Three Principles of Application Architecture For the Cloud Managed Platform

By David Shilman
September, 2014

Enterprises are rapidly introducing Software as a Service (SaaS) in place of both purchased and custom-built solutions deployed in on-premises data centers. SaaS business software has the potential to significantly streamline organizations' business capability engineering efforts by empowering business stakeholders to perform functions such as software configuration and change management, reducing business units' dependency on in-house IT support. More than 60% of businesses utilize cloud technology for performing IT-related operations.

Cloud Managed Platform (CMP), or platform as a service (PaaS), which is the focus of this article, provides comparably time- and cost-effective enablement for in-house software development, and, if done right, can further empower IT organizations to operate more efficiently. Examples of well-known CMPs include Amazon Web Services, Microsoft Azure and Force.com. Skillfully planned, engineered and executed CMP adoption can also increase an organization's ability to manage technical debt. Unfortunately, technical teams often consider cloud architecture without exploring the risks and challenges posed by existing technical debt. This article offers three principles for facing those risks and challenges, and for maximizing the CMP opportunity.

Commonly accepted principles of modern application architecture require software applications to be highly configurable and portable in order to support an open and cost-effective technology platform with efficient and responsive configuration and change management processes. CMP ecosystems have compelled application architects to redefine the context of modern application architecture, and expand the definition of legacy system to include newer assets that do not meet organizational requirements or industry standards for agility and total cost of ownership. Failure to embrace the new principles of modern application architecture increases the likelihood of engineering legacy systems and contributing to technical debt. Here are the new principles:

1. Cloud-based distributed application components must have an HTTP-centric API
2. Sustainable cloud-centric Service Oriented Architecture (SOA) requires an SOA platform
3. Cloud-based applications must be designed and tuned for efficiency at scale

First Principle: Cloud-Based Distributed Application Components Must Have an HTTP-centric API

As today's technical leaders become more aware of technical debt accumulation, enterprise architects seek to reduce new application footprint by reusing existing application components and creating lightweight web applications that are faster to develop and deploy than traditional multi-tiered versions. Such application architecture often requires using services and APIs to expose existing CRM and ERP systems distributed across cloud and on-premise platforms. Such services must be consumed by other distributed application components as well as mobile and web apps, and therefore need to be easily discoverable and accessible.

Remote Method Invocation (RMI) has proven to be a performance- and cost-effective integration pattern in a traditional on-premise data center environment, where such application and data services are often deployed on the same hardware platform separated by virtual partitions, or at least on the same network. RMI patterns using common platform integration technologies such as EJB, COM+, JDBC and ADO.net may no longer be practical or feasible for applications and services deployed on distributed, public multi-tenant CMPs. These technologies require precise system and network awareness, i.e. tight coupling, that is generally unavailable or difficult to achieve on CMPs. Reducing such cross system, platform and network dependencies during cloud migration therefore requires complicated system re-engineering. Application architects should therefore seek open, cloud-friendly application integration alternatives prior to the start of cloud migration.

HTTP-centric SOAP and REST API integration has become application architects' predominant choice for effectively addressing the needs of distributed computing within and across cloud ecosystems, primarily due to its interoperability, reliance on open standards and extensive development support by all popular technology and cloud platforms (Figure 1). Furthermore, loosely-coupled, small footprint, distributed application architecture that promotes reuse of existing technology assets via SOA creates a technology foundation to enhance Agile software development. The benefits of this approach include:

- **Increased return on investment:** The lifecycle of the existing CRM and ERP systems is extended through integration with new cloud applications.
- **Reduced time to market:** Development of new user interface and API components requires less time than the development of new multi-tiered web applications.
- **More continuous integration and delivery:** Individual application components can be updated and deployed independently and more frequently with reduced impact to system uptime and business operations.
- **Multi-channel content delivery:** Single-page, responsively designed web apps can be easily accessed from variety of desktop and mobile devices with Internet browsers.

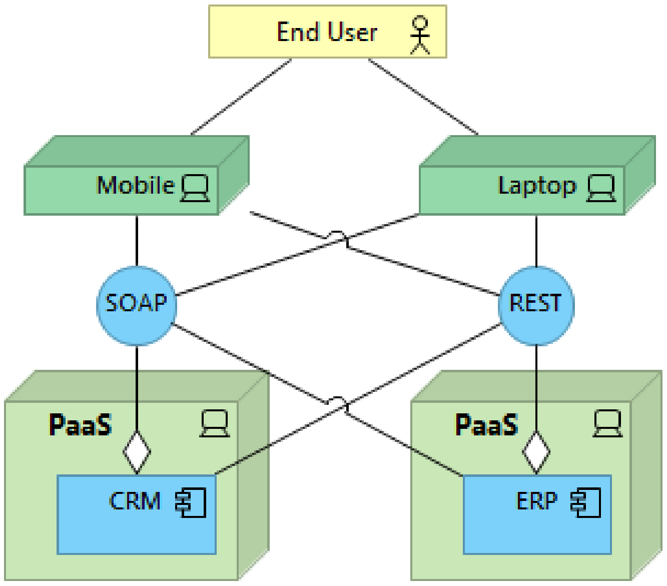

Figure 1 – Cloud-centric distributed application components exposed as standards-based services with APIs

Second Principle: Sustainable Cloud-Centric Service Oriented Architecture (SOA) Requires an SOA Platform

Point-to-point granular system integration is no longer sustainable or worthwhile in most enterprises. Coarse document- and Remote Procedure Call (RPC)-centric SOAP services offer design and performance efficiencies by presenting a business application such as an e-commerce transaction as a service with a well defined API. Such services implement atomic operations by encapsulating business logic, process and information management through an orchestrated workflow made up of granular application and data services.

As the number of service endpoints and service-oriented applications increases, so does the complexity of managing numerous point-to-point integrated solutions. SOA platforms allow consolidating and managing an organization's SOA capability on a single comprehensive platform. Examples of well-known SOA platforms include IBM Integration Bus, Tibco ActiveMatrix and MuleSoft Anypoint Platform. Such platforms further reduce awareness of domain service implementation, platform and network characteristics through service virtualization, also known as service federation. Many mature commercial and open source SOA platforms incorporate API and service management middleware such as an API gateway, enterprise service bus (ESB) and popular commercial cloud and on-premise CRM & ERP integration adaptors, all of which provide operational SOA capabilities such as (Figure 2):

- Security
- Service Orchestration
- Message-Oriented Middleware (MOM)

- Integration Brokers and Adaptors
- Business Activity Monitoring (BAM)

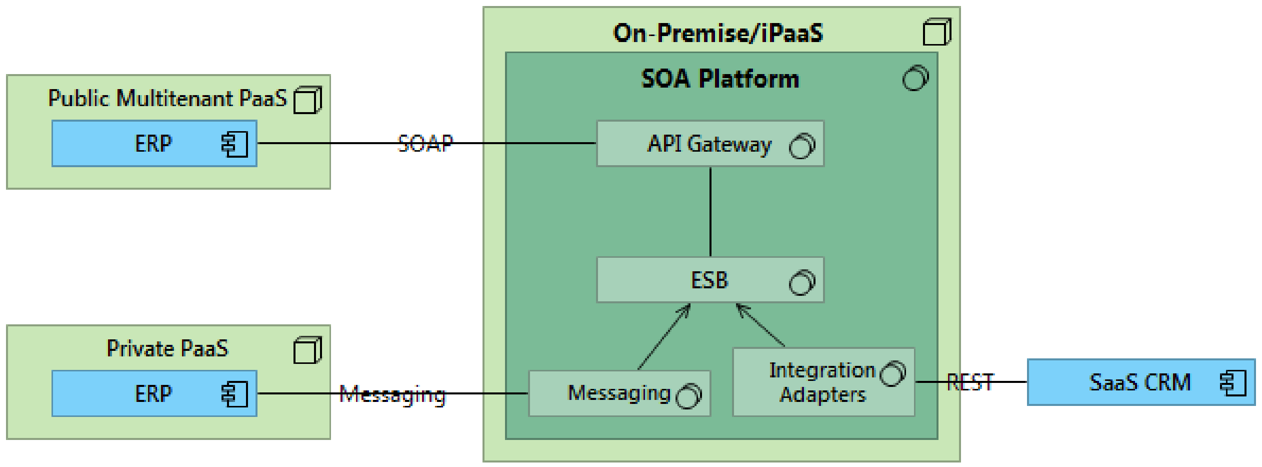

Figure 2 – Distributed systems integration through centralized SOA platform

Cloud-based Integration Platforms as a Service (iPaaS) provide an alternative to on-premise SOA platforms, and are gaining momentum as information and business processes management moves to the cloud. iIPaaS examples include MuleSoft AnyPoint Platform CloudHub and IBM WebSphere Cast Iron Live. iPaaS offers development and integration services that enhance information exchange with myriad cloud-based sources ranging from popular CRM and ERP SaaS solutions to social media outlets.

Third Principle: Cloud-Based Applications must be Designed and Tuned for Efficiency at Scale

It's a popular practice for many application architects to wrongly attribute responsibility for application scalability to infrastructure and middleware architects. Traditional architecture guidelines for high availability and scalability suggest application deployment on a redundant virtual and physical platform configuration with failover capabilities. Such environments can be highly cost inefficient, especially when system load is highly inconsistent throughout the business cycle, compelling companies to pay continually for peak resource requirements rather than for each usage. Popular PaaS providers offer public multi-tenant cloud hosting services, priced to reflect the usage of system resources rather than their capacity.

PaaS does not absolve application architects of their responsibility for optimal performance and scalability. Poor application architecture that doesn't efficiently utilize platform resources will not only negate the potential cost savings of PaaS, but may achieve the opposite effect. In order to achieve true efficiency at scale, solution delivery teams must engineer software that is designed and continuously tuned for optimal performance regardless of platform type.

Application performance tuning is a highly recommended practice even for software deployed in an on-premise data center. However, cost conscious IT organizations looking to embrace CMP as a cost reduction vehicle should make performance tuning a standard procedure built into their software development life cycles.

There are numerous ways to optimize a multi-tiered application for high performance and scalability. These include (Figure 3):

- **Static code analysis** tools can significantly reduce application technical debt by identifying code defects and poor coding practices that are likely to contribute to software poor performance.
- **SQL tuning** can help improve application memory and network utilization by reducing data scope in SQL statements. SQL tuning improves execution through retrieving and updating table rows based on indexed columns, and by optimizing database indices based on how data is selected and updated.
- **Batch update** uses SQL statements grouped together, and is a very effective technique for an atomic bulk data change. Developers can use stored procedures to encapsulate multiple data updates and inserts in a single call to a database. Both techniques improve application and database performance.
- **Optimistic concurrency control** reduces physical data locking in order to avoid deadlocks, and significantly improve database performance . This technique, which assumes that most transactions can complete without updating the same data, is incorporated in many popular open source Object Relational Mapping frameworks.
- **Dynamic application profiling** tools help visualize and analyze application execution paths during application runtime. Application developers use application profiling reports to identify code execution bottlenecks and application components responsible for poor performance.
- **Data cache** is a popular design pattern adopted by individual developers and by commercial and open source technology suppliers. Data cache saves expensive cross-network communication by storing frequently accessed static data in local memory.

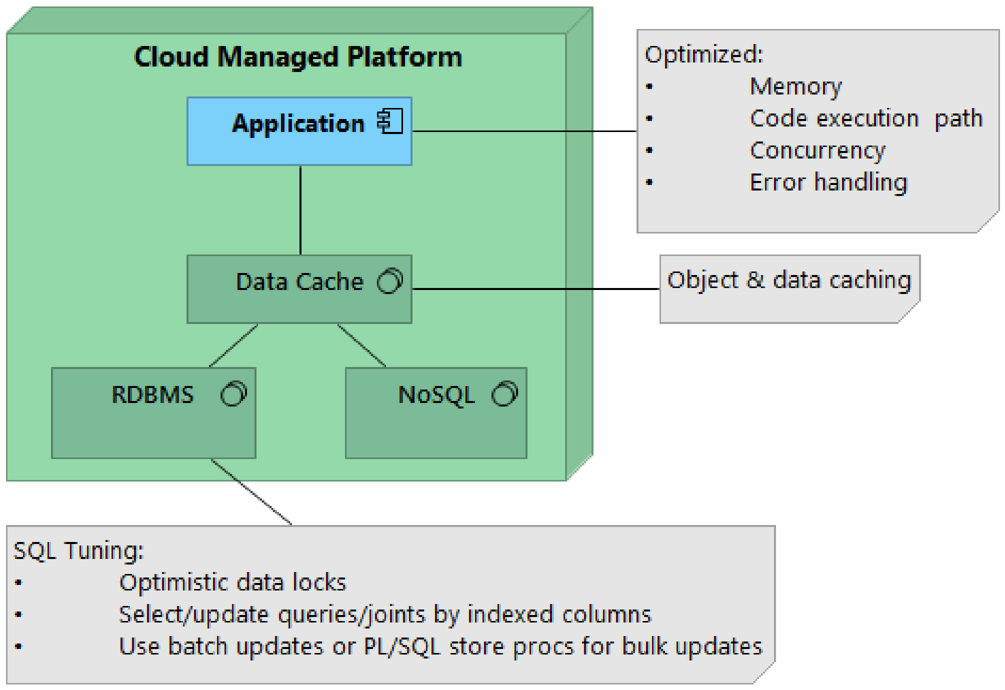

Figure 3 – Application tuning for optimal performance and scalability

Conclusion

Today, many IT organizations are beginning to embrace PaaS as a preferred platform in order to improve their software development and delivery capabilities. CTOs as well as technology and software architects need to be aware that the complexity of migrating their technology landscape to the cloud ecosystem depends on their systems architecture quality. Failure to incorporate cloud application architecture principles in today's software engineering will contribute to technical debt and make it more difficult to leverage cloud technology to deliver business solutions faster and at lower cost.

Many cloud vendors offer a cloud readiness checklist to their prospective clients as part of their sales strategy. Don't be fooled—cloud readiness assessment is an internal exercise that must be tailored for each organization to properly consider business, data, application and technology architecture. The best way to prepare for such assessment is to encourage the organization's architecture and software development staff to educate themselves about cloud architecture as it relates to their personal responsibilities and interests, identify target architecture attributes, and identify gaps in the baseline state.

Additional Resources

Cloud Patterns - http://cloudpatterns.org/

AWS Architecture Center - http://aws.amazon.com/architecture/

“Elements of an Effective Enterprise Cloud Computing Strategy” by Orbus Software - http://www.orbussoftware.com/resources/webinars/public/elements-of-an-effective-enterprise-cloud-computing-strategy/

"20 cloud computing statistics every CIO should know" by SiliconANGLE Network - http://siliconangle.com/blog/2014/01/27/20-cloud-computing-statistics-tc0114/

An earlier version of this article originally appeared in the Cutter Consortium Business and Enterprise Architecture Practice

About the Author

David Shilman is an Enterprise Architect with a Fortune 300 company. Throughout his career in the business technology sector, David has been an early adopter of technology and SDLC best practices in order to improve his effectiveness as a solution delivery leader, and to deliver complex technology solutions with most efficiency. David was an early adopter of Agile and Lean practices and methodologies. In his current role, David spearheads TOGAF awareness and adaptation in his organization. David has a B.S. in Mathematics, and holds advanced certifications in Enterprise Architecture, Project Management and Lean Six Sigma. David is a father of five, and enjoys playing sports with his kids.