ISSN: 2755-0214

Journal of Economics & Management Research

SCIENTIFIC Research and Community

Review Article

Open Access

Building Enterprise-Wide API-First Strategy for Medium to Large Organizations

Nilesh D Kulkarni[1]* and Saurav Bansal[2]

[1]Enterprise Architecture Leader, Fortune Brands Home & Security, USA

[2]Sr Architect, Fortune Brands Home & Security, USA

ABSTRACT

In this paper, we provided a comprehensive guide on implementing an API-first approach within medium to large organizations. The basic framework starts by defining APIs and their role in modern business systems, using an example of two companies sharing order-related information through APIs. We covered the concept of API-first as a design philosophy, the growth of API usage, and the complexities faced by medium to large organizations in API adoption. The paper also discusses different integration patterns, the importance of understanding API and integration strategies, and the key elements for a successful API-first integration strategy. Additionally, it emphasizes the role of API gateways, the need for standardized API documentation, and the avoidance of excessive reuse of APIs. The document highlighted the importance of measuring API success using business KPIs. The paper concludes with the real-life example of the large organization with complex ERP and data systems and how the guidelines provided in the paper can help abstract the complexities with the help of API-First approach, finally the paper covered the security and data consistency as an important factor in defining the strategy.

***Corresponding author**

Nilesh D Kulkarni, Enterprise Architecture Leader, Fortune Brands Home & Security, USA.

Received: February 10, 2023; **Accepted:** February 17, 2023; **Published:** February 24, 2023

Keywords: API-First, Integration, Microservices, Legacy Systems, Governance, Scalability, API Gateway, KPIs (Key Performance Indicators)

Introduction

API which stands for Application Programming Interface is a set of rules, protocols, and tools for interfacing software or applications [1]. Essentially, an API specifies how software components should interact, allowing heterogeneous software systems to communicate with each other using a language contract. APIs are used for integrating applications, to enable modern web and mobile experiences, and to deliver new digital business as private API's to be utilized within the organization's secure boundary or public APIs in partner ecosystems. APIs abstracts the complexity of a system by providing a simplified version of a system's functionality without exposing the underlying code or logic. This makes it easier to integrate and use certain functionalities of a system exposing those functionalities using APIs without needing to understand the entire system

API within Business Context

How does APIs Work within Business System Context

Let's understand the role of an API, with an example Let's say two companies wants to share the order-related information using APIs, where company "A" is a good manufacturer which supplies the goods to a retailer company "B". The conversation between two companies' computer systems using APIs might look like this (see Figure 1: API in a business process flow context)

1. Company A exposed Order Management partner APIs for retailers to use and send purchase orders.
2. Company B integrates Company A's Order Management API into its procurement system.
3. Company B's procurement system sends a purchase order request to Company A via the API, The API request includes details like product IDs, quantities, and delivery information, that is required as per company A's order management APIs.
4. Company A's system receives the order and processes it.
5. The Company A's API responds to Company B with an order confirmation, including an estimated delivery date.
6. Company A has also exposed another partner API that can be called to request order status which provides real-time status on order processing, dispatch, and estimated time of arrival.
7. Company B regularly queries the API for order status updates.
8. Once the order is delivered, Company A's API sends a delivery confirmation to Company B by calling their Company B's APIs.
9. The company A generates an invoice and sends it to Company B for payment processing by calling Company B's APIs.
10. Company B calls company A's APIs to make the payment, which is then processed by company A and a payment success response is sent by company A's API.

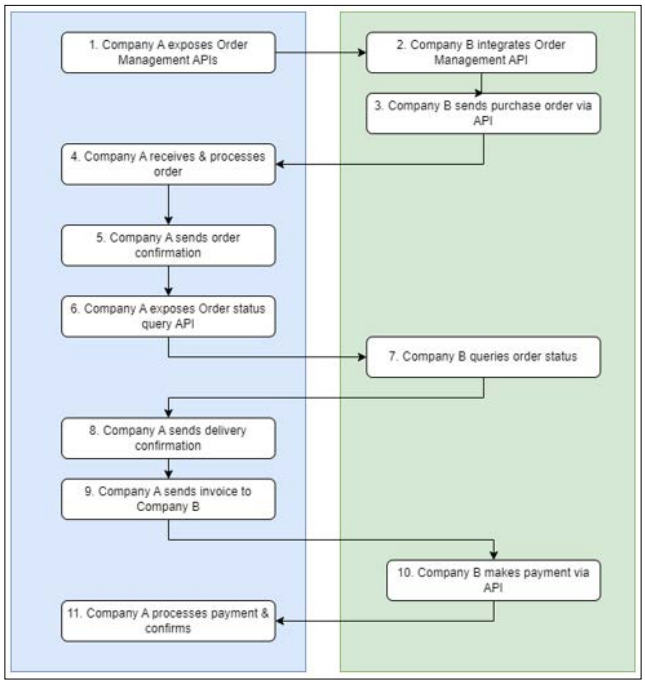

Figure 1: API in a Business Process Flow Context

In the Figure 1 API in a business flow context is an example of the APIs allowing two systems to borrow functionality and data from one another and become reusable building blocks to many business capabilities eliminating need of the manual interactions between two business entities company "A" and company "B".

API-First Approach

"API-first" means using APIs as the preferred method of accessing applications, platforms and data and is a design philosophy that prioritizes the creation of APIs at the outset by placing them at the core of the enterprise application's architecture.

APIs are a good option for standardizing interfaces and simplifying integrations because they improve accessibility to systems, services, or components, and are central to modern service delivery. However, the techniques of successful API-first integration implementations are not universally understood. The rapid increase in the number and use of Application Programming Interfaces (APIs) in software development and technology in response to the growing need for integration, communication, and functionality exchange between various software applications, systems, and services through increased adoption of SaaS and the rise of microservices development practices, has only raised the expectations.

Complexity within the Medium to Large Organizations

While APIs provide the flexibility to the organizations to share systems, services, or components it can also present unique challenges based on the size of the organization and compartmentalization of the work divided among many technical and business groups like

- **Integration with Legacy Systems:** Large organizations may have many older or legacy systems that weren't designed for API integration which can present a big roadblock.
- **Governance and Standardization:** Establishing uniform API standards and governance across various departments and teams is complex, requiring a balance between control and flexibility.
- **Security and Compliance:** Ensuring APIs adhere to stringent security protocols and regulatory compliance, especially when handling sensitive data, is crucial and complex.
- **Scalability and Performance Management:** As the number of APIs increases with multiple systems, ensuring that these API's perform efficiently and can scale to meet growing demands becomes challenging.
- **Cultural Barrier:** Shifting to an API-first approach often requires a significant cultural change within the organization, which can be difficult to manage.
- **Interoperability Issues:** Ensuring new APIs are compatible with various existing systems and technologies across the organization can pose a significant challenge.

Elements of Successful API First Integration Strategy

API Strategy Vs Integration Strategy is a crucial differentiation to understand while looking to adopt API-First strategy. While an API strategy is a comprehensive plan that outlines how an organization will use Application Programming Interfaces (APIs) to enhance business processes, customer experiences, and partner integrations by strategically defining the role of APIs in the organization's overall technology landscape. The Integration is simply process of linking together diverse computing systems and software applications, physically or functionally making independently designed systems work well together to act as a coordinated whole.

As a medium to large enterprise while defining the API-First strategy, it's important to understand the patterns of the integrations and how those are different from each other [2].

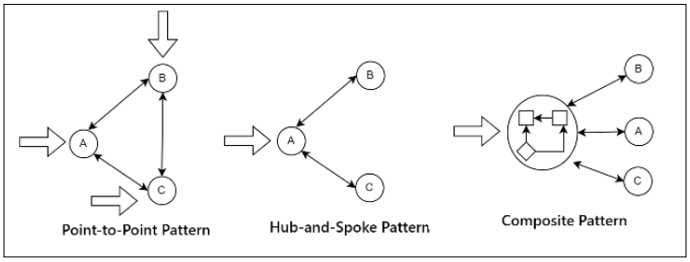

Figure 2: Three Patterns of Integration

- **Point-to-Point Pattern:** This is the simplest form of integration where each system or application is directly connected to every other system with which it needs to communicate. While easy to implement for a small number of connections, it can become complex and unmanageable as the number of connections grows, often leading to a scenario known as "spaghetti integration."

- **Hub-and-Spoke Pattern:** In this pattern, a central hub is used to facilitate communication between different systems or 'spokes'. Each system connects only to the hub, which then routes messages to the appropriate destination. This reduces the complexity of direct point-to-point connections and simplifies maintenance and scalability. However, the central hub can become a bottleneck or single point of failure.

- **Composite Pattern:** This pattern introduces a more sophisticated middleware layer, known as an Enterprise Service Bus, which standardizes and manages communication between different systems. The ESB decouples systems and provides services like message routing, transformation, and orchestration. This approach offers flexibility, scalability, and

the ability to integrate heterogeneous systems more efficiently.

While every integration pattern has the potential to address every integration need, each of the three basic patterns of integration (see Figure 2) emphasizes the need for different Best of Breed (BoB) integration capabilities.

Integration Capabilities Particularly Important for Point-to-Point Integration

- Flexible data provisioning and synchronization (e.g., change data capture) rules
- High-volume, complex data transformation
- Data governance support (e.g., access, obsolescence rules)

Integration capabilities particularly important for hub-and-spoke pattern

- Visual process modeling and improved user experience
- Stateful, scalable process runtime engine to orchestrate tasks
- Large adapter suite (for on-premises applications, SaaS, mobile, IoT, etc.)

Integration capabilities particularly important for composite pattern

- High productivity services composition
- Configurable API gateways (to expose well-managed services)
- Automated services definition generation (e.g., Open API WSDL, OAS/Swagger, WSDL)

Each Integration Capability Addresses Different Integration Needs

- **Application Integration Suites (e.g., enterprise service bus):** A general-purpose integration tool to integrate many applications and systems. Key features of application integration suites include communication functionality that supports protocol hopping and reliably moves messages among endpoints. It also offers: support for fundamental web API standards, functionality that dynamically binds consumer and provider endpoints, support for multiple interaction patterns, message transformation, asynchronous integration flows, content-based routing and typed messages.
- **B2B Gateway Software (BGS):** For integrating applications between companies, e.g., exchanging purchase orders from one company's procurement system and another's order management system. B2B gateway software can provide reliable, one-time-only communication between endpoints using managed file transfer and other protocols such as AS2, and even on-premises application integration in some cases.
- **Full Life Cycle API Management:** Tools to publish, promote and manage the usage of APIs, frequently in the cloud or on a scalable, on-premises environment. May incorporate some basic integration features, e.g., translation. Provides security when the data used or returned by the API is restricted or sensitive, and also provide additional features such as throttling, caching, tracking and analytics. It also includes API support documentation resources generally known as developers' portals.
- **Integration Platform as a Service (iPaaS):** A cloud-delivered application and data integration solution. iPaaS provides many of the capabilities addressed by ESBs and data integration tools, with an emphasis on addressing cloud services integration use cases, overall ease of use and SaaS-like delivery. Often used for a combination of data, multistep process and composite services integration. Typically includes at least rudimentary API management capabilities.

Building API-First Strategy

An API-first approach toward integration is using APIs to connect to a system, service or application as well as these APIs are well-defined and documented, and there are integration technologies to support the use of APIs to enable integration. The integration platform is responsible for connecting to cloud and on-premises applications, events and data sources using a mixture of APIs and other mechanisms such as native connectors, database connectors or via message queues.

Being API-first does not mean being API-only. There will be scenarios where other interface standards may be more appropriate, such as ODBC/JDBC , FTP , AMQP and MQTT . Based on the type of organization and level of system complexity a below guideline can be used while creating an API-First Strategy.

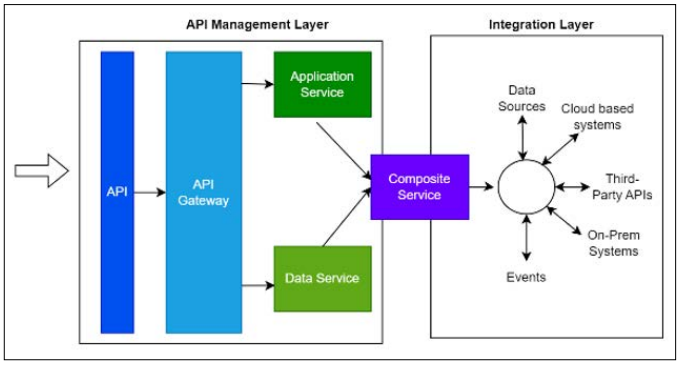

Figure 3: Reference Architecture for API-First Approach

An API gateway is a main component of API management, where it acts as a composition of the multiple API capabilities by exposing the underlaying systems securely and consistently. Whether you choose to create APIs indiscriminately or methodically, tracking the created APIs and protecting them from insider and outsider threats will be critical. It is imperative that the APIs created for internal or external purposes are published to the API gateway of choice, where necessary authentication and authorization policies can be applied.

API gateways are not an integration technology, but an enabler of integration. API gateways meets nonfunctional requirements like traffic management, load balancing, routing (mapping outer APIs to inner APIs) and enforcing security policies.

This composite service should not be implemented within the API gateway, as this will harm the performance of the API gateway and overload it with integration logic, rather it should be at the boundary of the integration services layer to eliminate future performance challenges.

Published APIs should also be added to an API catalog — and where they are intended for reuse, they should be published in an API portal and supplemented with standardized documentation to ensure your APIs are discoverable. In addition, you should collaborate with your security teams to ensure you discover your shadow APIs before attackers do. Conduct penetration testing on your APIs regularly, and implement reusable security policies to make your APIs bulletproof from a security standpoint.

Follow below checklist while developing the API standards across the enterprise to ensure API-First is governed

- Develop API Style guide to ensure technical consistency of APIs

- Establish standard and reusable security and runtime policies
- Build a community of practice in line with API best practices.
- Develop documentation standards ensuring appropriate documentation
- Centralize API gateway portal so that APIs can also be consumed in a consistent way

Stay away from too much reuse: Incorporating additional APIs into the integration workflow, apparently for reuse, can significantly extend the integration implementation timeline. This approach adds undue complexity and heightened latency. An API-first perspective entails prioritizing the API users' or consumer of API needs. If the sole user is an integration platform, then generating new APIs may be redundant.

Consider reuse as a secondary advantage of API-first integration, rather than the primary goal. When dealing with APIs and integrations, it's important to avoid overemphasizing reuse as the sole measure of success. Over-focusing on reuse can lead to counterproductive practices, like compelling developers to utilize APIs that don't align with their specific requirements.

Measuring Success of API-First Strategy Using Business KPIs

While defining the API-First strategy one must derive meaningful business KPIs from the business goals that the organization is currently trying to achieve. For example, typical commercial organizations define their goals in terms of increasing revenue, reducing costs, productivity increase and happy customers. The purpose of such metrics is not to provide organizational or financial reporting, but to define, quantify and create a baseline of values that technology leaders can use to generate awareness of the success of their APIs, platforms and strategies. This can also be used to track business value contribution and trending over reporting cycles. It is, however, important to avoid claiming direct credit for all (or part of) any revenue. Instead, use these metrics to create awareness and trending to gain and retain executive and business community backing, Some of the KPIs that can be used to measure the success [3].

Table 1: Measuring API Success with KPIs

Business Objectives	KPI to Measure API Strategy Success
Increase revenue	New revenue generated though API usage
Increase revenue	New revenue generated though API usage
Acquire new customers	Number of new customers joined partner API network
Productivity Gain	Cost reduction through automation enabled by APIs
Reduce capital spend	Avg. time to enable new capability using API
Reduce operational cost	Avg. number of defects closed/ time
Improve security posture	Count of API related breaches

The next step in the process is to identify which APIs to track against the identified metrics. Identifying the set of APIs designed to deliver the best business outcomes is key to measuring and tracking business value. It is crucial to apply the right metrics to the right set of APIs and to set the right expectations of desired outcomes. Organization may have many APIs deployed and contributing to various aspects of your business. These could be registered or unregistered APIs in one or more API catalogues. Typical API types include internal APIs, private or B2B APIs used between organizations, public or partner network APIs.

Few Important Considerations to become API-First Organization

Consistency and Discoverability of APIs: Make sure that data about certain business entities (e.g., customers, products, suppliers, employees, patients, citizens and assets) scattered across multiple databases and applications is in sync (e.g., the address of customer XYZ is the same across your CRM, ERP and billing applications). Create a federated API platform team in charge of API strategy, defining governance, operating the organization's API management platform, and the formation of a community of practice to improve the maturity and consistency of API designs [4].

API Security Considerations: API security can be divided into two broad aspects: API threat protection and API access control. API threat protection means detecting and blocking attacks on APIs, while API access control means controlling which applications and users can access APIs. To secure APIs, it is important to be aware of the ways in which your APIs can be breached like

- Unsecured API keys in repositories and storage. API keys or other keys, such as SSH keys or SSL/TLS private keys , may be discovered in cloud-based storage or in code repositories such as Github , which are left open to the public rather than being access controlled.
- Hardcoded API keys in applications. API keys or other credentials may be hardcoded in web and mobile applications and subject to decompiling attacks.
- API logic flaws, APIs may have bugs or other logic flaws which can be exploited
- Sniffed API calls. API traffic may be sniffed through a man-in-the-middle (MITM) approach, uncovering API keys or discovering unsecured APIs

A Sample Use Case Explaining the API-First Approach

The API-First approach, as highlighted in the Figure 4, is a strategic method applied within a complex organization which has many discrete ERP ecosystem, multiple data lakes, few third parties and multiple customers with heterogeneous information stored within various systems. As illustrated, there are many customers to the API ecosystem.

A Partner Customer (B2B)

- Need to place a purchase order for multiple brands associated with on or many ERP systems
- Need status of the order
- Request a replenishment of the partially filled order
- Get more accurate shipping information
- Need to inform about a broken package delivered to their warehouse

A Supplier

- Need to know the inventory of the raw material
- Request PO against the depleting inventory
- Understand the selling pattern, to plan for the internal production

A B2C Customer

- Place an order on a portal
- Receive shipment notification

- Make online Payment
- Receive refunds or add parts to the previous orders and so on.

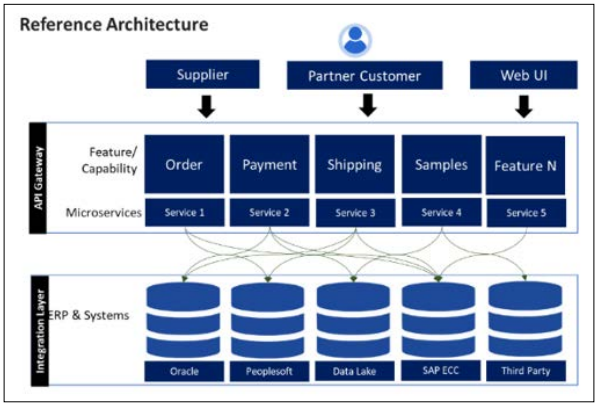

Figure 4: A Real Life Use Case Example

The API Gateway layer facilitates the capability exposure to the specific customer of the API based on the purpose, and allow the limited reusability of the APIs with security. This layer also has a microservices based construct to ensure every API service is modular and has a singular purpose, the feature composition is done within API Gateway layer to ensure separation of concern and reduced complexity from orchestration standpoint. The Integration layer is responsible to connect into each capability within the underlaying ERP, data systems or third-party APIs to abstract the complexity of the ERPs. It is also making the interface independent of the underlaying system, creating a uniform interface into the on-prem or cloud systems without exposing any system complexities.

The four guiding principles while designing the API-Frist architecture are

- **Microservices:** Microservices architecture is a design approach where an application is structured as a collection of loosely coupled, independently deployable services, each implementing specific business capabilities. Unlike monolithic architectures, microservices are small, modular, and can be developed, deployed, and scaled independently. This allows for agile development practices, easier maintenance, and rapid scaling to meet specific demands of different services within an application.
- **API-First:** API-First Architecture is an approach where APIs are treated as "first-class citizens" in the software development process. This means that the APIs are designed up front, before any code for the application logic is written, with a strong focus on the end-user's needs and the applications that will consume the API. This approach ensures that the API provides a well-defined contract, which can be used to guide the development of the application services that will implement it. It promotes a more decoupled, scalable, and flexible architecture, where the services can be consumed by various clients across different platforms, such as web, mobile, or third-party applications.
- **Cloud Native:** Cloud Native Architecture is a method of designing and running applications that fully exploit the advantages of cloud computing delivery models. It's characterized by containerization, dynamic management, microservices, and continuous delivery, enabling organizations to build and scale applications in modern, dynamic environments such as public, private, and hybrid clouds. Principles include automation, scalability, resilience, agility, and the use of DevOps practices. This approach emphasizes the use of services that are loosely coupled, managed by orchestrators, and continuously updated, allowing for flexible and resilient systems
- **Headless:** Headless architecture refers to a separation between the backend logic of an application and the frontend user interface. Essentially, the 'head' (front end, such as a website or app interface) is decoupled from the 'body' (backend, like databases and server-side processes). This allows developers to build and make changes to the frontend without affecting the backend operations, and vice versa.

Conclusion

APIs are a good option for standardizing interfaces and simplifying integrations because they improve accessibility to systems, services, or components, and are central to modern service delivery, but API-First approach ensure that the design approach is at the core of the enterprise application's architecture. While APIs provide the flexibility to the organizations to share systems, services, or components it can also present unique challenges based on the size of the organization and compartmentalization of the work divided among many technical and business groups

As a medium to large enterprise while defining the API-First strategy, the three patterns of integration need to be understood to identify where those patterns can be applied based on the specific use case.

A reference architecture outlining the components within API Management Layer and Integration layer and separation of concerns for medium to large organizations. While developing API strategy it is important to follow a checklist to ensure centralized governance.

Measuring the success of the strategy and impact of the strategy on business with the help of KPIs is very important.

At the end these elements are tied together, showcasing their relevance in a real-world scenario and underscoring the critical role of security and data consistency in formulating an effective API-First strategy.

References

1. What is an API? Available: https://www.postman.com/what-is-an-api/.
2. Benoit Lheureux, Keith Guttridge (2023) Choose the Best Integration Tool for Your Needs Based on the Three Basic Patterns of Integration. https://tech-prospect.com/technology/choose-the-best-integration-tool-for-your-needs-based-on-the-three-basic-patterns-of-integration-w/#.
3. Shameen Pillai (2023) How to Use KPIs to Measure the Business Value of APIs,Gartner. Available: https://www.gartner.com/en/documents/3980238.
4. Andrew Comes, Mark O'Neill, Shameen Pillai (2022) How to Organize Your API Development to Ensure Consistency and Quality. Available: https://www.gartner.com/en/documents/4015038.