The Perils of Single-Tenant SaaS

Unlocking the Power of the Cloud Via Multi-Tenancy

bolti systems
building a stronger SaaS business

Executive Summary

The benefits of multi-tenancy for cloud SaaS architectures are well understood. Yet many organizations settle instead for single-tenancy, sacrificing elasticity and cost flexibility – the very promise of the cloud. This means dramatically higher infrastructure costs, eroding margins and competitiveness. This white paper shows you the strategic advantages of operating a multi-tenant SaaS architecture, the challenges, and how Bolti Systems can help you succeed.

Preamble

One of the perks that comes with being a cloud architect is the many different architectures we come across and the different storylines they narrate. Systems and software architectures are more than just an aggregation of components working together: they are personalities infused by their creators, design trade-offs constrained by their environment, and sometimes, missed opportunities for improvement. One such example of the latter is the use of single-tenancy in modern SaaS architectures. In many ways, single-tenant architectures can put SaaS organizations at a disadvantage. In spite of this, some organizations continue to use this model and it typically it stems from a combination of factors:

- Legacy single-tenant ISV architectures being ported to cloud SaaS
- An application originally designed for one client suddenly being released to many clients
- Prohibitive costs of mass re-architecting
- Lack of awareness

Sometimes, choosing single-tenancy is the result of careful analysis and operating within the constraints of an organization. Pragmatism has its place in software architecture. However, choosing the single-tenant path can also mean that the SaaS organization will sacrifice on the very promise of the cloud: elasticity and cost flexibility. This white paper delves into the details of multi-tenancy, the challenges, and explains how SaaS organizations employing a multi-tenant architecture can reap strategic advantage.

Single VS. Multi-Tenancy

Let's start with a formal definition of these two types of architecture. Many SaaS organizations use a licensing model whereby each client is allotted a slice of the total system infrastructure pie. Individual clients then become tenants of this system. Each tenant, in turn, has a pre-determined amount of users that are allowed to use the service[1]. When building and deploying the service, organizations can choose between two extremities on the tenancy continuum:

Single-tenancy

An architectural paradigm where all infrastructure, processes and data are dedicated to one tenant.

Multi-tenancy

An architectural paradigm where infrastructure and processes (but not data) are shared by all tenants.

As noted, this is a continuum and not a binary option. Architectures can be hybrids of models.

The chosen architecture will have a huge impact on the cost structure of the offering. Let's compare basic infrastructure costs of these two extremes when deployed in a cloud environment. To do so fairly, we use a simple SaaS application consisting of a front-end, a processing layer (e.g. an application server with business logic) and an SQL database layer. This example has a SaaS organization that currently serves 16 tenants. As with any industrial strength SaaS application, each layer has built-in redundancy to mitigate hardware failures.

[1] Or some other form of resource allowance

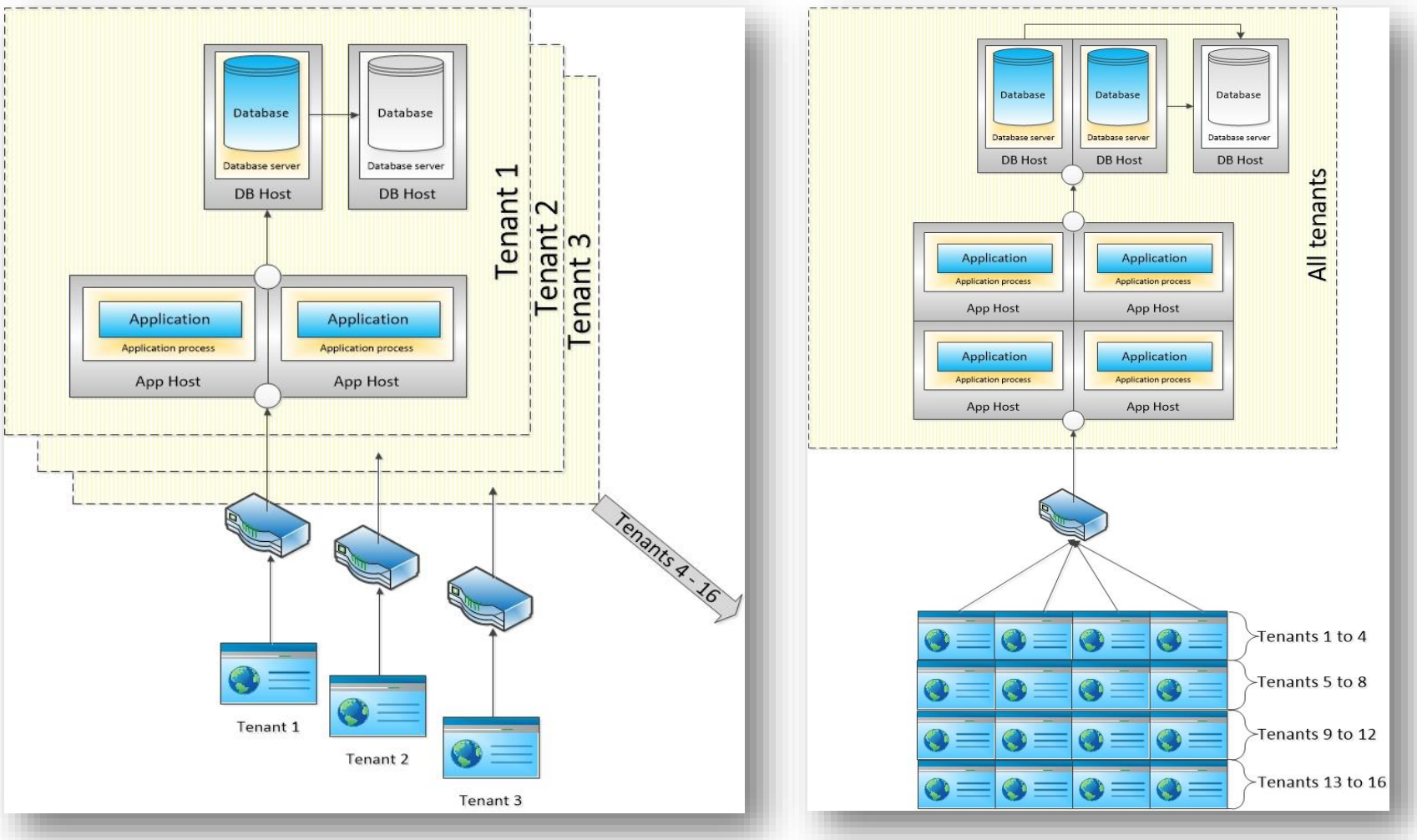

Figure 1: Single (left) vs. multi-tenant (right) architecture

The capacity model we'll use is one in which each application host can serve up to four tenants (each generating an equal amount of traffic) and that each database server can serve up to 16 tenants[2]. Also, we'll deploy this on Amazon Web Services public cloud and apply their price list to the model. While this is a fictitious example, it is not an atypical one. More importantly, though, is if we apply the same capacity model to both architectures, dramatic differences will emerge to the cost model.

In the single-tenant architecture on the left in Figure 1, each tenant has dedicated infrastructure and a minimum amount of hosts to ensure redundancy. Specifically, two database hosts and two application hosts are provisioned. On the right side, the entire set of infrastructure is shared and provisioned to handle all of the 16 tenants. The end result is that only eight infrastructure components (seven application and database hosts, one load balancer) are required to serve the entire tenant base. Whereas 80 individual components (64 application and database hosts and 16 load balancers) are needed for the single-tenant architecture. This will significantly impact the bottom line. Let's explore basic infrastructure cost differences between the two models.

Single VS. Multi-Tenant Cost Models

Figure 2 shows the cost model as additional tenants are added for each architecture type.

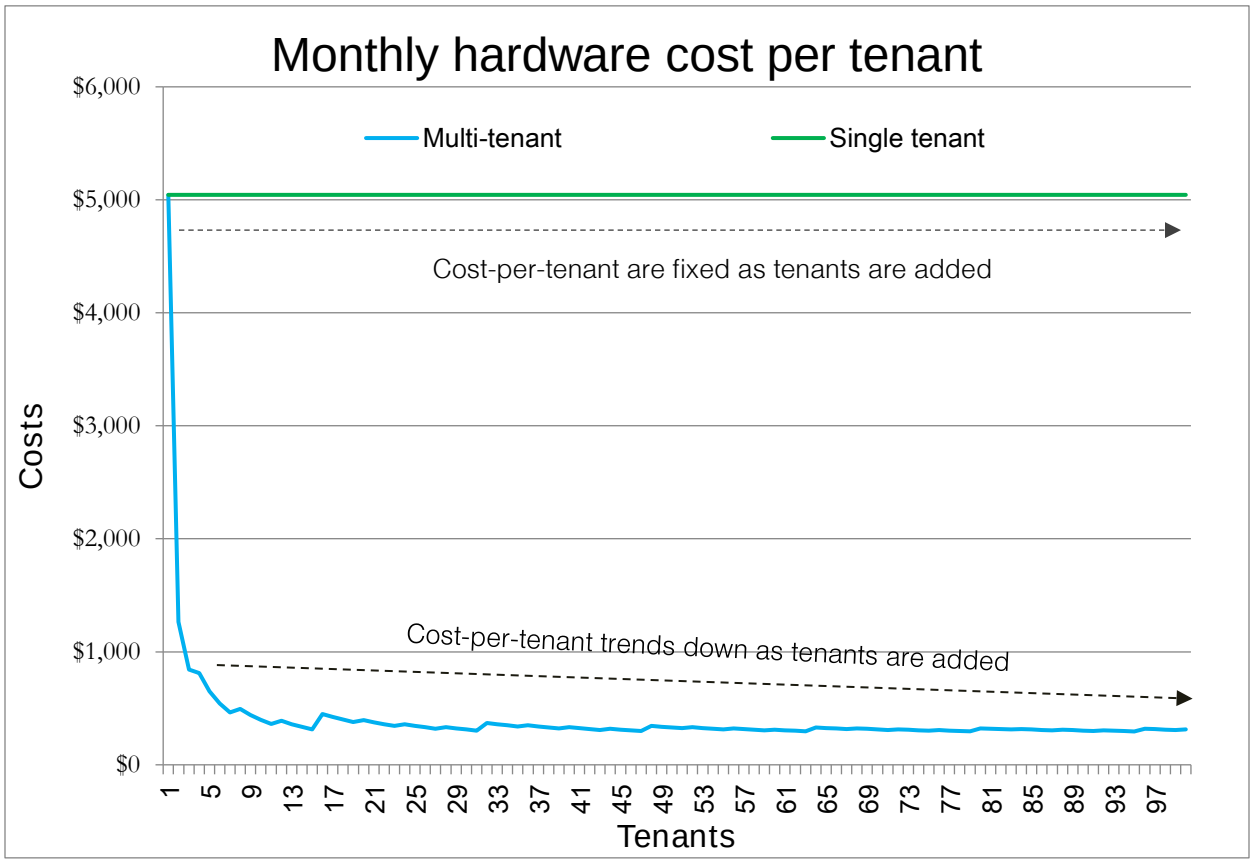

Figure 2: Multi vs. single costs per tenant

[2] See the appendix for the full capacity and cost model.

What's interesting is that not only does the multi-tenant architecture (blue line) start with lower or equal costs, but costs trend downwards as the business grows. Elasticity is the prime reason for this. In contrast, the single-tenant architecture (green line) has a constant cost regardless of the number of tenants. There are no economies of scale to be reaped because elasticity in not in play. The advantages are obvious.

There is also another hidden advantage. Consider the case where the number of deployments double from 16 to 32 tenants: the total costs will also double. The business must have adequate cash flow to support the doubling of costs from $81K to $161K per month. Compare that with the multi-tenant architecture, where extra costs would grow very slowly to about $4700/month for the same growth in tenant. This is shown in Figure 3.

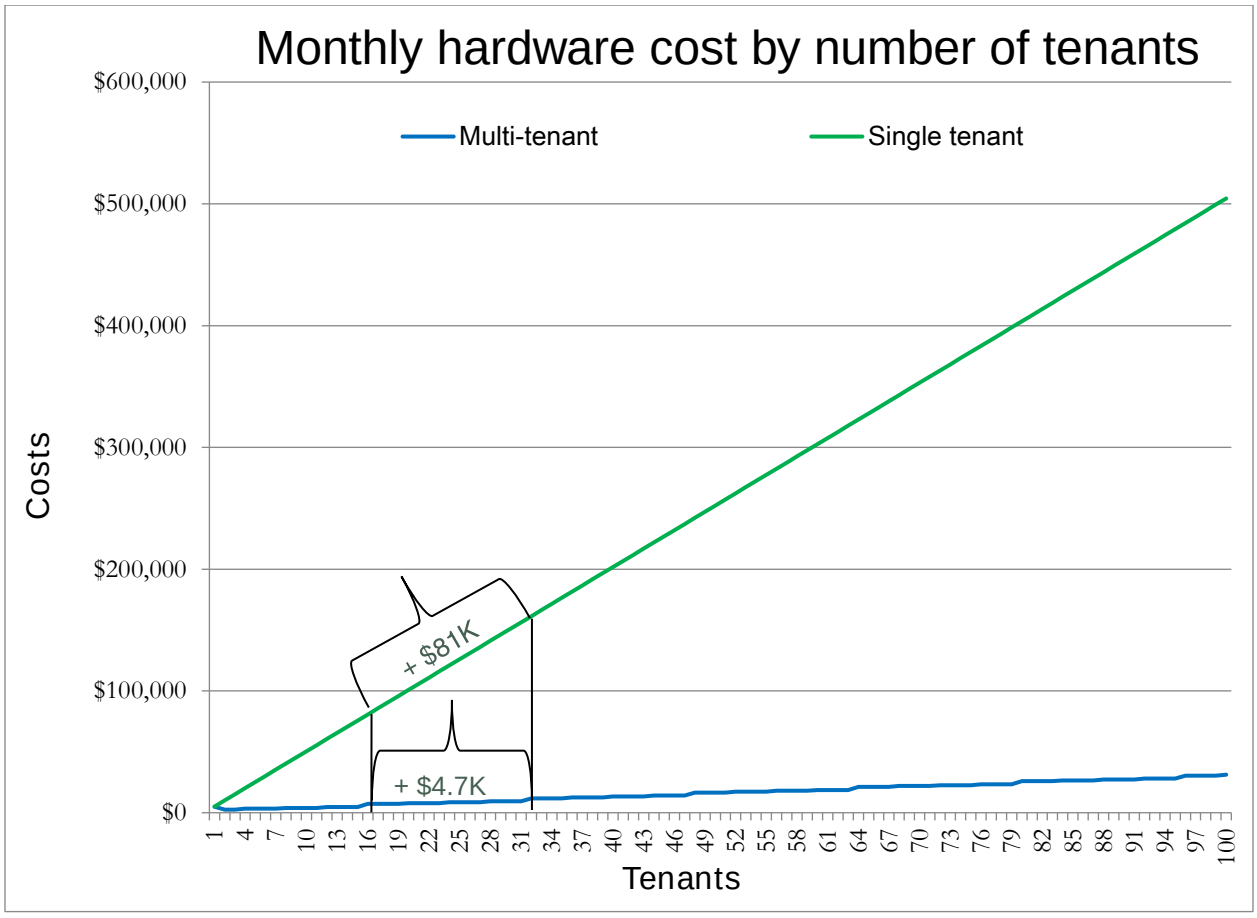

Figure 3: Multi vs. single tenant infrastructure costs

Over time, these differences can be massive. On an annualized basis, for example, costs would balloon to almost $1.8M. As shown in Figure 4, annualized cost differentials are compounded as the business grows. By the 100th tenant, they would have increased linearly to over $5M.

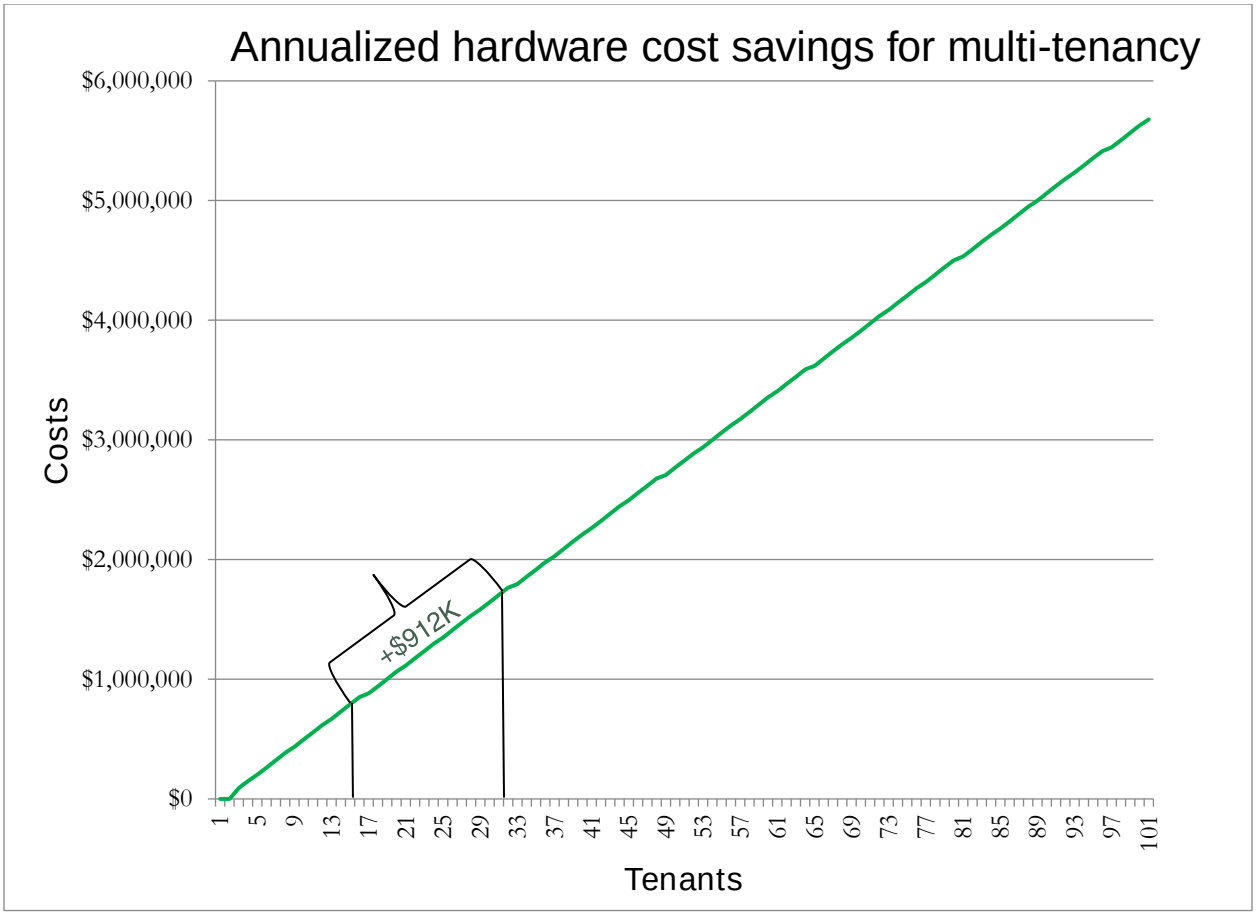

Figure 4: Annualized cost savings for multi-tenancy

Naturally, the numbers will differ when applied to your own unique architecture. The constant, however, is the differences in cost structure as the business grows. Of course it is still possible to profitably operate a single-tenant architecture provided that the market can support the higher prices. Eroding margins notwithstanding, this cost model would leave this single-tenant SaaS organization vulnerable to competitor using a cost-effective architecture. We explore multi-tenancy next.

Our Vision of Multi-Tenancy

We are believers in the power of multi-tenancy here at Bolti Systems. But we also believe in implementing it in a certain way, using these principles in order to reap all the rewards. These are:

One environment

There is only one deployed environment for the entire SaaS application.

Share-everything architecture

All infrastructure is shared indiscriminately amongst all tenants.

One software version

All tenants use the same version of the software.

Let's explore each of these separately.

ONE ENVIRONMENT

The first principle is that there is only one deployment environment for the entire SaaS application. This does not translate into one physical environment: the application could be distributed in multiple data-centers and regions to mitigate the risk of natural disasters. But there is only one logical deployment possibly consisting of multiple physical locations.

We've already shown how multi-tenancy offers big savings on infrastructure costs. One of the contributing factors is licensing costs for third-party software, which includes operating systems and database server licenses in this example. Common licensing models, such as the per-core pricing, make it advantageous to time-share the core amongst as many tenants as possible. Individual tenants can have diverging traffic patterns meaning they don't all peak at the same time. Multi-tenant resource sharing maximizes usage of the CPU core and therefore the license itself. In other words, sharing cores amongst tenants gives you the best bang-for-your-buck. Some vendors also offer per-user licenses, which tend to cost less than the per-core license. However, these are only advantageous for systems with few users and this is really not aligned with multi-tenant systems with many users.

Another cost savings is the reduction of operating staff. Having one environment reduces the team size. Less pairs of eyes are required to monitor one homogeneous deployment compared to the hundreds of potentially diverging ones.

To be sure, there are practical limits to cluster sizes and tenants served. Depending on the amount of individual components, it might become practical at some point to partition an overgrown environment into several pieces for better manageability. For example, environments could be grouped by similar tenant profiles or whichever other type of grouping eases operability. This would naturally negate some of the cost savings. Still, a handful of multi-tenant environments is better than hundreds of single-tenant ones. But like any good principles, caveats apply.

In all, however, the one-environment translates directly into reduced infrastructure and manpower costs.

SHARE-EVERYTHING ARCHITECTURE

The second principle is the share-everything architecture. Multi-tenant architectures share all infrastructure and this plays into the strengths of the cloud.

The promise of the cloud is about elasticity and cost-flexibility. By elasticity, we mean that the cloud offers a way to add processing power just in time and only if needed. This works in both directions of the continuum. That is, it is beneficial for up-scaling as much as it is for down-scaling.

Consequently, micro tenants with a small footprint can be served profitably – something difficult to achieve for single-tenant architectures where fixed costs either erode margins or the service is priced out of the market.

The second component, cost-flexibility, means that unlike owning hardware, organizations need not pay up front for infrastructure that they may or may not need. Infrastructure is only provisioned

when demand is present and de-provisioned otherwise. Single-tenant SaaS architectures weaken the value proposition of the cloud. The previous single vs. multi-tenant cost comparisons did not factor demand fluctuations per tenant. In the multi-tenant architecture, infrastructure can be provisioned using aggregate demand of all tenant traffic. This means that tenants can borrow computing resources from each other, (provided they don't all peak at the same time), without needing additional resources. Not so in a single-tenant architecture: tenants operate in silos and so each environment must be provisioned to meet maximum demand. In this way, elasticity is not fined-grained in single-tenant architectures. In a multi-tenant deployment, infrastructure costs increase slowly because this dynamic is at play.

Without elasticity and cost-flexibility, the advantages of the cloud over owned infrastructure are diluted because cloud providers demand a high-premium for these benefits, which remain unleveraged. In this sense, single-tenancy on the cloud is the worst of both worlds: it's more expensive than owning infrastructure without the benefits of elasticity.

Hidden in all this, but just as important, is the fact that on-boarding new tenants is a much cheaper and quicker proposition in multi-tenant architectures since the infrastructure is already in place. In a properly designed multi-tenant architecture, a configuration change is all that is required to add a new tenant. Furthermore, off-boarding is just as easy and cheap.

ONE SOFTWARE VERSION

The third and final principle of multi-tenancy is having only one software version in production. This principle can be used to the advantage of the SaaS provider allowing them to set the schedule for upgrades – leaving no tenant behind using an older version. In single-tenant environments, there can be many different versions deployed and engineering teams in SaaS organizations can be paralyzed by having to support all of those versions. Bug fixes need to be propagated to all previous versions. There are many software development tools and processes (e.g. Git and GitFlow) that can help but there is an inherent risk of error as this is still mostly a manual process. Every version to support in production translates to another source control branch to maintain and keep updated. The more source control branches, the slower the engineering team is to deliver on bug fixes and new features.

In contrast, having one software version is an outcome of a multi-tenant environment. This frees the engineering from having to maintain many different version and allows instead to focus on continuous integration and delivery processes, which are now achievable in multi-tenancy. This has a direct impact on software quality. Single-tenancy does not inhibit this principle but multi-tenancy enforces it.

Operation teams can also benefit from having one software version allowing them to focus on one set of software behaviors, leading to better monitoring, responsiveness and SLA conformance.

One of the countering forces at play is tenant pushback: some tenants may want to impose their own timetable for upgrades rather than being forced by the SaaS provider. A small provider may not have the leverage to drive the agenda against a larger tenant counterpart. We recommend making exceptions to the one-software-version rule by segregating these types of tenants into isolated environments. This will certainly go against the grain of our multi-tenant philosophy, but strategic accounts need strategic care.

Multi-Tenancy Is Challenging

We hope we've offered compelling arguments in favor of multi-tenancy, but it's important to point out that multi-tenancy presents its own challenges: organizations face barriers to entry. For example, legacy architectures designed as single-tenant can, in some cases, require significant changes and driving this effort requires champions within the organization.

Multi-tenancy also poses new risks:

- Inter-tenant data leaks (or co-mingling): software bugs could cause crosstalk of tenant data amongst the tenants. That is, data from one tenant can be accidentally served to another tenant.
- Security breaches: lapses in security can lead to multi-tenant deployments being more vulnerable because hackers now have access all tenants with one breach.
- Service outages: if the architecture does not properly factor regional redundancy, service outages can be more damaging because they can affect all tenants.

Additionally, multi-tenancy initiatives can fail to achieve their objectives if these are not properly addressed:

- AppOps: An application operation software layer, separate from the SaaS application itself, that manages the tenants (e.g. on-boarding, off-boarding, starting, stopping, upgrading, backups, etc.). Not having an AppOps layer can make it difficult to operate the service.
- Incompatible technologies: Some technologies are incompatible with multi-tenancy and can only be integrated by dedicated-per-tenant deployments.
- Continuous integration and delivery: new practices and processes are needed to fully take advantage multi-tenancy.

Bolti Systems

Is the cost of operating your architecture optimal or is your organization accepting something sub-optimal? Bolti Systems can help you assess your architecture and prescribe the right course of action. We are a boutique consulting company that helps clients SaaS-ify their products. We can help you with:

- SaaS architecture
- Software development
- Continuous integration and delivery
- SaaS business strategy

No matter how far down the path you are to SaaS-ifying your product, our team can help assess and re-architect your application. With decades of experience developing software, we know that making big changes is more than just a technical talk - it involves bringing all stakeholders together. That's why we can embed ourselves in your team and champion the recommendations we make to help deliver on our promises.

The Bolti Approach

Every organization has a unique set of circumstances. There are no one-size-fits-all approaches and this is why we adopt a pragmatic approach where we can assess your architecture's suitability for multi-tenancy. We can map out an evolutionary path to slowly align your architecture with the best of what the cloud has to offer. Here are some general guidelines and recommendations we make when designing multi-tenant architectures.

Q: How are relational database models made to be multi-tenant?

Relational databases are often central to multi-tenant architectures. They dictate much of the overall architecture because not only do they dictate how tenant data is stored but they also have an impact on the server redundancy and recovery. This, in turn, puts constraints on RTOs (recovery time objective) and RPOs (recovery point objective).

Database schemas can support multi-tenancy one of three ways: single schema/shared rows, where the entire user base of all tenants is stored in one shared schema, separate schemas/shared server, where each tenant has their own schema but the database server is shared, or separate schemas/separate server, where each tenant has their own schema and database server. The third option is the least aligned with the principles of multi-tenancy because it doesn't do much infrastructure sharing. This results in much higher costs. However, it does lower the barrier of entry into multi-tenancy and can be used as first step of long-term evolution strategy.

Evolving a single-tenant architecture to a multi-tenant one also impacts much of the application logic. A modularized approach, where a separate module determines how to find the tenant data, is better than embedded this logic throughout the application. This can drastically reduce the effort required to render the application multi-tenant.

Q: What are some of the ways to avoid inter-tenant data leaks?

In some SaaS systems, inter-tenant data leaks can be devastating to a SaaS organization and damaging to their reputation. If the database is using a single schema/shared rows approach, each table then contains a mix of all tenant data. A well-designed, tested and error-free persistence layer in the application would normally never co-mingle data. However, extra care can be taken to prevent such errors by using different encryption keys for each tenant. Any interaction with the database would be done by using distinct, per-tenant encryption keys. Any queries accidentally returning a mix of inter-tenant data would make that data unreadable to the application. The application would likely fail trying to decrypt the data.

Q: How can security breaches be minimized in multi-tenant architecture?

Security is even more important in multi-tenant environments. The Bolti team leverages the expertise of a security partner to audit every system architecture we produce. This way, security is baked into the architecture instead of being an afterthought.

Q: How can service outages be minimized in multi-tenant architectures?

Service outages can be more devastating on multi-tenant environments because they can affect all tenants. Here, inter-regional deployments can help. By use of geographically redundant architectures, installed on different data centers, floodplains and possibly even continents, we can dramatically reduce of the consequences of natural disasters and continue to operate the SaaS application from another region. However, service level agreements must be constructed with achievable an RTO/RPO, plus the architecture and base technology choices must support these objectives.

Q: Why is application operations important in multi-tenancy?

Application operations is a layer of software that integrates with the SaaS application to offer features such as seamless on-boarding of new tenants, opening tenant access to the system, controlling upgrades, closing access and off-boarding tenants. It is sometimes the forgotten piece in multi-tenancy. However, neglecting it can impair the operations of the system because no control is possible for individual tenants. A good multi-tenant architecture must also include an application operations portion that integrates with the rest of the system.

Q: How can continuous integration and deployment help?

The field of continuous integration and deployment has evolved rapidly in recent years. New trends have emerged whereby some organizations prefer to break down big feature sets into very small changes that can be delivered as soon as ready. This eliminates the need for scary monolithic upgrades that often fail and result in outages. Some organizations have pushed this idea to the point where they are constantly pushing changes throughout each day. The key is to have the entire pipeline aligned. Changes must be automatically tested, verifiable, integrated into the trunk and most importantly, any errors must be easily rolled back. But many tools and processes need to be in place before this can achieved inside an organization. By reducing the effort required to roll out changes, continuous integration and deployment can bring significant gains to SaaS organizations.

Q: Is multi-tenancy always the answer?

No. As with anything else in software architecture – it depends. We certainly believe in the value of multi-tenancy but context is also important. We provide cost-benefit analyses comparing cost-savings vs. engineering effort. Not all applications require extensive re-architecting. As well, hybrid approaches can move some parts of the architecture towards multi-tenancy – leaving other parts untouched. There are many possibilities and no one size fits all.

Q: Can hypervisors or containers be a good alternative to multi-tenancy?

Hypervisors such as VMWare are proven technologies and container-based technologies such as Docker increasingly popular tools that offer many benefits. They abstract the underlying platform and partition the physical hardware so that each seem to be using a separate physical host. Each partition could then be dedicated to one tenant and a means to synthesize multi-tenancy. But this approach is really just a variation of the single-tenant architecture theme. Although they facilitate deployments, they don't offer a compelling cost reduction story when compared to real multi-tenancy. So we believe that these technologies are great at solving the problems for which they were designed (e.g. abstracting the platform, hardware partitioning, environment portability) and could be used as part of complete multi-tenant deployment solution – not as a replacement of it.

Conclusion

This white paper has demonstrated some of the strategic advantages our team at Bolti Systems can give you. We hope to have the opportunity to meet with you. Please contact us at info@bolti.com or Len Madigan directly at 514-262-8756.

Appendix

Capacity model

- Each tenant has an equal amount of users with the same usage profile.
- Each application host can serve traffic for up to four tenants.
- Each database server can serve traffic for up to 16 tenants.
- One load balancer can serve traffic for up to 256 tenants.

Cost model

- The Amazon Web Services public cloud is used for the model.
- Prices are quoted in US dollars using the price list as of June 2016 for US East (N. Virginia) hosting.
- On-demand EC2 instances are used.
- The database server used is SQL Server from Microsoft and uses the i2.2xlarge image with 8 virtual cores and 61Gb of memory. The operating system is Windows R12 Server.
- The application host uses the m4.2xlarge EC2 image with 8 virtual cores 32 Gb of memory. The operating system is Windows R12 Server.
- Monthly costs for one database server: $1795.07

  2.459 (hourly cost) X 24 (hours in a day) X 365 (days in a year) / 12 (monthly average)
- Monthly costs for one application host: $717.59

  0.983 (hourly cost) X 24 (hours in a day) X 365 (days in a year) / 12 (monthly average)
- Monthly costs for load balancer: $18.25

  0.025 (hourly cost) X 24 (hours in a day) X 365 (days in a year) / 12 (monthly average)

Cost examples

- Multi-tenant architecture with 16 tenants
  Monthly rate for application host X (truncate(number of tenants / max capacity per application host) + 1) +
  Monthly rate for database server X (truncate(number of tenants / max capacity per database server) + 1) +
  Monthly rate for load balancer

  717.59 X (truncate(16 / 4) + 1) + 1795.07 X (Truncate(16 / 16) + 1) + 18.25 = $7,196 or $450 per tenant

- Single-tenant architecture with 16 tenants
  Monthly rate for application host X 2 + Monthly rate for database server X 2 + Monthly rate for load balancer

  717.59 X 2 + 1795.07 X 2 + 18.25 = $80,704 or $5044 per tenant