Telecommunications, Media, and Technology

# From box to cloud: An approach for software development executives

By: Santiago Comella-Dorda /// Chandra Gnanasambandam /// Bhavik Shah /// Tobias Strålin

January 2015

As the world moves to cloud-based software, many software development executives wrestle with transitioning from packaged to cloud products. Pointers from successful software vendors can ease both the decision and ultimately the move.

Recent growth in cloud-based software as a service (SaaS) is expected to continue at 20 percent each year through 2018, when the global market could reach nearly $85 billion. Switching from packaged or "on premise" software to SaaS has a number of benefits that include improved user experience and lower delivery and support costs. It also enables companies to access new markets and incorporate innovative third-party cloud software.

At this point, however, SaaS remains something of an afterthought in the portfolios of leading software vendors. A recent report estimates that only 8 percent of the revenues generated by the top 100 software vendors originate from SaaS models—and seven of the ten biggest companies draw less than 5 percent of their software revenues from SaaS. Other research shows that the SaaS penetration in most software app categories remains low today, ranging from 1 to 36 percent. By 2018, however, its share should increase materially, achieving up to 72 percent penetration with some apps. While many vendors have yet to jump onto the SaaS wagon, a few that have been delivering SaaS experiences for years are busy upgrading their technical architectures to implement the latest generation of cloud technologies. These include new persistence and database models (in-memory or NoSQL databases, for example), faster analytical platforms, adaptive user interfaces, and elastic computing, among many others.

*SaaS penetration will double by 2018—and reach more than 20 percent*

As companies attempt to transition packaged software to SaaS or upgrade existing SaaS solutions to leverage new cloud architectures, they often face a number of challenges. Conversations with senior software development executives surfaced a number of concerns and questions regarding this transition.

What kind of cloud-architecture should they target? Should developers use public infrastructure-as-a-service (IaaS) or platform-as-a-service (PaaS) solutions or choose the private cloud? Do they need to rewrite their entire codebase?

How does the organization manage the transition to its target state—from legacy architecture to cloud-based services-oriented architecture? How long should the transition take and what are the key steps? Should the company wait until cloud and on-premise products achieve parity?

What changes should be made to development and operating models? Should development methods be changed? How could this shift affect software release cycles? Will the company have to change the way it engages with customers?

What capability and cultural shifts does the organization need? How should a company build the necessary talent and capabilities, what mindset and behavioral changes do they need, and how do they select the right development, IT, and infrastructure approaches?

A deliberative process begins with a careful consideration of which codebase type, architecture modification, and cloud infrastructure is most appropriate. Then—to ensure successful execution on the choice made—software executives will need to make several commitments regarding the scope of the product, the approach to development, and the allocation of resources.

## Choosing the right approach

Software companies who are considering making the switch to cloud software face three critical decisions, and their optimal way forward will depend on their main objectives and starting points.

The first decision concerns whether it makes more sense over time to choose a unified codebase for both packaged and cloud software or to have a separate one for each. Ultimately, this decision comes down to a few key factors. First, the organization's long-term vision is important when determining the ultimate purpose of the application. Is the team trying to build an optimized application for the cloud or is it attempting to leverage specific benefits of the cloud, while providing additional options to customers? The second issue concerns the maintenance costs for two codebases. In this case, how long does the company plan to continue with both packaged and cloud software products, and is feature parity required? For many software vendors, it seems likely that packaged software suites won't go away anytime soon. The final factor involves talent and culture. Does the team have the desire and attitude required to learn new technologies and unlearn past coding practices?

When a unified codebase makes sense. A unified codebase might be preferable if current customers view the cloud as just another channel. That is, the company does not expect all of its customers to transition away from on-premise software in the short to medium term (see text box, "When less is enough"). Or the company might not need best-of-breed cloud architecture to take advantage of the basic cloud benefits (including elasticity, scalability, and low-cost). A unified codebase works when the company has to maintain and manage multiple versions of the product. From a practical standpoint, another reason to choose a unified solution is that a company has evidence that its development teams are willing not only to learn new technology but to unlearn past coding practices as well.

When to choose separate codebases. Maintaining separate codebases for packaged and cloud software may be ideal when managers see the cloud as the key channel for future growth and expect to phase out the on-premise product. If customers expect the cloud-based product to be different in terms of look and feel compared with the desktop version and also expect it to include features provided by other cloud-based offerings (weekly releases, better scalability, and support for social tools, for instance), then separate codebases may also be the right choice. Other reasons to opt for separate codebases may be the fact that the company doesn't have to manage feature parity between both cloud-based and on-premise products, since it will soon phase out the latter or the software team must completely rethink the user experience and has the required skills to execute.

The second critical decision: companies can choose to refactor and "re-architect" on the go or build an entirely new architecture. When making this decision, leaders should consider two factors: the viability of the current architecture given the projected road map of the company's software products and the time-to-market requirement.

When to refactor. Refactoring is typically much faster and preferable if the current architecture might not be ideal for the cloud architecture but does have basic structural elements such as identifiable layers. It also makes sense if developers can port multi-tier applications to cloud architecture without undertaking a complete rewrite. Another point in refactoring's favor is when the company needs to release the first cloud-based version as soon as possible. Fully-refactored, services-based architecture can help drive frequent and small releases but is not a necessity to get started with the cloud-based product.

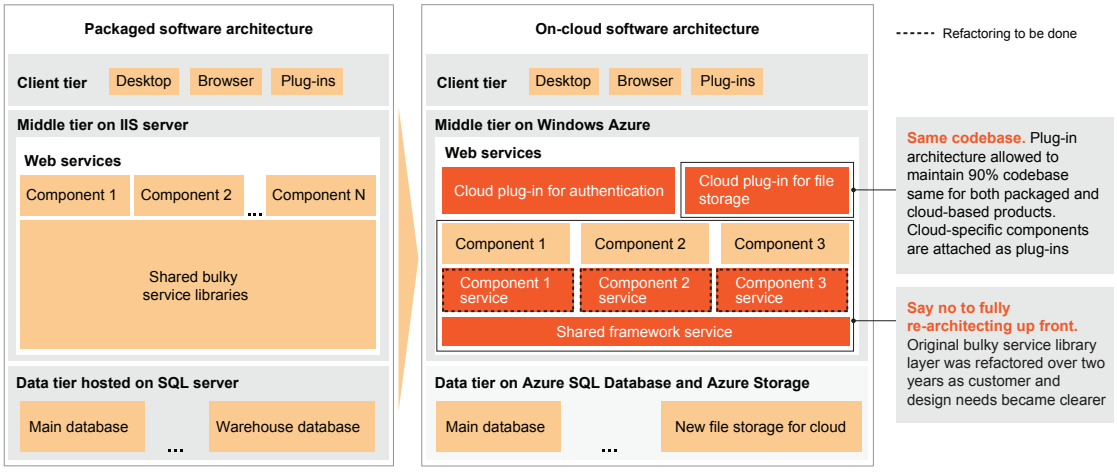
When to design a new architecture. Developing a new architecture makes sense if the current design is just not suitable for the cloud. For example, it might be a monolithic architecture that demonstrates symptoms of "spaghetti" code. Another consideration is the architecture's scalability. Sometimes an architecture originally built for on-premise isn't really designed to scale up to

# When less is enough: Software's measured journey to the cloud

One software company released the first version of its application lifecycle software suite nearly a decade ago. As is typical of many packaged applications, the product had a two- to three-year release cycle. Five years later, this company began to develop a cloud version to achieve some of the benefits the technology provides, such as scalability, elasticity, ease of deployment, and minimal up-front investment for customers. As the software team began its migration journey from packaged to cloud software, it made two key decisions:

**Use a single codebase.** The packaged version of its software will have a significant customer base going forward. The team decided early on that it would use the same codebase for both products and adopt a plug-in-based architecture for cloud-specific components. This decision allowed them to utilize 90 percent of the codebase for both versions of the software.

**Refactor as you go.** The packaged version is a three-tier application with the server running on the Windows platform. The product has a services-based architecture, but the services were not modular enough for a good cloud-based application. Since the team had not created the product in the cloud, it had difficulties making the transition. Team members chose refactoring in order to build a "minimum viable product" for the cloud and then continued refactoring existing codebase after releasing the product.




Source: McKinsey analysis

Team leaders highlighted several lessons they learned along the way. For instance, the use of advanced engineering systems and the team's "can-do" attitude were big transition enablers. They also learned that cloud-based products require three to four times more diagnostics capability compared with packaged software. Finally, they noted that the customer engagement model can be very different when product releases take place every three weeks instead of every two years. Today, approximately two million developers use the cloud version.

a larger number of users or it does not support multi-tenancy. Companies often build up related "technical debt" because of prior architecture decisions. For example, a payroll processing company decided to overhaul their current architecture to be able to move to open stack since portability is a key requirement for them. The company built some of the new system from the ground up, while tactically leveraging stable calculations engines and other components. Even while re-architecting much of the stack, the company didn't update a few mature components, including some on mainframe, since the risks of updating those components outweighed the potential benefits.

Another critical decision is the choice of public versus private cloud infrastructure. Companies can build their products on top of either privately hosted platforms or public IaaS or PaaS ones that rely on a service provider. This decision primarily concerns economies of scale, since the scale of infrastructure deployment, the company's tolerance for risk (data security or performance issues, for example), and regulatory requirements will ultimately drive it. IaaS platforms provide flexibility and control but entail the trade-off of additional complexity and the up-front effort required to build a user-ready service for them. Conversely, PaaS platforms often offer many capabilities that can help companies accelerate the transition to the cloud, but these platforms generally include proprietary or vendor-specific capabilities. As such, they require software created for a specific vendor's platform and stack, thus locking in those suppliers. While a small degree of vendor lock-in does exist with IaaS systems, it is relatively easy to plan around those areas.

*Data security and deployment scale are factors in making the public versus private cloud decision*

When to go for a private cloud. Private clouds work when the developer has sufficient internal scale to achieve a comparable total cost of ownership to public choices. That typically means it employs tens of thousands of virtual machines (VMs). It is also the right choice if at least one of the following four considerations is critical for the specific system or application and therefore precludes the use of the public cloud: data security, performance issues, control in the event of an outage, or technology lock-in. A final factor involves regulatory requirements that might restrict efforts to store customer data outside of set geographic boundaries or prevent the storage of data in multi-tenant environments.

When to choose the public cloud. Developers should consider the public cloud approach if the project lacks sufficient scale (will not involve tens of thousands of VMs, for example) or a high degree of uncertainty exists regarding likely demand. Using a public cloud is a more capital-efficient approach, since building a private cloud requires significant resources that the company could probably invest more effectively in the mainstream business. Another reason to go public: the system or application is latency tolerant. Experience shows that the latency levels on public clouds can vary by as much as 20 times depending on the time of the day. It also makes sense if there are no specific regulatory requirements that applications store the data in a particular place beyond performance needs. Even if companies decide to use a private cloud for their most critical applications, many decide to use public cloud for certain more basic use cases (dev/test workloads, additional temporary capacity, for instance).

## Six cloud-hopping design principles

Once executives have made their codebase, architecture, and infrastructure decisions, they begin developing their cloud-based software. To better understand how software players successfully make the transition, McKinsey reviewed a number of external cases and conducted in-depth interviews with leading software players. Those who succeed in the journey from on-premise to cloud software development share six commitments.

Shoot for the minimum viable product instead of feature parity. Organizations moving products to the cloud often discover that achieving full-feature parity could take several years. Instead, successful vendors often decide to release a minimum viable product (MVP) to customers in six to nine months. This strategy allows them to test their architecture and functionality. quickly. The approach also forces them to think deeply about which types of product functionality deliver sought-after core customer experiences and what they have to emphasize to get that functionality right. By putting a workable MVP with the most important features in user's hands as quickly as possible, the team is able to both gather crucial initial customer feedback and quickly improve on their cloud-based development skills.

Treat users as part of the day-to-day development team. Developers need to engage with their customers early and often—and shifting to a cloud model opens new ways of interacting with them. Teams can get feedback from customers in near real-time as soon as—or even before—they release a feature. User engagement also allows developers and product managers to ask customers to prioritize their needs via blogs when the product is in the concept phase and provide a basic product to specific early adopters. They can then codesign the full-featured version with them. Experience shows that collecting early feedback can help teams shape how they prioritize the features that are still in development.

Running the application centrally for all customers also opens up new capabilities. Developers can, for instance, employ logging and analytics to understand customer actions, taking a highly data-driven approach to tracking their usage patterns. Likewise, performing "A/B" features and functionality testing gives teams a data-driven approach to decision making. The 2012 Obama presidential campaign in the United States, for instance, used about 500 A/B tests on the interaction, copy, and images used on its Web page. The approach increased donations by nearly 50 percent and sign-ups by over 160 percent.

The cloud also enables teams to roll out functionality in a controlled manner (first 1 percent of customers, then 5 percent if all goes well, then 10 percent, etc.).

Expect and tolerate failures. Cloud infrastructure brings many benefits including the ability to grow or shrink resources for an application in real time. However, the shared nature of cloud infrastructure can pose challenges because of factors beyond the developer's control, such as hardware or network failures or slowdowns. And, as with all customer data centralized in the cloud environment, developers need to design an architecture for the application that can accommodate these failures and work around them.

*To be successful, companies need to develop a mindset that accepts failures that are beyond the developer's control and design an architecture to work around them*

For success, companies will need to develop a mindset that accepts failures. Without it, developers will hesitate to make changes, making release cycles grind to a halt. One Internet content provider learned this lesson the hard way after experiencing service disruptions due to a third-party Web services provider failure. In response, the company made its applications more robust in the face of such disruptions. Now, if similar problems occur, their apps are designed to provide a somewhat diminished customer experience rather than a complete crash. On top of this, to simulate random failures, the company created a special tool in the form of a script that will indiscriminately kill infrastructure services. This approach enables it to test application responses against failures that may eventually happen. It also helps teams learn about challenges specific to cloud-based development and incorporate customer inputs early.

Adopt agile and DevOps approaches. Companies should adopt agile thinking and DevOps,

a software development approach that focuses on product delivery, quality assurance (QA), feature development, and maintenance releases. DevOps builds on many "agile" concepts, like working in cross-functional teams and in short iterations all the way to deployment. Software executives also need to integrate their QA, operations, and security organizations with their R&D teams and schedule at least one release per month. Continuous integration, including integration into the main software branch, should be implemented with at least daily frequency. Releases should be scheduled as frequently as possible to ensure early user feedback. The release cycle can range from several releases per day to one per month. Once codebase is refactored into granular services, it is possible to achieve very short release cycles without destabilizing the entire product base.

The need for this shift goes to the heart of the differences between packaged and cloud software. With packaged software, releases are expensive because teams have only one true chance to launch a product. Consequently, releases occur once or a few times every 24 months. In a cloud environment, in contrast, most vendors find that incremental releases reduce the complexity of deployment and the magnitude of potential failures at the time of release. The incremental release approach leads to dozens of small releases for an individual product.

According to a recent study by Puppet Labs—a leading DevOps software provider—teams that embrace DevOps and continuous release practices deploy code 30 times more frequently, have half the number of production failures, and can restore services 12 times faster after a production issue. McKinsey research shows that embracing agile thinking increases team productivity by an average of 27 percent, boosts the timeliness of feature releases by 30 percent, and decreases the rate of residual defects by 70 percent.

Give developers QA and testing responsibility. Another hallmark of successfully moving from packaged to cloud software is the choice of companies to hold their software developers—not the code testers—accountable for quality. These companies seem to blur the boundaries between development and QA roles. The idea is to allow the software developers to resolve critical issues immediately as they become apparent. This approach requires them to deploy critical fixes continuously in addition to running short release cycles. It makes sense: despite the iterative nature of agile software development, cloud users will not accept or use apps with significant unaddressed issues. And it's also efficient: a developer can fix a bug introduced just two weeks ago much quicker than one that was introduced six months or two years back.

Another very important factor is that developers understand that fixing SaaS issues is fundamentally different from fixing problems with packaged software, which requires them to adopt new practices. It is normal, for example, to expect customers to take their servers offline to debug a problem for on-premise software. This is not an option for service-based software, since customers across multiple time zones are using the service. Building advanced diagnostics and tracing capabilities in the software is much more imperative for cloud-based software. Another similar example is NoSQL database adoption, which requires a significant unlearning of how developers worked with traditional, relational databases.

Invest in cutting-edge capabilities and automated test environments. McKinsey's observations of successful software developers suggests that hiring top development talent who can inject new external expertise into the organization at the operational and management levels is critical to making the switch to cloud software. Another crucial enabler is investing in tools and infrastructure to power the cloud-focused development model. The organization should shift all build, integration, and

*Box-to-cloud success requires ongoing, automated testing and integration and an advanced IT environment*

testing operations to a continuous and automated model that supports rapid release cycles. Leading cloud software providers—such as some of the world's largest search and social media companies—regularly build and test the entire codebase several times a day. Companies can reduce these intervals to as little as 15 minutes, but doing so requires a very advanced IT environment. Leading cloud players provide environment where developers can test code change against any of the portfolio products that could be potentially affected. This enables designers to conduct solid integration testing on their own before submitting the code for real integration.

□ □ □

Even if it generates significant buzz, cloud-based SaaS remains a relatively small part of most leading software developers' product portfolios. As the share of cloud-based software grows, developers will need to increasingly focus on transitioning away from packaged, on-premise software. Reaching carefully considered technology decisions and committing to several organizational and operational approaches to developing software as a service can help developers successfully transition from packaged to cloud software.

The authors wish to thank Buck Hodges, Engineering Director for Microsoft Visual Studio Cloud Services, and Roberto Masiero, VP ADP Innovation Labs, for their contributions to the article. The authors would also like to thank Alex Ince-Cushman, an associate principal, and Akash Shah, a consultant in McKinsey's Silicon Valley office for their contributions.

**Santiago Comella-Dorda**
is a principal in McKinsey's Boston office.
santiago_comella-dorda@mckinsey.com

**Chandra Gnanasambandam**
is a principal in McKinsey's Silicon Valley office.
chandra_gnanasambandam@mckinsey.com

**Bhavik Shah**
is an expert in McKinsey's Silicon Valley office.
bhavik_shah@mckinsey.com

**Tobias Strålin**
is a principal in McKinsey's Seattle office.
tobias_stralin@mckinsey.com