David Chappell

BUILDING SAAS APPLICATIONS ON WINDOWS AZURE

THINGS TO THINK ABOUT BEFORE YOU START

Sponsored by Microsoft Corporation

Contents

Whether you're part of an established software vendor or a brand-new start-up, there's a good chance that your next application will run in the cloud. Doing this means embracing software as a service (SaaS), probably building on a cloud platform such as Windows Azure. But ***creating SaaS applications is different from what you've done before—you need to think in new ways***. This paper describes some of the most important things you should consider as you design your first SaaS application for Windows Azure.

Illustrating SaaP and SaaS

Before looking at a checklist of design issues, it's useful first to think about how SaaS applications compare to on-premises applications that use the traditional Software as a Product (SaaP) model. As Figure 1 shows, the two approaches differ in fundamental ways.

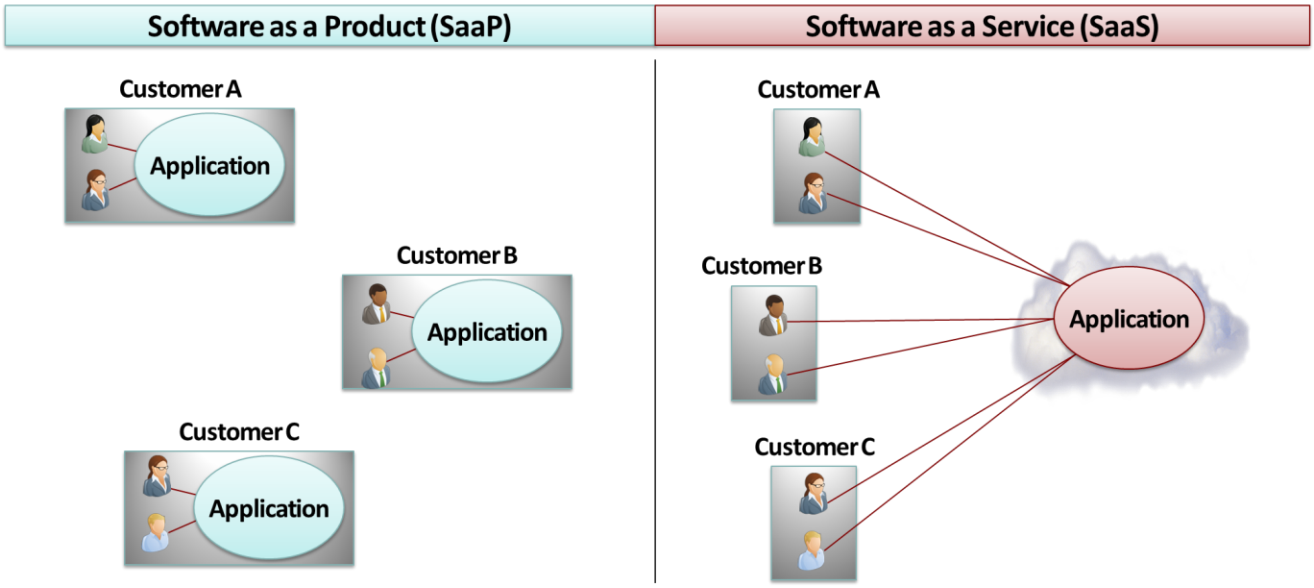

Figure 1: SaaS applications are different from traditional SaaP applications.

In the familiar SaaP world, each customer runs its own copy of an application, usually in its own datacenter. With SaaS, customers commonly share a copy of the application, and that application definitely isn't running in customer datacenters. This apparently simple change in where an application runs and how it's shared turns out to have large implications for how you design and build the application.

Design Issues for SaaS Applications

Creating a SaaS application isn't simple, especially the first time you do it. What follows looks at some of the most important things you need to think about as you design your application.

Multi-Tenancy

The biggest decision you'll make in creating a SaaS application is whether your software will be **single-tenant** ***or*** **multi-tenant**. Figure 2 illustrates the difference.

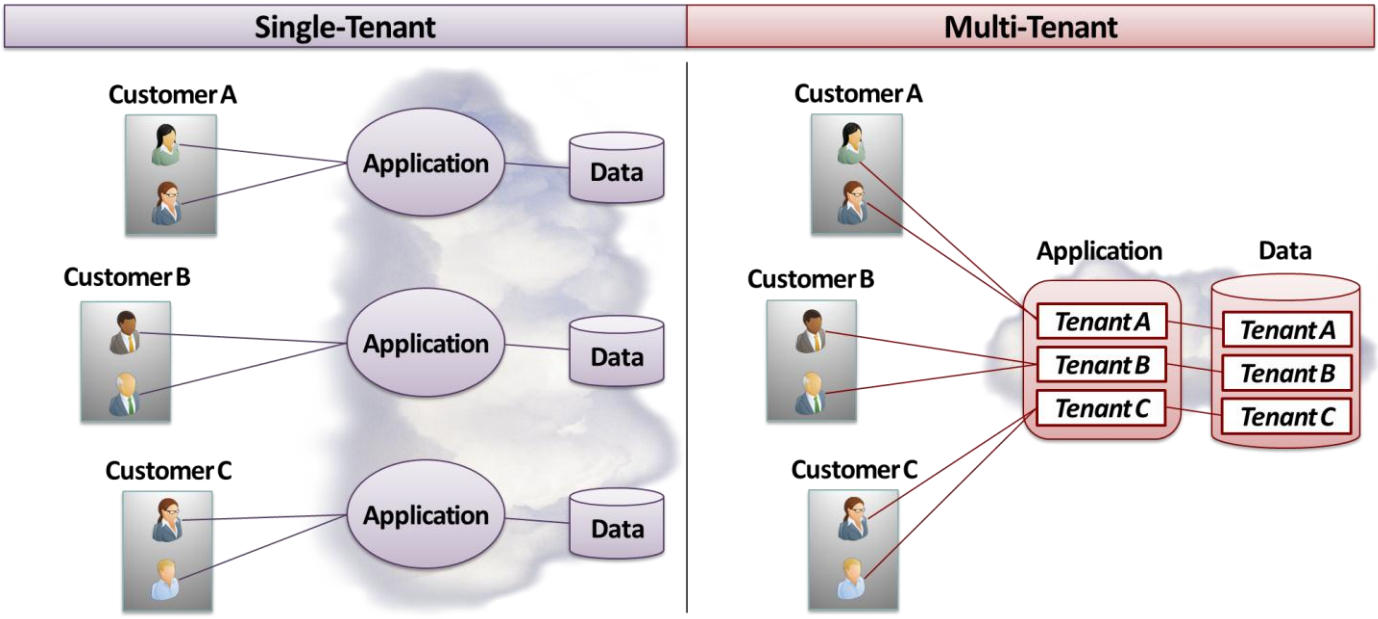

Figure 2: A SaaS application can be single-tenant or multi-tenant.

In a single-tenant application, each customer (such as a company or government agency) has its own instance of the application with its own distinct data storage. In a multi-tenant application, multiple customers share an instance of the application and the data storage it relies on. Which approach is better depends on what you're trying to do—each one has strengths and weaknesses. Figure 3 summarizes the main issues.

	Single-Tenant	Multi-Tenant
Implementation Difficulty	***Low;*** the application is essentially a SaaP application running in the cloud.	***Medium to high;*** there are many issues to consider, and existing SaaP applications usually must be modified.
Customer Perception of Security	***High;*** each customer's data is segregated from other customers.	***Low to high;*** this approach requires more trust in the application provider.
Customizability	***High;*** every customer has its own copy of the code, so each one can be modified as needed.	***Low to medium;*** shared code reduces the customization possible for any single customer.
Required Administration	***High;*** maintaining many instances of the same code gets expensive, especially with customizations.	***Low to medium;*** shared code makes administration and updates easier and cheaper.
Long-Term Costs	***High;*** costs rise significantly with each customer, making this option increasingly more expensive.	***Low to medium;*** each new customer adds some incremental cost, but much less than running a new application instance.

Figure 3: Single- and multi-tenancy each have pros and cons.

The main points in this comparison are easy to summarize:

- ***Single-tenant applications are easier to implement and to customize.*** This means they're typically faster and less expensive to create. They can also help customers feel more comfortable with the idea of SaaS, since many people are initially nervous about sharing code and storage with others. Beyond a certain number of customers, however, ***single-tenant applications can become quite expensive to operate***.

- ***Multi-tenant applications require more work to implement***, and so they take more time and money to create. Multi-tenancy also requires convincing your customers that a shared application can keep their data safe, and that the application can provide enough customization to meet their needs. But ***multi-tenant applications are significantly cheaper to operate*** over time, especially as the number of customers they serve goes up.

For independent software vendors (ISVs) moving from the on-premises SaaP world to SaaS, one option is to start by deploying a single-tenant version of an existing application. Especially in enterprise markets, where customers commonly demand high levels of trust and customization, single tenancy can be a good choice. This approach also lets the ISV gauge whether there's really a demand for SaaS in this market before investing the time and money required to create a multi-tenant application.

But servicing a large number of customers with a single-tenant solution is likely to be prohibitively expensive. Because of this, creating a multi-tenant SaaS application is most often the right long-term goal for existing ISVs. This is especially true for ISVs hoping to target smaller organizations with a lower-cost SaaS offering. Providing a lower price means having lower expenses, which in turn implies multi-tenancy.

For start-ups and anybody else creating a new SaaS application, starting with multi-tenancy is likely to make the most sense. Since your goal is to have lots of customers, why not design for this from the beginning? ***Unless you know your customers will have very stringent requirements for security or customization, multi-tenancy is the way to go.***

Defining SaaS

Some people believe that the term "SaaS" should be reserved solely for multi-tenant applications. Single-tenant cloud applications, they argue, are really examples of the older application service provider (ASP) model.

Yet customers typically can't tell the difference. To them, an application that runs in the cloud and provides cloud-style pricing (such as per-user/per-month subscriptions) is SaaS, regardless of whether it's single- or multi-tenant. And since creating a single-tenant application is sometimes the right thing to do, including both options under the SaaS umbrella makes sense.

To think clearly about a multi-tenant SaaS application, it's important to understand that a tenant isn't just a billing relationship, i.e., a customer. Each tenant is also a unit of the following:

- Data isolation: Even though many customers are sharing a single application, each one expects its work to be isolated from work done by the others. In general, the most important part of this is keeping their data safe from prying eyes. It's entirely possible that their major competitor is also a user of your application, and they need to feel safe about sharing storage with this tenant.

- Performance: Even though a customer might know that he's using a shared application, he's unlikely to be happy if that application's performance varies significantly from day to day. Building an application that provides predictable performance with varying numbers of users isn't easy, but striving for this goal is nonetheless important.

- Customization: Even though multi-tenant applications don't typically allow modification of the application code itself, they do commonly provide facilities for customizing the app's behavior (a topic discussed in more detail later). This customization must be done on a per-tenant basis.

- Access: Whether the application is accessed through a web browser or via an API, that access must always send users to the right place with the right data. This requires thinking of identity and access control on a per-tenant basis.

Reliability and Scalability

Reliability is important for every application, SaaP or SaaS. Both kinds of applications must also be scalable enough for the load they're expected to bear. With SaaS, however, both of these requirements become much more important.

To see why, look at Figure 1 again. With SaaP, each instance of the application supports users at just one organization. If it fails, only those users are affected. Similarly, the application's scalability requirements are limited by the number of users at its largest customer.

With (multi-tenant) SaaS, however, a single instance supports many users at many companies. If the application fails, all of those users are unhappy. And the application's scalability requirements also go way up—it's now required to handle all of the users at all of the tenants this instance supports. These changes are a big part of why creating a multi-tenant application takes more work.

Providing both reliability and scalability for an application depends on a common architectural approach: multiple instances of everything. Figure 4 illustrates this idea for a Windows Azure application running in a single datacenter.

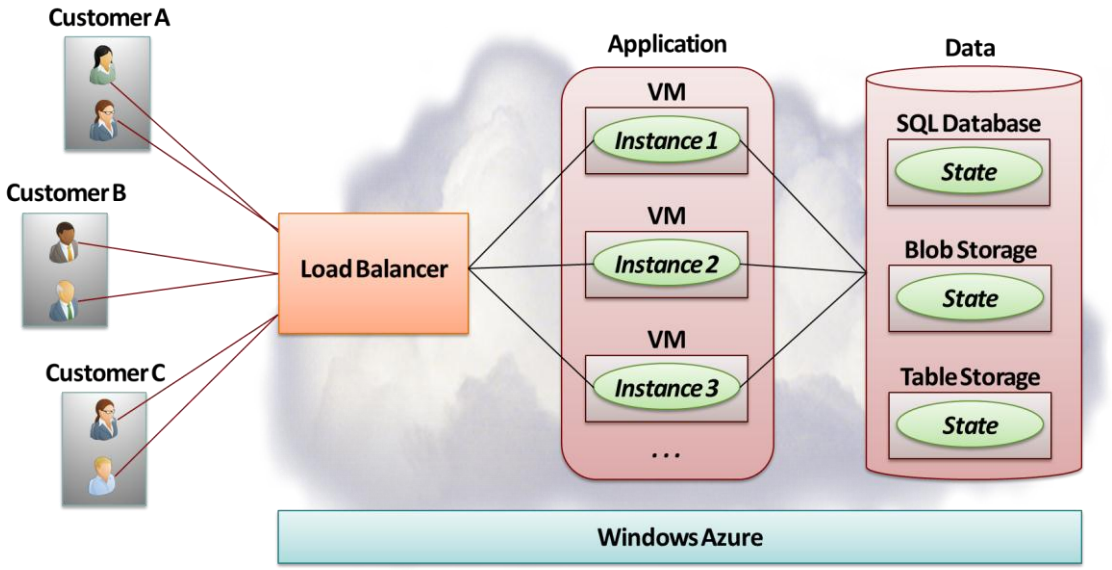

Figure 4: Reliability and scalability both depend on replication.

Whatever Windows Azure execution model option the application uses—Virtual Machines, Web Sites, or Cloud Services—a reliable and scalable application will provide multiple identical instances of its logic running in multiple virtual machines (VMs). Windows Azure provides built-in load balancing that spreads user requests across these VMs. If a VM fails, the load balancer will notice this and stop sending it requests. Once another VM has been created to take its place, the load balancer will begin sending it requests once again. And to handle increased load, the application or its administrator can create new VMs as needed, and the load balancer will once again begin sending them requests automatically. When the load decreases, those VMs can be shut down.

For this to work, the application can't maintain persistent state in any VM. Since any of those VMs can appear or disappear at any time, none of them can be special. As the figure shows, Windows Azure applications should store persistent state in a service designed for this, such as SQL Database, Blob Storage, or Table Storage. All three of these replicate data under the covers, providing their own reliability. (Although it's not shown in the figure, it's also possible to run a database management system, relational or NoSQL, in replicated VMs.)

In some situations, running an application in a single datacenter might be sufficient. A varying size set of VMs running identical software, all storing their persistent state externally, can provide both reliability and scalability. But what if the entire datacenter goes down? And what if your application needs to provide good performance to users spread around the world? Dealing with these challenges requires replicating the application across multiple Windows Azure datacenters. Figure 5 shows how this looks.

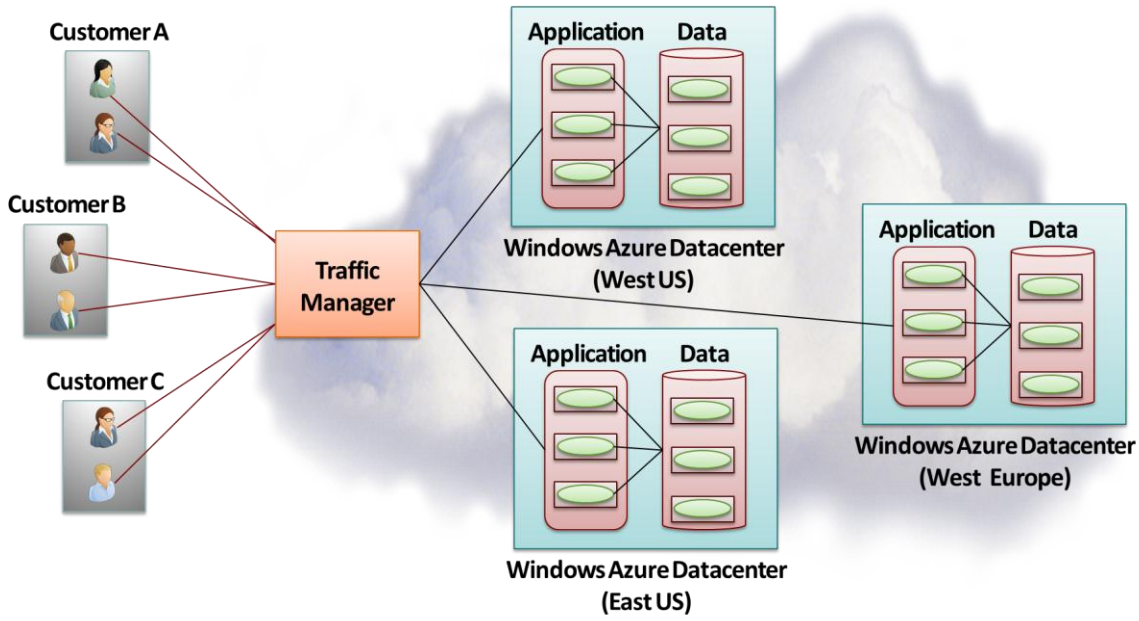

Figure 5: Running an application in multiple datacenters improves its reliability, scalability, and performance.

Windows Azure has datacenters spread throughout the United States, Europe, and Asia. Running an application in two or more locations protects against the complete failure of any single datacenter, increasing the app's reliability. Distributing requests across these datacenters also makes the application able to handle more users, improving its scalability. And since users can now access a copy of the application that's closer to them, they'll see better performance, too.

Windows Azure provides various services to help make this replication easier. To intelligently spread user requests across datacenters, the platform provides Traffic Manager, which acts much like a cross-datacenter load balancer. Another technology, SQL Data Sync, allows synchronizing data across databases in different instances of SQL Database. Similarly, data in Blob Storage and Table Storage is automatically copied to another nearby datacenter to guard against catastrophic failure.

Reliability and scalability must be designed into an application—they can't be afterthoughts. Assume that failures will happen—they will—and plan for the scalability required when your application is wildly successful. Whatever service level agreement (SLA) you offer your customers, recognize that the foundation of that SLA is the right application architecture.

Evolving Your Design

It's a safe bet that your first approach to a new application won't be optimal. As you run the application and gain experience with how customers use it, you'll learn more about how it should look. Because of this, expect to evolve your design over time.

Part of this evolution should be a relentless focus on lowering your costs. Competitive price pressure will probably force this on you, so dedicating resources here is important. Because cloud software commonly offers free trials, many SaaS applications have lots of people trying the application without paying (something that's especially true for applications that use freemium as a marketing strategy). Keeping the costs of running your SaaS app as low as possible is essential; continuing to optimize your design is a fundamental part of this.

Security

The biggest challenge most potential SaaS customers have today is convincing themselves that the public cloud is secure. Can they really trust an application running in some far away datacenter? Is the risk worth taking? Given this reality, making your SaaS application secure is critically important.

It's useful to think about this issue in two parts: datacenter security and application security. With a SaaS application built on Windows Azure, Microsoft is responsible for the first part. This includes ensuring the physical security of the machines your application runs on and guarding against network attacks to those systems. Ensuring the security of your application is primarily your responsibility, however. If you don't implement effective identity and access control mechanisms, for example, your application will still be insecure.

To help do this, the platform provides Windows Azure Active Directory. Despite its name, this technology is not a cloud version of Windows Server Active Directory. Instead, it's designed expressly to be used by SaaS applications. Among the services it offers are:

- A place to store information about users and the organizations they belong to.
- A way for users to log into the directory, then get a token they can use to authenticate themselves to SaaS applications.
- A mechanism that lets users log in using identities issued by other identity providers, such as Facebook and Google.
- Support for connecting Windows Azure Active Directory to Windows Server Active Directory, letting organizations use their existing on-premises identities with SaaS applications.

Like reliability and scalability, ***security must be designed into an application—it can't be an afterthought***. Being successful here requires understanding and exploiting what Windows Azure offers.

Metrics

Just about every application keeps track of some user metrics. Server applications maintain logs, and even desktop applications like Microsoft Word maintain a record of who's worked on a document, how much time they've spent, and more. With SaaS applications, both the necessity and the ability to track users increase significantly. ***Thinking about the metrics your application should track is an important part of the design process.***

Information you should consider monitoring for your SaaS application includes the following:

- The metrics you're using to price your application. These might be simple, such the number of current users from each customer. Depending on your pricing structure, though, they can also be quite complex. A SaaS application can price based on pretty much anything, including detailed measurements of customer usage: CPU, storage, or more abstract things that are specific to your application. Whatever pricing approach you take, making sure that you provide an accurate and reliable way to measure and track these elements is essential.

- Usage metrics for each customer. A SaaS application lets you see in detail how each user at each customer is using the application. This information can be incredibly valuable. Which customers aren't using your product much and so need more sales attention before their current subscription expires? Which parts of the application are most popular? Which parts are least used, and by which people at which customers? Tracking these things can help you understand what features customers really value. If you provide on-line help, you can even monitor which pages get accessed most often, letting you see what parts of your application are hardest to use.

- Aggregated usage metrics across all customers. Maintaining a data warehouse lets you perform ongoing analysis of usage patterns. It's significantly easier to do this with a SaaS application than it is with a set of on-premises apps installed in your customers' datacenters, since all of this data is immediately accessible to you.

- Performance and availability metrics. This includes traditional logging information, but it should also go beyond this to monitor behavior at each tier of your application. Among other things, this kind of monitoring can act as an early-warning system, letting you detect (and correct) problems before your application crashes.

Collecting all of this information makes sense, but where should your application store it? On Windows Azure, one popular choice for metrics of all kinds is Table Storage. It's relatively cheap, very scalable (up to a terabyte per table), and easy to query in simple ways. Blobs can also be useful, especially for unstructured binary data. And to help analyze large amounts of unstructured data, Windows Azure provides Hadoop. This service can be used through tools such as Hive or accessed via Microsoft Excel.

Design for Operations

With traditional on-premises SaaP, the customer runs the product in its own datacenter. ***With SaaS, the vendor is responsible for running the software.*** This places a significant burden on SaaS providers that doesn't exist in the SaaP world.

To meet this burden, SaaS applications need to be designed for operations. An important part of this is monitoring, as just described. An equally important part is creating an organization that knows what to do with monitoring information and can respond appropriately. To do this, ***SaaS organizations commonly adopt a devops approach, fielding a unified team with both application developers and operations staff***.

To see why this is so important, think about what happens when the continuous monitoring of business logic built into your SaaS application shows that a problem is developing. Detecting and fixing this problem before the application crashes—an important thing to do when a failure impacts so many customers—requires both operations skills and development skills. Creating a group that's good at doing this is important, as is building a SaaS application that provides the information this group needs.

Another aspect of operations is rolling out updates to your running application. Taking the entire app down while deploying a new version is rarely a workable solution, since some customer somewhere always wants to use the software. To address this problem, Windows Azure provides two options: a rolling update that deploys new code to different groups of VMs at a time, or an option that lets you deploy a new version of an application in a staging area, then switch atomically from old code to new. Both approaches let you update a running SaaS application with no downtime.

Portability

By definition, a SaaS application runs in the cloud. Yet it's common for some customers to insist on running the software in their own datacenter. Maybe they have regulatory requirements that mandate this, for example, or maybe they just don't yet trust the public cloud. Whatever the reason, ***quite a few software companies find that they need to create a SaaS application that can also run on premises***.

With a SaaS application built on Windows Azure, this means building the application so that it can also run on Windows Server. Figure 6 illustrates this idea.

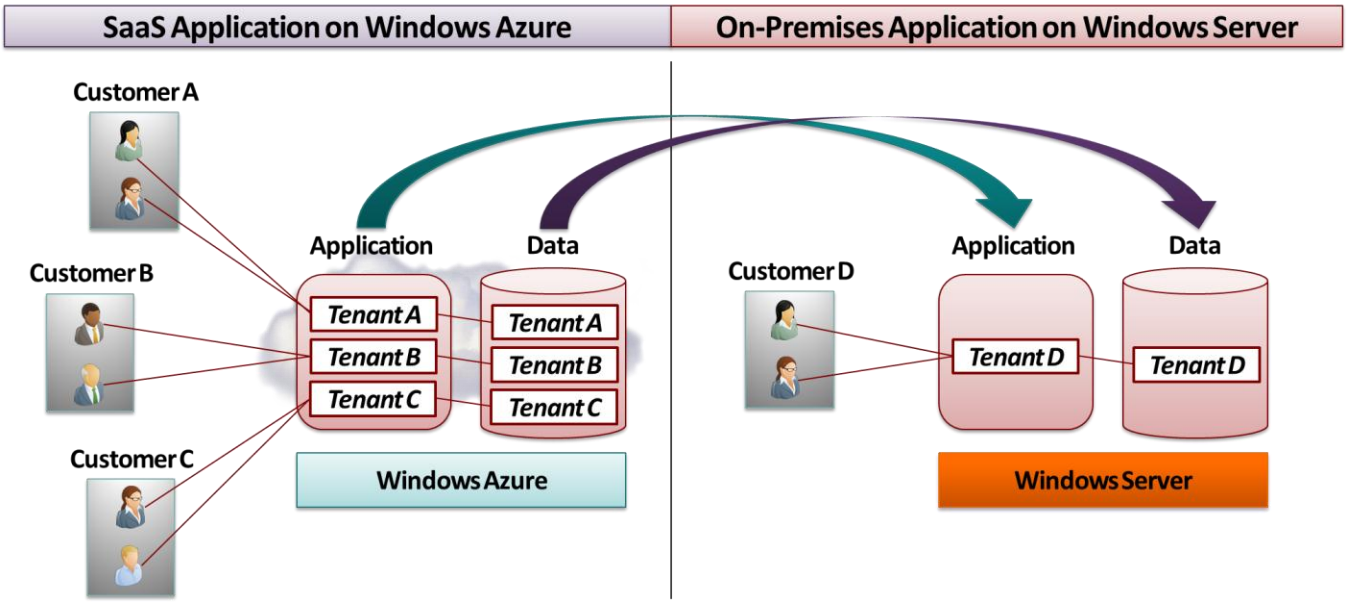

Figure 6: It's possible to build a SaaS application for Windows Azure that also runs on Windows Server.

As the figure shows, an on-premises version of a multi-tenant SaaS application might use just a single tenant. This isn't necessarily true, however. Some customers, especially larger organizations, might want to use separate tenants for different parts of the organization. Whatever choice the customer makes, building a Windows Azure

application that also runs virtually unchanged on Windows Server requires making the right design choices up front.

In general, this isn't especially difficult: All three of the Windows Azure execution models can support standard Windows software. The choices are:

- Virtual Machines, providing Infrastructure as a Service (IaaS). These cloud VMs run ordinary VHDs, and so moving applications between Windows Azure Virtual Machines and on-premises VMs is straightforward. While a SaaS application's architecture might be designed to provide more scalability than is required for an on-premises deployment, this doesn't constrain the app from running in customer datacenters.
- Web Sites, supporting IIS web sites and applications. Software built on this foundation can typically run unchanged on Windows Server.
- Cloud Services, offering Platform as a Service (PaaS). This execution model isn't based on VHDs, and so moving a Cloud Services application between Windows Azure and Windows Server requires moving code rather than VM images. Nonetheless, these applications are Windows applications—they can run in both places.

Whatever execution model you choose, an application that must run on both Windows Azure and Windows Server should avoid using Windows Azure-specific services such as Table Storage or Windows Azure Queues. Since there's no on-premises analog to these cloud technologies, making the application run on Windows Server will be problematic. These considerations also apply if you're trying to create a SaaS application that can run on other cloud platforms.

APIs

A truly successful application isn't just an app—it's also a platform for other software to use. This is true for on-premises SaaP applications, and it's even more true for SaaS applications. Since a SaaS application runs in the cloud, it can potentially be accessed not just by other software inside a single organization, as is typically the case with SaaP, but by software running anywhere, including mobile devices. To allow this, your SaaS application must expose one or more application programming interfaces (APIs), as Figure 7 shows.

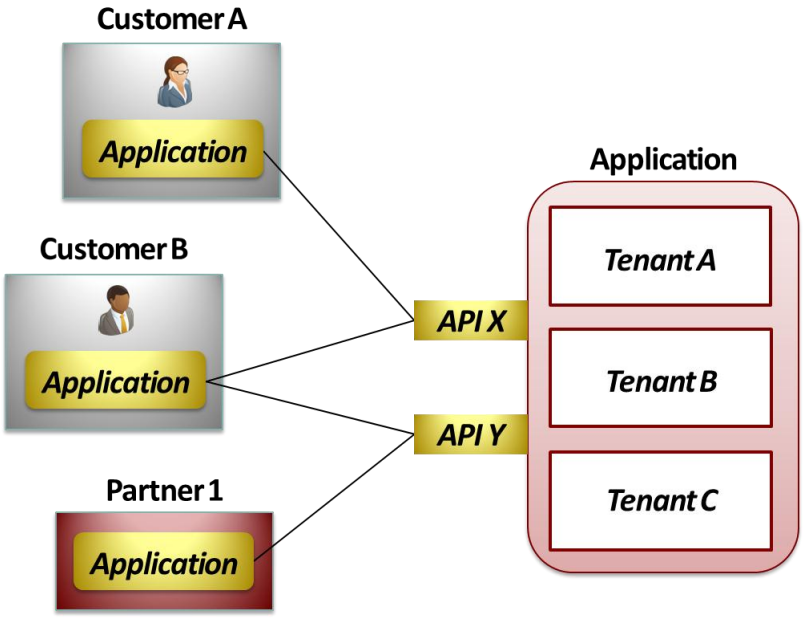

Figure 7: APIs let SaaS applications become platforms.

As with so much else, it's important to think about these APIs before you start building your SaaS application. What services will you want to expose to others? Which functions do you want to provide yourself, and which are better left to partners? What do you want your application ecosystem to look like? These decisions flow from your business strategy, but the implementation of those decisions is embedded in your technology.

Implementing a SaaS application's APIs requires some thought. If your application is multi-tenant, for instance, your APIs must reflect this, letting external applications interact with the right customers. You also need to think about the technical approach that's best for your market. It's common to create custom RESTful APIs that exchange data using standard formats such as JSON or XML, but this isn't always the best choice. There might be existing APIs you can implement instead, such as OData, that are already supported by other products and tools.

One more concern is scalability. An application that's capable of handling requests from people using web browsers might not be scalable enough to handle requests from other software through APIs. Software can issue requests much faster than people, and so you might need to implement some kind of throttling mechanism to limit how many API calls any single application can make in a given period of time.

Whatever technical choice you make, thinking about your application as a platform makes sense. One clear way to create SaaS software with staying power is by making it a foundation for innovation by others. To do this, you need to provide the right APIs.

Customization

In a single-tenant SaaS application, customization can be straightforward: Each customer has its own copy of the software. But even ***customers of multi-tenant SaaS applications often want to modify the behavior they see***. To accommodate this, you need to think about customization when you're designing the application. Figure 8 illustrates what's needed.

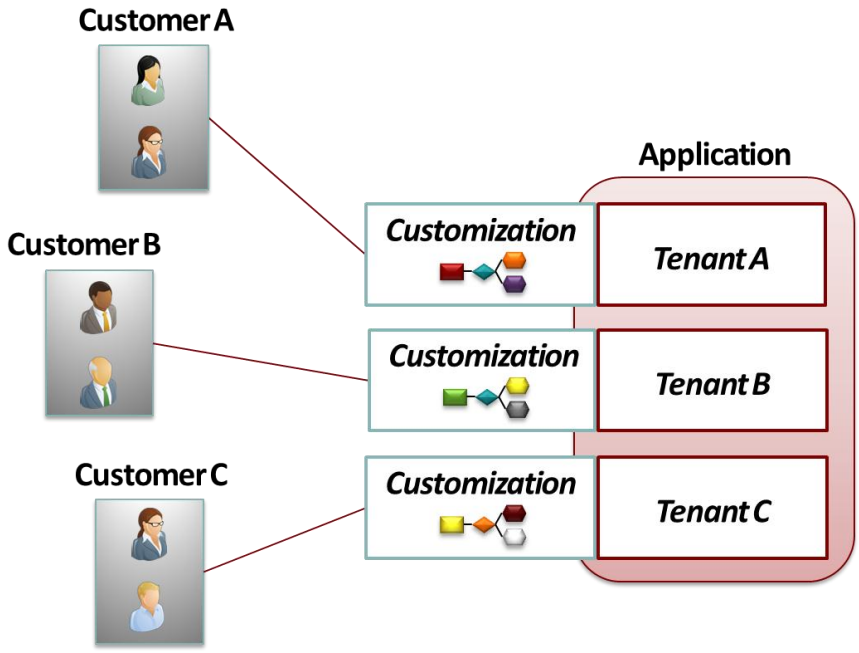

Figure 8: Multi-tenant SaaS applications commonly require some way for customers to customize their behavior.

Depending on what their customers need, different SaaS applications take different approaches to providing customization. Here are some of the options:

- Parameterization: Perhaps the simplest approach is to figure out what aspects of the app customers most want to configure, then expose per-tenant parameters that let them set these things.
- Metadata: A more general approach is to expand the set of parameters into full-fledged metadata, providing a way to control much more of the application's behavior. This metadata might include workflow definitions, letting each customer define its own processes. With a SaaS application built on Windows Azure, this approach can rely on Windows Workflow Foundation. An ambitious SaaS vendor could even create a domain-specific language for its customers to use.
- A development platform: Some SaaS applications offer their own accompanying platform for creating custom applications. For example, Microsoft CRM Online provides xRM, while Salesforce.com CRM provides Force.com. A SaaS application built on Windows Azure probably doesn't need to go this far—customers can instead write their own Windows Azure software that uses the SaaS application's exposed APIs. Still, this option can make sense in some cases.

Providing customizability for your SaaS application can get complicated, and it's not without costs. For instance, if a customer extensively modifies your app's behavior through metadata, you might see some performance degradation. More customization also implies more configuration or code that you can't break when you upgrade your application. It's fair to say that ***the more customization your application allows, the more complexity you'll have to deal with in both implementation and operations***.

Online Marketplaces

By definition, potential SaaS customers are online. This implies that they're likely to use the web to find new SaaS applications. One way they can do this is by searching online marketplaces, and so creating a SaaS application that works well with these marketplaces makes sense.

Multiple marketplaces exist today, each with its own requirements for how your SaaS application must interact with it. With the Windows Azure Marketplace, for instance, your application needs to be registered, then be able to create a subscription based on a message that the Marketplace software sends. You also need to accept and process an unsubscribe message sent when a user ends a subscription through the Marketplace. Understanding the requirements of the online marketplaces you plan to use and architecting your SaaS application to support them will avoid rework later on.

Other Issues

Everything discussed so far is important, but it's not an exhaustive list. Other things you might want to think about before building a SaaS application include the following:

- The ability to let different customers of a multi-tenant application see different versions of the software. Rather than guess what users prefer, SaaS applications commonly run tests. Some customers see one version of a new feature while others see a different version. This kind of A/B testing provides clear data about customer preferences, and so it's useful in deciding what path your application should take. To make this easy to do, an application can provide built-in support for deploying, then removing, new features to a limited set of customers. This kind of multi-version support should be possible in various tiers, not just the user interface. This functionality can also be useful for updating different instances of your application (and thus different customers) at different times. If an update requires retraining users, for example, some customers might prefer not to get it immediately.

- The right user interface (UI). The UI is important for every application, but SaaS makes this fact worth emphasizing. Here's why: with an on-premises application, you know that you can't change the UI until the next release, which is likely a year or more away. As a result, development teams commonly spend more time designing and building an effective UI for the app. With a SaaS application, however, the team can change the code much more frequently. This escape hatch makes it easier to skip serious UI design. Competitive pressures might be forcing a rapid schedule, for instance, and since it's a SaaS application, the team can always fix this later. Yet changing the user interface is no small thing—for one thing, it can annoy your users—so putting off the need to think through what the UI should look like is a false economy.

- Social software: Features such as support for user-to-user interaction can be built into most applications, SaaP or SaaS. With SaaS, however, multiple users at multiple customers can share these social features. And because the application runs in the cloud, it's accessible from anywhere. Given these realities, thinking more about social features for a SaaS application can make sense.

Conclusion

Building a new application requires making many decisions. With SaaS, that set of decisions expands, adding a number of new things to think about. The list includes:

- Whether your application should be ***single-tenant*** or ***multi-tenant***.
- How best to achieve ***reliability and scalability***, both in a single datacenter and across multiple datacenters.
- How you'll handle ***security***, including identity and access control.
- What ***metrics*** your application should track, along with how you should store and process this information.
- What ***design for operations*** means for your application, and what the application should provide for a devops team.
- Whether you need to worry about ***portability***, and if so, how that affects the design choices you make.
- Which of its functions your application should expose through ***APIs***.
- How much ***customization*** your application should allow and how it should provide that customization.
- What ***online marketplaces*** you'll support and what that support requires of your application.

This checklist isn't exhaustive—there are sure to be other important things that are unique to your situation. Still, ***everybody creating a new SaaS application should make sure they consider each of these issues up front***. Providing software as a service really is a new world. We need to make sure that we don't approach it in the same old way.

About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technologies.