Converting your web application into a multi-tenant SaaS solution

A checklist of considerations and steps to quickly turn your web app into a cloud application

Summary: You've built a single-tenant web-enabled application, but need to make it compatible with and effective in a cloud environment. What steps do you need to take to convert your application to a full-fledged, multi-tenant, cloud-ready SaaS application? The authors take a sample web application, discuss the necessary considerations and alterations to make it a cloud winner, and outline the steps you need to take to get it there. Then, as a bonus, they demonstrate the software they've designed to provide a "plug-in" approach to multi-tenancy.

Imagine you have a web application that you've been selling in the marketplace. You see the writing on the wall and realize that SaaS (Software as a Service) on a cloud infrastructure is the way the industry is headed. You know you need it and your own customers may already be pressuring you to offer a SaaS version your product.

The challenge is to do the conversion to SaaS quickly, efficiently, and in a way that will maintain or enhance your profitability.

There are many differences that must be taken into account for a SaaS application versus a regular web application. Some of these are technical and some are related to the change in business model that a company must adapt to when delivering SaaS.

Typical web app vs. SaaS

SaaS has as its central defining characteristic the ability for customers to use a software application on a pay-as-you-go subscription basis. They don't have to acquire the license for the software and arrange for it to be installed, hosted, and managed. Those operational aspects are the responsibility of the organization providing the SaaS application.

Multi-tenancy is the key to successful SaaS

Although the capability to subscribe to an application is minimally enough to satisfy the basic criteria of SaaS, in practical terms it falls short. In practice, **a SaaS application must also be a multi-tenant application**.

This has to do with factors which come down to being simply economic. The CEO's of leading SaaS companies agree, it is not possible to grow a SaaS business without multi-tenancy.

> "Multi-tenancy is a requirement for a SaaS vendor to be successful" - Marc Benioff, CEO, Salesforce
>
> "It's impossible to be successful is SaaS without multi-tenancy." – Treb Ryan, CEO , OpSource
>
> "We have achieved a phenomenally effective *(low)* cost of seat for computing…we have the lowest cost in the SaaS industry" *(as a result of multi-tenancy)* – Lars Dalgaard, CEO, SuccessFactors
>
> "Today ASP is attempting a comeback. However, the cloud data center approach still segregates customers using a *single tenant model* meaning it's still *very expensive to operate* and therefore to buy" – Zach Neson, CEO, NetSuite
>
> "We have the lowest cost in the SaaS industry" (because of multi-tenancy) – Lars Dalgaard, CEO, SuccessFactors
>
> " Multi-tenancy to us is really important…creates a much more efficient model" – Tim Wallace, CEO, iPipeline
>
> "Our gross margin went over 70 percent (due to multi-tenancy) " - Zach Neson, CEO, NetSuite
>
> "Multi-Tenancy is table stakes in the SaaS game." – Henry Olson, former CEO, Edge Dynamics
>
> "Multi-Tenancy enables the whole subscription model to work…" Jeff Kaplan, CEO, THINKStrategies

Multi-Tenancy is the Efficiency Lever for SaaS

The major leverage for efficiency comes from multi-tenancy, the ability to accommodate different users of the application while making it appear to each that they have the application

all to themselves. We are used to this concept as it applies to individuals users on a system, but its slightly different for a SaaS environment. In a typical enterprise SaaS application, the ***users*** are groups of employees of a particular organization and that organization is known as a ***tenant***. This is similar to what you would find in a simple web application if the organization purchased the application; they would have a group of employees who were users of the application and the organization would be the ***owner***. In a SaaS model, the organization is a tenant, not an owner, but the groups of employees are still users. Each user has an association to a particular tenant (organization) and SaaS provides each tenant with the experience of owning their own copy of the application, which their users can use.

Virtualization in the Cloud leverages SaaS

The difference between a simple web application and a cloud-enabled SaaS application encompasses two of the biggest capacity utilization features in IT today:

- Multi-tenancy (which was introduced above).
- Virtualization of hardware.

While the major source of application efficiency comes from multi-tenancy of the application architecture, the secondary leverage for efficiency comes from virtualization of hardware. A cloud does a better job of leveraging the value by increasing utilization percentages from a given amount of hardware by employing virtualization technology to reduce the unused capacity that results when an ordinary data center physical machine approach is used.

In addition the cloud offers the potential to re-allocate the hardware dynamically to the application based on its need for resources. This elasticity -- which can be over a short term (minutes) or a long term (months) -- helps to de-couple the decisions about hardware from one single application and spread them out over a large number of applications, smoothing out the variances and making the investment in hardware more predictable and manageable.

Now let’s look at the general steps you’d take to convert a more traditional web application to a SaaS-enabled one.

Steps to convert web apps to SaaS

To convert your web application to a SaaS application you have to do seven things:

1. The application must support multi-tenancy.
2. The application must have some level of self-service sign-up.
3. There must be a subscription/billing mechanism in place.
4. The application must be able to scale efficiently.
5. There must be functions in place to monitor, configure, and manage the application and tenants.
6. There must be a mechanism in place to support unique user identification and authentication.
7. There must be a mechanism in place to support some level of customization for each tenant.

Let’s look at each in a little more detail.

Support multi-tenancy

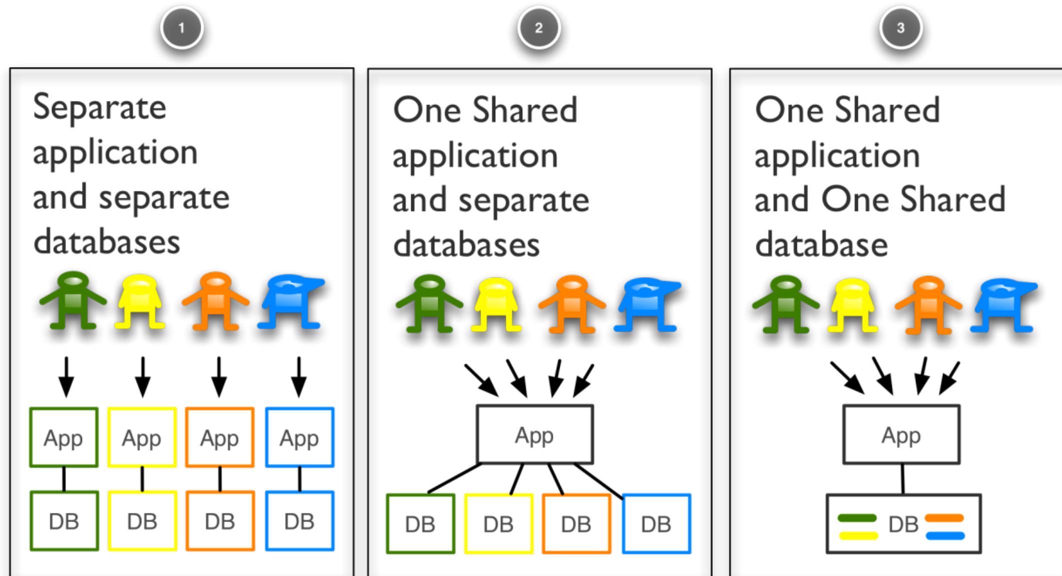
Multi-tenancy is the key determinate of SaaS efficiency. Typically an application supports multiple users, but with the presumption that all the users are from one organization. This model is fine for the pre-SaaS world where an organization would purchase a software application for use by its members. But in the world of SaaS and the cloud, many organizations will be using the application; they must all be able to allow all their users to access it, but the application must allow only each organization’s own members to access the data for their organization.

This capability to have multiple organizations (called tenants in the SaaS nomenclature), co-exist on the same application without compromising the security of data for those organizations defines the application as a multi-tenant one.

There are several levels of multi-tenancy:

1 – Simple Virtualization in the Cloud where only the hardware is shared
2 – Single application with separate databases per Tenant
3 – Single application and Shared database (highest efficiency, true multi-tenancy)

Multi-Tenancy Models



It can be argued that the first model is not really multi-tenancy at all, but it is often used in a Cloud with virtualized servers and promoted as a form of multi-tenancy. In reality it falls short and has only marginal advantages over the old ASP model with dedicated hardware.

The most efficient level is one in which the application fully shares a database and application business logic. Achieving this can be an onerous process that requires changes to the database schema to add a **tenant identifier** to every table and view, as well as a rewrite of every SQL access to add the tenant criteria to the filters. Missing one place in the code where this needs to occur can compromise the data security of the application.

In addition to the instances of data access that occur in the application, there may be other applications such as report writers or utility applications which must also be modified to include the tenant filtering necessary to keep the data of individual tenants accessible only to those particular tenants. The type of access that is not directly from the application itself presents problems that must be controlled. If any authorized user can write a report, they must be prevented from accessing data that does not belong to the tenant of which they are part.

Self-service sign-up

Your application must be available with some level of self-service sign-up, even if it is simply a request mechanism that results in a business process to add a tenant to the application.

Subscription and billing

You must provide a subscription and billing mechanism. Since SaaS applications by design involve a series of payments based on factors like number of users per tenant, application options, and perhaps usage duration, there must be a way to track and manage the application use and generate billing information that is accessible to the tenant administrators.

Scaling and managing the application

You must have the ability to scale up as subscriptions grow. The cloud infrastructure is a logical way to do this since it embodies many capabilities you will need for effective, efficient scaling.

Also, you must provide administration and application management functionality to monitor, configure, and manage the application and all the tenants.

User ID and authentication

You need to provide a mechanism to support user identification and authentication that allows for the unique identification of users. Because multi-tenancy requires that all the users that sign-on to the system be identified to determine which tenant they belong to, there has to be a definitive relationship that allows for users to be identified as belonging to a particular tenant. That user-to-tenant relationship is the key information that is used to restrict the data that can be accessed by the user.

Email addresses are a typical way of doing this so that uniqueness is assured and individuals can be recognized and identified as belonging to a particular tenant.

There are many authentication mechanisms and methods of integration with them, so a flexible mechanism for allowing a user to be identified is essential. It is often necessary that a particular tenant be able to utilize their existing LDAP or other directory service or authentication mechanism to support single sign-on to the SaaS application. Although this type of external authentication of the user is important, it’s the responsibility of the SaaS application to establish that the identified user is a member of the tenant they claim.

Per-tenant customization

You must provide a mechanism to support a level of basic customization for each tenant so that they can have a unique URL, landing page, logos, color scheme, fonts, and perhaps even language.

This basic level of per-tenant configuration is expected, but to truly meet the needs of multiple tenants, there will inevitably be a need for per-tenant customization that goes deeper than the basics.

Typical customizations required are similar to the type of customizations that would be done by a tenant with an in-house version of the application. They can involve adding fields or even tables, setting up special business logic, or integration with another application. Being able to do these types of customizations on a per-tenant basis without having to establish a separate instance that compromises the efficiency of a multi-tenant design is the hallmark of high-capability SaaS architecture.

Performance issues to consider

Performance issues for multi-tenant SaaS applications are typically the same as would be found for any web application accommodating the same number of users with the same level of activity. There are many best practices for dealing with increasing capacity demands in web applications; in general they are all applicable to multi-tenant SaaS applications as well. These techniques typically involve **horizontal and vertical scaling** and **load balancing** across a set of servers.

Horizontal/vertical scaling

Cloud infrastructures capabilities offer many opportunities to make this type of scalability happen dynamically and in an automated fashion, to provide the resources when needed and to scale back resources when performance **service level agreements** (SLAs) can be met with fewer resources. This elastic capability is something that can be tuned to respond in precisely the manner needed to provide the service without providing resources that will be underutilized:

- Horizontal scaling is typically employed for the application server tier.
- Vertical scaling is typically employed for the database tier.

Database clustering

With SaaS applications, the total number of users can be very high for a successful product; at some point vertical scaling of the database may not be the optimal solution. Many database technologies have the ability to provide a clustered database model that allows more capacity to be provided against the same database. DB2™ has several options that work well with this model and these can be used to create a database cluster.

Geography, partitioning, and synchronization

However there are other techniques that may be more appropriate depending on the SaaS application and its population of users. Since SaaS is inherently accessible from anywhere in the world, the population of users may be far away; this distance can cause performance degradations due to the long network topologies.

In these cases it may be advantageous to use clouds that are in different regions and either partition the data or use synchronization to maintain consistency. Which of these options is appropriate will depend on the nature of the particular application. Some will not be amenable to long-distance synchronizations.

The dark side: Separate databases

Of course the most radical method (for a SaaS application at least) to use when the database capacity cannot meet the demands is to establish a separate database. Anyone who wants a multi-tenant SaaS application has to consider that going to the dark side can lead to the untenable situation of supporting a database per tenant, which leads directly to the type of inefficiencies multi-tenancy strives to avoid.

And if the application servers must be divided up to service only one database, then the efficiencies of multi-tenancy are further eroded. In a competitive market, those efficiencies of multi-tenancy are critical success factors for SaaS companies.

Rescuing the Power of Load Balancing

One way to retain as much of the efficiency as possible even when, for whatever reason, separate databases must be used, is to have a multi-tenancy design which can allow any of the application servers in the load-balanced cluster to access the appropriate one when there are multiple databases. In this way the load balanced cluster efficiency can be maintained with all application servers being able to connect to any database for the User sessions they are supporting. This maintains the greatest level of efficiency within the constraint of multiple databases.

Capacity is not the only requirement

There can be legitimate reasons to have multiple databases besides just for the capacity, such as the demand for an encrypted version of the database for certain high-security tenants.

Capacity may not be the issue, so it is important to have a design that maximizes the efficiencies where possible, even when an inherently less efficient model is needed.

Security issues to consider

In survey after survey, security is usually listed as either the top concern for subscribers to SaaS applications; at least it is very close to the top. No SaaS provider can ignore security. But often, the concept of security of data is considered only within the context of the SaaS application itself.

Most SaaS application architectures take data security measures that prevent one tenant from seeing another tenant's data as a baseline requirement. But:

- A key capability that SaaS applications must have is to integrate and interact with other applications.
- Some of those other applications may be outside applications (not controlled by the SaaS provider).
- Not all SaaS architectures are designed with accessibility for outside applications in mind.

Those other applications could be in-house applications that need to access or share data; they could be analytic and report-writing tools that mine the data for trends. Even utility tools used by database administrators can be security concerns if the tenants can use them to access, or worse yet, manipulate data that does not belong to them.

A best practice architecture for SaaS must consider that not all the access to data may be under the control of the application; there must be mechanisms in place to allow the data to be secured for each tenant no matter if the access is through the SaaS application or through some external application.

Select your technology stack

There are always tradeoffs when you make decisions about technology stacks; in a SaaS application this is especially true because the decisions are being made for all tenants. Let's examine some of these considerations.

Operating system considerations

The operating system in web applications is probably the least relevant to the users because their interaction is through the browser. However there are financial as well as technical considerations that may come into play:

- If there is a dependency on particular code which is OS-dependent then the choices are constrained.
- They may also be constrained if there is a common need to integrate with outside applications and that is better done on one OS than another.

The relentless economics of the cloud will always push the choice towards an OS that has lower license fees and good performance. The primary means of scaling web applications involves horizontal scaling; this means that as the SaaS application grows, the total number of web application server instances will increase and that is a direct cost of operations.

Database considerations

The database in web applications is probably the not a critical concern of end users either because their interaction is through the browser and as long as the application is capable of storing and retrieving their data, it is largely irrelevant to them.

Again, financial and technical considerations come into play for the application developer. If there is application dependency on particular features of a database, then the choices are constrained.

The choice of database can be important for a number of application design reasons and it can also be affected because of the particular demands of the SaaS environment.

The demands on a database are higher in a SaaS application simply due to the number of users who will ultimately be on board; so database scalability is very important. The database scalability is usually done on a single instance with more and more powerful database servers used for an increasingly demanding application. But the scalability required of SaaS applications has the potential to outstrip the capabilities of vertical scalability, so the next step in database scalability involves having a clustering capability.

The ability to do this type of clustering in the cloud environment may influence the choice of database. For instance, DB2 may be selected for its ability to run on a wide variety of OS that provide vertical scalability and for its flexibility of choices for scalability through multiple-instance clustering and redundancy. For example, a DB2 HADR (High Availability Disaster Recovery) cluster can be set up in the cloud.

Application server considerations

The choice of application server, like the other technology stack decisions, is also primarily a decision of the SaaS application developer since the end user interactions are only through the browser. But there can be important differences for a number of application design reasons and it can also be affected because of the particular demands of the SaaS environment.

Applications which take advantage of special features or add-on components of a specific application server, will need to use that specific application server in the cloud.

The rationales for choosing an application server are usually made at an early point in the application life cycle because the application can benefit from some particular features or there are third party add-ons or integration capabilities that are important to the application. Sometimes it is simply because the organization’s expertise and internal standards dictate the use of particular components of the technology stack.

The decision points in selecting and configuring the technology stack to use in a SaaS/cloud environment involve balancing the technical and economic forces to achieve a good result.

Flexibility is key to technology evolution

Having the ability to make decisions and tradeoffs regarding the technology stack to use and the ability to revisit those decisions later is a valuable capability in a SaaS architecture. As technology changes, the leading cloud providers will incorporate new middleware applications and capabilities and it’s an advantage to be able to be able to adopt them for your SaaS application.

SaaS application architectures which force you to make decisions that lock you into a particular technology stack will compromise the ability of a SaaS application to evolve and adapt. The key concepts of a **Service Oriented Architecture** are ideally suited to providing the agility and flexibility necessary for SaaS applications. Loose coupling provides flexibility and flexibility enables strategic and tactical evolution.

In the remainder of this article, I’ll be using Corent’s Multi-Tenant Server™ to provide an example of how you can convert a Java™ open source billing web application into a multi-tenant SaaS version with minimal effort.

Automatically SaaS-ify an app with Corent's Multi-Tenant Server

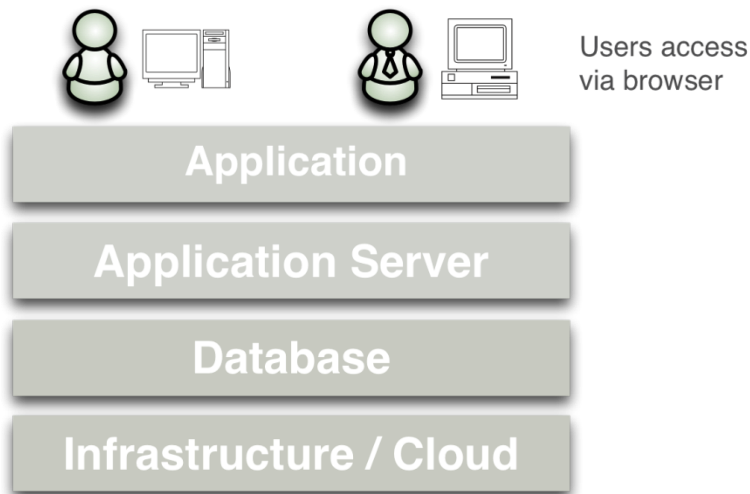
Cloud providers offer a wide array of capabilities and can accommodate almost any web application. To transform an application into a fully multi-tenant SaaS application usually requires extensive changes to the application code and a redesign and reconfiguration of the database.

Corent's Multi-Tenant Server enables ISVs to take a different approach, using a middleware layer to provide the critical multi-tenancy (thereby preserving the investment in the existing code). I'll describe the four steps necessary to converting jBilling, a popular, open source, Java web application that is representative of a typical single-tenant application, into a multi-tenant SaaS version.

Step 1: Transform database schema to an abstract model

The typical stack for a web application looks like Figure 1; an application communicates to a database, often through a data persistence layer such as Hibernate.

Figure 1. Typical web application stack in the cloud

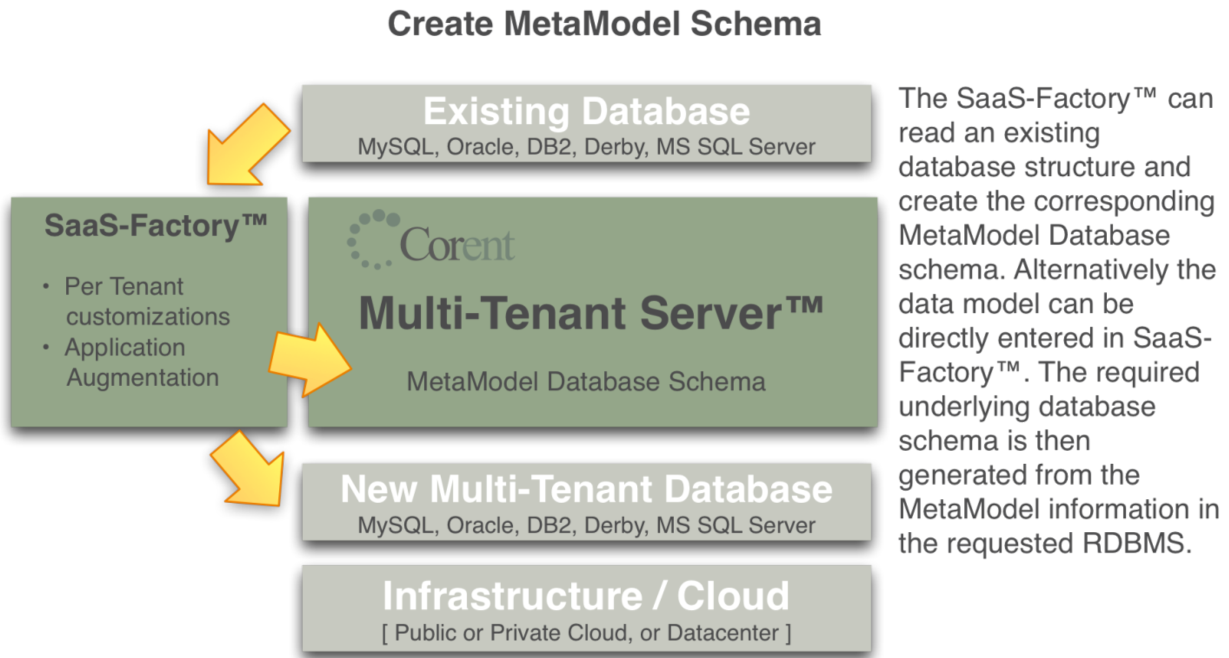


To transform an application into a multi-tenant application, the database has to be redesigned to have additional fields to manage the tenant identification data that will be needed to enable the data to be filtered by the tenant.

The SaaS-Factory<trade/> application is used to read the schema of the existing application database. It then creates a model of that database which is subsequently used to generate a

new database in MySQL that has the additional `TenantID` field in the tables. At this time several additional tables, including one for tenant information, that are used by the Multi-Tenant Server are created.

Figure 2. Creating the MetaModel Database schema



Because the MetaModel Database looks exactly the same to the original application with the same tables and fields, the original application can continue to interact with the MetaModel Database exactly the same as it did with the MySQL database. The real database underlying the model can be generated with the additional `TenantID` fields needed for a multi-tenant application.

The MetaModel Database is an abstraction only and does not actually hold any data: It is simply a model. Consequently, when the real database is generated, there is no reason it can't be generated in any of the supported RDBMS. This can be useful for cases where the ISV wished to change the technology stack by choosing a different RDBMS, perhaps to take advantage of some special features or achieve better performance for their application.

Step 2: Extend the user authentication process

Any multi-tenant SaaS application must have the ability to manage the necessary session information for authenticated users so that the tenant to which they belong can be established. There are many authentication methods for applications; consequently any application being converted must be analyzed and its methods of authentication understood.

In Step 1 I showed you how we establish a tenant table and added a user table so that this relationship can be maintained in the application data. The goal is to implement a method that will extend the application's user authentication process so that once a user is logged on, he is also matched with his corresponding tenant and that `TenantID` becomes part of the session information.

There are many ways to accomplish this; it depends on the implementation of the application. If Tomcat is used as the application servlet container, then the container-managed authentication can be leveraged to initiate a business rule that runs in the Multi-Tenant Server to perform a lookup of the user's associated `TenantID`. Alternatively, a servlet filter can be employed to detect and then set the thread local variable `TenantID`. For this example with jBilling the application handles the authentication directly by validating against its user tables. Therefore the method used to provide the enhanced authentication involved some simple changes to the authentication code to add in the processes to lookup the user's Tenant information stored in the new tables.

By having an authenticated user and knowing the `TenantID` to which they belong, there is sufficient information to be able to filter the data so that only data belonging to that tenant can be accessed.

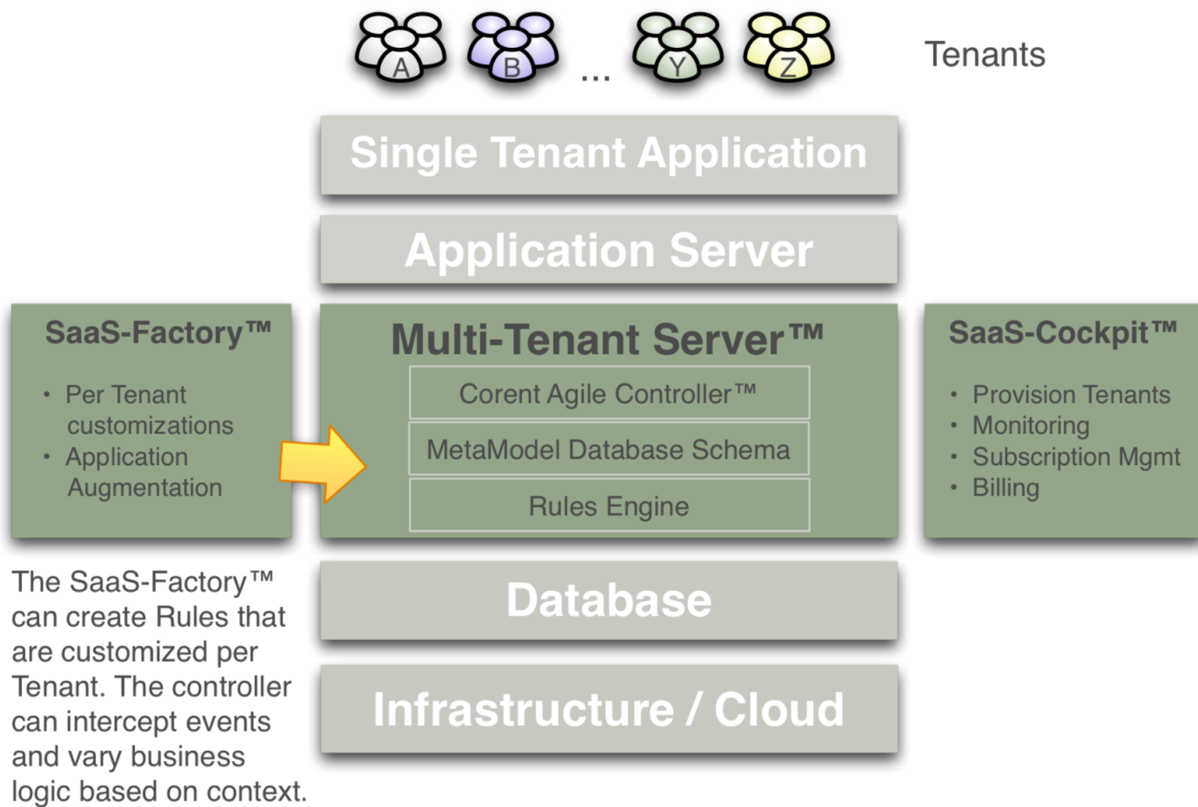
Step 3: Configure the database connection

The Corent Multi-Tenant Server is built on a service oriented architecture that uses a MetaModel Database (I discussed this back in Step 1). This MetaModel is used as an abstraction layer to model the original database of the application and the application's database communications are redirected to the MetaModel abstraction instead of the actual database implementation. This is done by simply reconfiguring jBilling's JDBC connection to access the MetaModel Database instead of the actual MySQL database. For applications that use Hibernate, the Corent dialect of Hibernate is configured.

As a result of these changes, the applications database calls are now directed to the MetaModel Database. These SQL events are intercepted by the Corent Agile Controller™ and then passed on to the Corent's ADBC™ (Agile DataBase Connector). The ADBC takes the SQL statement as it was submitted by the application to the MetaModel Database and parses it, then adds the filter criteria for the `TenantID` of the user whose session submitted the SQL (for example, where `TenantID = <myTenantID>`). This ensures that the security of the data in the shared database is always strictly maintained. Even when an external application like a ReportWriter connects, the data it can see is still restricted to the appropriate data for that Tenant. Because we know who is logging in, no matter from what application, we know which Tenant they belong and the proper filters are applied.

Because the manipulation of the SQL statements is done after they are submitted to the MetaModel Database and the user's tenant is known, the Corent Agile Controller can intercept and perform more sophisticated operations, including looking up tenant-specific processes and configurations. It is therefore possible to perform specific actions on a per-tenant basis, even substituting a different database connection so that one tenant could use a DB2 database even though all the others are using a MySQL database. Or one tenant could have additional SQL manipulation to restrict the data they could retrieve to records entered not more than 90 days before. All these per-tenant customizations can be done through the Agile Rules Engine<trade/> built into Multi-Tenant Server without any changes to the original application code.

Figure 3. The per-tenant customizations structure



The entire conversion of the jBilling application was accomplished with only a few minor changes to the application, mostly concerned with enhancing the authentication to include Tenant information in the session information, and to change the database connection to point to Multi-Tenant Server.

JBillling Changes Summary

Original Application	
Number of source files	897 (Java and jsp)
Total lines of code	76,621
Converted Application:	
Number of source files added	2 (standard template)
Lines of code modified	< 100
Application Business Logic changes	Zero

The jBilling conversion took less than a week including all the analysis and preparation to determine where the code needed to be modified. Many of the changes were simple repetitive one line changes in a series of Java code for JDBC access that is unnecessary if a database persistence layer such as Hibernate is used.

Step 4: Deploy the new multi-tenant SaaS application to the cloud

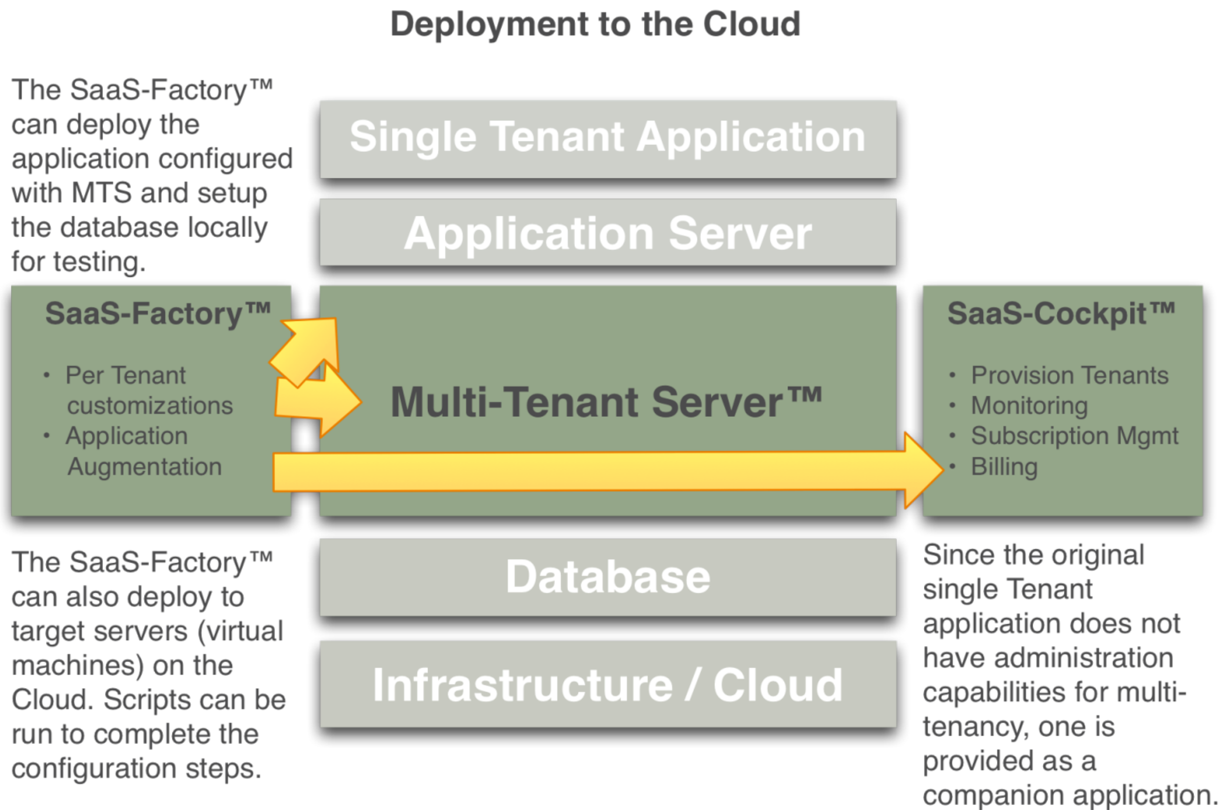
The SaaS-Factory can now be used to deploy a SaaS application to a designated server, including servers in the cloud.

After selecting a server for the application and database, the target database is generated (as a MySQL database on a Windows™ server for our jBilling application) and the fully configured Multi-Tenant Server application is deployed as a set of .WAR files to the application server selected. Multi-Tenant Server works with any modern J2EE Servlet container and has been certified for the WebSphere Application Server.

The deployed application is now capable of handling multiple tenants. However, the original application will not have an administration and management interface to manage tenants or to monitor a multi-tenant application.

The SaaS-Factory can also generate and install a companion application called SaaS-Cockpit™ which provides these basic multi-tenant services. It has access to the same MetaModel Database as the main application and has administration screens for provisioning tenants, assigning **Tenant Administrator** accounts, and configuring the basic parameters of the various per-tenant application configurations that are available. There are also administration facilities for monitoring and reporting on tenants and their users. Since one of the common characteristics of SaaS applications is the need to track tenant subscriptions and billing, facilities to do billing or integration with external billing systems is available.

Figure 4. Structure for deploying your new SaaS application to the cloud



This level of deployment is fairly basic as far as SaaS application deployments are concerned. The characteristics of the applications remain intact after conversion and standard types of deployment scenarios, including the use of cloud management tools to create templates and propagate instances of the application, can be used to deploy an operational architecture that meets the needs of the application for scalability, elasticity, resiliency, and redundancy.

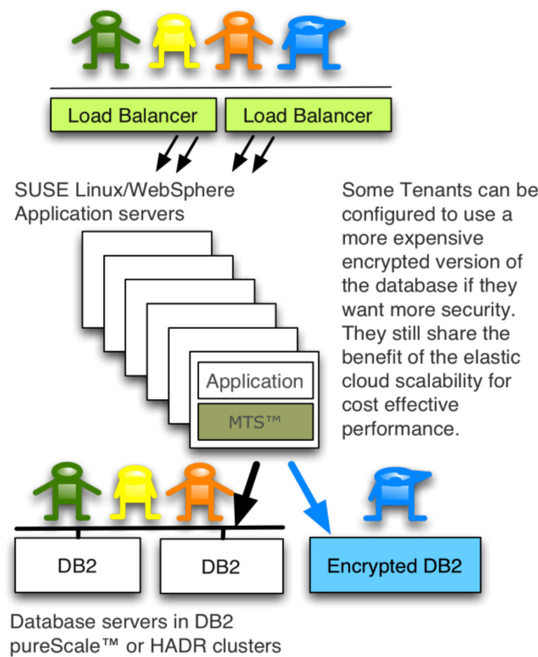
A typical SaaS application might have a set of application servers accessed through a load balancer and connected to a database server. The database server itself might be deployed as a cluster with the multiple database servers providing redundancy and scalability.

IBM™'s DB2 offers several technologies and configurations that can be used to achieve resiliency and guaranteed recovery times, as well as allowing the load on the database to be distributed:

- DB2 HADR (High Availability Disaster Recovery) configurations can provide both resiliency of availability and some load balancing through use of the secondary mode as a read-only database.
- IBM pureScale technology and TSA allows a clustered database environment with multiple nodes that share the processing load, but these have more restrictions on the hardware and OS configurations.

The trend in database availability and failover is becoming more sophisticated for SaaS applications because the number of customers/tenants affected by any outages is larger than that in traditional customer premise software. The trend is for capabilities which enable the application to run continuously even through offline reorganizations of data, schema evolutions, version upgrades, and heavy load variations (Figure 5). IBM's pureScale technology is one example of a trend of sophisticated database capabilities migrating to the world of Linux™ and cloud environments and enabling the kind of capacity handling and uptime previously reserved for dedicated mainframe environments.

Figure 5. SaaS deployment architecture for scalable resilient cloud operations



In conclusion

The information technology industry's evolution to SaaS is underway and as you might have surmised, it's already starting to cause some major changes in its landscape. Cloud computing is growing at rates far above any other IT wave and SaaS is the driver for that growth. This transition is forcing companies to rethink their business organization and to develop new ways of thinking about the delivery of services in a cloud-centric IT world. For software vendors there is an increasing urgency to understand, prepare for, and make the transition, or else be left behind on the digital dust heap of history.

The SaaS-on-a-cloud model differentiates itself from the traditional software vendor model in both the technical and business considerations. Those differences make the transition to SaaS a higher risk endeavor for ISVs. To reduce risk and accelerate time to market, ISVs can take advantage of proven technologies, products, and partners to assist in this transition.

Sidebar: Corent's Multi-Tenant Server

Corent's Multi-Tenant Server™ enables our customers to quickly transform existing single-tenant web applications into fully multi-tenant SaaS solutions that are cloud compatible without rewriting the application. Corent offers the most efficient and cost-effective approach to multi-tenancy. This plug-in middleware solution to multi-tenancy provides a secure, scalable solution that dramatically lowers the cost of service delivery.

You have the flexibility to choose your preferred technology stack, allowing you to use the OS, database and application server of choice. With Multi-Tenant Server's™ flexible architecture you can choose how to deploy your SaaS application, in-house or on any type of cloud. This enables the deployment of SaaS applications as public services or as Private SaaS™ that provides services to a limited audience such as a large enterprise with divisions that become the tenants.

Multi-Tenant Server™ comes with SaaS-Cockpit™ to enable you to provision, manage and monitor your SaaS customers, and with SaaS-Factory™ to enable per tenant customizations and augmentation of your SaaS application. With Corent's suite of solutions, SaaS transformation for applications can be dramatically accelerated.

Corent is currently offering a limited number of qualified candidates a no-charge Proof-of-Concept transformation, which will convert an ISV's web application into a multi-tenant SaaS solution running in the Cloud, and allow the ISV to test drive the resulting SaaS solution.

Resources

Corent Technology Inc. – Multi-Tenant Server™
www.corenttech.com

About the author

Scott Chate
scottchate@corenttech.com
Vice President, Products
Corent Technology

With a track record in software development, architecture, global operations and management for Fortune 500 companies, Scott is now experiencing the other side of the IT industry in an entrepreneurial organization with innovative technology. Through management consulting and product development at Oracle, TransCanada PipeLines, IBM, and Mercer, he has championed and implemented innovative solutions using emerging technologies to deliver efficiency, manage risk and enable opportunity.