. Simplify to stay nimble.

2. Everyone Contributes Everywhere

Engineering specialization is directly correlated to the growth of your startup. As you begin to scale, you will need to hire engineers who primarily or solely work on the front end, DevOps, and so on. But in the beginning, you should strive to empower generalist engineers to contribute across the tech stack. In practice, this means minimizing the number of languages and technologies you are using as much as possible. It will make your organization faster and more resilient. If there's only one engineer who can contribute to a certain layer of your stack, this creates a risk to your company if that employee leaves or needs to focus their contributions on other layers. You can avoid this liability by ensuring that all early engineers can contribute across your stack.

3. Optimize for Speed of Iteration

As CTO of a newly minted vertical SaaS startup, your goal should be getting your MVP to market as soon as possible. You need real-world user data to refine your product and land more customers. This means you should optimize your stack for iteration speed. The idea is to reduce friction to the point where you can deploy fresh code several times per day and quickly respond to feedback from early users. There are multiple ways to optimize a tech stack for iteration speed (e.g., embracing infrastructure-as-code and CI/CD) that will be covered in greater detail below.

4. Use Off-the-Shelf Tech

Launching a vertical SaaS startup doesn't require reinventing the wheel. You should leverage established frameworks for building and designing SaaS platforms whenever you can. This means you should use off-the-shelf solutions wherever possible, whether it's component libraries, cloud services, or hosting. Both open source and marketplace libraries are your friend. When you use pre-built tech, that frees your engineers to spend most of their time writing code that creates differentiating value.

5. Adopt Best Practices Early

Most startups will encounter unexpected challenges that require engineers to get scrappy and deviate from best practices. But in general, your engineering organization will benefit from adopting a culture where things are done the right way the first time—even if it comes at a slight cost to speed. The definition of best practices may vary from startup to startup and it's up to the CTO to decide what it means for their organization. Still, there are some universal best practices like local development, code reviews, fast and automated deployments, simple database migrations, and robust testing that are common to all successful startups. By adopting these best practices early, you will build a disciplined engineering team and save yourself a lot of wasted time in the future.

CASE STUDY

Joel Gottsegen

Qualia CTO and co-founder

Vertical	Real Estate
Founded	2015
Stage	Late
Prior Experience	Mosaic, DataFox

One Piece of Advice

“Optimize your stack and all of your early architectural decisions for speed. You want to get your product off the ground as quickly as possible and remain agile enough that when you inevitably find out that what you’re doing is wrong, you can trash your code and not feel bad about it. So choose the ones that create an environment where people can have fun and iterate quickly.”

Frontend

"Blaze is a Meteor tool that allowed us to quickly build a sophisticated and user-friendly frontend. Although it has fit Qualia's needs well, it probably doesn't make sense for most CTOs launching a company today unless they've also committed to the Meteor framework. I'd recommend checking out React instead, which has become the de facto frontend framework for SaaS companies over the past few years."

“Semantic UI’s components are really feature complete unlike a lot of frontend frameworks that are mostly focused on the CSS. With Semantic, it’s not just about the layouts. It also has a lot of abstracted functionality so there are a lot of different ways to modify components. When you’re making B2B software, the less your engineers have to think about design early on the better.”

Backend

“We wanted to use something new and modern that was fun to program on. Meteor was a great recruiting tool early on, but that has faded as the framework became older. At one point Meteor briefly stopped being officially supported so I’d recommend using more established tools that have big communities so you don’t end up holding the bag.”

Database

“MongoDB is paired with Meteor, which is part of why the framework is so cool and ultimately why we ended up using it.”

ƒ

The Vertical SaaS Tech Stack

A strong vertical SaaS tech stack is based on the five general principles outlined above, but the actual technologies deployed will be highly context dependent. The goal of every tech stack is to enable functionality that allows a company to serve their customers, which means the different workflow needs and user profiles of a given industry should ultimately drive technological decisions. For example, accounting for mobile capabilities is important in industries like construction where a lot of users work in the field, but may be less important in the legal industry where a lot of the workflow occurs in front of a desktop. Another common example is choosing HIPAA compliant tools for healthcare SaaS to enable more stringent data protection standards compared to industries that don't handle sensitive patient data.

The important thing when choosing a vertical SaaS tech stack is to deeply understand your industry and the workflows of the people who will be using your software. This means spending a lot of time with your advisors and potential customers to learn about how they work during the early days of your company. This will help you make intentional decisions about your tech stack that will drive engineering success while meeting the needs of users as your company grows.

The tooling recommendations made below are based on the experience of dozens of CTOs who have launched their vertical SaaS companies with Fractal's support. These technologies may not meet the unique needs of your startup, but they offer a useful reference for how to think about a particular layer of your stack and the tradeoffs involved in choosing a given technology.

Frontend Languages and Frameworks

The frontend language or framework for a vertical SaaS platform should be optimized for build speed, usability, and aesthetics. We strongly recommend CTOs use React with an existing component library such as Material UI or Ant Design. While we believe that good aesthetics and user interface principles are an essential part of a successful SaaS application, B2B applications don't have to be as visually distinct as consumer facing apps, especially in their early years. Although CTOs may opt for other frameworks such as Angular or Vue, the 2021 Stack Overflow annual survey of more than 80,000 developers found that React is by far the most popular frontend framework, with roughly 40% of respondents using React versus just 23% for Angular and 19% for Vue.

On top of this, you may want to consider using meta-frameworks such as NextJS to further simplify and accelerate your pace of development. These meta-frameworks are built on top of React and are tightly integrated with platforms like Vercel, Netlify, and AWS, which makes it easier for developers to test, deploy, and iterate new code.

1 · HTTPS://INSIGHTS.STACKOVERFLOW.COM/SURVEY/2021

CASE STUDY

Doron Roberts

CTO of Tradewing

Vertical	Industry Associations
Founded	2019
Stage	Growth
Prior Experience	Facebook, Adobe

One Piece of Advice	“Do the normalest thing possible.”

Frontend

“Everything is written in TypeScript. I was this close to ditching it in favor of vanilla JavaScript and that would’ve been a catastrophe. I love TypeScript, it’s been a huge benefit to us and plays nicely with GraphQL.”

Backend

“If I had a lot of experience with DevOps, I probably would have taken a few days to get AWS set up the way I wanted to in the beginning. But because of my limited knowledge in the area, it was really attractive to deploy our application in a few minutes. It’s pretty amazing how fast you can get it up and running.”

API

“GraphQL is powerful for a lot of reasons. The two that have been the most beneficial are that you can declare your GQL schema once but query it in a myriad of ways from the client without making any API or backend changes, and the ability to do static type-checking against the API. The biggest downsides versus REST have been that the backend permissioning layer is less straightforward, failure to use the Dataloader caching and batching abstraction leads to horrific backend performance, and it’s more difficult to mock API responses in frontend testing suites such as Cypress.”

Database

“Don’t let your lack of understanding of your domain leak into your database. The only time I could really see a document database being a real benefit is for a prototype application. You’re going to rebuild it from scratch when you launch anyway.”

Backend Languages and Frameworks

A Note on Mobile Development

For companies that will launch with a mobile MVP or plan to develop a mobile app shortly after launch, the balance tilts toward using JavaScript for everything because it makes it easier to use a mobile framework like React Native. Although you can't directly port a React web app to mobile, React Native uses the same principles as its web-based counterpart. This makes it easier for engineers to contribute across the entire codebase and also creates opportunities for building libraries that can work on both web and mobile. React Native is not without its downsides—it is notoriously inconsistent across iOS and Android—and you may want to consider alternative frameworks like Flutter, which is based on the Dart language created by Google. Bringing a new language into your stack may create friction and you need to carefully consider whether the benefits of the framework outweigh its costs in terms of development speed.

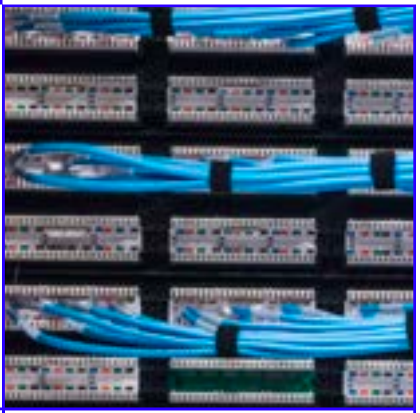
The backend handles all the business logic and will to a large degree determine whether your application can manage the different user types, business objects (e.g., accounts or inventory), and rules governing their interactions. Using React for your frontend can simplify the selection of backend technologies because it makes it easier to use JavaScript frameworks like Next.js or Node.js across both their client and server sides. But opting for a JS frontend doesn't preclude using other languages for the backend. The language and frameworks you choose for your backend should ultimately be based on the workflow needs of your users.

For example, a CTO trying to decide whether they want to use Python or JS as the basis for their backend may consider some of the following factors. If machine learning is a prominent part of a company's product roadmap, Python may be the preferable choice due to its status as the leading language for ML. This can future-proof a platform's backend by making it easier to implement features requiring heavy data analysis after the MVP launch. The tradeoff is it may make it more challenging for early engineers to contribute across the codebase because there are multiple languages involved.

Regardless of the backend language you choose, you will benefit

2 · HTTPS://MADNIGHT.GITHUB.IO/GITHUT/#/PULL_REQUESTS/2021/2

from the developer community's familiarity with both JavaScript and Python. Based on GitHub pull requests, JavaScript is the most-used language in the world and Python is a close second.[2] This means you will have a large talent pool to draw from when you hire your first engineers, and these engineers will be able to quickly begin contributing to your company's codebase. Since both JavaScript and Python are generally considered to be easy-to-use languages, even engineers who may not already be familiar with the language will be able to quickly learn its unique characteristics. Given the importance of development speed and iteration for early stage SaaS companies, you should give a lot of weight to the familiarity of the languages and technologies they add to your tech stack.



CASE STUDY

Daniel Blank

CTO at Stride

Vertical	Physical Therapy
Founded	2021
Stage	Growth
Prior Experience	Blackstone

One Piece of Advice

“CTOs need to strike a balance between choosing technologies that are modern enough to do what you need, but have been around long enough to stand the test of time. You don’t want to pick tools that are ancient, but you also don’t want something that has only had a few years of testing in the market.”

Frontend

“We briefly looked at Vue, but ultimately decided you can’t really go wrong with React. There are so many component libraries and tools, not to mention the market for engineers is so much bigger. It was really just a no brainer.”

“One of the major criticisms of Material UI is that every site uses it so your site ends up looking like every other site. But for us that was an advantage because all of our competitors’ sites look like they were made 20 years ago. We want our site to look modern and familiar.”

Backend

“Django has all the features you need to make an application HIPAA compliant, plus user authentication, and an entire admin site so you can interact with your database right out of the box. Plus, there are 20 years of StackOverflow history so there is a 0% chance you will encounter a problem that someone else hasn’t encountered before.”

“We're a healthcare company and that means you are going to be the stewards of some very sensitive patient data. Aptible is a healthcare-focused platform-as-a-service that basically handles all the database requirements for you so you can focus on writing code.”

API

“The Django REST framework allows us to build a really robust, secure, and fast API on top of Django. It will also make it easier for us to build a mobile application down the road.”

Database

“Our database decision ultimately came down to Postgres or MongoDB. You can be just as successful building with either model, but I decided that Postgres was the good solution because of my familiarity with it and its compatibility with Django.”

Databases

A Note on Mobile Development

An important consideration when building a vertical SaaS tech stack is ensuring that the technologies make it easy to test the application codebase. Buggy MVPs are a fact of life, but if you prioritize testing quickly and testing often you will remove friction around your product launch.

There are an incredible range of testing tools to choose from and the choice you make will ultimately depend on the technologies and languages in your stack. When it comes to testing, the benefits of using a single language like JavaScript across the entire tech stack are clear. This enables developers to use tools like Cypress that are specifically designed for end-to-end testing of applications built with JS. If multiple languages are used across the stack, this will likely mean using a suite of testing tools as well. For example, Jest can be used to test a frontend built with JS and a framework like Robot can be used for automation testing on a backend built on Python.

When it comes to choosing a database, the main decision you must make is whether to use a relational or a non-relational document database. Many CTOs with a background in developing consumer-facing software are familiar with relational database management systems such as PostgreSQL and MySQL, which remain by far the most widely used type of database.[3] But relational database structures may not be the optimal choice for vertical SaaS applications, especially during the early years of the company. The reason is that relational databases are designed for highly structured data. They work well when engineers already know the type of data their application will be fielding and need relationships between data fields to be perfectly consistent and non-redundant (e.g., in payment applications). Instead, you should consider launching your MVP with MongoDB, Firestore, or other NoSQL document database programs.

Document databases store their data in JSON files that sidestep the constraints of the tabular format used in relational databases. It adheres to the principle of tech stack simplicity insofar as it removes the need for querying databases in SQL since the database is queried with the programming language used in the application. Although it usually makes sense to use a document database in your stack, it's important that you understand the tradeoffs involved with this decision.

The schema-less structure of a document database is particularly

3 · HTTPS://DB-ENGINES.COM/EN/RANKING_CATEGORIES

useful for early-stage companies that are still in the process of learning their customer domain. Your understanding of the way businesses operate in your industry will become more sophisticated over time and if you build a relational database around your initial understanding of the domain, you may find that structure woefully inadequate in the future. It can be painful to overhaul a relational database once you have already launched your MVP, but the dynamism of a document database enables the data formats to evolve with your understanding of your customers' needs.

A further benefit of document databases is their inherent scalability. Whereas scaling relational databases involves using increasingly powerful computers with more CPUs and more memory, document databases can scale by spreading data across computing clusters. They were built for the cloud era and allow applications to scale dynamically based on user load. While this isn't as big of a concern for vertical SaaS apps compared to consumer-facing apps, it is an important consideration for early-stage companies that need their application to work for several years before they hire engineers dedicated to database management.

Of course, document databases have their drawbacks. One area of particular concern is the security of NoSQL databases. Both SQL and NoSQL databases are vulnerable to injection attacks where attackers hijack user input to run malicious queries on a database. Since NoSQL database queries are written in the application's programming language, this means attackers can execute commands in both the database and the broader application, which enables DDoS attacks and loss of server control. There are plenty of open source tools to help you ensure your document databases are secure before your application goes into production (e.g., NoSQLMap and Mongoaudit). But if you opt for a NoSQL database, you should exercise extra precaution to ensure that your application is secure. For example, if the application is written in JavaScript and JavaScript database queries are required for the application, it's critical that you build a process to validate all user inputs to reduce the risks from this common NoSQL attack vector. MongoDB offers many advanced security features such as encrypting data at rest and in transit, as do other document database systems. You should familiarize yourself with all the security best practices for your chosen system and ensure they are properly implemented before shipping your MVP.

With all the document database options available, it can be challenging to narrow your decision to a particular system. One suggestion is to start with Firestore, a cloud-hosted document database system maintained by Google that is optimized for rapid application prototyping. Firestore is a serverless architecture that makes it easy to scale an application and comes with native SDKs for iOS and Android, which simplifies the process of building out a mobile application. It is also essentially agnostic to the application's programming language. If your application uses a relational database, many SQL systems pair well with Prisma, an object-relational mapping tool that makes it easier to understand and manipulate your app's data models.

CASE STUDY

Evan Vandegriff

CTO at Greenspark

Vertical	Metal Recycling
Founded	2021
Stage	Growth
Prior Experience	Sevenrooms, Verst

One Piece of Advice

“Keep an eye on the features you want to build and look for out-of-the-box features that fit your needs. It will really minimize the number of migrations and changes to the infrastructure you’ll need to make down the road.”

ƒ

Frontend

“I started with the Create React app because you can spin up an entire working frontend app for you with like one line from the terminal. With that as a basis we could make personal touches and small changes around it.”

“Material UI gives you all these components like drop downs or date pickers right out of the box so you’re not reinventing the wheel for everything that a modern platform needs. It made it easy for new developers to get up to speed and jump in.”

Backend

“We went with plain JavaScript on the frontend and backend because we didn’t want to limit our options when we started hiring. A lot of people who are earlier in their careers have focused more on full stack JS solutions so we were able to find more candidates and get them up to speed faster.

API

“From the beginning I was leaning heavily toward a fully managed AWS solution. With AWS API gateway there ended up being some tradeoffs in getting people up to speed with the whole security framework and figuring out how everything fit together, but we’re pretty pumped about the ease of use from going this route.”

Database

“A lot of my early direction was what is the simplest way to get up and running quickly? AWS is really easy to spin up and start playing around with. You don’t have to spend significant engineering resources to maintain core functionalities when you spin up these serverless containers that scale automatically.”

Infrastructure: Servers, Billing, Email

Many vertical SaaS CTOs choose to launch their MVP with a serverless architecture so that machine resources can be allocated on demand. This is an optimal strategy because it reduces the amount of time that you and your engineers need to spend configuring a server. The less time engineers need to spend on low value activities like server configuration, the more time they can spend building features that differentiate the application and drive value to your company.

Today, cloud hosting is effectively a commodity and the provider that you choose isn't likely to have a major impact on the success of their business. Amazon Web Services, Google Cloud, and Microsoft's Azure are all equally good options, and you should choose the provider you are most familiar with that also fits your budget. In certain cases, it may make sense to look beyond the "Big 3" cloud providers to companies that provide servers that are more uniquely tailored to the needs of your application. For example, Vercel's serverless product is uniquely suited for apps using Next.js as their frontend framework because Vercel was the company that built Next. When it comes to choosing a server provider, the important thing is to make the decision quickly and move on to more important engineering challenges. Don't waste time splitting hairs about the unique features of each provider because for the first few years of your company's life, these differentiating qualities are unlikely to matter.

Other infrastructure decisions like email and billing APIs are similarly straightforward. In each case, you should select the simplest option. When it comes to billing, payments, and subscriptions that usually means using Stripe. There are more options for an email API and you should select the one that is easiest to use. Mailgun, Elastic Email, and SendGrid all support the most popular programming languages and will all work equally well for an early-stage vertical SaaS company.

CASE STUDY

Aaron Hammond

CTO at August

Vertical	School Health
Founded	2021
Stage	Early
Prior Experience	Stripe

One Piece of Advice

"Pick tools that you're familiar with that allow you to move quickly, but take your time with metaprogramming."

Frontend

"We're using React with TypeScript, which has been very helpful for us. We have effectively no tests on the front end and yet we have a very low defect rate because of the typing. Plus GraphQL plays really nicely with TypeScript."

"Ant Design is the most comprehensive open source component library I've ever seen, but it was a pretty big gamble because it is developed and maintained by Ant Financial, the Chinese tech behemoth. A lot of the source code comments are in Chinese, but fortunately someone developed a plugin that will automatically translate it."

Backend

"We're using Rails only on the backend, which is a non-traditional way of using it. One of the motivations for selecting Rails was the ecosystem around it. Even if we're not making use of all it's able to provide, it enables us to use all the other tools that work with rails. For example because of Rails we got our admin tooling, user authentication and mail templates basically for free."

API

"GraphQL handles our data binding between the frontend and backend. The flexibility of the query pattern has been helpful as we iterate on the API, and the Apollo client has removed the need for any kind of centralized state management on the frontend. The Ruby GraphQL gem also has robust support for extensions, so we've been able to metaprogram away routine platform tasks like wiring query params or exports."

Database

"We have a data domain that is exactly suited for a document database because we have a very low cardinality set of entities that are in a few different collections. Plus, the fact that Atlas, the cloud provider for MongoDB, maintains everything means we haven't had to think about databases at all, really. It's all turnkey."

Ready to Launch Your Own Vertical SaaS Company?

Fractal partners with exceptional engineers to give them everything they need to launch their own vertical SaaS company. As a CTO candidate in our Entrepreneur-in-Residence program, you'll get to choose a deeply researched company idea, partner with a vetted cofounder, and receive the capital you need to launch your business on Day 1. You'll join a community of dozens of other vertical SaaS CTOs and receive on-going support from Fractal's team of in-house experts who will share their insights from building some of the world's most successful vertical SaaS businesses.

Learn more about how Fractal can help you
launch your own engineering-led startup at:

fractalsoftware.com/engineering

fractal