eBook

Beyond ESB Architecture with APIs

How APIs displace ESBs and SOA in the Enterprise

By Ed Anuff

apigee

apigee

Table of Contents

Introduction

Powering interactions via apps requires a new architecture. Companies today seek to leverage the latest forms of mobile and rich-client engagement to build new experiences that can drive revenues or increase productivity. In recent years, apps have transformed from the means of content access and data entry to being the primary channels of interaction between a company and its customers and employees. This transformation has significant implications for enterprise architecture, which now must move from delivering web applications as the primary interaction channel to powering interactions in a secure, performant, and data-leveraged way across multiple interactive touch points, of which the web is just one.

ESBs and App Servers Are the Problem

The predominant web architecture in the enterprise is the application server. Whether WebSphere, Tomcat, or .NET, these were built to solve the problem of efficiently coupling dynamic web page generation with business logic in a monolithic stack for maximum performance. Most early web applications integrated with enterprise resources via the database tier, and, later, via transaction processing capabilities and integration via an enterprise service bus (ESB) to the company's service oriented architecture (SOA).

When web services in the form of SOAP and other standards emerged, they were initially implemented in the application server tier, partly because of the expectation that they would be accessed externally. Because the primary consumers of these web services proved to be internal systems, many architects came to view serving web services directly out of the ESB as a reasonable and efficient practice.

Modern apps and ESBs: an impedance mismatch

The challenge with coupling applications directly to the ESB is that there is a completely different set of assumptions for internal clients of the ESB that do not fit with the preferred architecture of modern mobile and HTML5 client apps or with highly scalable web tiers using elastic cloud capabilities. At a high level, it's essentially a case of impedance mismatch. ESBs are not designed for high concurrency, and modern mobile clients poll the servers more than traditional systems.

The ESB exists to shuttle non-optimized data on fast internal networks with largely coarse-grained security and little requirement for deep traffic analysis, other than for reasons of business activity monitoring of the health of the system.

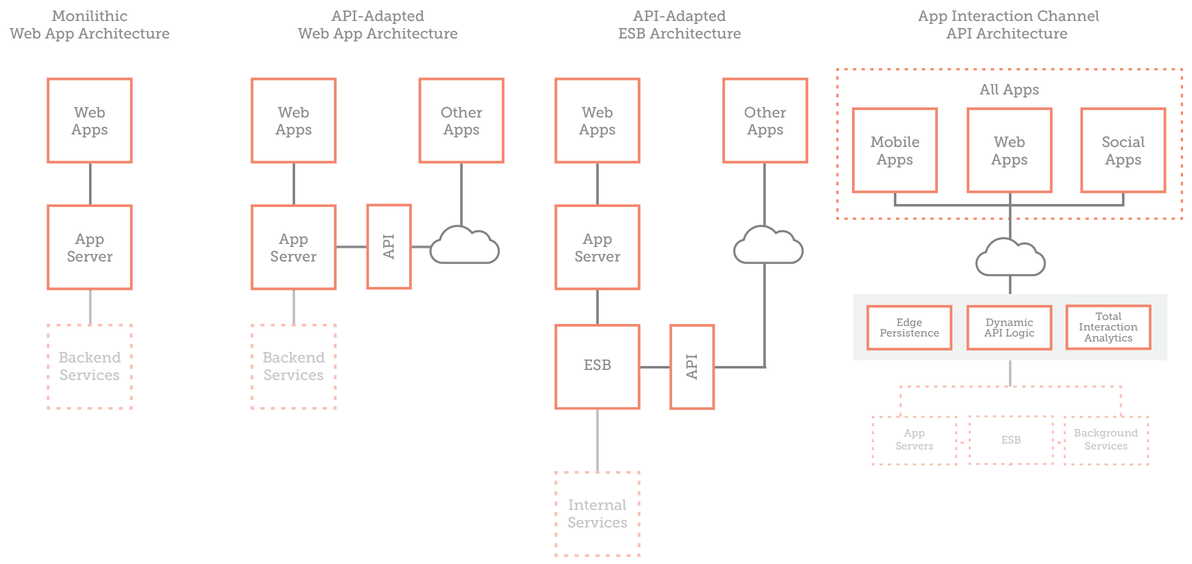
The predominant client of an ESB is a server-side application on the company's internal network, which has its access to other systems scoped to whatever level the developers of the connected applications and systems agree upon. The data is typically in large XML payloads that have low impact on latency in these fast internal networks; the powerful servers running the service-consuming applications can easily parse this data.

ESBs and App Servers Are the Problem

When external clients are added to the ESB, it's often under fairly controlled circumstances in the company's supply chain or other direct partners with both legal and programmatic contracts clearly defining rules for data usage and access patterns. In practice, most organizations opt for putting an application server with significant mediation logic between external clients and their ESB.

The best practice to enable external access by your modern mobile and HTML5 client apps is to build an API tier, either from scratch, or with the help of a vendor like Apigee, rather than trying to adapt the ill-suited ESB tier for this purpose.

The Case for the API Tier



Toward a unified app interaction channel

Moving past app server and ESB-centric architectures for total interaction insights

The goal of an API tier is to enable a large number of apps, from your partner channels or new app development teams, to access content and data from internal systems in a way that:

- **Grants individual users** appropriate levels of access control.
- **Shapes data to exactly** the size and format necessary for ease of app development.
- **Validates or processes data** with lightweight logic and combines it with or distributes it to multiple sources and services as necessary.
- **Enables deep analytics** to measure developer productivity and app and API usage growth as well as derive business-level insights by examining the interactions described within the contents of the API traffic.
- **Mashes up data** from external systems like “social” or “map” or “location” for more user-friendly behavior.
- **Caches frequently accessed data** thereby mitigating downstream processing needs.

Mobile apps: a new generation of apps

APIs are different because the new generation of apps is different in several ways.

Consider the humble email app and the frequency at which we check email on mobile devices compared to our behavior on older non-mobile email apps. This frequent access has significant impact on back-end systems.

Many new apps are running on mobile devices or in the browser within HTML/JavaScript-powered single page applications (SPAs). Even traditional model-view-controller (MVC) style server-side web applications are now different—the apps are running on elastically scaled app servers such as Amazon's Elastic Beanstalk, have little or no local state, and virtually every significant user interaction is backed by an API request.

In most cases, these apps run outside of the company's security perimeter. Because they must be considered compromised from the start, access to the company's digital assets often are secured with fine-grained access control based on the user's verified identity, rather than on the presumed identity of the application developer or the security of the application's execution environment.

Because of this, OAuth has become the preferred mechanism of securing API access, since it is issued against the identity of the end user after an authentication flow. Because it's end-user secured, access is based on the consideration of what actions the end user is allowed to perform, rather than the assumption that the application can be granted a wider scope of access than it exposes to the end user. From a security perspective, the app and the user are one and the same.

A qualitative and quantitative shift in the nature of data

There are significant differences in the data that is accepted from or delivered to today's applications.

Considering again the scope of security, it becomes evident that the data payloads transmitted on ESBs are often a mix of data that can be allowed outside the security perimeter as well as information that is for internal use only and should not be exposed.

Today's apps are often on remote and likely slow networks. The size of the data payloads being delivered is an important consideration to minimize the effects of network latency; the size and format of these payloads must be dynamically selectable by each different app at runtime.

Apps are often running in environments, such as memory-constrained mobile devices or in the JavaScript runtimes of browsers, where parsing large XML payloads will incur a significant performance hit. For these reasons, most application teams strongly desire that the data be shaped for their purposes. At a minimum, concise JSON communications is often preferred.

Further, valued marketing partners often use APIs to power business partnerships that involve expanding distribution and broadening market reach. Because these partnerships require reducing the time-to-market of the partner's app team to a matter of days rather than weeks, it's often important to quickly deliver a tailored payload in order to speed the application team's agility in delivering their apps.

Managing complexity and mass customization

The preponderance of such architectures inside an organization is a clear signal to the astute architect that an API tier, separate and distinct from the app server tier and ESB, is called for to manage the complexity and change requirements that will come from any serious level of API access via these mechanisms.

Modern API platforms and management solutions provide mechanisms for dynamically managing multiple versions of APIs, mass customized to meet business demands, in a high-performance runtime environment. These runtime environments provide configuration-based tools for performing data-shaping mediation and transformation activities as well as lightweight programmatic capabilities via the dynamic languages that have gained favor in recent years, including node.js as well as the enterprise-proven Java language.

"The preponderance of such architectures inside an organization is a clear signal to the astute architect that an API tier, separate and distinct from the app server tier and ESB, is called for to manage the complexity and change requirements that will come from any serious level of API access via these mechanisms."

While ESB products often boast graphical configuration tools, they don't enable the mass customization of these transformations in order to easily provide tailored API endpoints to large numbers of partners or disparate app teams.

Beyond operational metrics: building a data-driven business

The last missing ingredient in an ESB is analytics. This is often thought to mean just the operational analytics used by an IT team to ensure the health of the system or to audit its usage. However, once we get to the API tier and start to deal with greatly expanded usage, we realize that there is a much larger set of things to measure.

There are at least two important dimensions: the first is determining the success of the API program in terms of on-boarding developers and partners, driving usage, and delivering a rich set of innovative applications in an agile fashion; the second involves scrutinizing the API data itself to derive the business insights that can only be gained from a unified view of all the interaction traffic in the organization.

An API platform must provide the foundation of operational analytics if it's going to be suitable for use within enterprise IT architectures. However, many enterprises have multiple mechanisms for assessing system performance and health and having this level of analytics in the API tier is sometimes erroneously viewed as being redundant.

However, a good API platform provides a robust set of capabilities for measuring the key API operational analytics not available elsewhere—including access token failure rates, API-specific security threats, and performance gaps caused by the logic and systems behind the API tier. Because many capabilities, such as access tokens, are terminated at the API layer, often key data are not even available for analysis outside the API layer.

Such an API platform easily integrates and feeds into enterprise-wide systems. Because the ESB does not provide such information, often obtaining these data requires more custom logic in the code sitting in front of the ESB, and further dispels the notion that direct API access to the ESB tier is viable.

Understanding the channel

Completely missed by most ESB analytics is the fact that an API program is not just an integration strategy: it's a channel strategy, and measuring the success of the API program requires a unique set of metrics based on developer success.

The metrics studied to determine API effectiveness include the number of developers or partners registering to use an API, which applications are driving the most usage, and which API endpoints are receiving the most traffic; the list goes on.

Access to such analytics cannot be limited to the API provider; they must be viewable by the application developer. These are concepts that are not native to the ESB and, while they can be bolted on as an afterthought, they're typically not front-and-center for the managers of an ESB effort.

Implementing a web site or a commerce site today without deep analytics would be unthinkable. The same is true for APIs. The API is the application interaction channel representing interactions and, therefore, deep business-level insights. As a consequence, every user interaction performed on an app results (often in a one-to-one mapping) in an API request whose payload is the rich user activity data. For any type of application, being able to collect and analyze this data in a single place at any level of scale is an incredibly powerful capability.

"An API program is not an integration strategy, it's a channel strategy."

For customer-facing applications, this is game-changing. When implemented correctly, all customer interactions, whether via web, social, or mobile mechanisms, will pass through the API tier. This provides the single point of access where insights into customer activity across all touch points can be measured. On the other hand, by the time such data is split and stored in the various systems of record on the ESB, large swaths of actionable intelligence have been discarded and the data itself no longer resembles the original user interaction in any sort of useful way.

Summary

Although the ESB plays a critical role in the enterprise architecture, it is designed for a completely different set of concerns than are present at the API tier. It can be easy to confuse these concerns; they can appear similar in concept, though radically different in scale and granularity. Attempting to overuse the ESB tier to perform double-duty as an API tier results in the unintended consequence of increased reliance on ad-hoc app servers and custom code, as a rudimentary API tier is jury-rigged from custom logic.

In this scenario, needs such as mass-customizable security, mediation, and transformation for any number of distinct partners or apps are not met. This, at best, stifles the ability for application developers to execute in an agile fashion, and, at worst, opens up the ESB to additional vulnerabilities and Internet-scale performance demands for which it was not designed.

Lastly, an ESB solution completely misses the tremendous opportunity that comes from being able to derive deep customer insights from a single channel for application interaction across all user touch points. This eBook hopefully highlights for the thoughtful architect the challenges and false savings incurred by short-circuiting the separation of concerns between these two very different parts of the enterprise architecture.

About Apigee

Apigee is a leading platform for digital acceleration. Apigee empowers enterprises to gain the speed, scale, insight, and agility required to become a digital business. Through Apigee Edge API platform and Apigee Insights predictive big data analytics, Apigee helps businesses move at the new pace and scale of digital, while predicting and continuously adapting to change. Used together, APIs and predictive analytics create a powerful adaptive cycle of continuous improvement — and the faster an enterprise goes through this cycle, the faster it accelerates to become a digital business.

Many of the world's leading businesses, including 20 percent of the Fortune 100, use Apigee for digital acceleration. Apigee customers include global enterprises such as Walgreens, eBay, Shell, Live Nation, Kaiser Permanente, and Sears.

For more information, visit apigee.com.

About the Author

Ed Anuff is a leader in product and technology strategy at Apigee with direct responsibility for mobile and developer products. A respected technologist, a proven innovator, and an experienced entrepreneur, Ed has designed and created innovative consumer and enterprise products, defined product strategy at early-stage and publicly-traded companies, and founded and sold several technology companies.

Share this eBook