frontegg

The Complete Guide to **SaaS Multi-Tenant** Architecture

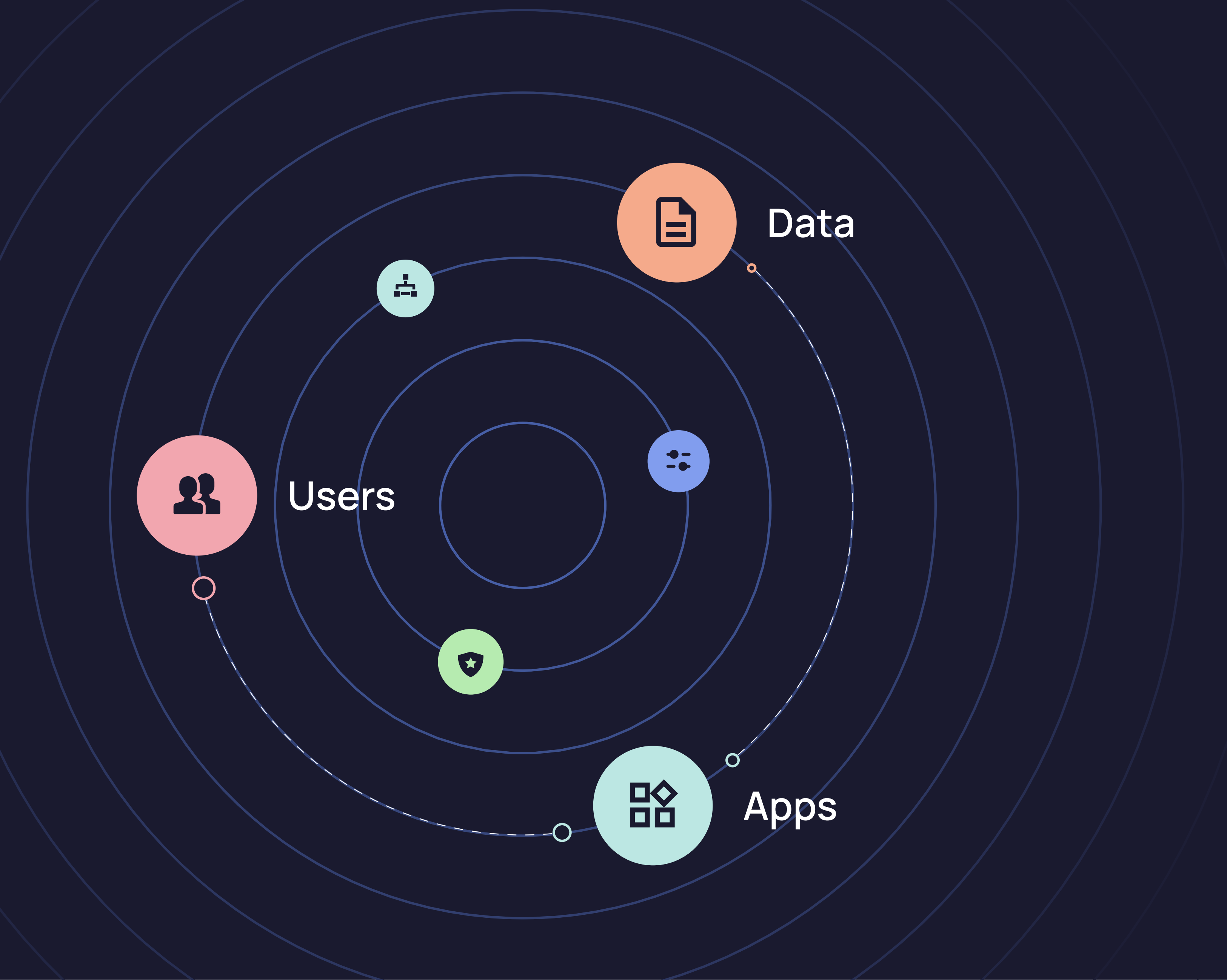


Table of Contents

Introduction

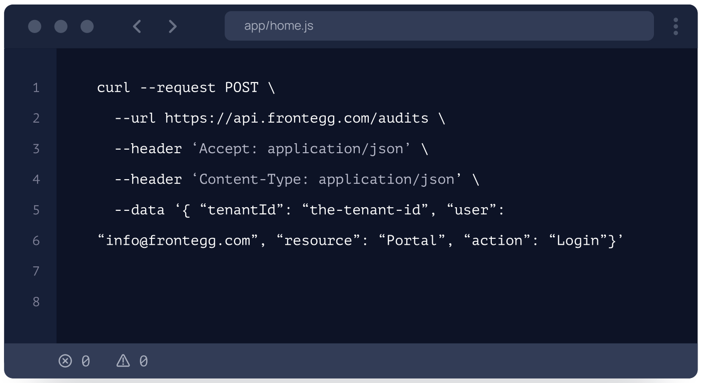
Software as a Service (SaaS) applications are essentially eliminating traditional on-premise applications thanks to their single-instance and multi-tenant architecture. These applications are hosted centrally and licensed on a subscription basis, making it a very efficient and manageable business model that can be scaled up fast. Salesforce, ZenDesk, Dropbox, Slack, HubSpot, and MailChimp are just a few examples of SaaS user-favorites.

As per a recent Frontegg survey, 100% of the respondents acknowledged the use of SaaS applications at work, which isn't surprising with the rise in remote work due to COVID-19.

This means that businesses, both SMBs and enterprise ones, now know that the benefits outweigh the risks in today's reality. But what does this mean when it comes to creating and developing the app? What's the importance of self service and multi-tenancy in the SaaS space today?

This guide will dive into the various SaaS architecture options you have today and cover the main challenges you will be facing while building your SaaS application. Without further ado, let's get started.

SaaS Multi-Tenant Architecture for Enterprise - The Why



```
curl --request POST \
  --url https://api.frontegg.com/audits \
  --header 'Accept: application/json' \
  --header 'Content-Type: application/json' \
  --data '{ "tenantId": "the-tenant-id", "user":
"info@frontegg.com", "resource": "Portal", "action": "Login"}'
```

A big reason for the emergence of SaaS applications is their improved security standards, making it easier for companies to move on from on-prem options.

The world is gravitating towards SaaS consumption because it allows them to save time, money, and resources when it comes to IT maintenance and troubleshooting. But what enterprise SaaS architecture is right for you? Which one should you go for to achieve optimal security standards and maximum performance? This article will shed some light over the top SaaS variations you can choose from today.

Introduction to SaaS

The SaaS methodology has been around since the late 90s, but it has taken off in a big way due to the massive spike in internet usage over the last decade. As per Gartner estimates, it has already passed the $100 billion mark, doubling the rivalling Infrastructure-as-a-Service (IaaS) methodology. Platform-as-a-Service (PaaS) and Desktop-as-a-Service (DaaS) also can't hold a candle to SaaS right now.

Table 1. Worldwide Public Cloud Service Revenue Forecast (Millions of U.S Dollars)

YEAR	2019	2020	2021	2022
Cloud Application Infrastructure Services (PaaS)	37,512	43,498	57,337	72,022
Cloud Application Services (SaaS)	102,064	104,672	120,990	140,629
Cloud System Infrastructure Services (IaaS)	44,457	50,393	64,294	80,980
Desktop as a Services (DaaS)	616	1,203	1,951	2,535
Total Market	242,697	257,867	306,948	364,062

So what is SaaS all about? It's a software licensing and delivery business model where you get partial or full access to the centrally-hosted application that is easier to consume and update. The Application Service Provider (ASP) is now essentially "shrink-wrapping" the software to business users via the internet, giving users a user-friendly and feature-rich experience with zero investment in maintenance.

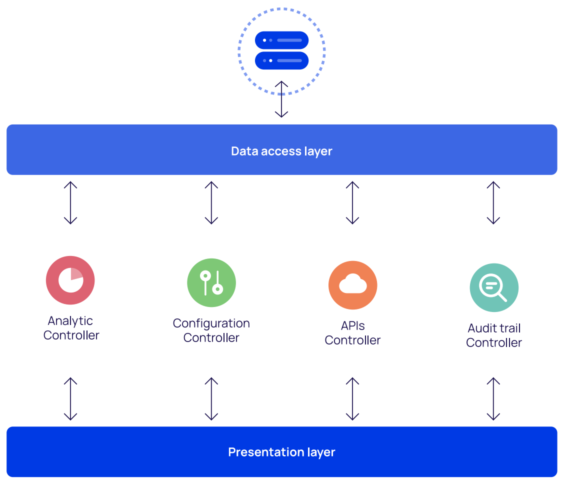
There are two main types of SaaS varieties today:

- ✓ Vertical SaaS - Industry centric solutions (finance, healthcare, etc.)
- ✓ Horizontal SaaS - Industry agnostic solutions focusing on functionality

Besides the aforementioned scaling and maintenance benefits, SaaS applications allow organizations to shift from a reactive approach to a proactive one. They can quickly add functionality to their applications with minimal investment in in-house development. Furthermore, integration is usually a breeze to ensure short time-to-value, thus enabling faster time-to-market with optimal quality.

The Evolution of SaaS Architecture for Enterprise: From Old Monolithic Models to Multi-Tenant Functionality

It all started in the monolithic era, where all APIs, databases, services, and the UI were baked together into a unified executable process. The Presentation Layer was responsible for communicating with the various Controllers, while the data layer took care of the Model. The Controllers were responsible for the logic part and the View took care of the presentation layer. A pretty straightforward arrangement.



The Model-View-Controller (MVC) approach

There were two main variations with MVC. First, there was the Active MVC pattern, where the Model notified the View when changes were made by the Controller. This was not the case with the other variation, the Passive MVC pattern. With this variation, the notifications tasks were performed by the Controller. Passive MVC was better to create separation between business and presentation logic.

Unfortunately, as DevOps started going mainstream, the monolithic based applications were simply not dynamic enough to handle the frequent changes made by the development teams. The Continuous Integration Continuous Deployment (CI/CD) pipeline started facing bottlenecks and performance issues. This is before we dive into problems with scaling up (and down) due to the rigid infrastructure.

There was a need for a more dynamic methodology that would align with the modern development practices. This is where microservices entered the picture. The main idea behind this new methodology was that every service was the owner of its own data. This segregation allowed quicker testing and faster CI/CD capabilities to improve scaling capabilities without compromising on quality.

The Different Types of SaaS Architecture

Let's take a closer look at the main SaaS architecture patterns that are in use today and learn more about the pros and cons that come with their implementation.

Monolithic Architecture

As explained earlier, this is the traditional way of doing things. The monolithic SaaS application is a single and indivisible module that cannot be split or segmented for optimizing development. You are basically looking at one big database and a solution that is built around a server-side and client-side interface. Everything is unified with all functions being managed and served in one location.

- ✓ Pros: Less cross-cutting issues, Easier to monitor, Straightforward testing
- ✕ Cons: Complex while scaling up, Difficult to make big changes, Technology barriers

Microservices Architecture

The microservice architecture is powered by Application Programming Interfaces (APIs). All functions are broken down to independent modules that can be deployed separately as required. The APIs then allow these modules to communicate with each other and sync their independent processes to work as one single entity. Each service can be upgraded, updated, and scaled separately for added flexibility.

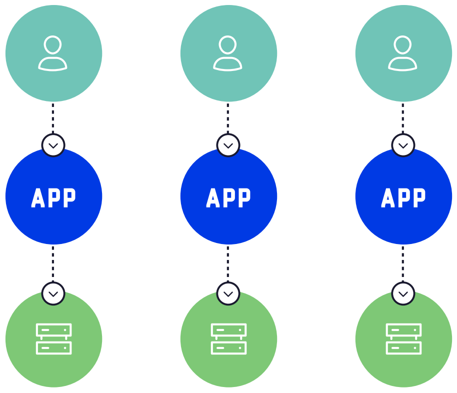
- ✓ Pros: Resource and time friendly. Enhanced scalability
- ✕ Cons: Cross-cutting issues, Complex architecture dynamics, Testing problems

Single-Tenant Architecture

As the name suggests, a single-tenant SaaS architecture basically caters to one customer at a time. The meaning of this is clear - there is a dedicated software instance, single infrastructure, and one database that is serving the entity that is paying for the service. There is no sharing whatsoever and the entire development environment is dedicated to one client at a time.

You also have the Multi-Single-Tenant (MST), where multiple customers can be served by setting up another environment with individually-provisioned instances.

- ✓ Pros: Improved security levels, Customization is easier
- ✕ Cons: Resource heavy, Costly to maintain

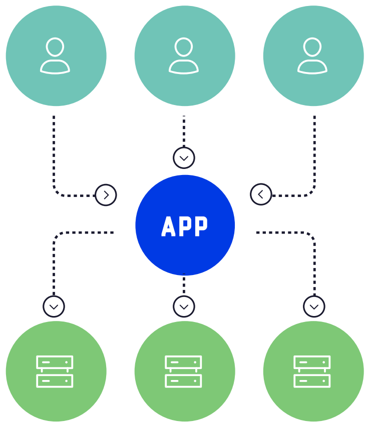


Single tenant app model

Multi Tenant Architecture (SOA)

Unlike Single-Tenant architecture, the Multi-Tenant variation is more focused on sharing resources, especially software instances and databases. However, each tenant's data is protected and saved in different places for obvious reasons. The financial benefits of this methodology are clear - having multiple customers allows to lower the environment and infrastructure costs significantly.

- ✓ Pros: Pocket friendly, Easier on the integration front, Smoother maintenance
- ✗ Cons: Harder to customize, Potential security loopholes, Complications with updates



Multi tenant app model

> "Multi-tenancy is really the future of our industry."
> Marc Benioff, CEO, Salesforce.com

Single Tenant vs Multi Tenant SaaS Architecture for Enterprise

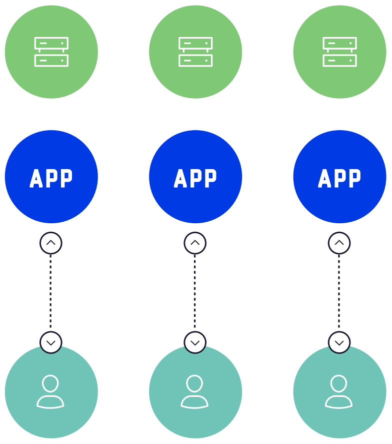
Single-tenant architecture is here to stay as it performs better on the security front, which also makes it easier for businesses to demonstrate ongoing compliance. The security risks are simply more isolated. You also have more control on what's going on the customization and personalization fronts. But that's where the advantages more or less end when it comes to single-tenant SaaS architecture. Single-tenant SaaS architecture allows seamless and smooth cost-sharing for serviceability, ongoing governance, and deployments. Resource utilization is significantly improved and so is the adding/onboarding of new customers. Furthermore, scaling up becomes much easier since you have more computing capacity and more free resources on-demand to get the job done fast.
It comes as no surprise that leading on-demand SaaS applications like Slack,Zendesk, Bogo, and other market-leaders have gone the multi-tenant way.

	Single–Tenant	Multi–Tenant
Security/Compliance	Every user/client has a dedicated secure database	Data leaks or breaches can cause more damage
Customization	It's easier to customize the dedicated architecture	Every architecture update affects multiple clients
Cost Effectiveness	A new instance needs to be created for every user/client	All users/clients share the same instance
Scaling Capabilities	Scaling up can become extremely challenging	Scaling up becomes smooth and seamless
Ongoin Maintenance	Requires large teams to build, maintain, and update	Significant resources and manpower savings

While multi-tenant SaaS architecture has clearly taken a lead over the single-tenant variation, the latter is not going away anytime soon as some use cases still require it. Think about a military application that requires the highest levels of security and customization for best results. The single-tenant SaaS architecture is the way to go in such scenarios. The same can apply for sensitive government setups.

Multi-Tenant vs Multi-Instance Architecture

The multi-instance SaaS architecture is another variation that is gaining popularity in the IT space today. Just as the name suggests, there is no resource-juggling involved with this methodology. There are separate software instances (data items) that simply run parallel to one another. Unlike multi-tenant scenarios, there are no buffers or remote machines involved, which significantly improves performance.



Multi-Instance SaaS Architecture Source:[Scaleway]

Another advantage multi-instances have over multi-tenancy is that data is completely isolated, which means that there are minimal security risks involved. Each team has its private database and ecosystem, which means that there is little to no incentive for hackers. Scaling up is also much easier and so is availability, since single faults can lead to multiple downtimes in multi-tenant environments.

Multi-instance setups have their downsides. They're less cost-effective and are harder to maintain and deploy frequently, an important DevOps requirement.

Summing it Up

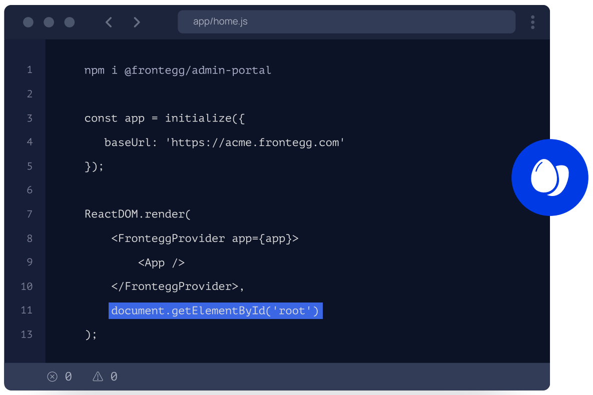
Now that we have taken a closer look at SaaS architecture and learned about the benefits of multi-tenancy, let's move on and examine how it can be implemented for improved productivity and optimized performance metrics.

"You need to get started in the SaaS space with accurate planning and finding the right solutions for your operational, compliance, security, and business needs. Besides being proactive, you need to adopt the principles of self-service and multi-tenancy. That's the only way you can achieve sustainable growth today."

Sagi Rodin, Co-Founder and CEO, Frontegg

SaaS Multi-Tenant Architecture for Enterprise - The How

It's no coincidence that Software-as-a-Service (SaaS) has leapfrogged rivaling methodologies like Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). Let's learn more and show you how to get started with your SaaS journey.



```
npm i @frontegg/admin-portal

const app = initialize({
   baseUrl: 'https://acme.frontegg.com'
});

ReactDOM.render(
    <FronteggProvider app={app}>
        <App />
    </FronteggProvider>,
    document.getElementById('root')
);
```

HTML, along with CSS and JavaScript, are powering the SaaS revolution.

Before getting started you will need to analyze your target audience and understand the market/s you will be operating in. After defining these basic needs, comes the stage where you need to understand the technical requirements your SaaS app architecture needs to accommodate. This article will dive into these exact specifics and introduce you to the best available options in the market right now.

Getting Started with SaaS App Development

Like any other project, the first step and most important one is planning.

Many organizations fail to pick the right type of SaaS application architecture, leading to a disastrous chain-reaction that becomes hard to deal with as scaling up starts. Beyond the complexity involved in maintaining bad architectural decisions, it also has a direct impact on the services you are offering and pricing flexibility, which are the biggest reasons you are getting started with SaaS in the first place.

Here are the three crucial factors you should inspect closely before getting started:

- Infrastructure and Hardware - This is where you need to pick the right cloud vendor for your needs and tech stack to go with it
- IT Administration - You need to take care of provisioning automation, while also ideally trying to automate installations, backups, and updating processes
- Licensing - Your SaaS model needs to be cost optimized when it comes to onboarding feature selection and other important usage matrices

TOP SAAS APPLICATION ARCHITECTURE CONSIDRATIONS

INFRASTRUCTURE | IT ADMINISTRATION | LICENSING COST

Picking the right SaaS architecture model for your business is very important. For example, if you have picked isolated app Virtual Machines (VMs) for your ecosystem. This will let you achieve optimal security and performance metrics, but these boxes will not be fully utilized. But if you take the other route and share the DB and app servers with multiple clients, a random spike can trigger performance issues.

Related: The Evolution of SaaS Architecture

Nothing is irreversible in the IT/SaaS space, but going the wrong route will lead to costly and time consuming realignment processes that will impact your business.

How to Pick the Right SaaS Architecture?

Now that we have established the basic best practices that need to be adopted in the initial planning stage, it's time to get familiar with the 4 SaaS architecture types.

- ✓ Type 1 SaaS Architecture – This type of architecture basically requires runtime and data isolation, but not necessarily on the cloud

  What's it right for? Non-cloud use cases like banking and finance businesses

- ✓ Type 2 SaaS Architecture – Unlike Type 1, this architecture needs runtime and data isolation on the cloud as well. The cost of isolation is put on customers

  What's it right for? Customers who don't want to share infrastructure and the app

- ✓ Type 3 SaaS Architecture – Here, the app is shared with all customers, which means that there is no runtime isolation whatsoever (minimal deployment)

  What's it right for? Privacy-conscious customers unwilling to share data storage

- ✓ Type 4 SaaS Architecture – This type of SaaS architecture is all about zero deployment where all customers share the app and database infrastructure

  What's it right for? High onboarding frequency scenarios. For example, Wordpress.

	DATA ISOLATION	RUNTIME ISOLATION	CLOUD USAGE	ONBOARDING FREQUENCY
SaaS Type 1	✓	✓	✗	LOW
SaaS Type 2	✓	✓	✓	HIGH
SaaS Type 3	✓	✗	✓	VERY HIGH
SaaS Type 4	✗	✗	✓	EXTREME

SaaS Architecture Types: Compared

As explained earlier in this guide, your customer requirements and behaviors should dictate your SaaS architecture choice. This is the only way to achieve sustainable growth and steer clear of situations where your development teams have to rebuild everything from scratch. SaaS is a profitable, smooth, and manageable methodology. But this is valid only if you minimize operational costs and work.

Related: The Evolution of SaaS Architecture

The Eternal Question: Single or Multi-Tenant?

If you have customers looking for extremely high levels of reliability, security, and customization, single tenancy might be the way to go. It is also easier to migrate to new environments and perform quick backups (and recoveries). Just keep in mind that the SaaS route will come at a price, literally. These SaaS ecosystems are often underutilized and very costly to configure, manage, and maintain.

Multi-tenant SaaS architecture is the better option if you want to create a scalable business model that is flexible and versatile. Here are some benefits:

- ✓ Lower Maintenance Requirements - All ongoing maintenance costs can be baked into your pricing models. Updates and patches apply to all users
- ✓ Cost and Resource Saving - Everything is shared - databases, applications, services, and resources. The result - lowered operational costs
- ✓ Feature Rich and User Friendly - Onboarding is significantly faster and easier. Self-service features enhance customer experience and brand performance
- ✓ Improved Efficiency - While this requires the right infrastructure and tools, all users can ideally get the same level of service and performance
- ✓ Better Scalability - All of the aforementioned benefits allow multi-tenant centric businesses to scale up faster thanks to the added elasticity

Just make sure you are getting your multi-tenant design right before getting started. This will allow you to make sure that the right information is going to your users without any unintentional mix ups or data leaks. Assigning unique tenant IDs is an effective way to achieve this goal. If your SaaS application is going to share tenancies, make sure it can accommodate flexible schema usage as well.

Last but not the least, make sure that you are secure and compliant with the latest data privacy laws like GDPR, CCPA, and other ones relevant to your geolocation.

Related: How Passwords are Breached?

Let's Talk About Serverless Architecture

Single and multi tenant SaaS architecture, with all of their variations, are not your only options. You should also take Serverless Architecture into consideration.

Serverless architecture can operate as a BaaS (Back-End-as-a-Service), where the application's backend is primarily on the cloud. It can also be a FaaS (Function-as-a-Service), where the application runs the code via various event triggers. In other words, functions are evoked on-demand, which allows businesses to enjoy a wide range of benefits that cannot be ignored.

Here are just a few of them:

- ✓ Quick(er) Deployment - Zero provisioning needs. Automatic scaling
- ✓ Integration - Agile-friendly, works well with microservices, less complex
- ✓ Focus on UI/UX - Your development teams can focus on features/frontend
- ✓ Flexible Infrastructure - Buy servers on-demand. Scale up (or down) fast
- ✓ Better Latency - Access points all around the world for better performance

But like everything in life, serverless architecture also has its flaws and limitations that need to be factored into your decision-making process. The biggest and more significant one is the vendor lock-in issue. Depending on an external third-party vendor will take away a lot of your control. There is also the "cold-start" issue, where the platform needs to initialize and fire up internal resources, causing delays.

SaaS Architecture for Enterprise: Best Practices

A well-planned and executed SaaS architecture can have high levels of fault tolerance and allow you to make sure disaster recovery will not be a disaster.

Here are the top three ways to make sure your SaaS app will succeed:

- ✓ Self Service - Make your SaaS application as self-serving as possible. The more your customer needs to contact the support team to get things done, the less likely he is to stay with your brand. Registering, setting up a password, learning about new features, and eventually upgrading memberships - all of these actions should be user-friendly with the end-user in mind.

- ✓ Multi-Tenancy - Once your SaaS application is able to serve more than one customer, you can enjoy more flexibility. This means you can scale up faster with no technical or operational constraints. Your application/s should also have the ability to accommodate new third-party applications and external tags for added functionality and customizations benefits.

- ✓ Microservices -Embracing microservices will help you further simplify your ecosystem for easier and smoother maintenance (and upgrades). It will also become easier to pin-point issues and solve them without painful downtime or roll-backs that can damage your brand reputation. Just make sure you have automated your database governance and management before doing so.

Related: Common Security Pitfalls of User Management

Working With Top Cloud Providers

Going the Amazon Web Services (AWS) route? For starters, your database should ideally be Amazon RDS. PostgreSQL is also a decent option.

Your SaaS tech stack should ideally be powered a Python, React, and AWS programming combo. Many organizations with non-complex offerings are also going with Node.js instead of Python. Your container orchestration platform should have Amazon Elastic Container Service (ECS) if you are a startup or have a medium sized operation. Big ops can pick Amazon Fargate or go for Amazon EKS.

With Microsoft Azure, you have MySQL or PostgreSQL, which can be procured from the Azure portal. Make sure you are using elastic pools for more flexibility.

On the container front, you will be working with Azure Kubernetes Service, also known as AKS. This will allow you to smoothly run microservices and run your containers (without managing servers) on Linux. Microsoft also offers seamless API Management, which runs in a multi cloud setting and allows you to adopt API architectures across multiple environments.

The Future Belongs to Multi-Tenancy

Multi-tenancy is becoming the obvious choice for organizations looking to scale up fast and optimize development pipelines. While there will always be specific use cases (not many) for the old single-tenant approach, all modern offerings have to go the multi-tenant way. This is the best way to implement new features across the board, cut setup costs, and speed up maintenance. Multi-tenancy is no longer a choice in the SaaS space.

While building your multi-tenant product, user management cannot be overlooked. Frontegg offers an end-to-end user infrastructure that enables you to focus all your resources on your core software product. Our multi-tenancy by design approach gives your customers a potent self-service mechanism to modify and control their security settings on the fly. Don't believe us? Check it out for yourself.

Start for free >