Comparison of Enterprise Service Bus and API Gateway in Light of Microservice Architecture

Table of Contents

Service Oriented Architecture Trend and The Need For Enterprise Service Bus

In the 90’s, Service Oriented Architecture (SOA) emerged in the IT industry as an architectural approach to break down large, monolithic applications into a suite of web services. Like every new concept that gets initiated to overcome deficiencies and boost effectiveness of existing architectural approach, Enterprise Service Bus (ESB) became a favored middleware pattern to address the limitations of traditional monolithic systems. ESBs were used to integrate and compile various components, including:

- SOA based web services for handling of typical synchronous operations.
- Message bus, queue, or topic-driven subscription services for handling of mostly asynchronous operations.

Unlike the focused nature of a well-implemented Microservices architecture, Service Oriented Architecture (SOA) aims to make software components **re-usable** through service interfaces. This means that services in SOA are not necessarily aligned with a specific bounded business context but rather optimized for **reusability**. The goal is to integrate these SOA services into new applications without the need to duplicate their functionality in new applications.

At a high level, an ESB provides the following functions (depending on the vendor of ESB):

- **Message Bus & Queue based Routing:** An incoming message is sent to a destination either through explicit application determined queue end point or routed to a queue via based on ESB controlled/configured analysis of its payload.
- **Event Topic based Routing:** Related to Message Bus, but for cases where multiple interested parties can subscribe and be notified of events, an incoming message for an event is distributed to several endpoints via topics.
- **Message Transformation & Mediation:** An incoming message is transformed from one format to another (such as to/from XML/SOAP).
- **Protocol Mediation:** If ESB provides both message bus, and message routing, an incoming message sent from origin in http protocol can be redirected to another destination with a different protocol, such as; a message broker specific MQ protocol.
- **Audit:** When all message and/or service routing handled by ESB, ESB becomes the single component that can provide enterprise level transactional audit.
- **Security:** ESBs can also provide security services (including and not limited to) such as encryption, decoupling some aspects of security specific protocol overheads from SOA services.

In the realm of SOA based integrations, ESB has long been the preferred solution due to its robust integration capabilities as depicted below:

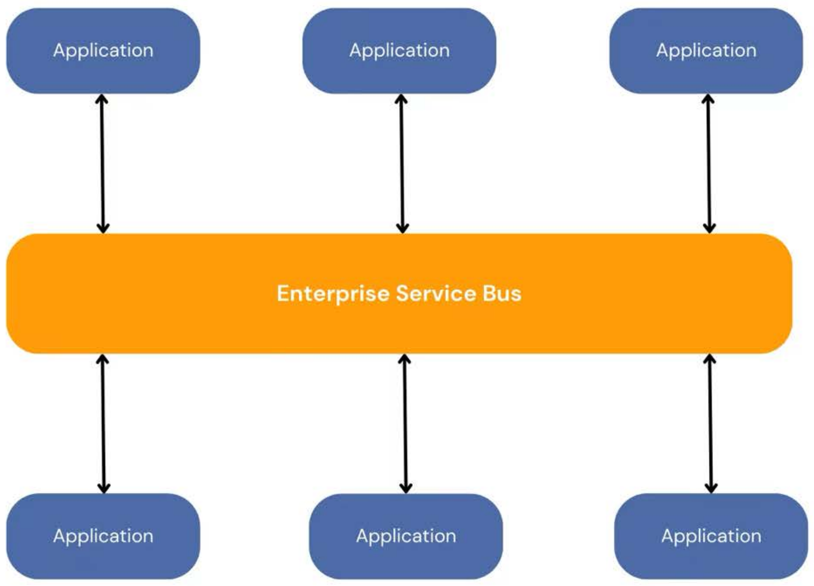

Figure 1: Enterprise Service Bus

Problems of SOA and ESBs paving up to Microservices Approach

As enterprises expanded and their automation needs increased, many SOA projects encountered challenges, leading to failures or the development of highly operationally expensive (OPEX) systems that deviated from the original goal of achieving business agility with SOA approach that favors reusability.

In instances of poor SOA implementations or even in initially successful SOA projects, in many caes the design deteriorated over the project's lifecycle. The following issues arose:

- Designing SOA services with main goal for **reusability** resulted in modeling the solution more as a set of technical reusable services and impacted the scalability of the enterprise architecture.
- When multiple business requirements from different business areas hit the same domain of reusable SOA service, it became practically impossible to scale for autonomous development and independent deployment. Plus, testing costs for time to launch were rising with an ever-growing fragility of the system, where an updated SOA service that works for one department, unexpectedly introduced defects for the other department.
- ESB itself was becoming the spaghetti hub and becoming the single point of failure and impacting operational scalability.

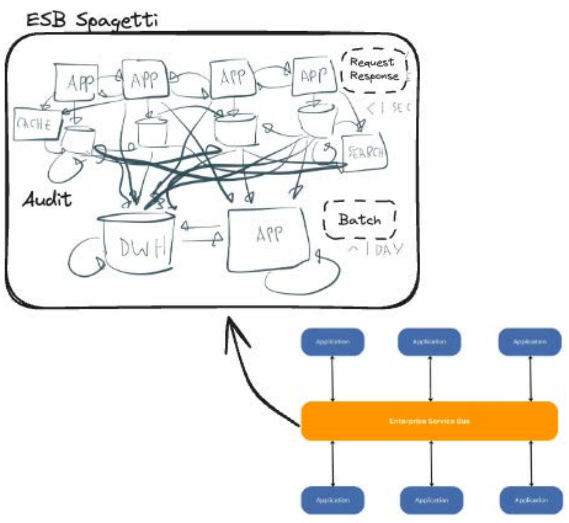

Figure 2: From Design to Real-life ESP

The first reaction to this came in early 2000s. At that time, it was so called DDD-Domain Driven Design (Eric Evans) which is an approach based on modeling the problem (aka Business) domain as an approach to designing the software. Thus, so called **bounded context** concept becomes part of software where business domain specific particular-terms and rules apply in a consistent way. With the analogy that bounded contexts are like fire breaks in a forest**,** if software is aligned with those firebreaks, then it means that the enterprise system is going to naturally be less coupled (against ever changing business demands), because different business domains generating demands for change are less coupled at those points.

From SOA/ESB to Microservices

This, the problems of SOA and ESBs not being able to scale up against enterprise business demands resulted in a new architectural approach when applying a distributed service-oriented approach to break the monoliths, so called Microservices.

Even though Microservices concept was stabilized in first half of 2010s, the concept became famous in Martin Folwer's and James Lewis's famous 2014 Microservices article.

One of the most repeatedly referred definitions of Microservices is:

"… an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms…", Microservices

Even though Microservices is also a service based distributed architecture approach, it is crucial to understand what is different than SOA. Microservices approach:

- Favors business scalability over reusability
- Aligned with bounded context and focuses on one task (responsible to a single business domain).
- Trades off code reuse where needed and resulting in the effect detailed in next bullet point when properly applied.

- Fosters autonomous development, and unit testing when simultaneous changes requests raised by different business teams, also boosting delivery scalability.
- Results in independently deployable business units driven services (aka Microservices)
- Thus results in a more loosely coupled set of business capability services.

Questioning Enterprise Service Bus with Microservices Approach

Every new concept that gets initiated to overcome deficiencies and boost effectiveness of a chosen an architectural approach. Now, let's take a closer look on the Enterprise Service Bus (ESB) facilities versus Microservices approach also considering the evolving technology standards.

- Technology standards and best practices evolved. When implemented with a web service driven approach, Microservices communicate (unlike most SOA based web services that utilize heavier weight XML based RPC style like SOAP protocol) over a very light weight standard http protocol, where the actions are http methods (POST, PATCH, GET, DELETE etc.) and the payloads typically encoded in JSON (http-REST).
  - So, no need for a centralized ESB for: **Protocol Mediation**, **Message Transformation & Mediation**
- Technology standards and best practices evolved. When implemented with an event/message driven service approach, Microservices communicate (unlike most SOA based event driven services that utilize proprietary and/or heavier weight messaging protocols: JMS variants, IBM Web Sphere MQ etc.) over light weight and distributed Message brokers such as; Kafka, Rabbit MQ, MQTT variants that utilize lighter wait messaging protocols.
  - So, no need for a centralized ESB for: **Message Bus & Destination Routing, Event Topic based Routing.**
- Technology standards and best practices evolved in security arena. There exists dedicated security standards and protocols such as OAuth, Open ID Connect with dedicated and NOT centralized components.
  - IAM-Identity Access Management Servers: that provide federated identity services out of the box and provide role association management for RBAC-Role based Access Control
  - Entitlement Servers: that provide enhanced fine-grained security for RBAC-Role based Access Control, UBAC-User based Access Control, A/PBAC-Attribute/Policy based Access Control
  - So, no need for a centralized ESB for: **Security**
- Technology standards and best practices evolved in operational monitoring/logging and auditing arena. There exists out of the box frameworks that enable independently deployable microservices to generate correlatable logs, provide health check monitoring end points and provide TPS, latency metrics. All this information can be collected in separate LOG Management, Monitoring and Audit Management components that result in even more scalable enterprise landscape.
  - So, no need for a centralized ESB for: **Audit**

Microservice Approach and The Need For New Components: API Gateways

We are in 2024 and worldwide many enterprise projects are favoring API-first, Cloud based microservices based approaches interwined with agile delivery approaches. Especially, with the rise of *microservices* and cloud-native architectures, *Thus, ESBs are gradually being replaced by architectures that align more closely with the agility, resilience, and scalability demands of modern business.*

In this evolving landscape, the API gateway emerges as a new choice for enterprise integration. **Positioned as a key separate and independently scalable component in microservices architecture**, the API gateway plays a crucial role in connecting, managing, and securing microservices, addressing the limitations of ESB considering more dynamic API focused Management requirements in distributed architecture based systems.

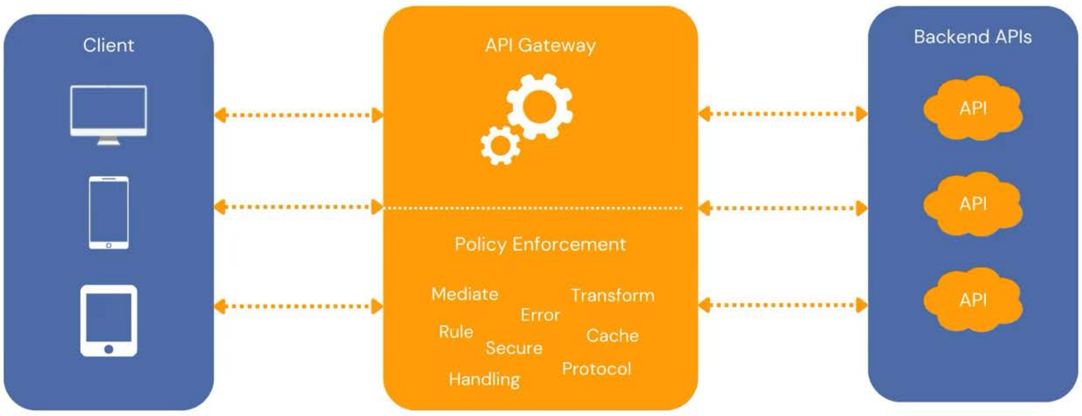

Figure 3: API Gateway

In microservices architectures, API Gateway performs mainly the following:

1. **API Routing, Versioning and Forwarding:**
    - Based on URL paths and HTTP-Methods, headers and/or query parameters, routes incoming API requests to the appropriate backend services.
    - Related to routing, it allows multiple versions of the microservice APIs to coexist and ensures backward compatibility via proper routing for the clients.
2. **Protocol Translation and Transformation:**
    - Translates incoming requests from one protocol (e.g., HTTP, HTTPS) to another (e.g., WebSocket, gRPC) as needed by backend services.
    - Converts request and response formats between different data structures (e.g., JSON to XML, XML to JSON) to ensure compatibility between clients and services.
3. **Traffic Control and Rate Limiting:** API Gateways manage traffic control and governance by enforcing rate limiting policies to prevent API abuse and ensure fair usage. They handle API throttling, prioritization of requests, and management of API quotas to optimize resource usage and maintain system stability.

4. **Caching and Performance Optimization:** API Gateways can implement caching mechanisms to store and serve frequently accessed API responses, reducing latency and improving scalability. They support caching at different levels (e.g., edge caching, server-side caching) and provide caching policies and expiration rules.
5. **Integration with Backend Services:**
    - Acts as a proxy for backend microservices, legacy systems, databases, and external APIs, abstracting the complexity of backend infrastructure from clients.
    - Orchestrates API workflows by composing multiple backend service calls into a single API request/response cycle.

For sure, similar to additional functions provided by an ESB, ***when microservices lack some of the capabilities,*** API GWs can also be used to quickly supplement some of the following functionalities by acting as the front-end to the microservice:

- **Request and Response Management:** Validations, handling of error responses.
- **Security and Authentication:** Implement authentication, authorization mechanisms such as OAuth, Open ID/Connect on behalf of the microservice missing this capability.
- **Logging, Monitoring and Audit:** Provide logging, monitoring, and audit log generation on behalf of the microservice missing this capability.

Conclusion

As the new digital era unfolds and business dynamisms becomes a must, and technological architectures continue to evolve with microservices, API gateway approach emerges as a solution for modern enterprise architecture and have clear advantages over ESB. API gateways with their lightweight design, flexibility, and focus on API management, make them the best candidate for the rapidly changing and developing business requirements of today, when it comes to API management.

In addition tp API gateways, technology standards and best practices evolved and some of the past centralized and multiple functionalities of ESB, are now offered as separate components that foster more independently scalable and functionality focused components such as:

- ESP (Event Stream Processing and/or Messaging) Platforms; Kafka, Rabbit MQ etc.
- IAM (Identity Access Management) Platforms: WSO2, Keycloak.
- BPM/Workflow Engine PLatforms: Such as Camunda where more stateful process oriented requirements exist (such as order management etc.) helping to keep the microservices stateless and independently deployable.

However, there is no straight answer to say that one implementation is the right choice. It ultimately depends on the requirements of your organization. For instance, if you are already running legacy software then switching to API gateway and microservices can be a slow and tedious process and then it might be better to stick with ESBs.

But when you are designing your enterprise architecture from scratch or extensively changing most of the elements, then microservices should be the way to go.