SaaS Migration Strategy – White Paper

Contents

1 MIGRATION STRATEGY: CONSOLIDATING SINGLE INSTANCE SOLUTIONS TO MULTI TENANT APPLICATIONS 3

1.1 MAJOR BENEFITS FOR VENDORS 4
1.2 MAJOR BENEFITS FOR CUSTOMERS 4

2 SAAS MIGRATION STRATEGY 5

2.1 MIGRATION STUDY 6
2.2 APPROACH 8
2.3 DESIGN 9
2.4 DEVELOPMENT 10
2.5 TESTING 11
2.6 DEPLOYMENT 13
2.7 GO LIVE 14

3 ABOUT TECHCELLO 15

4 CONTACT US 16

1 Migration Strategy: Consolidating Single instance solutions to Multi Tenant applications

SaaS & Cloud Application delivery model has become the game changer and a huge paradigm shift in the way applications are developed, deployed, delivered, marketed, sold and serviced. With SaaS delivery model, ISVs are no more just software vendors rather service providers adding value to their end customer's business. Many existing software vendors and enterprises are unable to embrace the benefits offered by Clouds and Virtualization platforms due to the fact that the current applications are designed to work as single instance solutions. These Solution providers are continuously looking for a simple and effective mechanism to SaaS enable by consolidating their existing single tenant solutions to a multi-tenant based model.

Various technical articles and whitepapers have talked about the methodologies to build SaaS applications but very few talks about migrating existing single tenant applications to a completely shared true multi tenant applications. Multi Tenancy as an architectural design is the fundamental for any SaaS/Cloud applications, the existing SaaS applications in the market such as Workday, SalesForce, Gmail, Yahoo etc exhibits one or the other form of multi Tenant models. Although there are different multi tenant maturity models exists, out of which **Single Code based or fully shared Multi Tenancy** is the Cost effective and Low TCO model which benefits both Vendors as well as the Customers.

1.1 Major Benefits for Vendors

- First and foremost, TCO
- Single Code base, so no more multiple copies of the same software
- Easy Patches, Updates, enhancements and bug fixes
- Complete Sharing [Infra, H/W, S/W] leads to saving
- Fewer resources to maintain and manage single code base

1.2 Major Benefits for Customers

- Attractive rates – lower expenses for vendors will translate to more attractive pricing.
- Seamless Patch fixes, Updates, enhancements and bug fixes
- Availability of new features

In general developing new SaaS applications is fairly straight forward and easy to build with the availability of newer software and technologies, but migrating existing solutions to cloud is far complex and effort consuming process. Apart from cost involvement, it requires deep analysis, thought process, expertise and proper planning to deal with migrations.

This Whitepaper primarily details the various phases of migration process on how to migrate or consolidate existing single Tenant Solutions to Multi Tenant based model.

2 SaaS Migration Strategy

Everybody knows that there is no magic bullet which can convert existing Single Tenant applications to Multi Tenant model. Apart from development effort, time and cost it needs Expert knowledge and real time experience in the areas of architecture, design and SaaS. The migration effort, time and cost may vary from one product to another depending upon the existing product size, complexity, current customer & user base, database model etc. When it comes to application consolidation, Careful analysis is required while merging all the customizations done in the respective Client specific Source code, then merge into a single source code which can then be migrated into a multi tenant model.

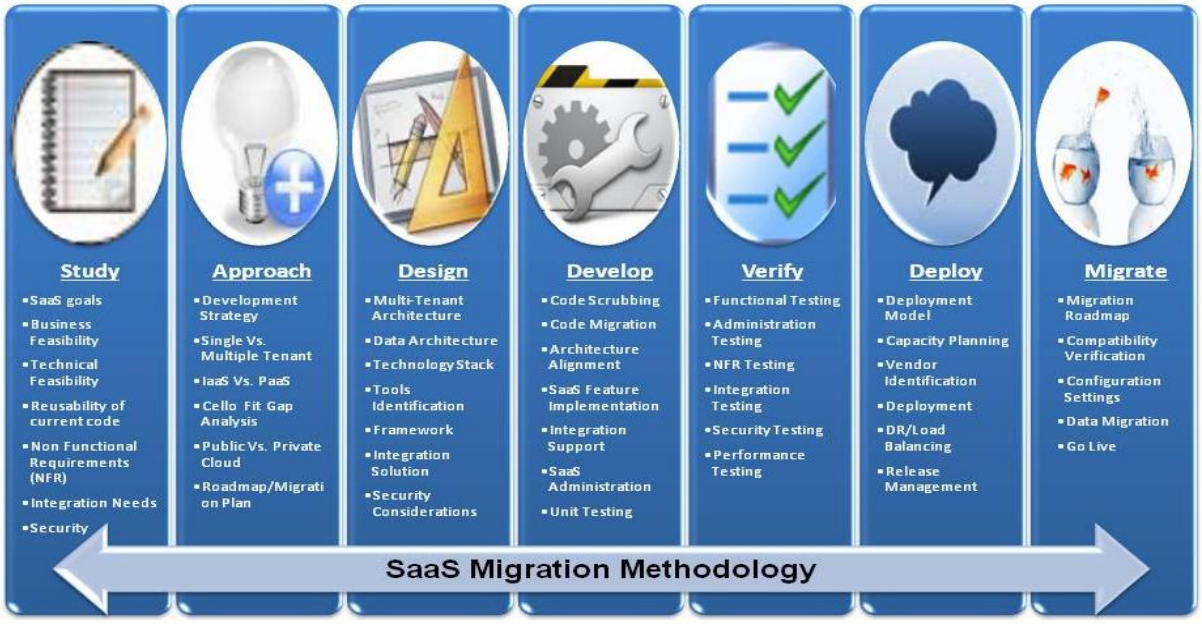

Figure 1: SaaS Migration Methodology Flow Diagram

The entire application architecture has to be re-architected to support multi tenancy and yet provide the maximum configurability and customizability for the end customers with minimal impact to the source code. Web user experience has to be drastically improved aligning with the

existing application accessibility, because building or altering look and feel might introduce learning curve for the existing users who were using the application with the older interface.
Generally, while rolling out the new application it is advisable to release it in a phased manner such as the beta version with part of the existing application and gather customer experiences, feedback and comments. These suggestions can be constructively incorporated into application or backlogged based on the priority.

The Entire migration exercise involves multiple phases. The various phases involved in a migration process are

Figure 2: Phases of Migration Process

2.1 Migration Study

The Application study is the first step towards the application migration process, the study phase plays a crucial role as it may have a huge impact on the cost and effort.

1. **Identify Goals** – Identify the clear goals of the migration need as to what will be the out-come or benefits. it could be financial benefits, aligning with the market, value added service, customer/user benefits, web presence etc.

2. **Business Feasibility** – Research the Business feasibility whether it is feasible to convert from the existing model to multi tenant SaaS model and its impacts and so on. Many migrations effort failed simply because of challenges related to business model, legal requirements, compliances, acceptance from end customers..

3. **Technical Feasibility** – Understand whether the features provided by existing solution can be serviced in the multi tenant model and identify whether the chosen technology supports such requirement. Applications that demand rapid response times in milliseconds are not a right candidate for web or SaaS application.

4. **Reusability of Existing Code** – While re-engineering an existing solution, the wise option is to reuse the existing code to the maximum extent to reduce the overall development time and cost, but it is truly dependant on the way it is written/coded. Carefully analyse the source code and find out the percentage of code which can be reused, which will help in the process of migration effort estimation.

5. **Identifying Non functional Requirements [NFRs]** – Identify all the non functional requirements to be supported by the application. The below snapshot provides some of the key NFRs that are typically supported by a SaaS solution.

Parameters	Challenges
Scalability	• Design to handle current and future loads • Optimum use of hardware and other resources
Performance	• Response time • Bandwidth constraints
Availability	• SLA Compliance • Offline mode of working
Security	• Physical and Network Security • Role based access • Data Encryption
Integration	• Support for in-bound and out-bound integration • Standards compliance (HL7, cXML, etc.)
Extensibility	• Custom field support • Support for dynamic forms
Multi Tenancy	• Single code base • Independent Schema/Shared Schema
Configurability	• Personalization/"Org"analization • UI/Business Rule/Workflow
Auditing	• Entity/Data level tracking

6. **Integration needs** – Identify the integration needs of the product, because single tenant applications are allowed to customize per tenant and they have the full control of the source code, whereas multi tenant applications are single Source code based and driven by metadata. Enough integration hooks and connectors have to be implemented which can enable the customers to connect with external applications and services.

7. **Application Security** – More than business aspects, security plays the major role in any SaaS application especially applications dealing with financial domain. Strong security

framework has to be implemented in order to make sure the application is accessible only to authorized users. What if an unauthorized user is able to login into the application and do some malicious activities? In some cases, the security framework has to comply with standards such as PCI and HIPPA.

2.2 Approach

The next phase involves coming up with an approach based on the facts collected from the migration study phase. Many of the key decision making steps happen in this phase, and hence its recommended to spend quality time in this phase.

1. **Development Strategy** – Should we reuse the existing code base or scrap the existing solution and build the solution from ground up. In certain cases, building the application from scratch is better than re-engineering the existing product, but such decisions purely depends on the way the existing solution is built.

2. **Single vs Multi Tenant** – Recognize the suitable delivery model to serve the customers. The decision of Single vs Multi Tenant is influenced by various factors such as the Extreme level of Customization, Application Complexity, Application isolation, legal aspects, and in certain cases end Customers may also influence outcome.

3. **Identifying IaaS vs. PaaS** – Choosing the appropriate service provider and deployment platform to host the application is a key and in some cases it influences the purchasing decision for the end customers. Main drivers are the Cost, Ease of Deployment, Cloud tools, development tools etc

4. **Identifying the Gap Analysis** – Identify the various features/functionalities that have to e built/enhanced in the current system in order to deliver the application in a multi-tenant mode. This exercise involves going through the current applications architecture, layer separation, high level designs and in some cases may also have to look at the potential performance bottleneck areas (ex: complex calculations)

5. **Identifying the types of cloud [Public /Private/Hybrid/Community Cloud]** – Choosing the type of cloud based on the type of domain/vertical that the application deals with. With the availability of multiple types of cloud with advanced infrastructure and introduction of new virtualization technologies solution provides have a plethora of choices to choose from. Choose a cloud based on the application and domain type. In general CRM, ERP etc are the right fit for Public cloud and financial applications are more suitable for Private cloud. Many SaaS companies prefer to take the hybrid model route, where they have an option of deploying the application in an on-premise model (for large customers).

6. **Cloud Independency** – It is always advisable to build cloud agnostic solutions, so that applications can be ported from one cloud to another with zero to minimal effort. In case if you are deciding to build applications specific to particular cloud such as Azure or Amazon, careful thought process is required to analyse the dependency that is being created, and the impact of breaking the dependency in future.

7. **Preparing Migration Plan** – Make a detailed migration plan including migration strategy, limitations/risks, migration schedule template, user communications process, validation, issue management, decommission etc.

2.3 Design

While consolidating all the multiple instance applications to single Code based SaaS solution, the whole architecture design, components design, messaging pattern, security, configurability etc has to be well designed and get it right at the first time.

1. **Multi Tenant Architecture** – Design and build strong multi tenant architecture, considering how the security is handled throughout the application, how the web layer interacts with the business layer, and business layer communication with the Data layer on so on. Although this is same for any kind of application but what matters is how multi tenancy is weaved through out all the layers of the application. For example it is not wise to handle security validation at the web layers when the application is SOA based.

2. **Database Isolation** – A True multi tenant application should provide complete Data Isolation models, because the primary considerations of SaaS application is Data Isolation, i.e. how a SaaS application isolates one tenants from another. The Migrated application should be able to handle all kind of Data Isolation Models, the frequently referred Data Isolation models are

   - Separate DB per tenant
   - Separate Schema per tenant
   - Separate Table per Tenant
   - Shared DB, Shared Schema, Shared DB

   More the isolation, more the complexity and cost.

3. **Technology Stack** – It is always advisable to choose a technology stack and programming language aligning with existing application, by doing so ISVs/Enterprises can retain the existing resources and expertise for the migration/development effort. Going with a different technology or programming language is a right fit if the existing stack or programming language is legacy and not exists.

Note: If the existing app is written in Asp.net 2.0 /3.5/4.0 and it follows web forms it is advisable to migrate to MVC architecture**.**

4. **Tools and Technology** – Businesses are always under the pressure of Time and Cost Constraints, so find the best tools and technology that can help in reducing migration

time and effort. Many a times development team tends to reinvent the wheel i.e. trying to solve an already solved problem, which is unnecessary and explorative in nature. Moreover if the development team is doing such functionality for the first time may not get it right and it might require iterations and rework to get it done. Look for existing frameworks/components that can save you significant time and effort, and also reduce the technical risks to a great extent.

5. **Framework** – The primary goals for consolidating existing single instance application into multi tenant one is to have a single source which is easily maintainable, manageable, cost effective, Scalable, Quality etc. To fulfil all the needs the application has to be well written, well commented, easy to understand, Modularized, fine tuned, Proper usage of Patterns and best practises and so on. Unless the team has proper expertise and experience in architecting or building such applications in the past, it is hard to build a High Quality, scalable applications. The best approach would be choosing a framework approach which is proven, well built and brings in all the non functional components to the table.
   With frameworks, developers will get undivided attention to concentrate more on the business aspects rather than contemplating on the technical aspects.

6. **Integration Solution** – Unlike single tenant application, SaaS should be more flexible and extendible and provide enough endpoints or hooks to customize as per the organization's need. Hence, so the application design should be integrate-able which can allow the customers to connect their application with heterogeneous services.

7. **Security Considerations** – Application Security is the heart of the SaaS application especially in the case multi tenant application, consider the security concerns in all layers of the product, such as Application level, Database level, Network and File level. The application level security is further subdivided into functional security and data security, functional security deals with providing access to specific features and modules based on the plan subscribed the tenant, data privilege deals with entitling data level security as to who can view/edit/delete the records or columns etc.

Apart from this, the whole web application has to be secured from SQL Injections, Cross Site Scripting and Access from unauthorized users etc.

2.4 Development

1. **Code Scrubbing** – During the migration process, reusing the existing code and logic is a better idea, but then in many of the cases the code cannot be used as such and it requires code scrubbing i.e. the process or removing Code Smell or removing dirty code. Because a piece of logic can be written in different ways, but then an ideal code is one which is simple, effective, non repetitive, Easy to Read, Easy to Maintain and mainly performs well.

2. **Code Merging** – While migrating to Multi Tenancy the initial job is to merge the different versions of the same product into a single source code, because there might be custom business validation, algorithm, hard coded configurations present, which has to scrubbed and merged as a single Source Code.

3. **Code Migration** – Once the Code is cleaned and merged, then introduce or write code that weaves multi tenancy into the application. Hard Coding is not an option at all while building Multi Tenant applications, because the application has to be flexible, customizable and configurable as possible without touching the code base. So build the application totally metadata and configuration driven.

4. **Architecture Alignment** – Aligning or sticking with the architecture helps the developers from deviating from the defined patterns such as making sure every developer follows the same architectural style, messaging pattern, business goals etc

5. **SaaS Feature Implementation** – SaaS application requires many SaaS related functional and Non functional components other than business functionalities, they are many such as Tenant Management, SaaS Licensing, Metering, Configurations Management, Business Rules, Workflow, Reporting and Notifications etc.

6. **Integration Support** – While re-writing the application, make sure the application provides enough integration hooks with which customers can integrate with their existing applications either located at on premise or cloud. For Example, in an e-commerce application, as soon as an order is placed, the same has to be intimated to QuickBooks to maintain the inventory, but not all customers might use Quickbooks some may choose Xero or GnuCash. Threfore, the application must be flexible enough to integrate with other applications.

7. **SaaS Administration** – All the administration related features such as Tenant Management, Security Management, licensing Management etc requires an effective yet intuitive dashboard in order to configure and customize. The Administration screens have to be incorporated to manage the entire application using the web interface rather managing with XML configurations or using back end.

2.5 Testing

1. **Functional Testing** - Test the features and components to check whether it meets the user requirement. Build multiple tests cases and validate it fulfils the various functional testing. Functional includes,

    a) **Mainline Functions** - Testing the primary business specific functionalities of the product. For example, in a simple registration page, checking whether a user is able to register using the page with positive inputs, replicate the same with Negative data.

b) **User Experience Testing** – Validate whether a user can easily navigate throughout the application without any complexity, especially in the ajax based applications where navigations and pop ups are showed based on the inputs given by the user.

c) **Accessibility** – Checks the accessibility of the system.

d) **Error Conditions** – Test whether appropriate error messages are displayed to the user in the occurrence of unexpected exceptions. Error messages should be more clear and precise. Server error messages or business specific error messages should not be disclosed to the end users which may lead to malicious practices.

2. **Administration Testing** – Effective testing has to be done to verify whether all the administration screens work as expected. Administration screens shares the equal weightage similar to business functionalities. Any defects or bugs in the administration screens might stop application managers to configuring and customizing the application.

3. **NFR Testing** – The Non Functional Testing has to be carried out to validate the Scalability, Reliability, performance, availability and other non functional aspects. Generally, Testing NFRs is even more complicated than testing business specific components.

a) Performance Testing

b) Load Testing

c) Volume Testing

d) Stress Testing

e) Security Testing

f) Deployment Testing

g) Penetration Testing

h) Compatibility Testing

4. **Integration Testing** – Integration testing is very important for application migration scenarios especially when we integrate one module with another and especially integrating functional and non functional elements. Integration test can only be done when the individual modules or code logic unit tested and confirmed.

5. **Security Testing** – Multi Tenant application has to be thoroughly tested for security, accessibility and privacy. Security testing includes simulating a user by applying different roles and privileges and validate if the user is able to access the particular functionality.

6. **Performance Testing** – Satisfying all the requirements of the customers and users is not sufficient to qualify for a good application, rather the key is to fulfil the requirements without compromising Speed, Scalability and Stability. Identify the most used functionality of the application and test it for performance, the testing plan could be duplicating 1000s of customer accessing the application at the same time, users from different locations etc. Various Performance testing types includes:

a) Scalability Testing

b) Load Testing

c) Stress Testing

d) Endurance Testing

e) Spike Testing

f) Volume Testing

2.6 Deployment

1. **Deployment Model** - Identify the deployment model, it is very important to identify the deployment model i.e. 1. Non Distributed 2. Distributed Deployment, the second option is always advisable for Cloud based as solution as Scalability is one of the non functional attributes and the solution is expected to scale up and scale out as and when it is necessary. One of the aspects that help scaling the application is "Specialization" i.e. isolating mostly used features or components separately.

2. **Capacity Planning** – Calculate the load that your server can handle at any given point of time, this can be done using techniques such as "Transaction Cost Analysis" – which calculates the cost of the important user actions, and "Predictive Analysis" – which basically forecasts the future resource utilization based on the past history.

3. **DR/Load Balancing** – When the distributed deployment model is chosen, it is necessary to employ a network load balancer to take care of request and response. NLB is a defacto component provided by all the Cloud provides, choose an appropriate routing algorithm that suits the application.

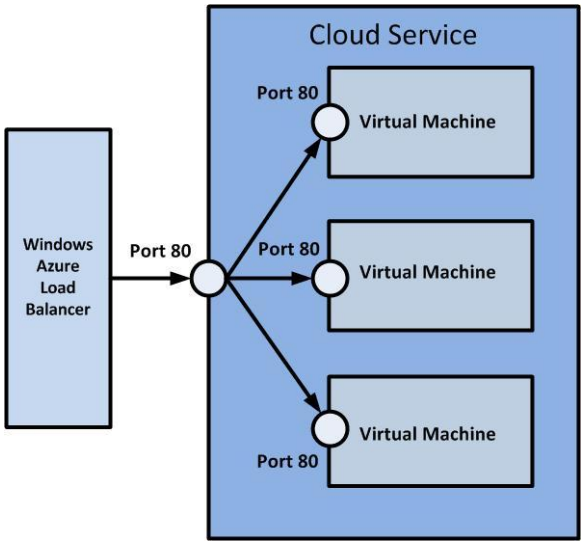

Image Source: http://www.windowsazure.com

4. **Release Management** – SaaS applications are all about innovations and enhancements, the app will continue to evolve every now and then, and the development teams

are forced to build new features, enhance existing functionalities, fix bugs and patch it etc. So a best release mechanism has to be adopted, Cloud providers offers release management as an add-on e.g. Azure's WARM etc.

2.7 Go LIVE

1. **Configuration Settings** – Multi Tenant applications are metadata driven, so based on the configurations the application run time behaves differently. Capture the various settings and parameters and consolidate against each tenant.

2. **Data Migration** – While re-architecting the entire applications, the existing data models might have been modified to suit multi tenancy, if so the existing data has to be migrated to the new environment accommodating all the data level changes required. Depends on the size of the applications and the number of existing customer, the migration effort and process might vary.

3. **Migration Roadmap** – Devising a plan for full fledged product roadmap as to what are all the new features or existing features will be built into the product, the timeline, the versions etc.

4. **GO LIVE** – Rolling out the migrated product to production environment.

5. **Application Decommission** – Decommissioning the old solution.

3 About Techcello

Techcello is a cloud-ready, multi-tenant application development framework built on Microsoft .NET. ISVs, Enterprises and BPO/KPOs use Techcello to build software products and applications within .NET - better and faster. Applications built on Techcello can be hosted anywhere from public cloud (such as Amazon, Windows Azure) or private cloud to on-premise serves.

Techcello provides a complete SaaS lifecycle management solution, which allows ISVs to build, manage and operate their product in a SaaS business model.

Techcello provides complete freedom, flexibility and control of custom development, without vendor or platform lock-in and still saves you from all the complexities and cost overheads of building and maintaining your own engineering framework. Techcello framework consists of all the basic engineering components & administration components in a ready to use form (API, WCF services and inheritable classes) so that application developers can focus on building their domain specific business features rather than engineering / multi-tenant features.

To understand more about techcello, Refer

SaaS : Take a Tour

Enterprise Application: Take a Tour

Techcello Architecture

Techcello Features

SaaS application Benefits / Enterprise Application Benefits

White Papers / Case Studies

Watch Techcello Videos

Request for a Demo

4 Contact us

Techcello

4819 Emperor Blvd,

Durham, NC 27703, US

Tel: +1 888 706 1604

General Enquiry : info@techcello.com

Sales Enquiry : sales@techcello.com

Support Enquiry : support@techcello.com