How to Successfully Implement API Management

Published: 17 May 2019 **ID:** G00370408

Analyst(s): Kevin Matheny, Matt Brasier

APIs are fundamental to digital business; however, exposing API endpoints without protection, monitoring or analytics creates risk and limits value. Application technical professionals responsible for publishing APIs must implement an API management solution to provide these essential capabilities.

Key Findings

- API management (APIM) requires an API gateway for endpoint protection, but a gateway alone is insufficient. Gateways do not offer analytics, monitoring, developer support or governance capabilities.
- Implementing APIM without first ensuring that you have a support plan for the APIs you will manage risks failure. APIs must be supported by teams with knowledge of the systems those APIs represent.
- Successfully implementing APIM requires upfront planning to ensure that the solution you deliver will meet the needs of multiple parties, including API developers, API consumers, product owners and business stakeholders.
- Although API gateways offer mediation capabilities, using these capabilities to implement mediation such as multipart orchestration or complex business logic is unwise; it burdens the gateway and creates lock-in.

Recommendations

Technical professionals responsible for implementing an API management solution:

- Treat your APIM platform as a product, and staff and operate it as a long-term capability for your organization.
- Guard against postdeployment issues by clearly defining responsibilities for API development, onboarding and support. You should have a clear governance plan in place before deploying an APIM solution.

- Define strategies for testing, deployment, mediation, access control and monetization before beginning implementation to avoid misalignments and reduce the chance of unpleasant discoveries.
- Avoid implementing complex business logic in an API gateway. Instead, pair a gateway with integration platform software or iPaaS when APIs require complex transformations.

Table of Contents

List of Tables

List of Figures

Problem Statement

"How do I safely and securely publish APIs so that developers can find, understand and use them to deliver business value for my organization?"

APIs are fundamental to modern integration strategies. Whether you want to modernize an aging application portfolio, move to the cloud or pursue digital business in some other fashion, you will create and use APIs. These APIs will expose data and functionality that enable you to achieve your goals. You need to both protect and manage those endpoints to enable the delivery, promotion, operation, measurement and continuous improvement of APIs.

API management is, at heart, about curating and exposing your APIs to the consumers, either internal or external, who will use them to create value. To do this, you need key technical capabilities such as an API gateway, management console and developer portal. However, implementing API management goes beyond setting up an API gateway, or even a full APIM solution. In addition to the technical platform, to be successful with APIs and API management, you must also have processes in place to:

- Onboard new APIs
- Ensure API quality and usability
- Keep documentation up to date
- Coordinate the release of new versions of APIs
- Manage end-of-life and retirement of APIs
- Communicate with consumers
- Handle support requests and bug reports
- Report on API usage

This guidance framework should be used by technical professionals who are responsible for the delivery and operation of an API management platform for:

- Internal APIs
- External "private" APIs leveraged by business partners
- External "public" APIs used to drive new business

This document is a companion to "A Guidance Framework for Evaluating API Management Solutions," which covers the process for selecting an APIM solution, and "A Guidance Framework for Designing a Great API," which provides a detailed process for designing APIs.

The Gartner Approach

Implementing APIM is not about the APIs themselves. It is about creating the framework and capability that enable your APIs to succeed, which, in turn, enable your API consumers to succeed. Gartner's approach to implementing APIM focuses on the decisions and actions you need to take to deliver the platform that provides access to valuable business functionality and data. Our approach emphasizes information gathering and planning prior to implementation. This does not mean that we recommend a waterfall approach; we do not.

Plans are worthless, but planning is everything.[1]

Avoid the mistake of trying to fully plan out all stages of APIM implementation before beginning the work. Equally, avoid the mistake of starting without any planning or discovery work. Our research for this document found that clients frequently cited a lack of planning as a cause for issues both during and after implementation.

We recommend an agile, iterative process, where you build on what you learn as you go through the process. This document provides a framework that you can use to organize your implementation of APIM, focusing on clearly identifying the outcomes you need to deliver and then working to achieve them.

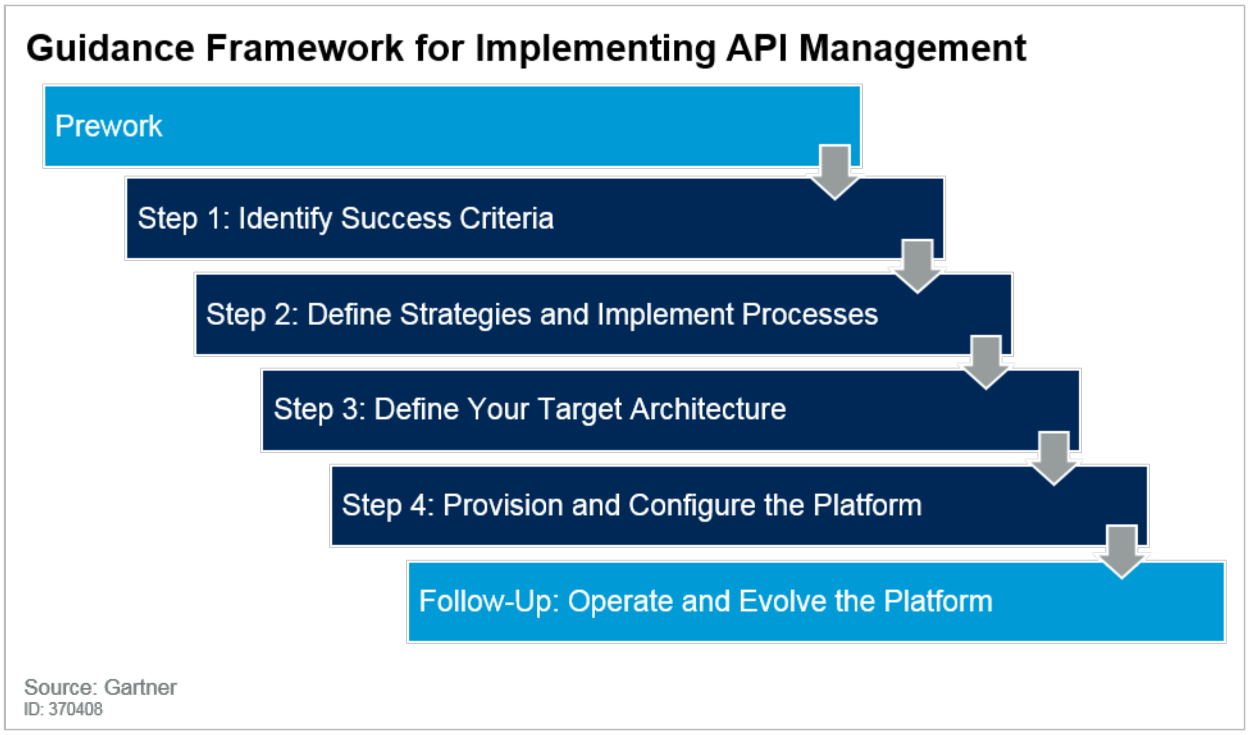
The Guidance Framework

To successfully implement API management, you should:

- Understand your organization's goals and criteria for successful implementation
- Define the key strategies for your APIM solution, and implement the processes that you will need to operate it
- Define the technical architecture of your solution
- Do the work of provisioning and configuring components, importing APIs and documentation, and setting up users and reports
- Operate your APIM as an organizational capability

This process is shown in Figure 1.

Figure 1. The Guidance Framework


Guidance Framework for Implementing API Management
Prework
Step 1: Identify Success Criteria
Step 2: Define Strategies and Implement Processes
Step 3: Define Your Target Architecture
Step 4: Provision and Configure the Platform
Follow-Up: Operate and Evolve the Platform
Source: Gartner
ID: 370408

Prework

API management as a capability is about publishing a curated set of APIs to developers. Implementing APIM is an investment in enabling technology. To make the most of this investment, you should clearly understand:

- Your business goals for both APIs and API management
- What technology choices have been made, how the chosen technology works and how well it fits your needs “out of the box”
- Your standards for API design

Understand Your Organization’s Goals for APIs and API Management

If you do not know what your organization wants from APIs and APIM, you will be working blind. It’s possible to have an implementation project that finishes on time, on budget and with the full agreed-on scope, but fails to deliver what the organization actually wants and needs. To eliminate this risk, begin by understanding what your organization wants from APIs and why you are implementing APIM.

Goals for APIs

Understanding the goals of the APIs that your platform will manage is key to understanding the capabilities that your platform must provide. For example, if your organization is expecting APIs to directly generate revenue, your platform's capabilities for monetization will be front and center. However, if your organization is relying on APIs for integration of critical business processes, your platform's reporting, rate-limiting and security capabilities will be in the spotlight.

The two primary business cases for creating an API are:

- **Value generation —** In some cases, organizations look to monetize APIs directly. In others, APIs are a means to acquire partners, enable new customer-facing or employee-facing applications, enable innovation or enter an ecosystem. For-profit businesses measure value in terms of revenue, while nonprofit or public-sector organizations measure citizen satisfaction, patient outcomes or other nonrevenue metrics.
- **Cost reduction —** This may be straightforward and near-term because the API replaces an existing higher-cost implementation, or longer-term because the API is intended to be the basis for multiple applications over time, amortizing its cost across multiple efforts. Even for an API that is used by a single consumer, creating a formal, well-documented interface makes it much easier for clients to develop and operate their software. Reduction in cost may be mitigation of risk, such as when an organization creates an API rather than a point-to-point integration, enabling the capability to be reused in the future.

Information that provides insight into the business purpose of your API can be found in several ways. Project documentation, including business-focused presentations and artifacts, can be a rich source of data. Business leaders responsible for APIs or API product managers can provide insight into the expectations that the organization has for the API. Key questions to ask when reviewing documents or interviewing leaders include:

- What is the core business purpose (value generation or cost reduction), and how does the API enable this result? What capabilities must the platform provide to meet these objectives?
- Is the context of delivery:
  - An internal API, where we own both sides?
  - An external "private" API with a small set of known developers?
  - An external "public" API with a large set of unknown developers?

If the initial answers to questions about the purpose of APIs indicate that your organization is seeking "agility" or "flexibility," dig deeper. You need to know what benefit that agility or flexibility will deliver for the organization.

Goals for APIM

As noted above, APIM is about exposing a curated set of APIs for use by consumers. However, you are adopting APIM for a reason, and your organization has a set of expectations about the outcomes you will deliver. Knowing the expected benefit is crucial to delivering success. Example goals for adopting APIM include:

- **Assert control over existing APIs —** You will want to focus on processes to discover and onboard APIs, as well as put rate limits and policy enforcement in place quickly.
- **Create consistency across existing APIs —** You will want to focus on API design standards and API mediation to create a consistent API consumer experience.
- **Make discovery and usage of existing APIs easier —** You will want to focus on creating an easily navigable API catalog, providing good documentation and creating a smooth process for gaining access to APIs.
- **Enable creation and publishing of new APIs —** You will want to focus on outreach to API development teams, processes to onboard new APIs and guidelines for both API design and API documentation.
- **Develop new business opportunities with APIs —** You will want to focus on analytics and consumer-facing support and outreach. You will also want to identify how you will share usage data with product owners for use in prioritizing features and building new APIs.
- **Apply policies for secure API access —** You will want to focus on creating reusable security policies, working in conjunction with security and identity management teams.
- **Managing consumption of third-party APIs —** You will want to focus on creating access policies and identifying which external APIs require management.

Select API Management Technologies and Conduct POCs

This guidance framework assumes that you have chosen an APIM product to implement. If you have not done so, you should use the process in “A Guidance Framework for Evaluating API Management Solutions.”

In addition, we assume that you have conducted a proof of concept (POC) implementation using your chosen APIM technology to validate that it meets your functional and nonfunctional requirements. Examples of things to validate in this POC:

- Ability to handle the protocols and formats of your existing and planned APIs.
- Connectivity to your existing identity and access management (IAM) solution.
- Connectivity to your API endpoints, particularly if these are within a data center.
- Usability of the administrative and developer-facing interfaces by your target users.
- Integration with your existing monitoring solutions for uptime and performance.
- Compatibility with your security policies.
- Ability to implement the traffic management policies and patterns you intend to use.

- Deployment of on-premises components. You should clearly understand the hardware and operating system requirements of any on-premises components, and discuss these with operations and security teams.

As you conduct this POC, you are likely to discover gaps between the capabilities of your selected APIM product and your requirements. No APIM product meets all needs. For any such gaps, you will need to determine if you can forgo that requirement, or if you will need to find another means of meeting it, such as using an additional product or writing custom code.

Locate or Define API Design Guidelines

To make use of an API, consumers need to understand not only the data or functionality offered, but also the API's format, versioning, pagination strategies and much more. Each aspect of an API that consumers need to learn increases the amount of time needed to understand and use it. API design guidelines provide API developers with the information they need to create APIs in a consistent fashion. This increases the usability and, therefore, the adoption of APIs.

In addition, your API guidelines provide you with valuable input into the capabilities that your platform will need to support. These guidelines should describe the protocols and formats of your APIs, as well as versioning strategies.

Your organization may have an existing set of API design guidelines. If it does not, you should develop these guidelines in partnership with the API development teams. It is better to start with an existing set of guidelines and adapt them for your circumstances than to start from scratch. For more information on creating API design guidelines, including a checklist of items to include in your own guidelines, see the section "Step 1: Establish API Design Guidelines" in "A Guidance Framework for Creating Usable REST API Specifications."

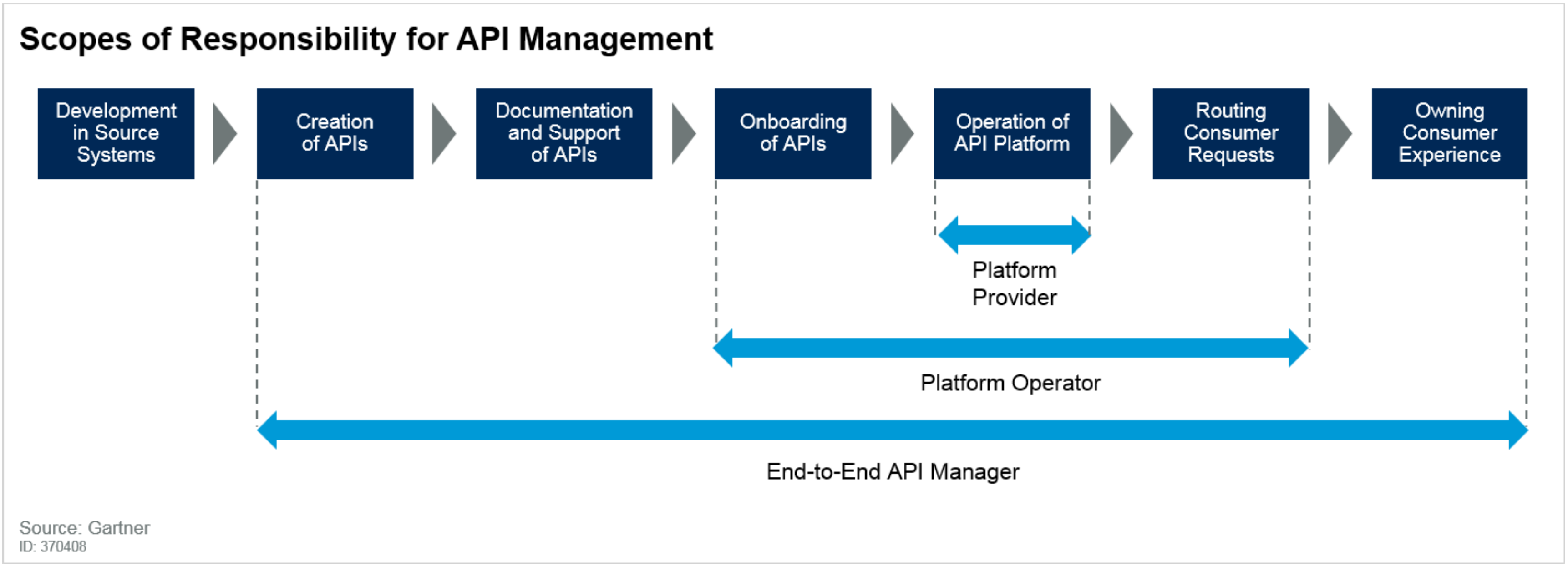
Step 1: Identify the Success Criteria for Your Implementation

You should begin the process of implementation by ensuring that you know the criteria for success. This means understanding the scope of your own responsibilities and the capabilities you will need to deliver, in terms of both technology and processes. Note that this is not about making detailed plans for implementation; it is about understanding what success and failure will look like.

Determine the Scope of Your Responsibilities

The first and most critical thing to determine is your own scope of responsibilities. This will drive all of your other decisions and actions. There are three common scopes of responsibility, as shown in Figure 2: platform provider, platform operator and end-to-end API manager.

Figure 2. Common Scopes of Responsibility


Scopes of Responsibility for API Management
Development in Source Systems
Creation of APIs
Documentation and Support of APIs
Onboarding of APIs
Operation of API Platform
Routing Consumer Requests
Owning Consumer Experience
Platform Provider
Platform Operator
End-to-End API Manager
Source: Gartner
ID: 370408

Although this is not a decision you are making, you may be able to influence it. Work with your leadership team to understand their expectations of you, ensure they are aware of the consequences and the resources you will need to meet their expectations, and set your priorities accordingly. The key attributes of these roles are:

- **Platform provider —** In this model, you are responsible only for providing the platform itself. This is a platform ops variant; the platform capabilities that you provide are used by the teams responsible for each of the APIs you offer. Teams creating APIs are responsible for adding their APIs, creating and updating documentation, validating APIs against API design guidelines and providing support to API consumers. You will need to coordinate between teams to ensure consistency, but this is the smallest scope of responsibility.
- **Platform operator —** In this model, you're taking responsibility for getting APIs into the platform and for the experience provided to API consumers. This means you'll be doing things such as importing APIs, making sure that documentation is up to date, routing support and feature requests to the development teams. This is a middle-of-the-road model — you're ensuring that the API experience is consistent for consumers, but not taking responsibility for the APIs themselves.
- **End-to-end API manager —** In this model, you are responsible for ongoing support of the APIs hosted by your platform. This model is more common in smaller organizations, where a dedicated team creates all of the organization's APIs.

If you are accepting the role of end-to-end API manager, be aware that this model places a significant burden on your team.

Effectively creating and supporting an API requires knowledge of both the source system and the consumer's needs.

You will need to forge strong relationships with application development teams responsible for your source systems, and you will need to incorporate API product management responsibilities into your team. See the section "Treat the API as a Product With Product Management" in "A Guidance Framework for Designing a Great API" for more on this topic.

Create a RACI Chart to Clearly Delineate Responsibilities

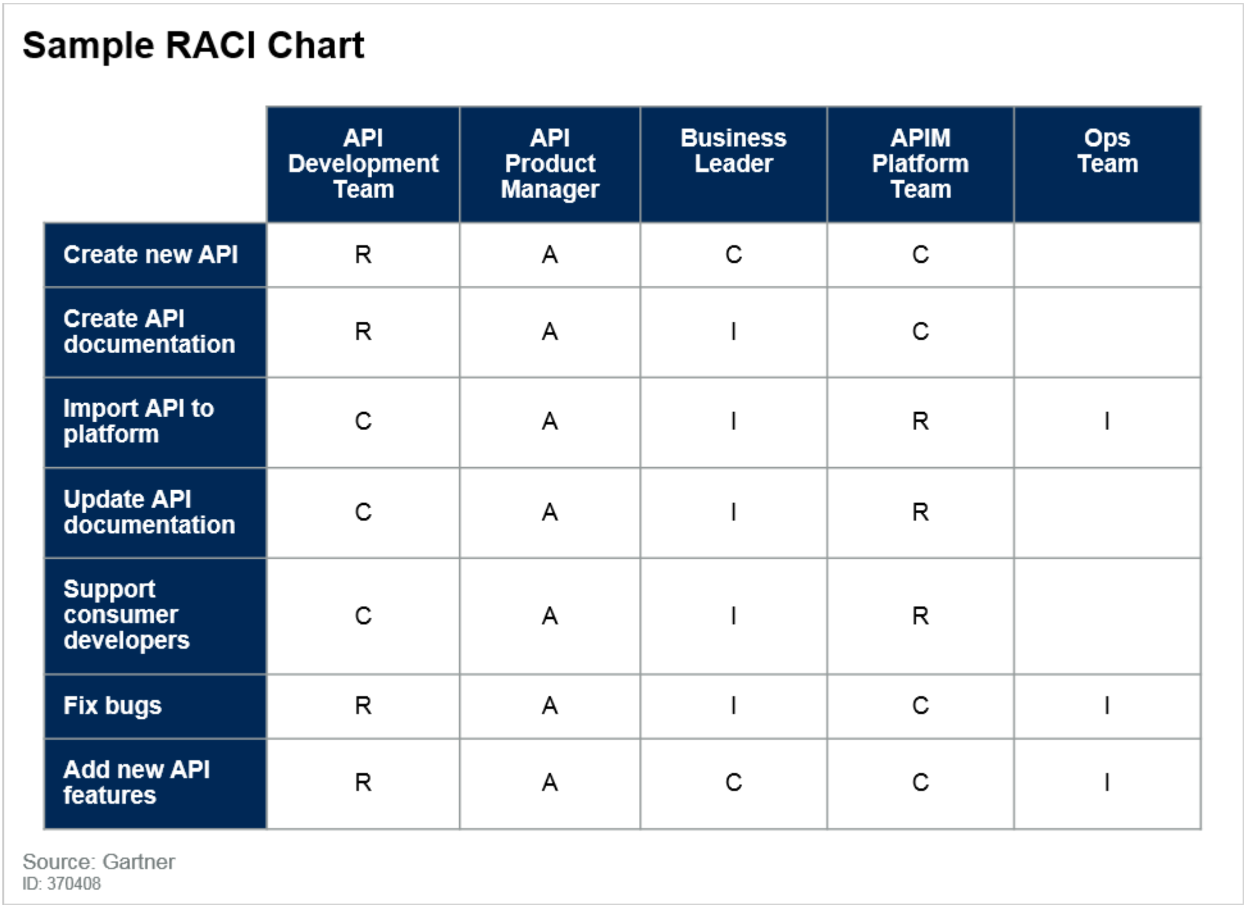
A responsible, accountable, consulted and informed (RACI)[2] chart is an excellent tool for clearly mapping responsibilities to stakeholders. You can use this visualization to help your leadership understand the consequences of the scope of responsibilities you have been assigned. A RACI chart is a simple table, with stakeholders on one axis and decisions/tasks on the other. For each intersection, you will have one of the following roles:

- **Responsible —** People who must complete the task or make the decision.
- **Accountable —** The one person who must sign off or accept the task or decision. There can be only one "A" in the chart. This person also makes sure that the "R" is assigned properly.

- **Consulted —** People who need to give input during the process of doing the work or making the decision. These are active participants (if not, they're "I" instead of "C").
- **Informed —** People who are updated on progress or results. They're not active participants, but they care about the results.
- **(Blank) —** Not every stakeholder has a role in every task — it's OK to have blank cells.

Figure 3 shows an example of a RACI chart for a team that has accepted the platform operator role.

Figure 3. Sample RACI Chart for API Management



Sample RACI Chart

	API Development Team	API Product Manager	Business Leader	APIM Platform Team	Ops Team
Create new API	R	A	C	C	
Create API documentation	R	A	I	C	
Import API to platform	C	A	I	R	I
Update API documentation	C	A	I	R	
Support consumer developers	C	A	I	R	
Fix bugs	R	A	I	C	I
Add new API features	R	A	C	C	I

Source: Gartner
ID: 370408

Determine the Developer Communities You Are Serving

In 2014, Daniel Jacobson of Netflix described two kinds of audiences for an API, calling them a "small set of known developers (SSKDs)" and a "large set of unknown developers (LSUDs)."[3] The key difference between these audiences is that API development teams know who the SSKDs are:

“They may be engineers down the hall from the API team, a contracted company hired to develop an iPhone app, or an engineering team in a partnering company....

More importantly, however, the API team and the providing company care about the success of these implementations in a different way than they might care about the applications developed by the LSUDs.”[3]

From an API management perspective, you need to do the same thing. If the audiences that the APIs in your platform are serving as LSUDs, you will need tools to help you scale support and enablement.

To determine which of these audiences you are serving, consult with your API product owners. They should have information about the number of API consumer developers and their needs. For additional information, see “Step 1: Identify and Engage Your API Consumers” in “A Guidance Framework for Designing a Great API.”

Will You Need a Developer Portal?

If you’re dealing with SSKDs, you need fewer tools and a greater emphasis on direct engagement. For example, one Gartner client launching an API program with a handful of APIs and a single partner consumer chose to launch without a developer portal. Instead, the team elected to directly engage with those developers, using email and sharing documents. Mindful that this approach does not scale, the client has a clear roadmap with plans to introduce capabilities for developer self-service as the number of external partners and developers increases over time.

Note that you should use caution when de-emphasizing developer-facing capabilities. Multiple clients interviewed for this research expressed regret that they had not planned for scaling early enough in their implementation. If your API program will serve more than a few API consumers, you should put developer-facing documentation in place from the start. If your program will scale to support many users, you should either start with a self-service developer portal, or have a plan to implement one quickly. If you do not, you will find that your team is overloaded with support work in short order.

Determine the User Roles You Must Support

From your perspective as the operator of the APIM platform, there are two kinds of users: API consumers and API providers.

- **API Consumers —** Developers, business analysts and others who are using the APIs that the platform manages. They will need access to those APIs, API documentation and administrative functionality for their own accounts, to do things such as change passwords, retrieve API keys and view usage data.
- **API Providers —** Internal users of the platform itself. Likely roles:
  - **Platform administrators —** You and your team. Users with this role can both see and edit everything. Using this role should be an escalated action to prevent accidental editing of the platform's configuration.
  - **Platform operators —** You and your team. Users with this role can see, but not edit, everything. As a good security practice, you and your team should have this as your default role, with access to the administrative level of permissions limited to cases where you intend to make changes. Staying signed in to a superuser-level account at all times is risky.
  - **API developers —** Teams responsible for creating APIs. They can upload APIs and update documentation. They can also view and change configuration about their APIs — unless you want to take that on, which is a lot of work.
  - **API product managers —** Owners of APIs from a business perspective. If your product managers are technical, they may administer their APIs, or they may just have analytics access.
  - **Community managers —** Team members responsible for outreach and support, focused on consumers. If your APIs are targeting external consumers, you may have this role.

You should create a summary of your desired roles and access permissions, and validate it with your stakeholders, including the development teams responsible for APIs. Follow the principle of least privilege, granting only the permissions necessary for a given role to execute its responsibilities. Table 1 shows an example of a set of user roles and associated permissions.

Table 1. Example User Roles and Permissions

	API Consumer	API Developer	API Product Manager	Community Manager	Platform Admin.
Access API	X	X	X	X	X
View documentation	X	X	X	X	X
Add new API		X			X
Edit my API's documentation		X	X	X	X
Edit documentation for all APIs				X	X
Edit configuration for my API		X			X
Edit configuration for all APIs					X
Add new users				X	X
View user contact information				X	X
Edit own contact info	X	X	X	X	X
Edit contact info for any user					X
View analytics		X	X	X	X
Edit reports			X		X

Source: Gartner (May 2019)

Will You Need to Monetize Your APIs?

If you are seeking to realize direct monetary benefit from the APIs that you are managing, you need to identify the monetization model that you must support. The monetization model you adopt will drive your need for usage logging and monitoring to identify chargeable API calls. Table 2 outlines the most common models and associated monitoring requirements.

Table 2. Monetization Models

Model	Description	Requirements
Freemium/ Tiered	This is free up to a certain usage level, with tiered pricing thereafter.	Monitoring of calls at the per-API and customer level that can be aggregated over a billing window.
Subscription	Clients subscribe to a predefined level of API usage monthly, with alerts and/or overage charges if the subscription limit is exceeded.	Usage monitoring at the subscription package level. For example, if a subscription package consists of 10 APIs, the monitoring must aggregate usage across all 10. This monitoring may need to be near real time to deliver timely alerts to customers when they exceed their predefined level of usage.
Pay per-API Call	Pricing is based directly on usage, such as per-API call, or per megabyte of data consumed. In some cases, consumers may prepay for a fixed number of API calls.	Usage monitoring is at the individual API and customer aggregated levels.
Revenue Sharing (End Consumer Pays)	A portion of the revenue resulting from execution of the API is shared with the developer, based on predefined business outcomes. This revenue typically comes from the user of the application powered by the API, so this model can also be called “end consumer pays.”	You can use logging of API requests for audit purposes, but you will have to largely trust that the consumer identifies the outcomes that require payment and pays accordingly.
Revenue Sharing (App Developer Pays)	The developer pays a portion of their application revenue (such as app sales) to the API provider, based on a business agreement with the API provider.	You must be able to correlate business outcomes with the API requests that triggered them. This will require that you are able to correlate requests with successful outcomes (such as a transaction being completed). Separate out the API calls that trigger revenue sharing (e.g., successful completion of a payment in an app store) from those that just prepare for the outcome (e.g., requesting a payment).

Source: Gartner (May 2019)

In most of these cases, you will need to identify a way that near-real-time data on usage levels can be exposed in your developer portal, so that customers can see how much they have used. If this is not supported by your chosen platform, it will require custom development effort. In addition, monetization of your APIs will require that you package or bundle your APIs to create a logical container that your APIM platform can associate with the rate plans and conditions of your monetization approach.

For additional information on API monetization strategies, see “Choose the Right API Monetization and Pricing Model.”

Step 2: Define Strategies and Implement Processes

The next step in implementation is to make key decisions about how you will implement your chosen platform and putting processes in place to ensure success once you have launched.

Determine How You Will Package APIs

A successful APIM platform will include APIs for many tasks and with many target users. Although these may be developed by different teams, they do not exist in isolation. Collectively, the APIs in your platform represent an enterprisewide API catalog. You should organize your API catalog into logical groupings to help consumer developers find the right APIs, and to manage them effectively yourself. In this research, we refer to these logical groupings as "packages." However, the term used for this concept varies depending on the specific product you have chosen.

Identify the optimal method to group your APIs into packages. Grouping methods include:

- **By target user —** Mobile APIs versus web APIs, for example
- **By business function —** APIs to access CRM functionality versus APIs to access marketing automation functionality, for example
- **By security requirements —** Separating public versus private APIs, or APIs with sensitive data such as customer personally identifying information (PII) from those with less sensitive data such as product information, for example
- **By rate plan —** Using packages to manage the rate plans and conditions of your chosen monetization strategy (if monetizing your APIs)

Packaging also creates a structure that allows you to manage your APIs by grouping APIs with similar configuration requirements. In some APIM products, management configurations such as security roles, rate limits and access policies can be applied to packages of APIs, removing the need to manage each API individually.

If you have multiple APIM products (for example, due to acquisitions, or if different products have selected different platforms), then grouping APIs into packages helps make clear why different APIs exist in different places. Conversely, when creating a new API, you must consider which other APIs it will be grouped with when deciding where to implement it.

Although packaging your APIs is useful, do not overdo it. Fewer packaging approaches are better.

The reason is simple: Each packaging approach that you add beyond the first increases the complexity of navigating your API catalog. You will find that you hit diminishing returns quickly.

If your packages of APIs are very different from each other, you may benefit from having multiple documentation portals. This is particularly the case when you have different branding associated with different packages of APIs, or different packages that are deployed on different infrastructure. This will affect your deployment architecture choices.

You should validate your chosen packaging approach against your actual catalog of APIs and target architecture. If you find that your plans do not match reality, change your plans accordingly.

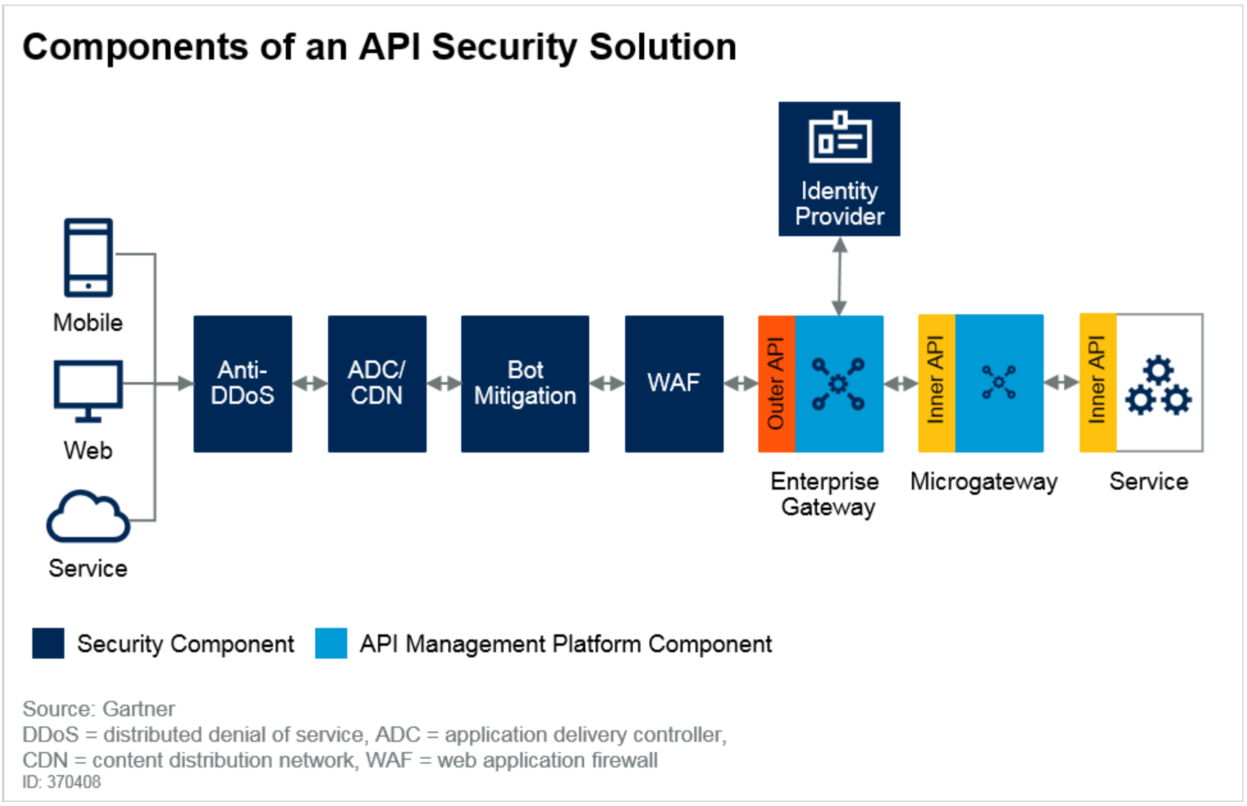
Determine Your Security, Identity and Access Management Strategy

API gateways and microgateways — the runtime components of your APIM — are part of an overall API security architecture that includes both IAM and API protection. In addition to an API gateway and/or microgateway, appropriate protection usually includes:

- A separate IAM system
- Distributed denial of service (DDoS) protection
- Bot mitigation
- Web application firewalls (WAFs)
- Application delivery controllers (ADCs)
- A content delivery network (CDN)

Figure 4 shows the relationship between the components of API security.

Figure 4. API Gateways and Microgateways as Part of a Complete Protection Solution


Components of an API Security Solution
Identity Provider
Mobile
Web
Service
Anti-DDoS
ADC/ CDN
Bot Mitigation
WAF
Outer API
Inner API
Inner API
Enterprise Gateway
Microgateway
Service
Security Component
API Management Platform Component
Source: Gartner
DDoS = distributed denial of service, ADC = application delivery controller, CDN = content distribution network, WAF = web application firewall
ID: 370408

You should identify the full set of your capabilities for security and IAM, and determine how your APIM solution will interact with them.

APIs represent potential attack vectors for data breaches or application security and performance problems. You should evaluate APIs that expose functionality or data from critical business applications with the same level of scrutiny and rigor that you exercise over the applications themselves. To do this:

- Implement granular security to lock down both authorization and access to the lowest level of functionality or data possible for an integration flow.
- Ensure that sensitive data is encrypted in transit and at rest, and use Transport Layer Security (TLS) 1.2 or newer to safeguard connectivity between endpoints.
- Deploy and use runtime protection technologies such as WAFs, cloud application security brokers and API gateways to reduce the attack surface of your integrations and to inspect traffic for threats.
- Create a set of reusable security policies to apply to APIs that are enrolled in your APIM solution. Base these policies on the scenarios you plan to encounter, such as mobile access.

An analysis of web application and API attacks, and the mitigating technologies is provided in "Protecting Web Applications and APIs From Exploits and Abuse." For a full description of web application and API protection services, see "Defining Cloud Web Application and API Protection Services."

Implement Support and Escalation Procedures

You will be responsible for supporting two types of users: platform users and consumers of the APIs published in your platform.

Supporting Platform Users

You will have a community of users of your APIM solution. This will, at a minimum, include your team. It should also include the developers responsible for APIs, and API product managers and business stakeholders. For these users, you will need to provide:

- Secure access to the platform itself.
- A means of requesting access to the platform. This may be accomplished within an existing ticketing system or may be an entirely new process. It should be clearly documented and visible to users.
- A regular audit of access and permissions to ensure that all users with access to the platform require this access. If you are using an external IAM solution for authentication, you may be able to rely on this to handle termination of accounts. However, unless you are also using that solution for authorization, you will not be able to rely on it to capture changes in role.

Supporting API Consumers

Your role in support and escalation for the consumers of the APIs in your platform will vary depending on the scope of your responsibilities:

- **Platform provider —** You will be responsible only for providing the capability for consumer developers to make requests or report issues, and for the development teams to respond.
- **Platform operator —** You will be responsible for routing support and feature requests to the development teams. This will mean some level of contact with consumer developers, particularly for access requests, and in acknowledging support and feature requests and bug reports.
- **End-to-end API manager —** You will be responsible for handling all communication with end consumers. You will still need the ability to escalate to development teams for bugs or features that your team cannot handle.

Example channels for consumer communication:

- **Social media —** If you have a public API and your organization has a social media presence on Twitter, Facebook or LinkedIn, you may receive communication via this channel. You should work with your organization's social media staff to understand policies for communication in these channels and follow them carefully.

- **Email —** Use shared mailboxes or aliases rather than exposing specific individual email addresses for support. This makes it easier to handle vacations and staff turnover without disruption.
- **Chat —** If you have embraced ChatOps, you may elect to use chat channels to provide consumer support. This has the advantage of being a shared experience, where consumers can help one another, providing additional leverage for your teams.
- **Forums —** These may be your own forums hosted on your developer portal, or public forums such as Stack Overflow. Like chat, forums have the advantages and disadvantages of being a shared experience for consumers. Ensure that the staff who are engaging in these forums are skilled technical support staff, so that they can understand and answer detailed questions about your APIs.
- **Help desk —** You may elect to expose your own ticketing system to partners, or to use a third-party system as part of your APIM solution.

In all these cases, you should create policies and standards for support, and ensure that all team members who will be providing support are familiar with these policies and standards. Any time you are supporting APIs in a public channel, you have the chance of your interactions with customers to reflect on your brand. This can be for good or ill.

> For any support experiences that are in public spaces, such as social media, ensure that you are following your organization's policies.

The *last* thing you want is an incident arising from a mishandled consumer request on social media.

Step 3: Define Your Target Architecture

Before you can provision, configure and launch your platform, you must define the architecture you will be implementing and the strategies you will employ when using it. To do this, you must first determine the types of API gateway you will need, and then create a map of your deployment architecture showing the components you need and where they will be located.

Define the Types of API Gateway You Will Need

There are two types of API gateways: enterprise gateways and microgateways.

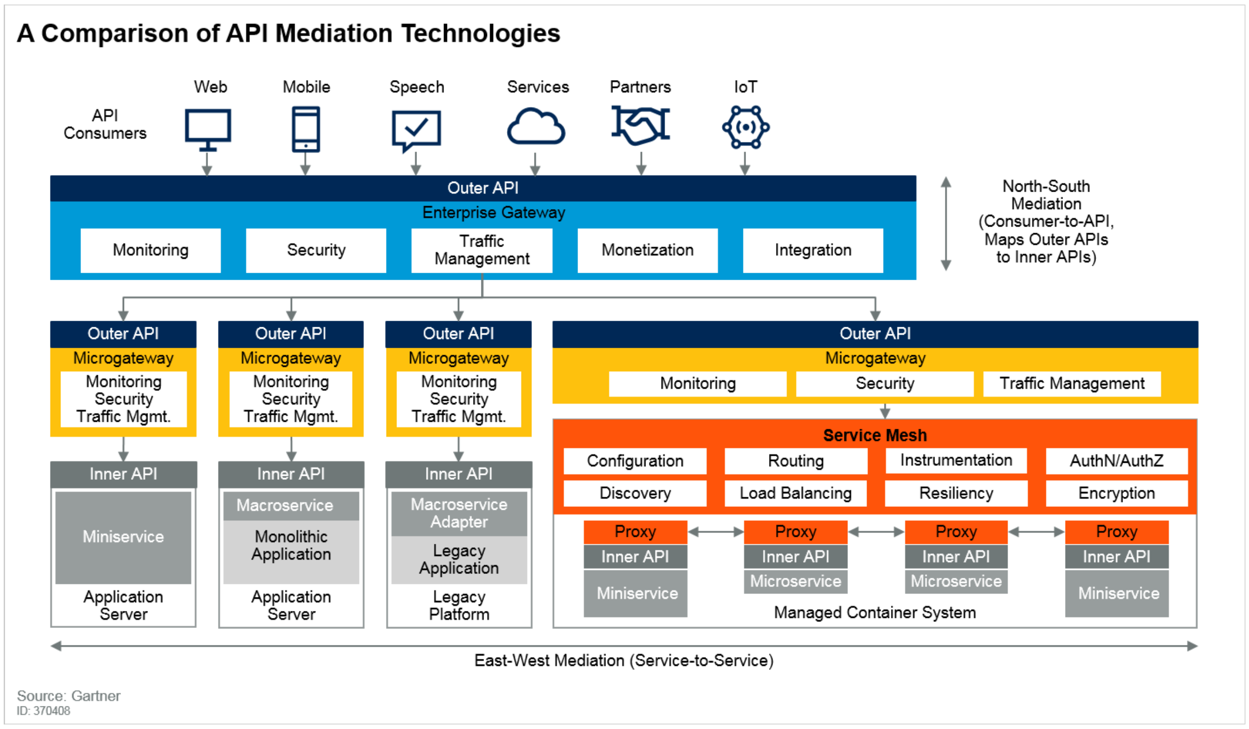
- An **enterprise gateway,** sometimes called an "edge gateway," uses a centralized model to provide a first line of defense at a perimeter entry point, and to manage traffic between API consumers and back-end services (north-south traffic). Example enterprise gateways include Apigee Edge and CA API Gateway.

- A **microgateway** uses a distributed model to provide last-mile defense at an API endpoint. It manages traffic between API consumers and back-end services (north-south traffic) and between services (east-west traffic) when the services are deployed in application servers. Example microgateways include Kong and Tyk.

Another option, **service mesh,** uses a distributed model to manage service-to-service communication (east-west traffic) for miniservices and microservices deployed in a clustered container system. Example service mesh technologies include Istio and Linkerd. Because a service mesh does not offer the capabilities of an APIM platform, it is not a replacement for an API gateway, and is not in the scope of this document. See "Assessing Service Mesh for Use in Microservices Architectures" and "How a Service Mesh Fits Into Your API Mediation Strategy" for more details.

Figure 5 shows the relationships of these three technologies.

Figure 5. API Gateways and Service Mesh



As API management and API gateway technology matures, the line between enterprise gateway and microgateways is blurring. Some products, such as Kong, can be deployed in both scenarios, and some vendors, such as CA API Gateway, offer both enterprise gateway and microgateway products as part of their offering. In addition, service mesh offerings such as Istio are expanding the capabilities they offer to encompass the capabilities of microgateways. While the architectural need for a microgateway is not going away, the idea of a stand-alone microgateway product may be.

There are three key criterion driving your choice of gateway type:

- **Location of consumers —** If your API consumers are located outside of your organization, you should choose an enterprise gateway product, or deploy your product configured as an enterprise gateway, to provide robust security and analytics functionality. If your consumers are located inside your organization, you may need only a microgateway.
- **Need for interface translation —** If you intend to handle lightweight API mediation in your gateway, rather than using a dedicated integration solution, you should choose an enterprise gateway product. Use caution when performing integration work in your gateway — see the section "Limit the Work Done by Your Gateway" below. If you will not need to mediate APIs, a microgateway may be sufficient for your needs.
- **Monetization capabilities —** If you need the ability to monetize your APIs, you will need to use an enterprise gateway product. If you do not need monetization, then a microgateway may be sufficient.

Table 3 shows the key differences between the three technology options.

Table 3. Comparing API Gateways, Microgateways and Service Mesh

	Enterprise Gateway	Microgateway	Service Mesh
Intended deployment	Edge of enterprise	Between applications	Inside microservice applications or between services
API mediation capabilities	Strong	Limited	None
Monetization capabilities	Strong	None	None
Topology	Manages north-south traffic	Manages north-south traffic and can manage east-west traffic	Manages east-west traffic and can manage north-south traffic
Traffic management	Advanced routing, load balancing and throttling	Basic routing, load balancing and throttling	Dynamic discovery, routing, load balancing, retries and circuit breaker
Monitoring	Monitors two-way traffic through the gateway	Monitors two-way traffic through the gateway	Monitors microservice-to-microservice traffic, automatically instruments traffic and supports distributed tracing
Security	Authentication, authorization, encryption, token verification and API attack protection	Authentication, token verification and some API protection	Tokens and credential verification, encryption, key and certificate management, and service identity

Source: Gartner (May 2019)

Note that there is no technical reason you must use a single gateway technology or provider. Many Gartner clients use different vendors or open-source technologies within their APIM platform, reflecting the different needs. For example, you may choose to use Apigee Edge as your enterprise gateway for handling APIs used by external partners, but the open-source Community Edition of Kong for your internal APIs.

The advantage of using multiple products is that you can more closely target your specific needs for each use case. In addition, using open-source technologies for internal use cases can lower the upfront costs of licensing. The disadvantage is that you are increasing the overall complexity of your platform. You should also be aware that supporting open-source technologies can have high ongoing costs. While they are “free, as in beer” they are also “free, as in kittens” — you are committing to the care and feeding of these technologies for the term of your use.

As with all architectural decisions, you are making trade-offs. There is benefit to having a “single pane of glass” to view and manage your API gateways, and having consistent policy enforcement across gateways is valuable. Using a single product or suite also gives you consistency; for example, API keys and user accounts will be consistent across your API landscape. If you use multiple gateway technologies or vendors, you will need to do extra work to gain these benefits, or do without them.

Determine How You Will Handle API Mediation

Although APIs are a powerful, flexible tool for application integration and business enablement, they also create new challenges. The more APIs that you expose, the more you expose sensitive functionality and information. Each API represents an additional attack surface for your systems. In addition, shared, general-purpose APIs are poorly suited to the specific needs of consuming applications. And, in many cases, the APIs supplied by existing packaged applications and legacy systems may be incompatible with modern needs. SOAP APIs with XML payloads are still common for legacy systems and even for APIs created within the last few years. Many Gartner clients are dealing with a variety of formats and protocols for their internal APIs.

To bring order to this chaos and to create a more unified and more usable API experience for consumers, the mediated APIs approach abstracts away the complexities of the service implementation. It also offers a single point to implement access control, security and monitoring. This is part of the reason you’re adopting API management.

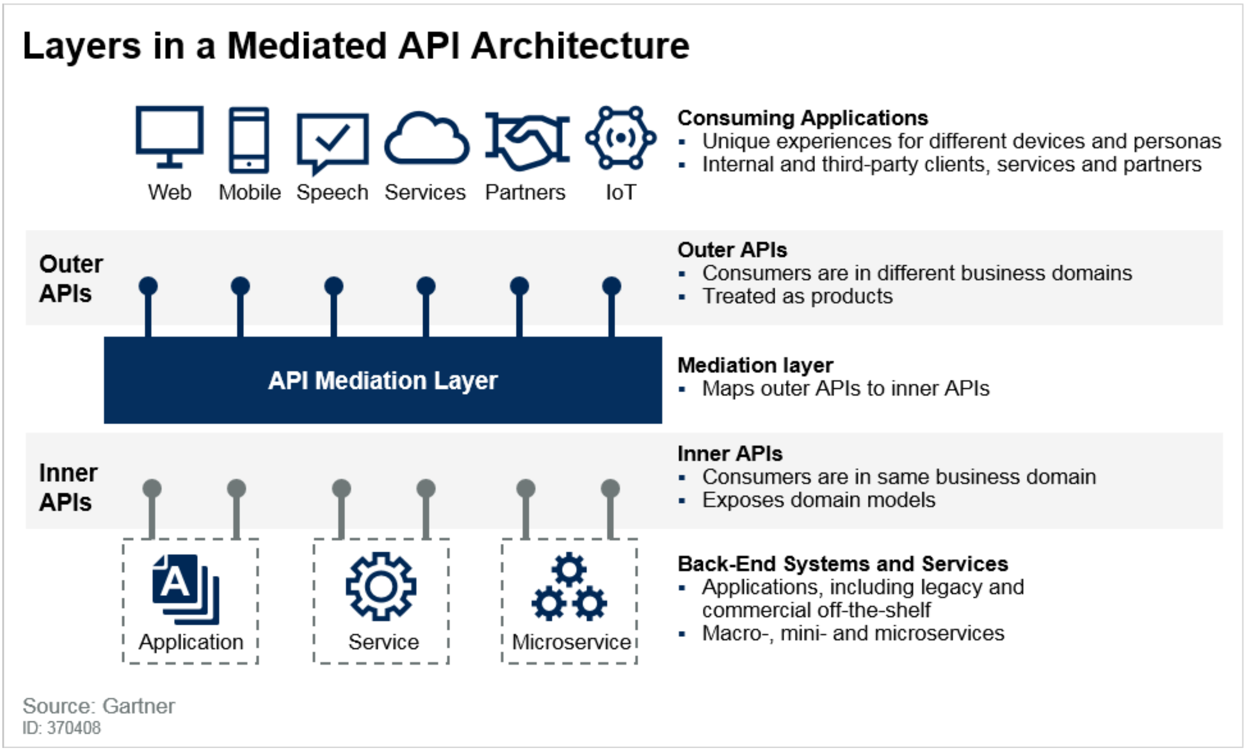
A mediated API approach involves implementing three distinct layers:

- **Inner APIs** enable the service’s native domain and interaction models. These may be existing APIs from packaged or legacy systems or even new APIs created using older protocols or formats. Or they may be those that directly expose existing domain models.
- **Outer APIs** provide a consistent layer for apps to access back-end functionality. Apps may share an outer API or have a unique API. Apps use outer APIs as part of their platform and, therefore, treat the outer APIs as individual consumer products.
- **API mediation** maps the outer APIs to the inner API, monitors and logs interactions, and enforces runtime policies for security and traffic management. This includes acting as an

intermediary to translate protocols, formats, data structures and other transformations as requests pass between inner APIs and outer APIs.

This approach provides the highest degree of architectural agility because it allows you to add, modify or replace individual apps, APIs, mediators and services, with limited impact on the rest of the mesh application. Figure 6 shows the conceptual model of API mediation.

Figure 6. Mediated API Pattern


Layers in a Mediated API Architecture
Web
Mobile
Speech
Services
Partners
IoT
Consuming Applications
▪ Unique experiences for different devices and personas
▪ Internal and third-party clients, services and partners
Outer APIs
Outer APIs
▪ Consumers are in different business domains
▪ Treated as products
API Mediation Layer
Mediation layer
▪ Maps outer APIs to inner APIs
Inner APIs
Inner APIs
▪ Consumers are in same business domain
▪ Exposes domain models
Application
Service
Microservice
Back-End Systems and Services
▪ Applications, including legacy and commercial off-the-shelf
▪ Macro-, mini- and microservices
Source: Gartner
ID: 370408

Limit the Work Done by Your API Gateway

Modern API gateways offer a wide range of functionality, including the ability to enforce policies and perform transformations in the gateway. You should use this functionality with caution. The base heuristic is that you should limit the work done by the gateway to things that are directly required to deliver APIs.

Simply because you can do something in the gateway, does not mean that it is a good idea to do so.

You should balance the desire to simplify your platform against the degree of lock-in you are creating and the runtime cost of executing policies and transformations in the gateway. Each policy or transformation that you put into your gateway will consume capacity and increase complexity of the gateway. Instead, leverage other technologies, some of which may already be in place:

- Integration platform software such as an enterprise service bus (ESB) or distributed integration platform (DIP)
- Integration platform as a service (iPaaS)
- Data virtualization technologies
- Mobile back-end services
- Publish and subscribe (pub/sub) event middleware
- Custom mediators written in-house or by third parties.

Work that should be done in the platform:

- **Policy enforcement** determines who gets access to your APIs is a key capability that you should enforce directly at the gateway.
- **API usage monitoring** uses an API gateway, for example, as a good point to gather API usage information for monitoring or analytics.
- **Traffic management,** where throttling, rate limiting and circuit breakers are baseline capabilities of APIM solutions.

Work that is a good idea to put in the gateway, but can be done elsewhere:

- **Protocol mediation** such as SOAP to REST. This is a standard capability of API gateways, and carries little risk. Note that the capabilities offered by gateways are not sophisticated. The REST that is created from existing SOAP endpoints is suboptimal.
- **Format mediation** such as XML to JSON. This is also a standard capability with low risk.
- **Simple orchestration,** where a single inbound call requires multiple backing services to satisfy. For example, if you have product images and product descriptive information in two separate APIs, implementing an orchestration to combine the results into a single response makes sense. This is a common use case for API mediation. You should be confident that the orchestration will remain simple and that the endpoints involved all support the least common denominator of service requirements.
- **Simple payload modifications**, such as rewriting URLs from internal format (for example, changing “http://internal-prod-api/v1/products” to “http://api.company.com/v1/products”) to ensure that hyperlinks inside payloads work both inside and outside your network.

Work that is a bad idea to put in the gateway:

- **Complex orchestration** where a single inbound call triggers a series of calls to backing services with business logic.

- **Complex payload modification,** such as redacting content to implement security (for example, removing the "costOfGoodsSold" attribute from a product data feed when the calling user is not authorized to see that data).
- Anything involving **custom code implementing business logic** that is deployed into the gateway itself.

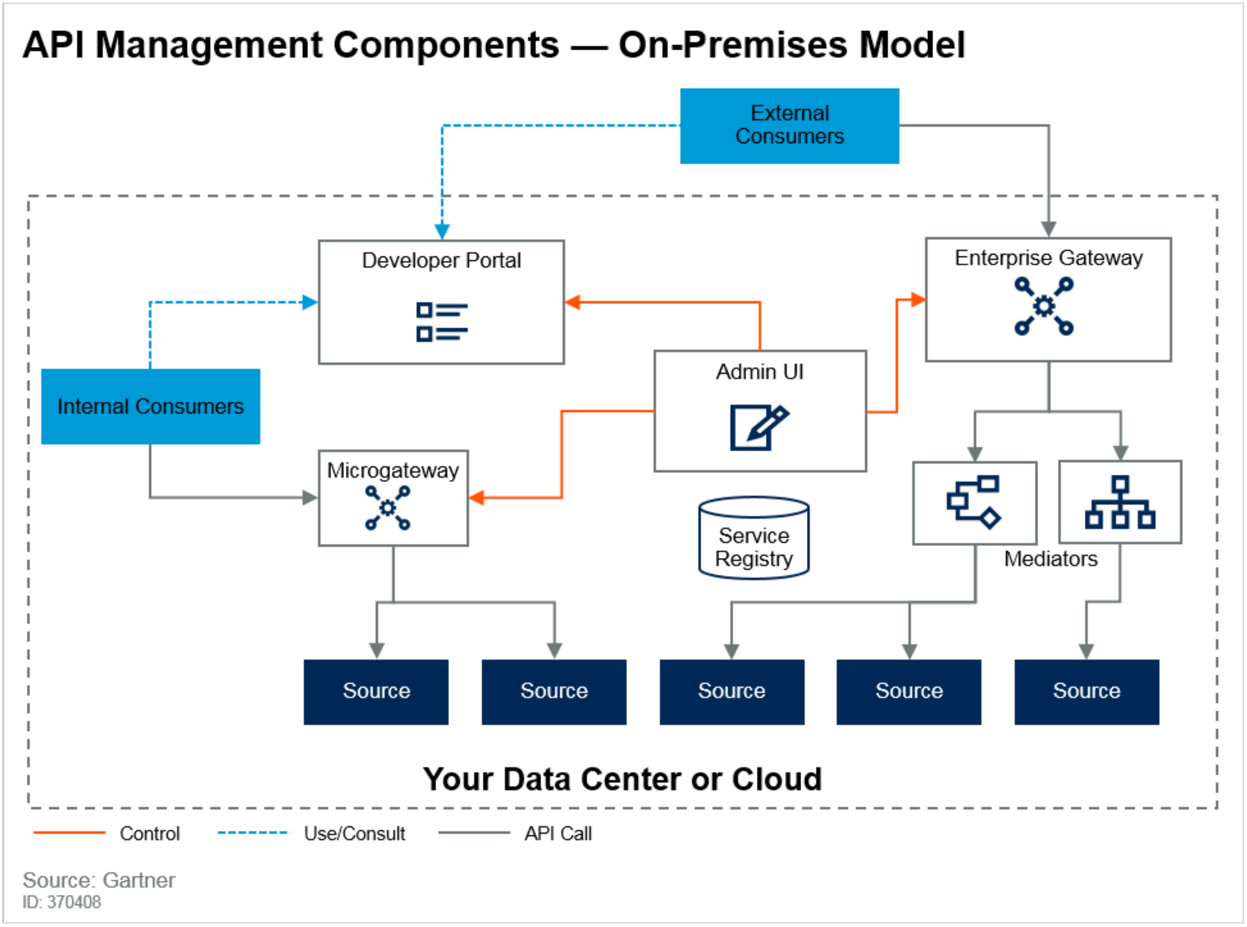
Map Your Deployment Architecture

The deployment architecture of your platform needs to take into account both the reality of your existing network architecture and the needs of your API developers and consumers. To aid in planning, create an architectural diagram that lays out the high-level physical architecture of your solution, incorporating the following key components of the platform:

- **API gateways —** The working heart of the APIM solution, where endpoints are exposed and policies are enforced. If you are deploying multiple gateways, such as using both an enterprise gateway and one or more microgateways, you should clearly indicate where these are deployed.
- **API administration —** The administrative user interface for the APIM solution. If you are using a solution such as Kong Community Edition, which is managed from the command line, there may not be a deployed component for this functionality. You should still include this functionality on the diagram. You will need to ensure that you have the access necessary to control the platform. For example, if your network team does not normally allow command line access to instances, you will need an exception to allow it for this case. If you are using a solution such as TIBCO Cloud Mashery, which is managed from an administrative UI hosted by Mashery, you will need to ensure that you have access to the cloud-hosted control panel.
- **Service registry or repository —** The specific technology used by your APIM solution to store configuration and usage data. Commonly linked with the API administration component.
- **Developer portal —** The developer-facing component of the solution. As noted above, this capability is optional, and may be part of your provider's offering or may be something that you develop independently. This may be a single portal used by both internal and external developers, or may be separate portals for internal and external audiences.
- **Sources —** The service implementations behind the APIs that the platform is managing. These may be internal applications or third-party endpoints.
- **External consumers —** The external applications that are using the APIs that the platform is publishing. These will be using your APIs through an enterprise gateway.
- **Internal consumers —** The internal applications or services that are using the APIs that the platform is publishing.
- **Mediators —** If you are incorporating additional technologies for API mediation such as an ESB, DIP or iPaaS, you should include them in this diagram.

Use this diagram to ensure that your operations teams are aligned with your plans for deployment. Figure 7 shows an example of such a diagram for an on-premises deployment of API management serving both internal and external audiences.

Figure 7. Components of API Management for an On-Premises Deployment



We see three topologies for API management in use today: Cloud-only, on-premises-only and hybrid.

- **Cloud-only:** The API gateway, API administration and developer portal are delivered as a cloud service. All API requests are routed through the cloud provider for policy to be enforced. This reduces the administrative overhead of the APIM solution, but incurs additional hops and latency for API calls between on-premises applications and on-premises APIs.
- **On-premises-only:** All components are deployed on-premises. All API calls are routed through on-premises gateways. This reduces the hops and latency of calls between on-premises systems, but incurs operational overhead for the platform. It is also inefficient for cases where you are managing calls between cloud-hosted or software as a service (SaaS) applications and cloud-hosted APIs, such as from Workday to Salesforce.

- **Hybrid:** Either or both API administration and developer portal are hosted exclusively in the cloud, while the API gateways used to enforce policy at runtime are deployed where needed. API requests are routed through the most logical gateway — cloud-to-cloud API calls are not routed through on-premises gateways, for example.

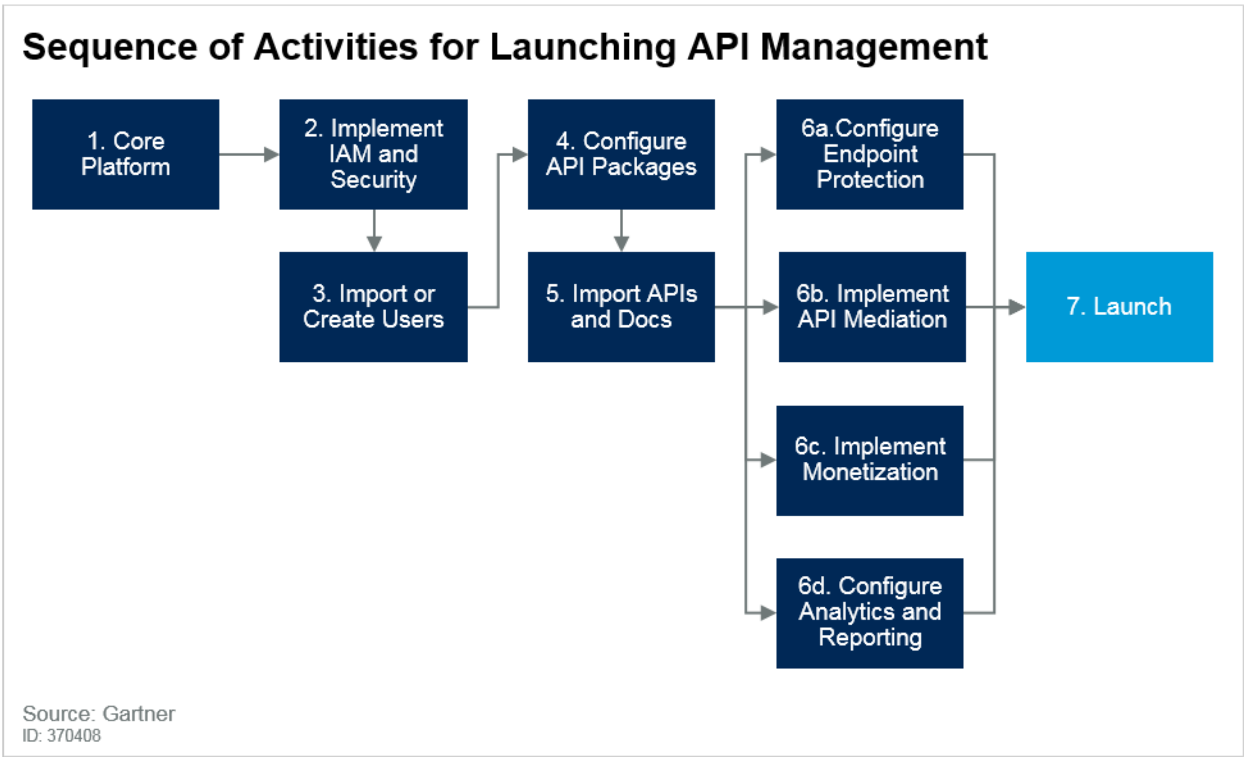
An additional topology is **multicloud,** where an organization deploys the same API management technology across multiple cloud providers. As of this writing, this is aspirational — while it is possible to do this, Gartner has not yet seen a client with this topology in production.

The product you select will strongly influence, if not define, the possible deployment topologies that you can implement. In both the cloud-only and on-premises-only models, you should pay attention to the routing of API requests. If you find that your chosen product's available deployment options are creating inefficient routes that perform poorly, you should consider extending your solution with additional components. In inquiry, most Gartner clients prefer the hybrid model for its greater flexibility. Bear in mind that you are not limited to a single provider's offering. As noted above, you can choose to use multiple API gateways to meet the disparate needs of your internal and external customers, or to handle API endpoints located both on-premises and in cloud environments.

Step 4: Provision and Configure the Platform

In this step, you put your decisions into action, deploying and configuring your APIM solution. Use this section as a checklist for activities to perform during deployment. Figure 8 shows the suggested sequence of these activities.

Figure 8. Sequencing API Management Configuration and Launch Actions


Sequence of Activities for Launching API Management
1. Core Platform
2. Implement IAM and Security
3. Import or Create Users
4. Configure API Packages
5. Import APIs and Docs
6a.Configure Endpoint Protection
6b. Implement API Mediation
6c. Implement Monetization
6d. Configure Analytics and Reporting
7. Launch
Source: Gartner
ID: 370408

1. Provision the Core Components of the Platform

The first task in this step is to provision and configure the core components of your platform. You cannot implement any of the remaining sections until you have a platform in which to perform them. The specific actions you take will vary based on the product you are using and the deployment architecture you have chosen.

- If you are using a SaaS solution such as Microsoft Azure API Management or TIBCO Cloud Mashery, you will be provisioning components in a web-based interface.
- If you are using an on-premises product such as CA API Gateway or IBM API Connect, you will be working with operations teams to deploy and configure application instances.
- If you are using a hybrid deployment model, you will be provisioning cloud-based components and deploying application instances.

Use the map of your deployment architecture that you created in Step 3 as a guide in this process. Do not make the mistake of treating your planned deployment architecture as a fixed plan. You are likely to find opportunities to correct or improve your deployment as you actually implement it, and you should take advantage of these opportunities.

2. Implement IAM and Security

Your next step is to implement access control for your platform's administrative UI, followed by the security infrastructure that will protect both the platform and your APIs. Not only is every API a potential attack surface, your APIM itself is a target for attackers. Work closely with your security team to secure your platform as you:

- Link your platform to your authentication and authorization systems.
- Lock down all other access. Ensure that default administrative access accounts are removed entirely or completely disabled.
- Review the platform's default roles, and edit them to conform to your security model. Remove any superfluous roles, and ensure that all roles have the minimum permissions necessary to accomplish their responsibilities.
- Ensure that your DDoS protection will shield your platform.
- Ensure that bot mitigation, WAFs and ADCs are in place.
- If you use a CDN, ensure that you have correctly configured its caching policies to work with your APIs. You do not want to create security or privacy risks by caching sensitive content in your CDN. Equally, you do not want to cache and serve stale data.

For additional guidance on API security, including the types of threats that APIs face and the approaches for defending against them, see "Protecting Web Applications and APIs From Exploits and Abuse."

3. Import or Create Users

Before you, your team or the API developers who will use the platform can do any work in it, you must import or create user accounts for platform users. In most cases, each person with access should have a unique account. If your development teams are pairing and using shared team-level accounts, review their processes for changing passwords when team members leave to ensure that they are secure. Also review their internal auditing and change control processes. You do not want to break or slow the development process, but you do need to ensure that platform access is being handled securely and responsibly.

Also note that some individuals may have more than one account. For example, you and members of your team may have both a user account and an administrative account, so that you are not operating with superuser privileges when doing things such as reading documentation or viewing reports.

Key actions:

- Validate the list of initial users with team leads, supervisors or business stakeholders.
- Import users, if possible, to reduce human error in setup processes.

- Test permissions of each user role to validate that users can complete their necessary tasks, and that they have no ability to escalate their level of access.
- Communicate the new access to users.
- Implement your access request and audit processes.

4. Configure API Packages

If you are using packages to group your APIs, you should implement these packages before importing APIs. Doing so will make it easier for you to group APIs as you add them, decreasing the effort required to manage your APIs. Remember that you are using packaging to make your APIs easier for consumers (and you) to navigate and use, so you should limit the number of packages you create and the number of layers in your packaging. Navigating through a multilayered hierarchy is frustrating.

- If your API management platform has built-in features for grouping APIs, use these to manage your packages. For example, Apigee Edge provides functionality (called "API packages" in Classic Edge and "API product bundles" in New Edge) that allows you to create "a collection of API products that is presented to developers as a group."[4]
- If you cannot use features provided by your platform, you must architect a solution. For example, you could use different instances of the APIM platform or developer portal to separate APIs into packages.
- If you are using packages to manage security and access control policies at the group level rather than for each individual API, you should create and apply those policies to the packages at this step.

5. Import APIs and Documentation

You are now ready to bring APIs and the associated documentation into your platform. The responsibility for this will vary depending on the scope of your responsibilities:

- **Platform provider:** Teams creating APIs are responsible for adding their APIs and creating and updating documentation. Your role will be to support them in this process.
- **Platform operator:** Your team will be importing APIs and documentation into the platform. The APIs themselves as well as the API specifications and documentation will have been created by the API development teams.
- **End-to-end API manager:** Your team will be importing APIs and documentation into the platform. The APIs, API specifications and documentation will have been developed by your team.

Key actions:

- If your platform allows you to use an OpenAPI Specification (formerly known as Swagger Specification) file to partially or fully automate the import and configuration process, you should leverage this.
- When you import APIs, add them to packages to eliminate or reduce the amount of work needed to configure each API.
- API documentation includes the API specification, any markup of that specification and any other documentation such as sample code or data dictionaries.
- Publish your APIs with their supporting documentation in your developer portal.

Note that you should use your API packages to aid navigation in your developer portal. They are logical groupings and will help consumers find the APIs they are looking for.

6a. Configure Endpoint Protection Capabilities

One of the most critical capabilities of API gateways is their ability to provide protection for API endpoints. There are two types of endpoint protection capabilities: Rate limiting and throttling limit access to an API endpoint, while circuit breakers protect both the API endpoint and its consumers from failure.

Rate Limiting and Throttling

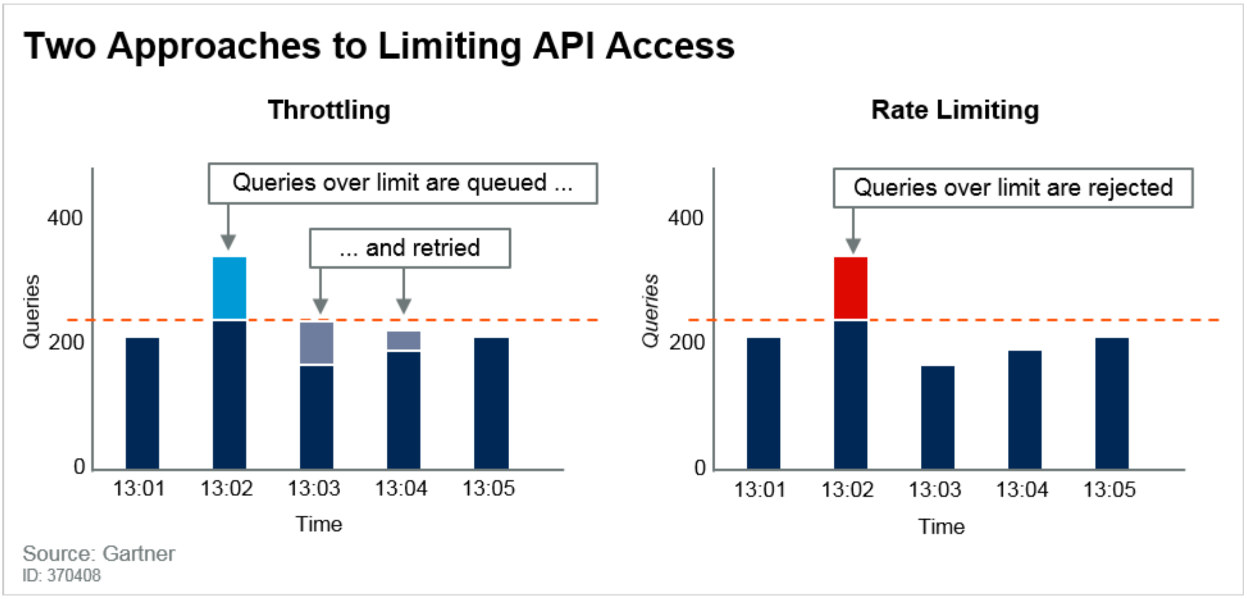
These are closely related concepts and are often confused with one another. Both of them impose a limit on the number of queries that a caller may make to an API endpoint within a given time frame, such as five queries per second. The difference is how they handle requests that exceed that limit.

- **Throttling** queues overlimit requests for retry in a future time frame. The advantage to this approach: The consumer's request will eventually be met. The disadvantages are that it:
  - Creates additional load on the gateway to queue and retry requests
  - Runs the risk that the query will take a long time to finish
  - Can generate out-of-sequence responses from the API
  - Is vulnerable to denial of service (DoS) attacks because requests are not timed out

  If you will use throttling, you should ensure that you have DoS protection for your endpoints.
- **Rate limiting** simply rejects requests that exceed the rate limit. The advantages of this approach: It limits load on the gateway, eliminates the potential for long-running requests and lowers the chance of out-of-sequence responses. However, it can create a frustrating experience for a calling system if the rate limits are set very low, and it requires the calling system to catch and retry failed requests.

Figure 9 shows the difference between the two approaches.

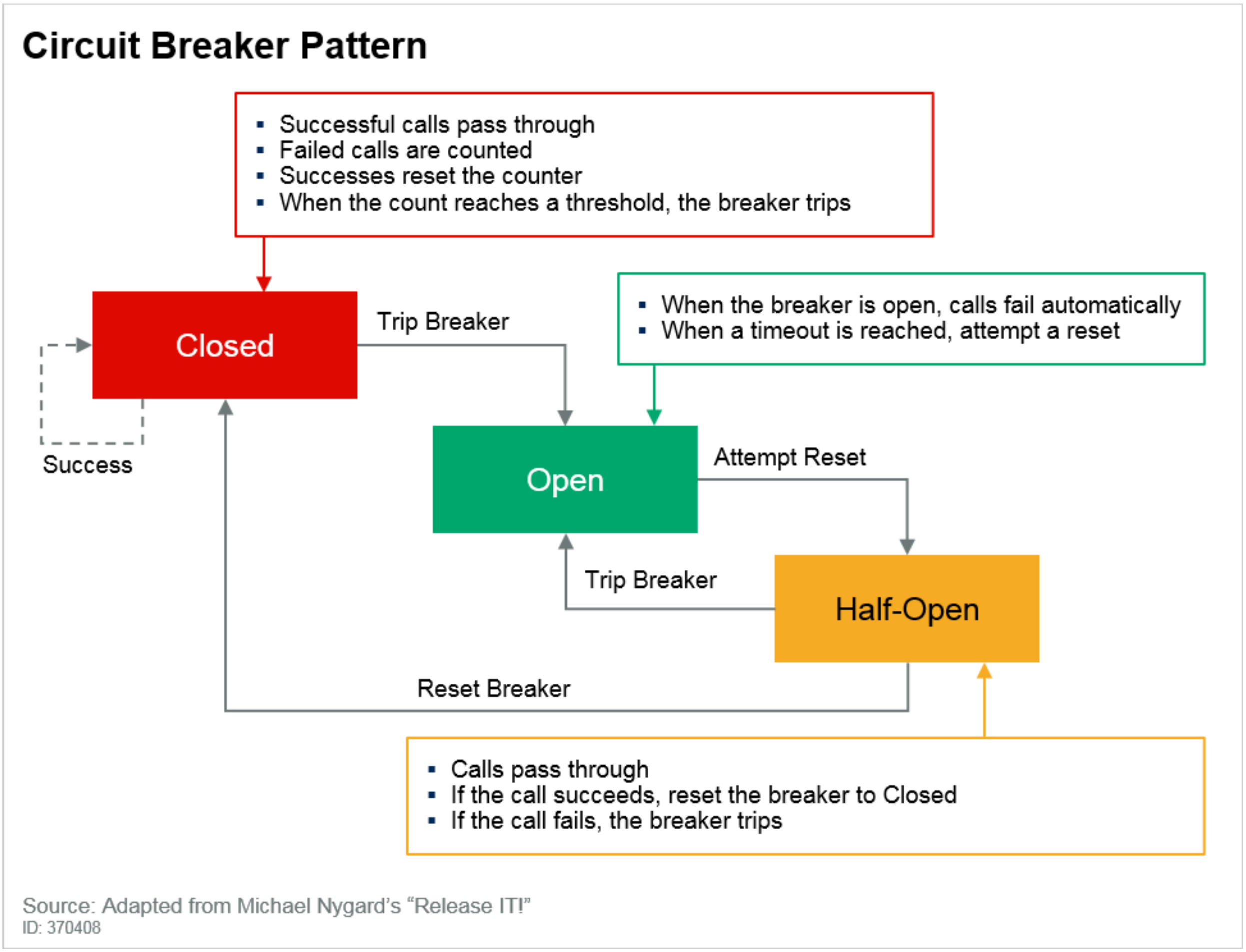
Figure 9. Comparison of Two Approaches: Throttling and Rate Limiting



Circuit Breakers

The circuit breaker pattern shown in Figure 10 helps prevent failures of one service from causing cascading failure and helps prevent services from being overwhelmed as they recover from outages. The idea is that, by emulating the circuit breakers that keep our houses from burning down due to an electrical failure, we can limit the damage of any single failure in a system and keep it from spreading.

Figure 10. Circuit Breaker Pattern


Circuit Breaker Pattern
Successful calls pass through
Failed calls are counted
Successes reset the counter
When the count reaches a threshold, the breaker trips
Closed
Trip Breaker
When the breaker is open, calls fail automatically
When a timeout is reached, attempt a reset
Success
Open
Attempt Reset
Trip Breaker
Half-Open
Reset Breaker
Calls pass through
If the call succeeds, reset the breaker to Closed
If the call fails, the breaker trips
Source: Adapted from Michael Nygard's "Release IT!"
ID: 370408

In the context of API management, circuit breakers are extremely useful. If an API begins to fail under load, the circuit breaker will trip, and subsequent requests will fail immediately at the API gateway, rather than waiting for the API implementation to time out. This prevents calling applications from consuming resources because they are waiting for responses from the API. As an additional benefit, requests are not passed to the implementation while the circuit breaker is tripped, which removes load and allows the endpoint to recover. For more details about the pattern, see "CircuitBreaker" at Martin Fowler's blog.

6b. Implement API Mediation

Your API gateway is the starting point for your API mediation. You will use it to enforce security and access control policies, and to monitor and manage traffic to your APIs. You will also do some amount of transformation in your API gateway, such as converting XML data packets to JSON. Use caution when implementing transformations in your gateway — as noted above, doing so both burdens the gateway and creates lock-in to your current vendor.

If you need more complex mediation, such as complex modification of API payloads or orchestration of API calls across multiple inner APIs, you should leverage a separate integration system. Because mediators live “in between” source APIs and your APIM:

- If your APIM is deployed on-premises and publishing APIs from on-premises systems, you should prefer an ESB, DIP or custom mediator located on-premises.
- If your APIM is cloud-hosted and publishing APIs from on-premises systems, you can either choose an on-premises solution, or use an iPaaS as a mediator.
- If your APIM is cloud-hosted and publishing APIs from cloud-hosted systems, you should prefer an iPaaS as a mediator.

If you are doing this kind of complex mediation, you will be creating outer APIs that are distinctly different from your inner APIs. For each such API, you should follow the process defined in “A Guidance Framework for Designing a Great API” to ensure that your APIs are designed in a consumer-centric fashion. All APIs need API specifications and documentation.

For each outer API, you will need to identify an owner so that you can route support requests and bug reports appropriately. If your team is the owner for outer APIs, ensure that you have a good line of communication with the teams responsible for the inner APIs on which your outer APIs are based. You do not want to be trying to establish these relationships when you need their assistance in troubleshooting issues.

6c. Implement Monetization

If you are monetizing your APIs, you will need to ensure that you can gather metrics to identify the value each API is generating. Simple metrics such as number of calls to each API at global and per-customer levels give a good overview; however, you must remember that billing for usage is only one of many options for monetizing APIs.

- If you are charging for usage, you will need to provide capabilities for billing consumers. You can use a subscription management product such as Aria, Recurly or Zuora to handle API subscription plans and billing, or you can export the usage information out into an existing billing system.
- If you are billing consumers, you will need to provide them with a means to pay you. Subscription management systems such as Aria, Recurly and Zuora can provide payment processing capabilities, or you can use an existing system.
- If your monetization approach relies on revenue share from application sales, you will need to trust that the partners that are using your API to power their applications are reporting accurately.
- You should use log aggregation technologies such as Elasticsearch, Logstash or Splunk to trace API calls through to back-end transactions, to see which API calls result in business revenue.

If you are charging for API access, you will need to implement additional features within your architecture that allow you to collect and enforce your monetization policies:

- IAM that allows you to identify requests at the “paying client” level. This is not as simple as it first seems, because a client may have multiple systems that access the same API. Ensure that at least one identity or authorization token can identify the client at the level required for billing, to ensure that rate limiting and tiered access policies can be applied correctly.
- An access control mechanism that allows clients to access the specific packages of APIs that they are paying for. This may also require that you generate aggregated usage metrics and rate limiting policies at the package level, rather than individual API levels.

6d. Configure Analytics and Reporting

Analytics and reports are critical tools for understanding how your platform and the APIs it publishes are performing. They also provide valuable evidence for key stakeholders that your platform and its APIs are meeting expectations and can identify cases where it is not. The data produced by your platform is useful to many audiences, and you should ensure that all of them are getting what they need. For example, if you are monetizing APIs on a per-call basis, then you will need to provide access to information on usage and costs to the developers who are consuming your APIs. This will allow them to understand and manage their costs.

- Start by listing the consumers for analytics from your platform. For each, summarize:
  - Their reason for needing analytics — How will they use this data?
  - Their preferred location for accessing data — Will they come to your platform, or do you need to send reports via email or provide access to your platform’s APIs?
  - The type of access they need — Will they be creating ad hoc queries, viewing dashboards or consuming a canned report?
  - The specific metrics they need — What do they need to know?
- For each case where your platform’s native capabilities meet the needs of your analytics consumers, configure the dashboards and reports as needed and provide access.
- For cases where your platform’s capabilities do not meet your consumers’ needs, you will need to develop custom solutions. You should prefer solutions that use your platform’s APIs. Avoid granting direct access to your platform’s databases; doing so will couple your platform to the reporting consumers.

Table 4 shows examples of analytics and reporting consumers, showing the value they get from analytics, the type of access they need, frequency of scheduled reports and example metrics. Use this as a basis to create your own list of analytics consumers.

Table 4. Audiences for Reporting and Analytics

Role	Value of Analytics	Preferred Location	Access Type	Example Metrics
Your team	Insight into platform usage	APIM platform	Ad hoc queries, dashboards	Latency by API Error rate by API Error rate by calling app
API product managers	Guidance for developing features and new APIs	APIM platform	Dashboards	Active developers Conversion rate of new developers
Business stakeholders	See overall health of API program	Email	Scheduled reports	Active developers Total number of API calls
Security team	Detect potential breaches or issues	Security dashboards	Log files	Error logs Most active users Errors by API by type
API developers	Understand usage patterns to guide new development	APIM platform	Dashboards	Latency by API Errors by API by error type
API consumers	Track usage against rate limits or monetization caps	APIM platform	Dashboards	Usage by time period

Source: Gartner (May 2019)

7. Launch the Platform

With all activities complete, you are ready to launch your APIM platform. You should avoid a “big bang” launch if possible, preferring a “soft launch” strategy, where the platform is publicly available in advance of any promotional activity. This allows you to have a chance to spot any issues and remediate them without the spotlight of attention.

If you cannot do a soft launch, for example, if regulatory or compliance reasons prevent it, ensure you have staff available to handle any issues that arise. Prefer launch windows that are timed to provide maximum staff availability. You do not want to wake up people in the middle of the night if you can avoid it.

Follow-Up

Operate and Evolve the Platform

Once your platform is live, your job has just begun. You must now operate the platform as an organizational capability. Both the platform and the APIs it publishes are products and must be treated as such. Key activities include:

- **Discovering new APIs —** Create an ongoing process of discovery to find new (or existing) APIs within your organization so that you can bring them into your APIM solution. Use the information that you gathered about organizational goals for APIs and API management to guide this work. Sadly, this is a manual process; despite many user requests for this functionality, no APIM products currently provide functionality to automatically discover APIs.
- **Inducting APIs —** Once you have discovered APIs, you will need to add them to your API registry. At a minimum, this means importing API specifications and documentation, and may require updating packages, security policies or transformations.
- **Sunsetting and versioning APIs —** APIs do not live forever. Some of them are unsuccessful and will need to be retired. Others are so successful that they need to change in a way that breaks the API's contract with consumers. Each API should have a clearly defined versioning strategy, including both a method to specify the version and a process to implement breaking changes. When either retirement or version changes occur, you will need to assist in implementing the change in the platform and, potentially, in communicating changes to consumers.
- **Managing communication with API consumers —** Ensure that your API consumers are being listened to and taken care of. They are the reason that your APIs exist — Without consumers, there is no point in having APIs. Monitor the communication channels and track the timeliness of responses. This is particularly important for publicly visible communication channels, where slow responses are visible to others. Potential API consumers will use multiple dimensions to evaluate your APIs, and public forums with unanswered questions that are days or weeks (or months) old is a sure-fire way to decrease their confidence in your APIs.
- **Managing the platform roadmap —** Your APIM platform is itself a product, with a set of capabilities that are likely to change over time in response to the changing needs of your organization. As your organization matures its use of APIs, you will need to grow your platform's capabilities. For example, if your organization invests in IoT or begins to explore event-driven architectures, you may need to handle webhooks or Message Queuing Telemetry Transport (MQTT) or Advanced Message Queuing Protocol (AMQP) within your portal. Set aside time to create and regularly update a roadmap for your platform that takes into account future needs and opportunities.

Risks and Pitfalls

Risk: Pressure to Begin Implementation Immediately

You may face pressure to jump directly to provisioning and configuring your APIM solution without first discovering your success criteria, defining strategies and implementing processes for support. Doing so creates the risk that your implementation choices will not reflect your organization's needs, and that you will not have processes in place to handle things such as discovering new APIs, providing support and fixing bugs.

To mitigate this risk, you should push back against efforts to move directly to implementation, and put processes in place for API discovery, consumer support and remediation of defects.

Risk: Analysis Paralysis

The converse of moving too quickly is moving too slowly. Organizations that are used to waterfall project management and strong change control can fall into the trap of trying to plan for all possible contingencies. This is not useful; the start of any project is the point in time when you know the least you will ever know about that effort.

To mitigate this risk, time-box your planning activities, and treat your plans as living documents rather than final decisions.

Risk: API Developers Reject the Platform

If you are in the position of bringing order to an existing set of APIs, you may need to do extra work to persuade development teams to bring their APIs onto your platform. API developers may see the platform as needless overhead, or believe that the API gateway adds latency to their APIs.

To mitigate this risk, point out that the APIM platform provides implementations of technologies such as OAuth, so development teams do not have to implement these themselves. In addition, the platform takes care of developer onboarding for APIs, key management and security policy enforcement, and provides capabilities such as rate limiting and circuit breaking.

Risk: Changing Expectations

Change is inevitable. Your organization's expectations for API management may change for any of a number of reasons. New leadership, mergers and acquisitions, changed competitive or regulatory circumstances, new business opportunities or simply new information can lead to dramatic changes in your API programs.

To mitigate this risk, you should focus on frequent, iterative delivery. By working in increments, you can ensure that the impact of changes in expectations, goals or objectives is minimized. Use a methodology such as Scrum or Kanban to organize your work.

Pitfall: Treating API Support as an Operations Problem

Organizations with an unhealthy separation between development and operations teams may treat support for APIs as an ops — rather than a development — issue. The problem with this approach is that the ops team cannot know as much about the application and business domain behind the APIs as the development team responsible for developing the APIs.

To avoid this pitfall, you should secure commitments from development teams to provide support for their APIs and the services behind them, and establish clear communication and escalation paths for routing support requests and bug reports to development teams. Even better, implement a cross-functional team structure and adopt DevOps practices. See "Extending Agile With DevOps to Enable Continuous Delivery" for more information on DevOps practices.

Pitfall: Out-of-Date Documentation Creates Consumer Frustration

If your API development teams are not using consumer-centric, contract-driven development approaches to API development, your API documentation may become outdated. Documentation that does not match APIs creates frustration.

To avoid this pitfall, you should encourage your API development teams to implement contract testing into their software development life cycle (SDLC) and continuous integration pipeline. This means that they should treat their API specification as a formal contract for the interface rather than a description of the API. You should also implement testing of documentation against the API into your own processes.

Related Guidance

This research is a partner document to "A Guidance Framework for Evaluating API Management Solutions," which provides detailed guidance on the process to select an APIM solution. Technical professionals responsible for selecting APIM technology should also read "Selecting the Right API Gateway to Protect Your APIs and Microservices," which provides detailed guidance on the security capabilities of leading API gateways.

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

"A Guidance Framework for Designing a Great API"

"A Guidance Framework for Creating Usable REST API Specifications"

"Protecting Web Applications and APIs From Exploits and Abuse"

"Choosing an Architecture for Managing APIs and Services"

Evidence

[1] From a speech by Dwight D. Eisenhower to the National Defense Executive Reserve Conference in Washington, D.C., 14 November 1957; "Public Papers of the Presidents of the United States," Google Books.

[2] RACI is a project responsibility chart. "The RACI Matrix: Your Blueprint for Project Success," CIO.

[3] "The Future of API Design: The Orchestration Layer," Daniel Jacobson blog.

[4] "Managing API Product Bundles," Google (Apigee).

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp