Powering the API world

EBOOK

API Infrastructure: ESB versus API Gateway

KongHQ.com

Content

Introduction

For the modern enterprise, providing delightful experiences founded on customer obsession has become an all-encompassing task. Forrester research shows customer-obsessed companies report 2.5 times higher revenue growth and 2.2 times better customer retention. Gartner predicts that digital self-service channels coupled with predictive analytics of customer behaviors will soon be the most valuable capabilities for service organizations.

However, implementing systems that put the customer at the center requires harnessing all of the organization's resources in a way that is greater than the sum of its parts. Thus connectivity is a key part of any modern IT infrastructure. Back in the day, the Enterprise Service Bus (ESB) was the primary provider of connectivity for a service-oriented architecture (SOA).

However, times are changing. Organizations are moving towards decentralization, replacing monoliths with microservices, and abandoning the idea of having whole organizations use a single tech stack in favor of having a diversity of languages, platforms, and solutions. They're liberating siloed data and modernizing legacy applications, and they're using APIs to do so. In addition, organizations are migrating their services and applications to the cloud.

In the midst of all this, connectivity to provide connected customer experiences is even more critical than before. API gateways are a modern solution for connectivity which free up teams to provide the innovation and efficiency that modern enterprises require.

In this eBook, we'll first look at why connectivity is becoming more important each day. Next, we'll cover the ESB — what it is, how it arose, and where it was used. Then, we'll take a look at another solution for connectivity — the API gateway — and compare it to the ESB. Finally, we'll discuss how present-day organizations can move forward.

Connectivity for Delightful Experiences

Reliable APIs

This world is becoming increasingly digital. Digital transformation, or the integration of digital technology into all areas of a business, is continuing its expansion. It is no longer simply the integration of a new piece of software or hardware, but a cultural change that fundamentally affects how organizations operate and what they target.

One of these new targets is customer obsession. It is no longer enough to simply say that you are a customer-first company. Customer obsession comes from having an organization-wide focus and commitment to placing customer needs at the center of everything that the company does. From sales and marketing to operations to support, every department needs to share this commitment.

From there, it becomes possible to generate delightful experiences for your customers.

In the age of social media, organizations that go beyond merely satisfying customers' needs and seek to actively delight them will gain outsized influence and mind share.

While none of this is new, each piece of it is reaching new heights. The focus and data needed to drive all this is unforgiving and requires the organization to relentlessly pursue it. Connectivity plays a key role in providing that delightful experience.

The importance of connectivity

In order to provide customers with great digital experiences, it is important to connect the sum of all your organization's interactions with the customer. This ability to connect marketing, sales, IT, and all your departments with regard to your customer is also known as Customer 360. By linking and synchronizing information about your customer from every angle, the organization has a more holistic understanding of the customer — order history, demographic information, loyalty data, complaint history, and more.

The data from this 360-degree view is then used to provide an omni-channel digital experience. The organization can then provide a delightful experience to the customer that is intricately personalized and perfectly in-the-moment. The customer no longer feels like he or she is interacting with a nameless bureaucracy, but is being treated as a whole person.

In order to generate this 360-degree view and the resulting omni-channel experience, connectivity is key. Every system and business process within the company must be connected and kept up-to-date in order for the customer to have a seamless, delightful experience.

The ESB and connectivity

Connectivity has always been important. However, in the past, the focus was on organizational efficiency. The Enterprise Service Bus (ESB) was originally envisioned as a connector for all the different kinds of services that existed.

In a service-oriented architecture (SOA) — which was still growing and evolving — the challenge of connecting different services with different standards and protocols was significant, and the ESB rose to meet that challenge.

In an environment becoming more centralized, the ESB became a significant part of enabling a monolithic enterprise architecture. With the need to connect large, on-prem applications and databases, the ESB enabled that connectivity. Let's look at how it achieved that task.

The Enterprise Service Bus

What is an ESB?

An ESB is a middleman that connects applications in a SOA. Instead of direct point-to-point or client-server communication, the ESB integrates with all endpoints. The idea of an ESB really took off in the early 2000s and has since grown to become an important part of large enterprise IT infrastructure.

Since there isn't a single standard for all ESBs to follow, this eBook will refer to the capabilities of ESBs generally. A given ESB implementation may not contain every feature, but there are large similarities across the ESB space.

The role of the ESB

An ESB serves to decouple the various services and applications that exist in an SOA-based IT environment. Each service must set up just a single integration with the ESB. From there, the ESB makes that service available to all other services connected to it, typically handling format transformation, protocol negotiation, queueing, and in some cases even additional business logic. In doing so, the ESB serves as a one-stop shop for any applications or services looking to consume or publish data.

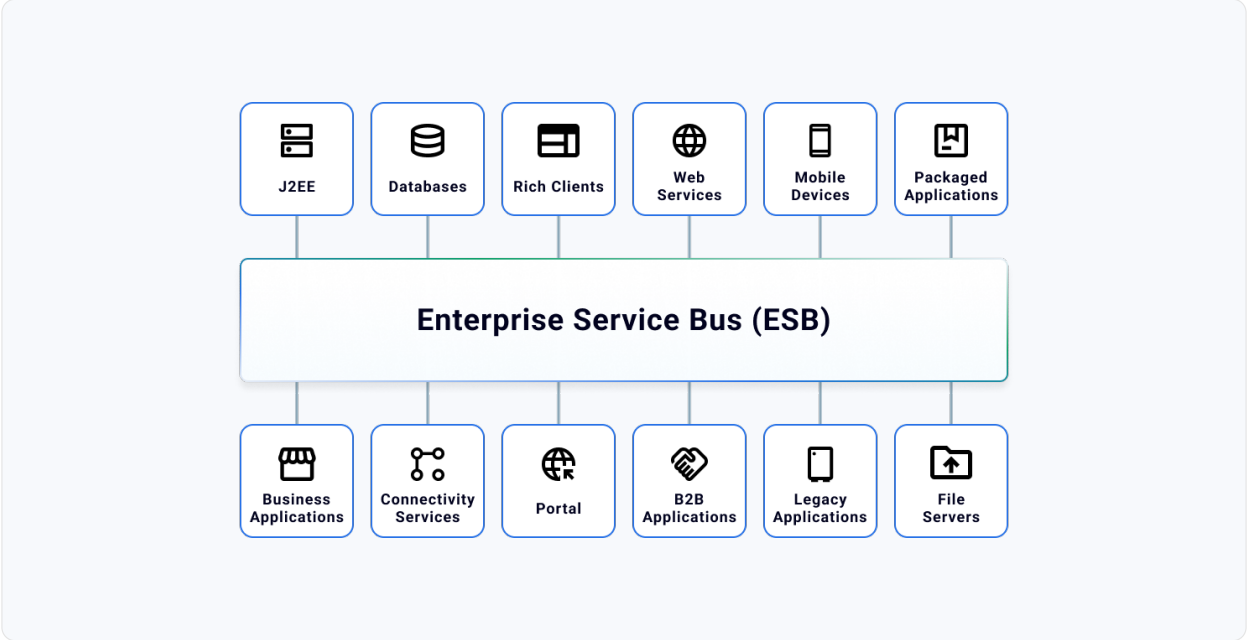

Figure: The Enterprise Service Bus

The key here is that the ESB is an intermediary for all service-to-service communication, which typically delivers the best benefits.

Centralized, on-prem architecture

In its role as a negotiator and mediator of inter-service communication, the ESB tends to become a central hub of IT infrastructure. The ESB provides many features that allow it to integrate with just about every service available, including legacy services.

As more services go through the ESB, the drive to bring every new service through the ESB increases. There is a "network effect" similar to that of large social networking apps. The more people that are on the network, the more draw there is for others to join.

As such, the ESB eventually becomes an essential monolith service of its own. Each integration with the ESB tends to go just a bit further and contains a little more logic. Before long, the business logic no longer lives in the individual services, but rather in the ESB.

The ESB team — central IT

As the ESB grows, it naturally requires more maintenance and attention. This responsibility typically goes to a dedicated IT team, tasked with the upkeep of the ESB. Since the ESB acts as a centralized hub for all service-to-service communication, the ESB team must function similarly, communicating and working with the various application teams. Coordinating new features and rollouts avoids breaking downstream dependencies

Benefits and drawbacks of an ESB

Now that we have a general understanding of the ESB, let's take a look at the specific challenges ESBs solve, the benefits of ESBs, and some drawbacks.

What challenges do ESBs solve?

In an era where service formats and even protocols were non-standardized, the ESB provided a way to centralize those integrations. Consider an enterprise context where you had five services. Without an ESB, 10 point-to-point integrations would be required for each of those services to communicate with one another. Add one more service, and the total jumps to 15. With each new service, the number of integrations required grows exponentially, as do the resources required for upfront development and maintenance. With an ESB, each new service only requires one new integration — directly with the ESB.

Additionally, since ESBs have centralized access to all services, ESBs provide orchestration benefits — that is, the ability to aggregate data from multiple services and present it as a single service. As the number of services grows and the needs of applications become more complex, orchestration allows simpler application development and interfacing.

Benefits of ESBs

One major benefit of an ESB is service discovery. Since services must connect to the ESB to access other services, the ESB also serves as a directory of the services in the organization. This is one of the benefits of the ESB's network effect: In addition to simple integrations, the ESB can serve as a continuous up-to-date source of documentation about any given service. It contains not only general information about the service but also technical details of how to consume and use the data from the service. Of course, this information is geared for developer consumption, providing engineering-level understanding rather than business-level understanding.

ESBs also add a measure of resiliency to the service architecture. As an intermediary to service communication, ESBs can add middleware-like functionality to that communication. Many ESBs serve as a message queue, allowing for the decoupling of services, as they are no longer required to be online at the same time in order to communicate. This makes inter-service communication more resilient, less prone to network outages, and able to withstand the failure of any individual service.

In addition to messaging, ESBs also add load balancing and transactions. Load balancing allows for increased service availability, and transactions allow for performing complex operations as atomic units with rollback capability.

Finally, as ESBs grew in prominence, some added even more features. Some features include adding business logic and rules in a centralized way — with no-code and low-code solutions available for those in the organization — and viewing and generating graphical modeling of the architecture. Additionally, more adapters were introduced for consuming large enterprise applications and even EDI communication. While each of these features served as a value-add, they also increased the scope and complexity of the ESB application. This leads us to our next point: How did ESBs fail organizations?

How ESBs fail organizations

While ESBs provide many benefits, they also introduce their own challenges and issues. Perhaps the most apparent challenge is that maintenance of the ESB became its own task, often requiring an entire dedicated team. As each service produces and consumes data in its own way, the ESB team becomes responsible for all the integrations with the ESB.

Additionally, with the growing functionality of many ESBs, the ESB team is required to take on development and maintenance of that new functionality as well. As such, while the ESB originally promised to decrease the complexity of the overall IT architecture, it introduced its own complexity and overhead.

More importantly, the centralization of the ESB results in a high coupling of teams and decreased team independence. Since every service needs to integrate with the ESB, the development team for that service or application needs to work closely with the ESB team on each change to the service.

Additionally, the growth of ESBs and the inclusion of business logic means that integrations are no longer merely about the communication or translation of information. ESBs become not just another monolith service, but a monolith service that all other services are required to work with.

This high coupling of teams results in slower application development, more overhead on every new service, and less agility for both individual teams and the enterprise as a whole.

The centralization of the ESB also made migration to the cloud a complicated endeavor. With services highly coupled and dependent upon the ESB for their business logic, organizations seeking to move to the cloud have few options beyond a simple "lift and shift" approach that keeps the ESB central but merely hosted in the cloud.

The future of ESBs

With these issues, what does the future hold for ESBs? Let's take a look at the modern IT landscape and how ESBs fit in.

The role of service connectivity in the modern IT landscape

In the modern IT landscape, service development has moved towards an API-first approach. We are no longer in the SOA world, and APIs have become much more standardized with regard to protocols. Additionally, development of the modern API has moved towards a spec-first approach. This means that teams that want to consume and integrate with a given service no longer need to wait for that service to be developed before working on their own application. With the technical contract already in place, development can happen in parallel

IT is moving towards decoupled, distributed teams. As organizations continue the search for agility and innovation, small development teams require increasing amounts of autonomy. To the modern development team, being coupled to a single point of integration or oversight is anathema.

Similarly, IT environments are becoming increasingly distributed as well. Organizations are no longer on-prem or even cloud-only; rather, they are working with hybrid-cloud and multi-cloud environments. There can no longer be points of integration that only work in one given environment; those points of integration must be able to span various types of environments.

Challenging the old ways of connecting via ESB

Finally, the move towards microservices is fundamentally at odds with the traditional, monolithic ESB. By breaking down the monolith ESB into multiple focused services, this retains many of the advantages while still increasing flexibility and agility.

With an understanding of ESBs and the shifts that are occurring in the modern enterprise, it is time to look at a new model of integration: the API gateway.

The API Gateway

What is an API gateway?

An API gateway is a modern infrastructure component between clients and services. The API gateway acts as a single point of entry for clients. This is in contrast to an ESB, which handles all inter-service communication.

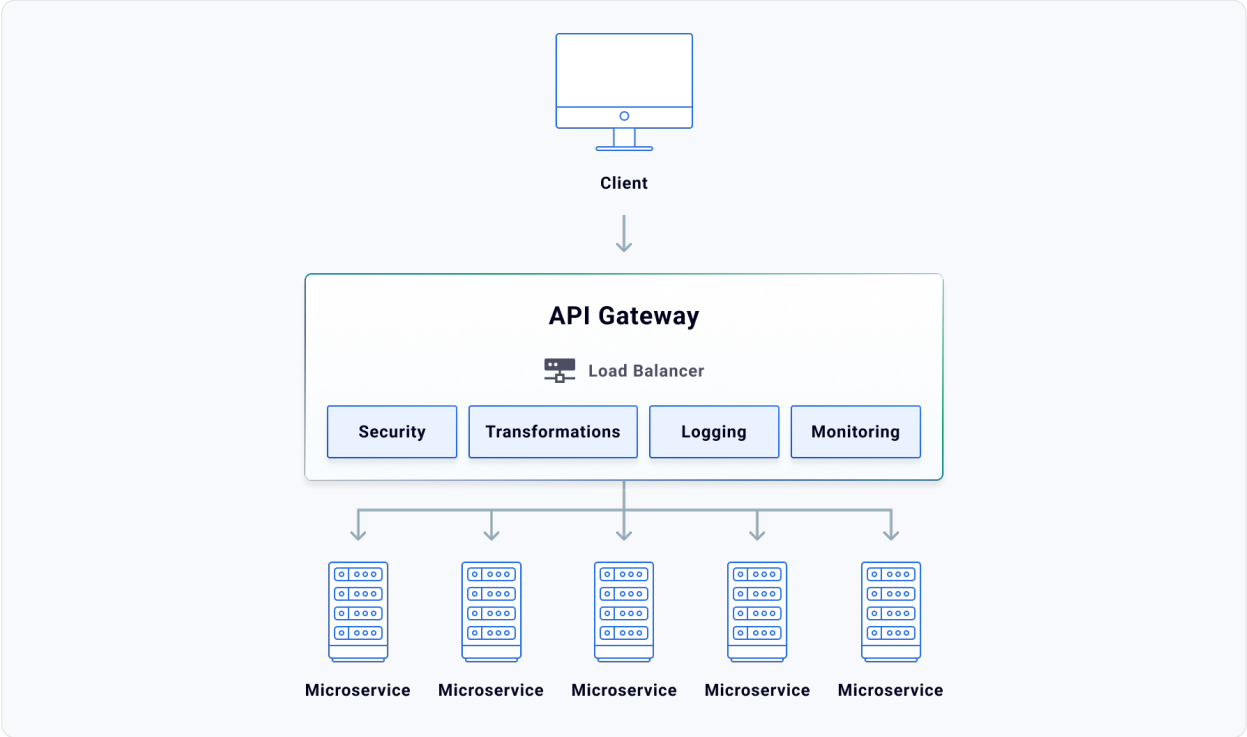

Figure: API gateway

Similarly to ESBs, API gateways serve the role of connecting together disparate services and integrating that information. However, with the rise of APIs, the task of connectivity is more focused.

A shift from service-first (ESB view) to API-first

APIs provide the standardized contract that was missing from the SOAP environment. In their original form, ESBs were a technical solution to a former standards problem. Additionally, with the advent of spec-first API development, the contract between client and service no longer needs to wait for the service to be developed, further decoupling development teams. In addition, this approach focuses on business requirements, prioritizing business and customer outcomes rather than merely cobbling together a spec for a backend system. API-first design leads to better reuse and relevance for business-led “products.”

Why is API Management so important to the modern enterprise?

With the advent of microservice-based architectures, the number of services to manage and oversee has increased dramatically. This is where API Management comes into play: It simplifies this oversight task for you by considering the entire API lifecycle — design, documentation or a developer portal, deployment, discovery and connectivity with an API gateway, and Day 2 concerns.

What role does the API gateway play?

The API gateway allows you to simplify the task of connecting to any given API. An API gateway handles cross-cutting concerns such as authentication, logging, and monitoring. Additionally, an API gateway can also handle orchestration to reduce roundtrips. Finally, the API gateway can provide the correct API for each client.

What benefits does an API gateway provide?

Now that we have a general understanding of what an API gateway is, let's take a closer look at its benefits and use cases.

Leaner microservices

First, API gateways allow the centralizing of common functions to reduce overhead. Rather than reinventing the wheel with every service, cross-cutting concerns such as authentication, logging, and monitoring can be handled at the gateway level. This makes each service leaner and reduces development time by allowing developers to focus on the actual business logic and competitive differentiation of each service. It also decreases overall system complexity, as these cross-cutting concerns can be implemented once in the gateway.

Decoupled clients and services

API gateways also decouple the clients from the implementation details of the services. This allows for the orchestration of multiple microservices into one client API. Similarly, differing clients can receive different APIs tailored to their needs, in a variation on the "backend for frontend" pattern.

API Discovery

API gateways serve as a natural place for discovering APIs. This speeds up the development of new clients and features. A developer portal working alongside an API gateway promotes API adoption and reuse by ensuring that APIs are highly discoverable, well-documented, and easy to consume.

Decreased number of requests required

API gateways can also increase performance by reducing the number of requests and round trips required for a given API call. Through orchestration, there are several API calls on the backend that can be aggregated into one roundtrip from the client to the API gateway and back.

This can improve the user experience. Through caching, API gateways can determine which requests can receive cached responses with fresh-enough data, thereby reducing unnecessary load on services.

Logging, monitoring, and analytics

Due to its position as a client-service intermediary, the API gateway is well positioned to provide logging, monitoring, and analytics data. Again, this is similar to an ESB, although

ESBs provide this for all service-to-service communication. API gateways only do so for requests between the client and the API gateway.

Consistency through plugins

The API gateway can take care of providing consistent, best practice governance, security, observability, and handling of all other cross-cutting concerns. This is most easily done through plugins and policies applied at the API gateway level. Aspects that the API gateway manages include:

- Authentication and authorization
- Traffic control and shaping
- Request and response transformation
- Security
- Logging
- Routing

How do API gateways compare to ESBs?

When comparing API gateways to ESBs, the similarities are clear. Both solutions occupy a similar place in the architecture: the centralized intermediary for communication with services. However, API gateways offer advantages as well as a more modern approach to achieve those advantages.

The main advantage is that API gateways have a clear scope. ESBs were envisioned as the end-all, be-all solution to communication between all applications and services. As they grew into that role, additional features were added, allowing for business rules and logic to be incorporated into the system. As such, the ESB became too convenient; what began as a project to reduce system complexity evolved into a massively complex system of its own.

On the other hand, API gateways play a more focused role. First, the API gateway is not responsible for (as much) transformation and protocol negotiation. As API standards have matured, the API gateway can be leaner than an ESB, focused specifically on cross-cutting concerns. Additionally, the API gateway is focused primarily on client-service communication, rather than on all service-to-service communication. Again, this focus allows it to stay lean and specialize.

This specificity of scope allows API gateways to avoid scope creep, keeping them from becoming yet another monolith that needs to be broken down. When selecting an API gateway, it is important to find a product with a clear identity rather than an extensive feature set.

In contrast to the centralized and highly-coupled nature of ESBs, API gateways allow for decentralization and distribution. This aspect empowers both kinds of enterprises — those that are on the journey to the cloud and those that are taking a hybrid approach.

When to use an API gateway?

API gateways are a good fit for the modern business that is focused on moving faster and enabling innovation. Let's consider why organizational culture matters.

Why is the right culture so important to the modern enterprise?

In the modern enterprise, the focus is on increasing agility and fostering innovation. This occurs through distributed teams with the independence and ability to do their work. In general, the shift toward decentralization is not only technical, but it is also cultural. As such, it becomes critical to select the right tool for the job — not just for its technical specifications, but also for the culture alignment that the enterprise wishes to foster.

ESBs fail this standard, as they are large, central monoliths leading to increased coupling between teams and decreased independence. In contrast, API gateways — thanks to their scope specificity — free up teams to focus on their individual tasks. By handling the cross-cutting concerns, API gateways enable teams to be leaner and more specialized.

API gateways — with developer portals — also foster a design-first approach to APIs and promote a discovery-led consumption approach. By providing the right API for each client, API gateways can enable increased adoption, reuse, and iteration velocity. This also facilitates consuming and discovering APIs across the organization and enables the use of no-code or low-code tools. Again, the focus is on the enablement of independent teams rather than coupling to the API gateway team itself.

How to select the right API gateway in a sea of options?

When considering the adoption of an API gateway, the selection of an API gateway is critical, involving several criteria for consideration.

Deployment complexity
First, consider the technical requirements of the gateway. How many pieces must be set up to get to a minimally viable installation? Many require setting up an additional database. The initial deployment complexity is also relevant when considering future maintenance requirements. Your deployment complexity also hinges upon your infrastructure approach — will your systems be in the cloud or on-prem or a hybrid? Will you leverage a container orchestration platform like Kubernetes?

Proprietary versus open source
Consider the extensibility of the platform as well as the probability of long-term support. Introducing an API gateway into your environment is a significant architecture-level change that affects many teams, and it is not something you want to be forced to remove due to the end-of-life situation of a third party.

Ease of scaling
Since the API gateway will be in the critical path for every client-service interaction, scaling is critical. Consider your growth plans and whether the API gateway can scale both horizontally as well as vertically. Scale management is best achieved by providing support for modern, orchestrated, and declarative approaches like Kubernetes.

Feature set
An API gateway, at the very least, must support the full API lifecycle. Beyond this, however, whether or not an API gateway has the most features should be a secondary concern. As we have seen from the ESB journey, scope creep and feature bloat lead to a less effective product. Consider carefully the role of the API gateway, and look for a product focused specifically on fulfilling that role, regardless of feature set size.

API gateways in the modern enterprise architecture

In the modern, API-first enterprise architecture, API gateways are a purpose-built fit. The proliferation of microservices means that there are an infinite number of services that need to be identified, discovered, documented, protected, controlled, and secured. API gateways provide an anchor and a point of reference that enable clients to interact with myriad services in a manageable way.

While avoiding the monolithic nature of ESBs, API gateways also provide some similar efficiencies from centralization, allowing your microservice teams to operate faster and leaner. The API gateway enables development teams to focus more, increasing their independence and velocity.

Finally, by avoiding the centralization of business logic, API gateways allow this logic to be distributed closer to the services that require it. This, again, avoids coupling teams to the API gateway and increases team independence.

Putting It All Together

Now that we have a clear understanding of both ESBs and API gateways, let's put all of this together. Perhaps you already have an ESB and are considering adopting an API gateway, or perhaps you are already in the process of introducing an API gateway into your architecture. How do you move forward from here?

The power of ESB + API gateway for the modern enterprise

The journey of the modern enterprise involves moving toward agility and rapid innovation in order to delight the customer. By increasing team independence and enabling those teams to remain lean and focused, this journey is possible.

To do this, IT organizations must become technically heterogeneous and diverse rather than homogenous. They must embrace the best-of-breed solution for each use case. This requires diversity in technical solutions and approaches. After all, the shift in direction is multifaceted. Here are some examples:

- On-prem or cloud-only → Hybrid-cloud and/or multi-cloud environments
- Centralized → Distributed
- Monolith architecture → Microservices
- Servers → Serverless, functions, Kubernetes, containers
- Organization-wide languages → Polyglot teams and organizations

With regard to integration platforms, the focus should now move to APIs. API connectivity is the new competitive battleground, and API gateways are a solution specifically for this purpose.

In most situations, a gradual hybrid approach is the best starting point. Start by implementing an API gateway with new APIs, and slowly bring over more services as opportunity and time allow. Over time, this gradual approach will break apart the monolith ESB. Take the opportunity to extract the business logic inside the ESB and distribute it into new microservices. The goal is not necessarily to replace the ESB entirely, as it still has a place with legacy services that may never get upgraded. However, the focus is on moving the ESB out of the critical path for new development.

Additionally, focus on those new APIs and the API Management capabilities unlocked by the API gateway. Generate value through connectivity, and the underlying implementation should naturally shift towards the place with the most value.

The long-term outlook

Every IT business should focus on generating value for the customer rather than any specific implementation. This is because the needs of the business will evolve and change over time. The systems you connect to and the data you consume will change as well.

However, in the mid-term, the API contract will be longer-lived by building business-level dependencies on top of them. As such, a focus on API connectivity will generate value for the modern enterprise.

Interested in learning more about how Kong's API gateway can enable your organization to become customer-obsessed? Learn more here.

Powering the API world

Konghq.com

Kong Inc.
contact@konghq.com

77 Geary Street, Suite 630
San Francisco, CA 94108
USA