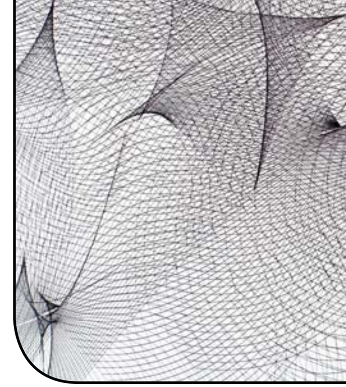
Efficient Software Delivery through Service Delivery Platforms



by Gianpaolo Carraro, Fred Chong, and Eugenio Pace

Summary

Delivering software as a service (SaaS) has gained a lot of momentum. One reason this one-to-many delivery model is attractive is that it enables new economies of scale. Yet economy of scale does not come automatically—it has to be explicitly architected into the solution. A typical architectural pattern allowing economy of scale is "single instance multi-tenancy," and many Independent Software Vendors (ISVs) offering SaaS have moved to this architecture, with various levels of success.

There is, however, another means of improving efficiency which ISVs have not adopted with the same enthusiasm: the use of an underlying Service Delivery Platform (SDP). Adoption has been slow mainly because service delivery platforms optimized for line-of-business applications delivery are still in their infancy. But both existing and new actors in the hosting space are quickly building compelling capabilities. This paper explores the goals, capabilities, and motivations for adoption of SDP, and describes the technology and processes related to efficient software delivery through SDP.

Economies of Scale and Application Architecture

Operating systems evolved as a way of simplifying application development, operations and management. Rather than requiring each application to (re)create the full stack of subsystems needed for it to run with expected quality levels, an operating system provides an infrastructure where common, general purpose applications services are encoded and reused.

Often these services are complex technically and require expertise and specialized skills to develop. ISVs quickly understood the value proposition of an operating system and standardized their development on top of one or two operating systems. Leveraging the common underlying infrastructure allowed them to spend more time on the domain-specific part of the application, which is, in the end, what software buyers are paying for.

There are cases where the specific needs of an application go beyond what a general purpose operating system offers. Under this circumstance parts of the operating system are replaced by custom developed modules providing the specific level of support. For example, databases often write their own file systems, and in the case of Microsoft SQL Server, its own scheduler and memory management. But these very specific needs are very much the exceptions than the norm.

Enterprises and ISVs continually factor out common components from their solutions into "horizontal" application frameworks. These frameworks provide further abstracted, common, shared components (and procedures) that lead to increased reuse, increased productivity, and in general, a more predictable development and operational environment.

"THE DOWNWARD FLOW OF HORIZONTAL CAPABILITIES FROM ISV "PLUMBING" INTO FRAMEWORKS, AND THEN INTO CORE PLATFORMS CAN BE GENERALIZED FURTHER TO COVER THE SCENARIOS OF SOFTWARE DELIVERED AS A SERVICE. IN THIS SENSE, AN SDP BECOMES AN "OPERATING SYSTEM" FOR SERVICE DELIVERY."

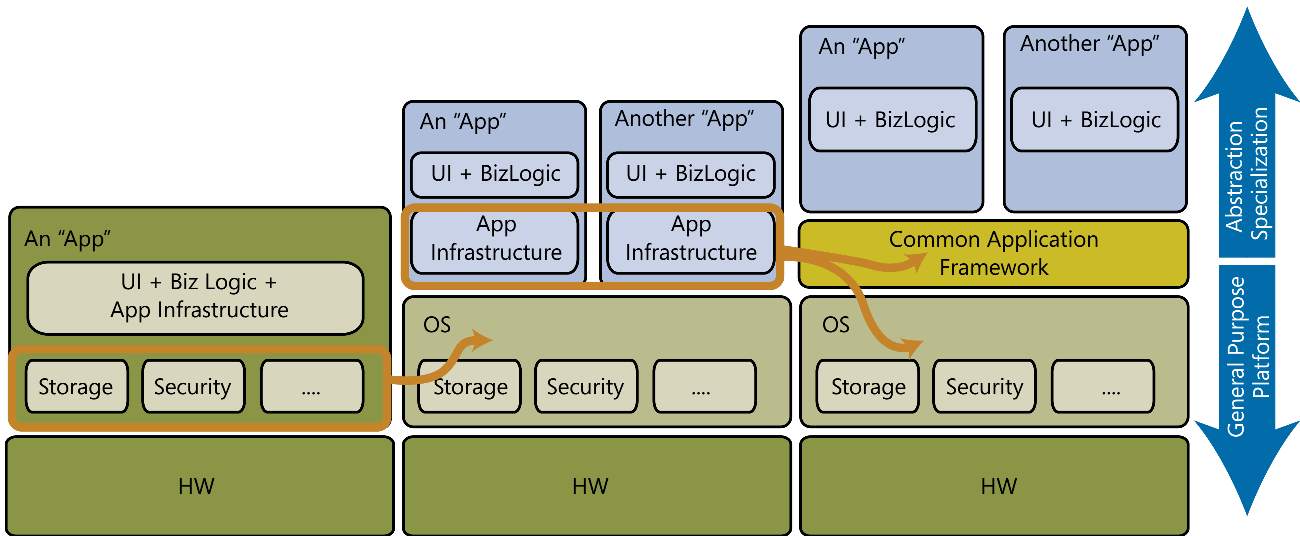
The problem with building rich frameworks is that they often do not directly contribute to the value of the software itself. They are "necessary evils" that facilitate the longer term maintenance of the application, and ISVs and enterprises would rather have someone else own them. This reinforces a win-win situation where "plumbers" can concentrate and sell horizontal functionalities so "domain experts" can concentrate on "vertical" functionalities where the value proposition for the buyer is higher. Let's face it, nobody buys a solution because it has a better exception handling mechanism.

As illustrated in Figure 1, there is a natural and ongoing process of extraction and generalization of functionality from applications into frameworks and from frameworks into core platform components. By increasing the amount of shared elements, this flow improves economies of scale.

The downward flow of horizontal capabilities from ISV "plumbing" into frameworks, and then into core platforms can be generalized further to cover the scenarios of software delivered as a service. In this sense, an SDP becomes an "operating system" for service delivery, specialized in the horizontal characteristics and requirements of SaaS-delivered applications.

Traditional hosting companies are natural candidates to implement and offer an SDP given their background, expertise, skill set, and installed base. However, other actors might as well consider entering this emerging space. ISVs that are currently self-hosting their SaaS solution could consider monetizing the self-hosting environment they

Figure 1: Factoring out commonality into a reusable platform



have built for themselves by offering it as a general-purpose SaaS delivery environment. System integrators with world-class operational excellence gained through their business-process-outsourcing offerings could leverage that knowledge in the fast-growing SaaS space.

"THE KEY POINT IS THAT AN SDP'S EFFECTIVENESS IS HIGHLY DEPENDENT ON THE ARCHETYPE SERVED. THE MORE KNOWLEDGE OF THE APPLICATION AN SDP HAS, THE GREATER ITS ABILITY TO INCREASE THE EFFICIENCY OF RUNNING AND OPERATING IT, AND THE GREATER THE DEGREE OF SHARING."

Success Factors of a SDP

The degree of success of an SDP is determined by the following attributes:

1. *Core operational capabilities:* the ability to provide strong service level agreements (SLAs) on nonfunctional requirements, such as availability (fault tolerance and uptime), scalability and disaster recovery; and on generic service features, such as multiple geography presence, 24-7 customer support, and multilayered physical security.
2. *Services depth:* the degree of sophistication of the services it provides, such as billing support for multiple payment options.
3. *Services breadth:* the completeness of the platform; in other words, the support for the different stages of a SaaS-delivered application life cycle, such as billing service, application staging, capacity planning services, or methods for patching and upgrading applications.
4. *Ease of use:* usability from the ISV perspective; in other words, the cost of learning and using the services provided by the SDP. Easy-to-use SDPs have short ramp time, complete documentation, well-planned interfaces and SDKs, sample code, templates, wizards and training content.

Note that the four attributes above are not meant to represent a maturity model. Observation of existing SDP offerings seems to indicate that two strategies are currently offered: a breadth strategy, optimized for providing a comprehensive, end-to-end platform that covers every step of a typical SaaS-delivered application life cycle (see Figure 2), with albeit shallow coverage in some capabilities; a depth strategy, focusing on certain stages only, but offering sophisticated features on these. The same survey also indicates that the "ease of use" attribute is not yet fully incorporated in the offerings as many of them rely on human intensive ad hoc processes.

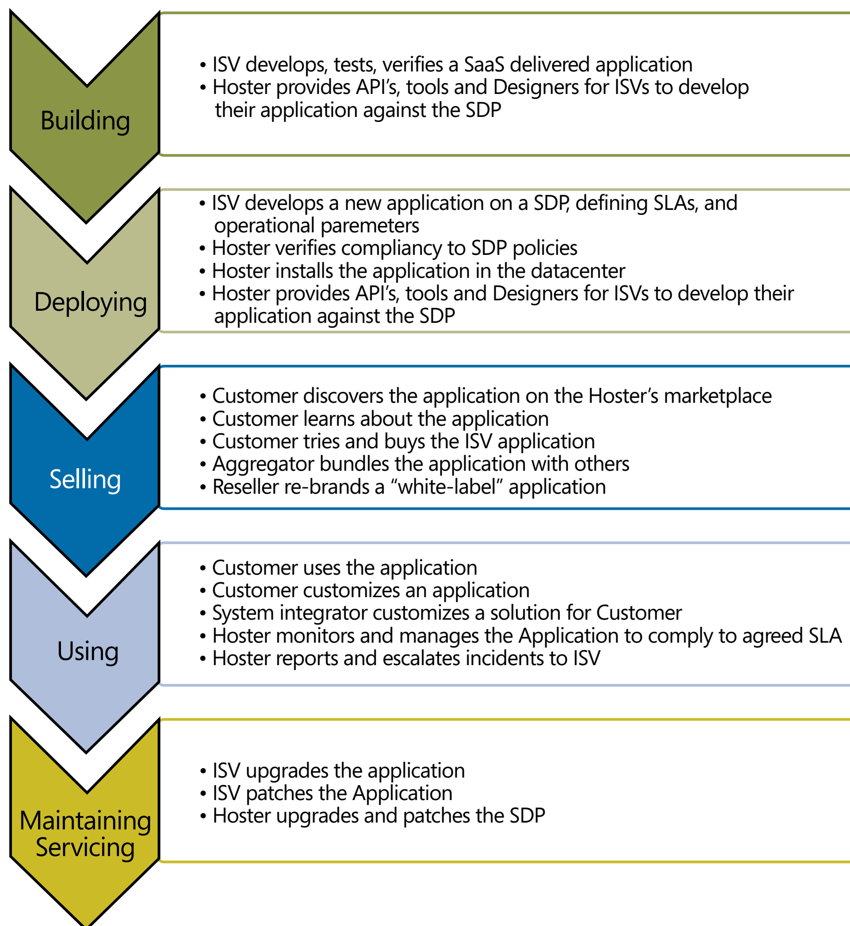
Application Archetypes: Does One Size SDP Fit All?

Business applications can be classified in different archetypes based on their characteristics and requirements. A few examples of these archetypes are:

- *Online transaction processing systems (OLTP):* characterized by low latency, high responsiveness, data integrity, predefined UI workflows. Instances of this archetype are e-commerce sites, e-banking systems, CRM systems.
- *Analysis systems or online analytic processing (OLAP):* characterized by their ability to produce complex analytical and highly customizable queries on large multidimensional datasets, with low latency responses. BI systems fall into this category.
- *Batch systems:* capable of performing operations on large datasets efficiently, coordinating jobs to maximize CPU utilization with recovery policies when exceptions occur.

Each of these application families has its own constraints, characteristics, and optimal design patterns that can be applied to solve the specific challenges they present. Very often, these challenges have conflicting

Figure 2: A typical SaaS-delivered application life cycle and associated activities



goals. For example: OLTP will optimize for low latency, whereas latency for batch systems is not as important. OLTP scales better horizontally and benefits from a stateless architecture, while batch systems scale vertically and tend to be stateful. The infrastructure and services to support each is consequently significantly different.

The key point is that an SDP's effectiveness is highly dependent on the archetype served. The more knowledge of the application an SDP has, the greater its ability to increase the efficiency of running and operating it, and the greater the degree of sharing.

The Capabilities of a Service Delivery Platform

Figure 2 illustrates a typical SaaS-delivered application life cycle. Each stage of the life cycle is characterized by a number of activities performed by the ISV, the hoster, and other actors that will become more relevant as the degree of sophistication of the SDPs increases (system integrators, value-added resellers, for example)

The four scenarios below describe the capabilities and the experiences enabled by increasingly sophisticated SDPs.

Scenario A: The Basic Service Delivery Platform

The capabilities of the simplest SDP are essentially those offered by most of traditional hosters today. These services are focused mainly on core, very generic infrastructure components such as: CPU, network access, Internet security features, and storage (disk and databases); commonly referred to as "ping, power, and pipe".

Economy of scale in this scenario is limited to the sharing of the lowest levels of infrastructure: the building for the data center and the IT infrastructure, such as servers and network equipment.

In this basic scenario, deployed ISVs applications use their own standards for application architecture and little of it is exposed to the hoster or known by it. ISVs are expected to develop and provide their own multitenant management infrastructure, security infrastructure, tenant management, and so on.

Infrastructure services requests such as registering a new domain, a new virtual directory or a Web site, creating a new database, allocating storage are often performed manually or are automated with custom scripting.

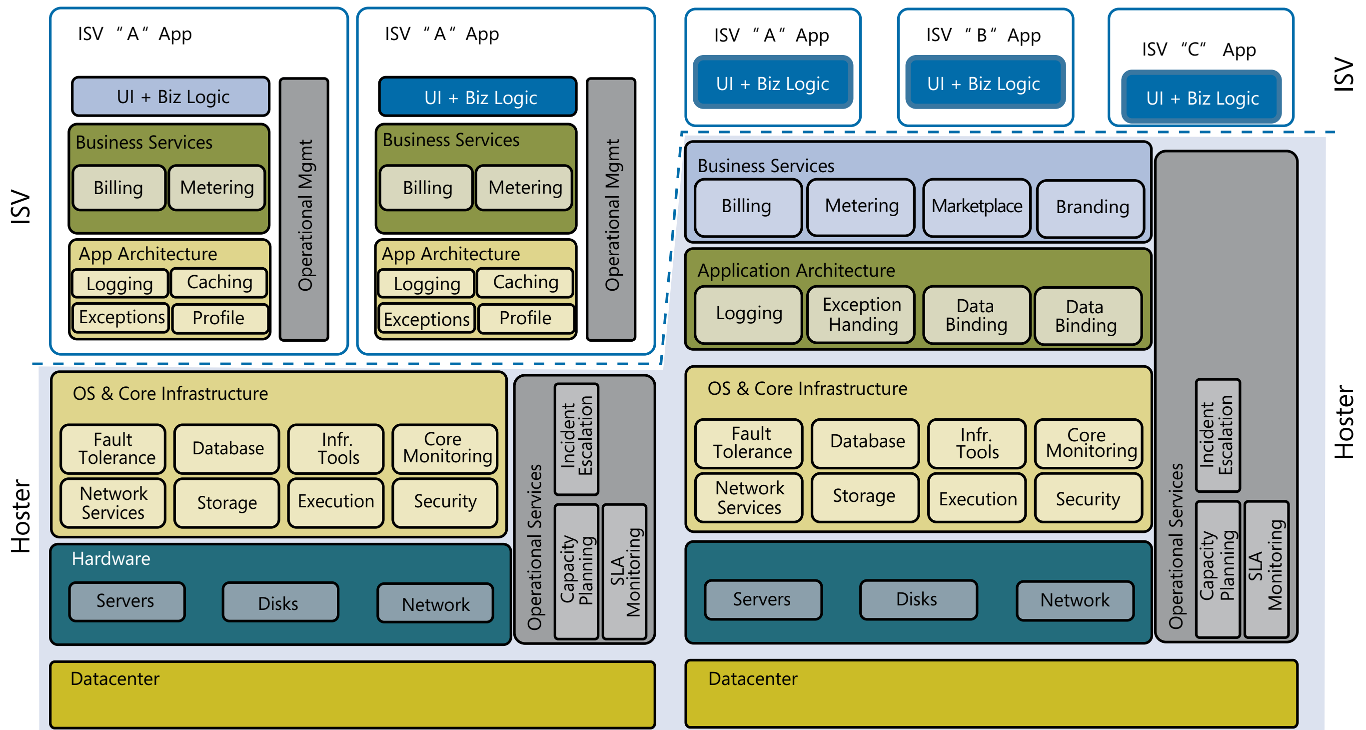
Installation and deployment of the solution require the ISV to provide the hoster with detailed instructions for configuration and setup; multiple nonstandard tuning is required (such as handling XML config files, registry keys, file paths, connection strings) because little of the application architecture is known or exposed to the hoster. These ad hoc procedures hinder the hoster from scaling its procedures to thousands of applications because it has to treat each one as an exception.

Each application is a "black box" of potentially conflicting components, competing inadequately for shared (and limited) resources, a reason many ISVs request dedicated servers that will guarantee complete isolation from other applications. However, dedicated servers goes directly against the goal of efficiency through shared infrastructure.

The hoster can only watch and act upon broad indicators to monitor the solution. They can only measure general, broad performance counters like CPU and memory workloads, bandwidth, SQL contention; and maybe some application-specific performance counters that the ISV provides on an "as needed" basis. There are little or no agreed-upon, standard mechanisms for finer-grained management of the application.

Quite remarkably though, even with very little or no knowledge of what the application does, hosters are usually able to give the ISV detailed information on database performance. This is because database artifacts are common to all ISVs (every application has tables, stored procedures, indexes, triggers, views, and so forth) and the platform where these artifacts are instantiated (the database server itself, like SQL Server) is instrumented at that level of abstraction. Hosters can therefore provide detailed contention, locking, and performance reports, and even suggest improvements on such artifacts.

Figure 3: Driving economies of scale through an increased shared infrastructure



"EVEN WITH VERY LITTLE OR NO KNOWLEDGE OF WHAT THE APPLICATION DOES, HOSTERS ARE USUALLY ABLE TO GIVE THE ISV DETAILED INFORMATION ON DATABASE PERFORMANCE BECAUSE DATABASE ARTIFACTS ARE COMMON TO ALL ISVS AND THE PLATFORM WHERE THESE ARTIFACTS ARE INSTANTIATED IS INSTRUMENTED AT THAT LEVEL OF ABSTRACTION. "

The salient point here is that these artifacts exist on every database-based application regardless of the ISV authoring it. Comparatively, using a Web application as an example, the only artifact shared among different ISVs is the "Web page" identified by an URL, an element too coarse-grained to be efficiently managed. Questions, such as what part of the page takes longer to load, what components are instantiated through a particular page, or which one of these dependent components is causing contention or run-time problems, require code inspection, use of advanced tools and/or deep knowledge of how the application is built—procedures that inherently don't scale to thousands of applications.

Upgrading everything (except core operating system, networking equipment, and other core infrastructure) requires manual or ad hoc procedures as well and human interactions (e-mail, phone calls) between ISV experts and hosters.

SaaS applications operating in this basic environment are usually implemented using a wide variety of not necessarily compatible technology stacks and libraries. For example, you might find a Terminal Services hosted and delivered VB6 application, a Web app using various available frameworks and run-time components. In other words, the exception is the rule.

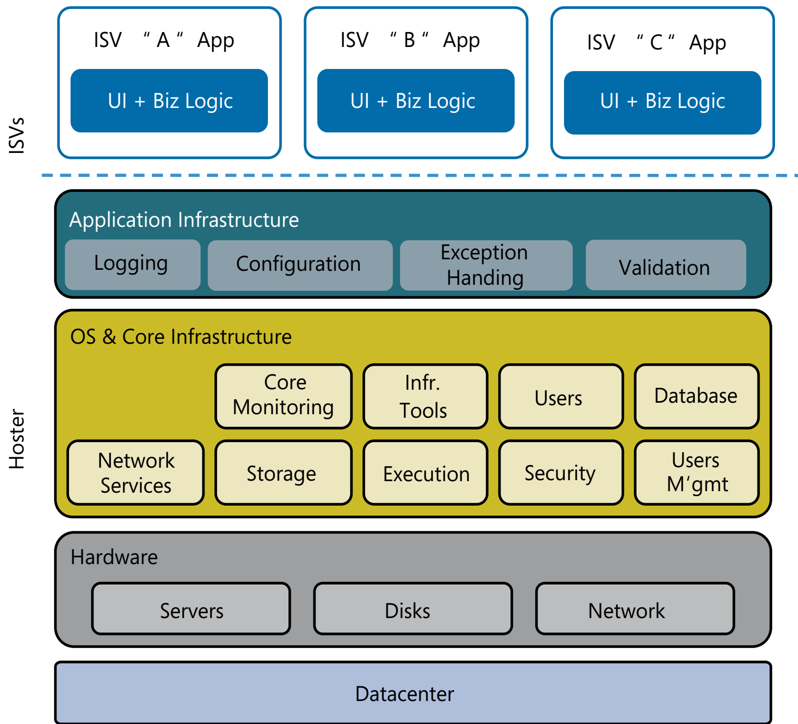
This scenario is mainly characterized by a sharing of operational infrastructure. In summary, economies of scale in this scenario are limited to fairly low-level components. One of the key factors affecting the SDP efficiency is the heterogeneity of the hosted applications.

Scenario B: Improving Efficiency Through Deeper Knowledge of the Application Architecture

An increased amount of shared components leads to higher levels of efficiency, so the question is which are the most natural candidates to be "extracted" from applications into the SDP? The obvious candidates are those referred to application infrastructure services such as: application configuration, run-time exception handling and reporting, logging and auditing, tracing, caching, data access. Every application needs them, yet they are frequently written once and again by ISVs.

An example of a common, standard and widely adopted application infrastructure framework is Microsoft's Enterprise Library.
By exposing these basic services publicly, the SDP has a much increased ability to automate common procedures and offer more advanced operational management capabilities. Thus, finer-grain tuning, customization and troubleshooting is available. Notice that the hoster

Figure 4: Increased reuse through an application infrastructure



does not need to understand in detail what the application does, but rather how it does it (for example, where are connection strings to the database stored? How are run-time exceptions logged and notified?).

Because ISVs would be developing against these APIs, the hoster is required to publish an "SDP SDK" including documentation, samples, and even some basic tools for ISVs to download and use.

Hosters in this scenario can dramatically scale out basic operational procedures as all of them are common, and exceptional cases become, well, exceptions. Applications that don't comply to the standards can of course lead to premium offerings for increased revenue streams.

Additionally, hosters can offer a higher range of differentiated services with different monetization schemes. For example, the hoster knows that all applications will log run-time exceptions using the same policies and procedures, so basic run-time exception logging and reporting could be offered in the basic hosting package, and advanced run-time exception logging, notification and escalation could become a premium offering. (See Figure 5.) Notice that with this approach the ISV application does not change, because all this logic ("plumbing") resides on the SDP side.

Scenario C: The SDP Beyond Operational Management

Scenario B is clearly an improvement over A, but the SDP can be further enhanced with the addition of more sophisticated application services, like advanced infrastructure provisioning, security-related services (user profiles, user roles, personalization) and new business services such as: business events, application metering and billing and usage intelligence.

Because contracts between services and applications can be very well defined, SLAs at the services level can now be established. Traditional SLAs at the macro level (like "uptime," "bandwidth," and "CPU utilization") are necessary but not sufficient. Much finer-grained SLAs can be negotiated and traded. For example, an SLA stating "provisioning a new tenant in less than X min" can be defined, monitored, and enforced.

The SDP SDK can be further enhanced to cover the new services provided, with stand-alone implementations of the services that enable offline development. For example, it is highly unlikely that the hoster will share with the ISV the full billing system just for development purposes. The ISV will most likely code against a "mock billing service" with equivalent interfaces and contracts and a minimal simulated implementation. The goal here is the creation of a development environment that mimics all SDP capabilities with minimal dependencies to any concrete implementation. This also avoids unnecessary exposure of SDP internals and intellectual property to the public. To support this scenario, a much deeper integration and interaction between the hoster and the ISV's own software

Figure 5: An example of differentiated offerings for the same function

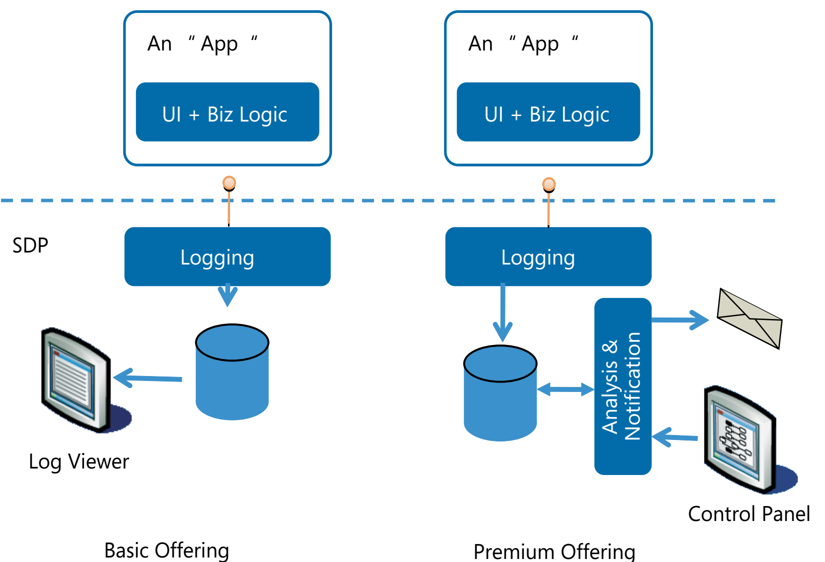
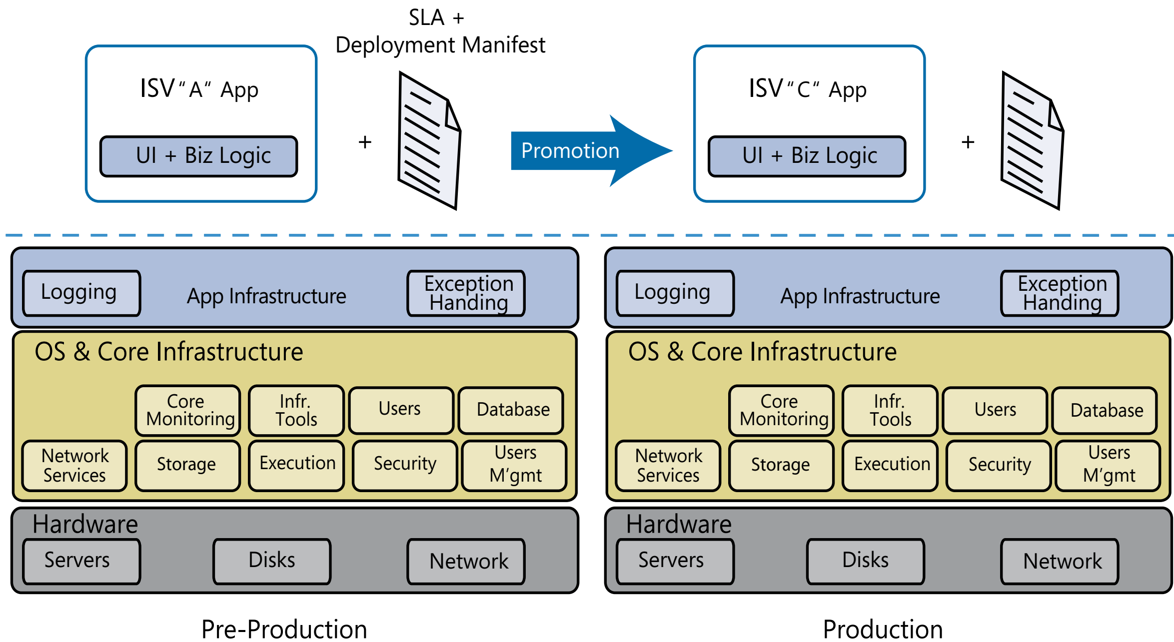


Figure 6: Deployment includes code and runtime definitions



"VERSION CONTROL, INTEGRATED INCIDENT MANAGEMENT, STAGING AND VERSION PROMOTION BETWEEN STAGING ENVIRONMENTS ARE OFFERED THROUGH THE SDP— THIS INFRASTRUCTURE ENABLES PROGRAMS LIKE "BETA TESTERS" TO RUN ON CONTROLLED REGIONS OF THE SDP, COLLECTING FEEDBACK, USAGE PATTERNS AND BUG RESOLUTION, BEFORE FEATURES ARE PUSHED INTO THE FINAL RELEASE."

development life cycle (SDLC) is required.

Also in this scenario, the deployment of solutions into the SDP is done through highly automated procedures and minimal manual intervention. These procedures include advanced features for automatic validation and configuration of the application requirements (prerequisites, server locations, connection strings, for example).

The deployment into the SDP goes beyond the code and configuration scripts to include the negotiated SLAs, operational requirements such as automatic capacity management and incident escalation, billing parameterization.

The SDP is also able to provide staging environments, where the application can run and be verified prior to being promoted to production. Staging environments allow simulation of billing, tenant creation, faults, and so on, to enable more complex modeling of real world scenarios. These can be seen as "unit tests" for the SLAs between the hoster and ISV. (See Figure 6.)

Interestingly this leads to a new breed of services to be offered beyond the runtime: for example, fault simulation, load testing, performance analysis, optimization could be available for the ISVs. Smaller ISVs could have access to temporary resources that would be too costly for them to own.

In production, because of the deeper knowledge of how the application works, the SDP can offer automatic resource management capabilities and tuning. More advanced SDPs can dynamically assign new machines, add CPUs to a cluster, assign higher communication bandwidth, and whatever other infrastructure is required to keep the application at the agreed-upon SLAs.

As applications are fully instrumented, hosters can offer intelligence on usage patterns and finer-grained analysis of how the application performs and works, feeding back this information (possibly as a premium service) to the ISVs' product planning pipeline.

Because of the highly automated quality assurance and deployment procedures, ISVs have an opportunity to offer almost real-time product enhancements based on the intelligence collected and improve their products constantly based on more accurate user feedback.

Selling, branding, bundling, and aggregation of applications are enabled through well-defined composition rules, therefore, the hoster can create bundled offerings with common services across different solutions (such as Single Sign On or common role management). Also, "white label" branding is enabled for aggregators and third parties to create specialized, customized offerings based on the same code base.

Scenario D: The Ultimate SDP

The ultimate SDP doesn't exist yet. This section is a little bit of extrapolation and a little bit of speculation. In the most advanced scenario, in addition to all the features described above, the SDP includes services for complete application life cycle management,

allowing further integration of development, versioning, deployment, management, operations and support of an SaaS-delivered solution, allowing an extended ecosystem to contribute and collaborate to deliver specialized solutions.

Discovering, learning, trying, developing, extending, versioning, tuning, are use cases supported by the SDP. This is a full integration of the ISV Software Development Life Cycle (SDLC) and the SDP Software Operations Life Cycle (SOLC). In this scenario, aspects of software development are offered as a service itself: bug tracking, software version control, performance testing, and so on. (Figure 7.)

All SDP capabilities are expressed in machine-readable formats. Models of software components and SDP capabilities are fully integrated into the tools. Verification, analysis and simulation of these models can be performed by independent members of ecosystem: ISVs, third parties, aggregators, enterprises, and others collaborate to create complex systems.

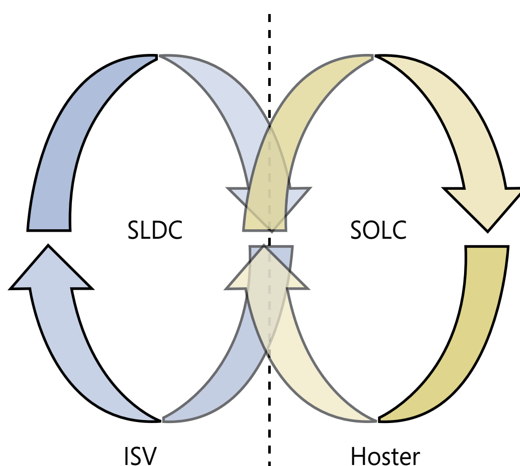
Application metadata includes not just operational parameters, but also information required to publicize the application in the SDP Marketplace (the equivalent of the "Add New Software" application registry found on Windows), provide "learn more" content, guidance content, documentation, training, additional value added professional services, and so on.

Version control, integrated incident management, staging and version promotion between staging environments are offered through the SDP, allowing ISVs, aggregators, system integrators to develop, customize and maintain different concurrent offerings. This infrastructure enables programs like "beta testers" and "early adopters" to run on controlled regions of the SDP, collecting feedback, usage patterns and bug resolution, before features are pushed into the final release.

The Marketplace service provided by the SDP offers a mechanism to discover, learn and try new offerings and is made available utilizing the metadata that accompanies every application.

As said, it will take some time for these advanced SDPs to become mainstream.

Figure 7: Integration of ISV's SDLC and hoster operational life cycle in the "Ultimate SDP"



Conclusion

SDPs represent an exciting new opportunity for traditional hosters to create differentiated, high value offerings; lowering the bar of entry for a larger number of ISVs to offer solutions with world-class operational levels. First movers into this new market category will attract large numbers of ISVs, especially those in niche or small- to medium-sized business segments that cannot economically self-host due to their skill set or the financial feasibility of their business models.

Microsoft's Architecture Strategy Team is actively investing in developing architecture guidance to help ISVs, hosters and enterprises realize the benefits of software and services, leveraging the Microsoft Platform.

Resources

MSDN Architecture Development Center
http://msdn.microsoft.com/architecture

SaaS Section on MSDN Dev Center
http://msdn.microsoft.com/architecture/saas

LitwareHR – A sample SaaS-delivered application developed by Microsoft Architecture Strategy Team
http://www.codeplex.com/litwarehr

About the Authors

Gianpaolo Carraro, director of Service Delivery – Microsoft Architecture Strategy, drives SaaS thought leadership and architectural best practices for Microsoft. Prior to Microsoft, Gianpaolo was cofounder and chief architect of a SaaS startup; he is a former member of technical staff at Bell Laboratories. Gianpaolo is a published author and a frequent speaker at major international IT conferences. You can learn more about him through his blog: http://blogs.msdn.com/gianpaolo

Fred Chong, architect - Microsoft Architecture Strategy Team, is recognized by the industry as a subject matter expert on the topic of SaaS architecture. Previously, he has designed and implemented security protocols and networking components for Microsoft products and customer solutions. Fred has also conducted research at the IBM T.J. Watson Research Center and the University of California at San Diego. You can find his blog at http://blogs.msdn.com/fred_chong

Eugenio Pace, architect - Microsoft Architecture Strategy Team, is responsible for developing architecture guidance in the SaaS space. Before joining AST, he was a product manager in the patterns and practices team at Microsoft where he was responsible for delivering client-side architecture guidance, including Web clients, smart clients, and mobile clients. During that time, his team shipped the Composite UI Application Block, and three software factories for mobile and desktop smart clients and for Web development. Before joining patterns and practices, he was an architect at Microsoft Consulting Services where he led the development of several enterprise solutions throughout the world. You can find his blog at http://blogs.msdn.com/eugeniop