A Guidance Framework for Evaluating API Management Solutions

Published 20 August 2019 - ID G00383316 - 78 min read

By Analysts Matt Brasier, Gary Olliffe

Initiatives:Integration Architecture and Platforms for Technical Professionals **and 1 more**

API management is critical to the success of APIs as a fundamental feature of modern integration architecture. Application technical professionals selecting an API management solution must evaluate its ability to successfully control, promote, operate and measure APIs throughout their life cycle.

Overview

Key Findings

- APIs are an essential component of any modern application and integration architecture. Their adoption has been driven by their role in microservices, web and mobile apps, cloud services, integration strategies, and regulatory requirements.
- APIs reinvigorate the relevance and value of SOA by focusing on the needs of service consumers. To achieve this, your APIs need to be supported, secured, monitored and managed throughout their life cycles.
- Managing API delivery requires more than deploying an API gateway. The core policy enforcement features of an API gateway are augmented by catalog management, testing, documentation, monitoring and governance features to create an API management solution.
- The set of APIs that an organization must manage extends beyond a single environment when an organization uses SaaS, hybrid or multicloud applications. These APIs must still provide a consistent experience, so they must be governed in a consistent way.

Recommendations

Application technical professionals responsible for selecting API management solutions:

- Evaluate API management capabilities by identifying and prioritizing requirements across five functional areas: deployment and operations, developer enablement, API life cycle management, communications, and measurement.
- Scope and select your API management solution with a developer-centric view. Choose tools that encourage productivity for developers who are both API providers and API consumers.
- Prioritize requirements for integration between API management and the rest of your application and security infrastructure, including identity management, monitoring and automation.
- Favor solutions with a flexible, hybrid deployment model that supports consistent management of APIs using both cloud-hosted and on-premises components.

Problem Statement

How do I evaluate an API management solution?

Application programming interfaces (APIs) are essential to any new integration architecture. They play an important role in mobile apps, modern web architectures, cloud services, digital strategies of leading enterprises, and, increasingly, compliance with industry and government regulatory requirements. [1] , [2]

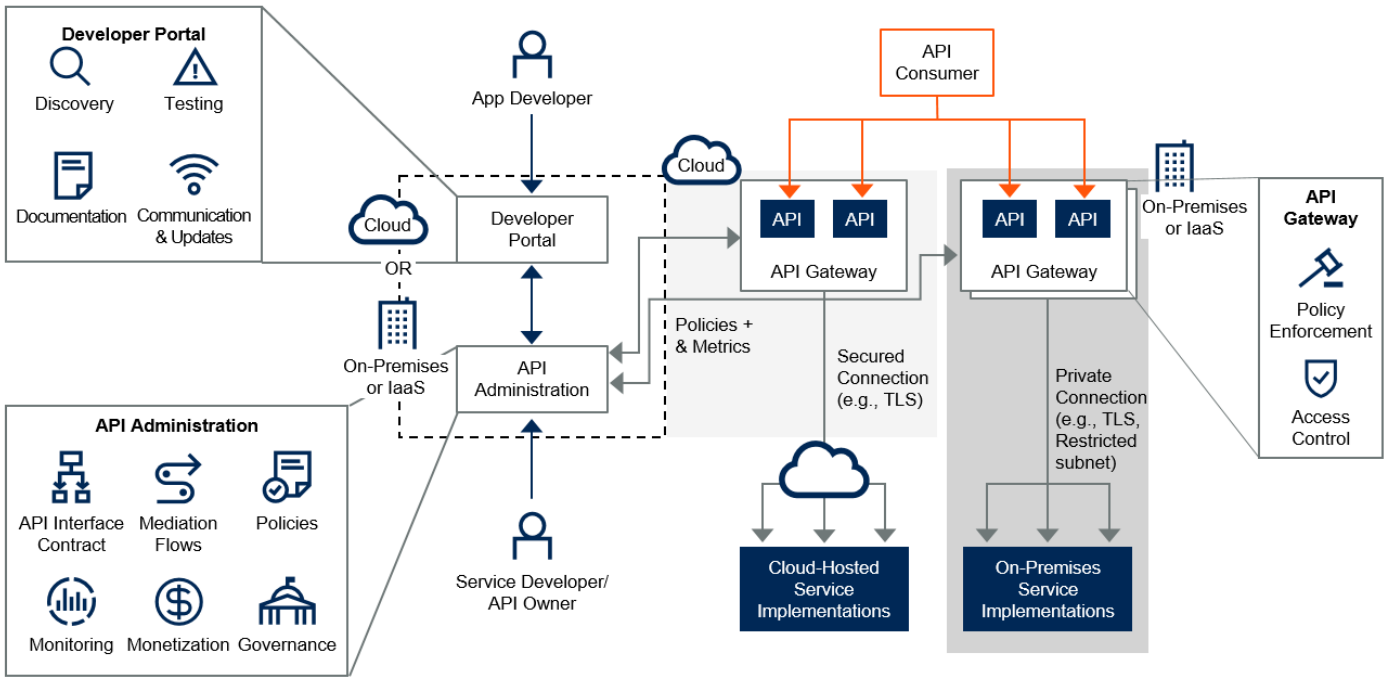
API management capabilities enable the successful delivery, promotion, operation, measurement and continuous improvement of APIs. Although focused primarily on REST-based web APIs, API management capabilities are applicable to a variety of service APIs, including:

- Event-driven APIs
- Simple Object Access Protocol (SOAP)-based APIs (which may or may not use HTTP as a transport)
- Custom APIs

Gartner uses the term “full life cycle API management” to describe the discipline of end-to-end governance for modern application programming interfaces. This report covers both basic API management capabilities and more advanced capabilities that support the successful delivery of web APIs. Figure 1 shows the capabilities and features of a full life cycle API management platform.

Figure 1. Capabilities and Features of a Full Life Cycle API Management Platform

Capabilities and Features of a Full Life Cycle API Management Platform



Source: Gartner
ID: 383316

API management capabilities are delivered using one or more of three logical components or services that must be deployed and integrated into your operation environment:

1. **An API gateway**: One or more policy enforcement points that apply policy to requests at runtime. This might be a cloud-hosted service, a resilient physical deployment (e.g., two or more clustered physical or virtual appliances), or, increasingly, a distributed software deployment with multiple software-based gateways deployed on-premises and in the cloud.

2. **An administration portal**: The policy administration point. This is usually a web-based application for configuring, managing and monitoring API policies and usage activity from an API provider's perspective.

3. **A developer portal**: The API discovery and self-service channel. This is a web-based tool for developers to discover, learn about, register for, collaborate on and monitor usage of the APIs from a given provider.

API management capabilities benefit companies hosting APIs for internal and external consumption. You don't need every capability to start delivering APIs, but as your API program grows, so too will your need for the full set of API management capabilities. Prioritize your acquisition of API management capabilities based on the use scenario, your organizational requirements and the business value implications unique to your project, and have a planned reaction for when your API program is wildly successful.

This guidance framework will prove useful to project teams evaluating API management capabilities to support:

- Delivering a private web API to support mobile, web, internal integration and/or business-to-business (B2B) integration requirements
- Delivering a public web API as a direct or indirect revenue channel or information-sharing channel (e.g., digital government)
- Delivering an enterprise platform web API to enable digital business and/or bimodal IT strategies

This document is a companion to "A Guidance Framework for Designing a Great API," which provides a detailed process for designing APIs, and "How to Successfully Implement API Management," which provides guidance on implementing the API management solution that you have chosen.

The Gartner Approach

Gartner's approach to this evaluation is to identify the API management capabilities that need to be assessed and prioritized. This document provides the "menu" from which you can select these capabilities based on the needs of your API program. Once selection is complete, you should prioritize your requirements across all function areas and use this as the basis for direct vendor evaluation.

Establishing and operating a successful web API begins with a product-centric mindset — that is, treating the API as a product. According to renowned business strategist and Harvard Business School professor Clayton Christensen, the goal of any product manager should be to deliver a product that helps its customers get a job done more easily. [3] When your product is successful, you will have consumers who depend on it as a platform to build their services on, and you must deliver a stable yet up-to-date experience for their developers.

API management platform capabilities support the API governance processes and activities you need to implement. They are used by API providers to offer programmatically consumable interfaces to API consumers (who are often developers of mobile applications or modern web applications, or application integrators). These API consumers may work within the organization, externally in a well-defined business partnership or as part of the growing ecosystem of third-party developers participating in loose relationships with API providers.

API management platform capabilities are delivered in the form of software or cloud services by a variety of market participants, including:

- API-focused players, such as Google (Apigee), Kong and Perforce's Akana
- Enterprise integration stalwarts, such as Axway, Broadcom-CA Technologies (Layer7 API Management), IBM, Salesforce (MuleSoft) and TIBCO Software (Mashery)

- Open-source/open-core vendors, such as IBM (Red Hat 3scale API Management), Tyk and WSO2

API management features may be integrated with or delivered through other vendor offerings, including integration platform as a service (iPaaS), application platform as a service (aPaaS) and mobile back-end as a service (mBaaS) offerings. As a result, although you may choose to mix and match best-of-breed components from across the API management capability spectrum, your ultimate choices cannot be completely decoupled from other technology platform decision making.

> **API management platform capabilities enable API providers to successfully deploy, maintain, promote and sustain an API strategy.**

Gartner's approach to this evaluation is to identify the API management capabilities that need to be assessed and prioritized. This document provides the "menu" from which you can select these capabilities based on the needs of your API program. Once selection is complete, you should prioritize your requirements across all function areas and use this as the basis for direct vendor evaluation.

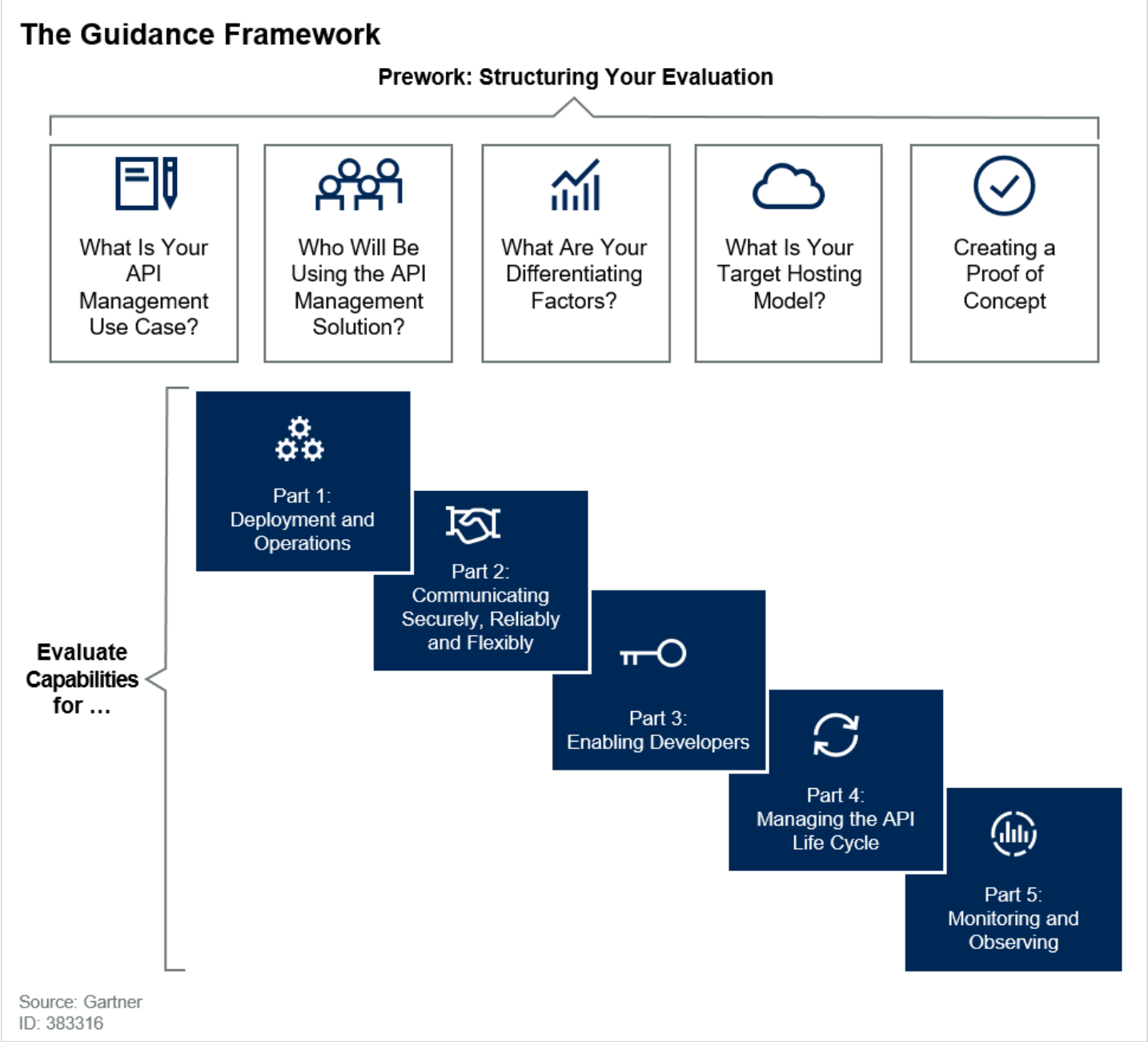
The Guidance Framework

The number of API management capabilities can be overwhelming. This guidance framework starts with a prework section, where you will define the personas that will use the API management solution, and the type of API management you are doing. Next, you will perform an ordered evaluation of characteristics in the following areas:

- **Deploy and operate:** Capabilities that affect how the API management functions are deployed and operated. While nonfunctional in nature, an API management solution is part of your application infrastructure and must fit well within it.
- **Communicate securely, reliably and flexibly:** Capabilities to support the secure and scalable consumption of APIs and the integration of an API to its underlying implementation.
- **Enable developers:** Capabilities to help developers discover and use APIs productively.
- **Manage the API life cycle:** Capabilities to help providers publish, support, version and retire APIs.
- **Monitor and observe:** Capabilities to support the monitoring, analytics and monetization of API consumption.

If you are just beginning your API program, it is unlikely that you will need all these capabilities from the outset. However, it is important that you understand the role they play and their priority for your API delivery program over time. For example, more sophisticated developer enablement capabilities, such as a self-service API developer portal, may not be a priority for your first few API use cases. However, without a good "developer experience," it will be impossible to attract and support the number of developers needed to meet the objectives for your API program (see Figure 2).

Figure 2. How to Evaluate API Management Capabilities



Prework

Before beginning your evaluation, determine which features of the API management solution will help resolve your immediate needs. These priorities will prevent you from investing effort in evaluating features you don't need or selecting a product because of features that you will not use.

Start with three questions:

- What is your API management use case?
- What personas will be using the API management capabilities?
- What is your target hosting model?

The answers to these questions will help to prioritize the features you need, and to ignore those that will not be relevant. This is the work you need to do upfront to understand the problem. This work will also contribute to the creation of a proof of concept (POC).

> **A technical proof of concept is the primary method used to evaluate the suitability of your API management platform. It should be used to evaluate your highest-priority features and their usability by your target personas.**

What Is Your API Management Use Case?

When evaluating API management solutions, a common mistake is to look for a product that offers every feature possible, or one that offers features that you think you would like to use in the future, but are not ready for now. Another pitfall is looking for the market-leading products and assuming that, because they meet the needs of many other clients, they must be able to meet your needs as well.

When making purchasing decisions, it is tempting to overbuy and select a product with more features than you need in the hope that you can avoid the expense of running a procurement process again in the future. However, API management is a fast-changing market where new capabilities (such as support for non-REST APIs) are continuing to emerge. While moving from one API management solution to another requires significant work, it is not as complex as migrating from one integration platform to another, due to the prevalence of standards such as OpenAPI Specification and AsyncAPI.

To help you understand what features are important to you, Gartner has identified the following three broad use cases:

- **Exposing APIs externally:** In this use case, the API management capabilities are used to ensure that any APIs used to communicate or transact with other companies are well documented. Because these APIs are externally facing, they will need to be well supported and reliable, and to be presented and behave in a consistent manner to better serve the other organizations you do

business with. You will also need ways to track usage of the APIs for monetization or marketing purposes.

- **Internal API enablement:** In this use case, APIs are used internally to enable more ad hoc integration and citizen development. In this case, API management capabilities are used to enable standardization internally within the interfaces. In this case, the emphasis is less on the user experience of the people using the APIs, and more about what features are available to enable developers creating the services to discover existing APIs and expose new ones.

- **Service governance:** This use case involves using your API management solution as a tool to enable agile governance of services. This entails using your API management platform for purposes such as enforcing standard data models and standard interaction patterns across your applications, and ensuring you don't have overlaps in provided services. Your API management platform provides your API catalog, access to standards documentation and dependency tracking features.

What Personas Will Be Using the Platform?

You must evaluate not just whether your selected API management solution provides the features you need, but also whether it can expose those features in a way that makes them accessible to your target users. Focus on identifying the personas who you expect to use or interact with the API management capabilities as either users or providers of the APIs under management.

Gartner has identified four personas that can use API management tools:

- **API Developers:** These are the software engineers who create the services that implement and expose APIs. They usually have strong software development and programming experience, but may not if services are developed using low- or no-code platforms. They need to be able to find information about relevant API standards and existing APIs, deploy their API definitions and provide routing information for the back-end endpoints.

- **API Consumers:** The people who consume APIs will often be software developers with similar skills to those above. However, because they are often external to your organization or business unit, they will not be as familiar with your terminology and your business processes. They need good documentation, sample code and test harnesses to help them consume APIs without having to contact the team that created them directly, and will need to create and manage their security tokens and API keys. If you are monetizing your APIs, consumers will want to access billing data.

- **API Product Managers:** Responsible for the API life cycle, API product managers need to understand who is using APIs, how much they are using them, what they like and what they don't like. They will want to manage versions of APIs, communicate with consumers and producers, and define and deploy API consumption policies.

- **API Platform Team:** This is the team that is responsible for managing and running the API management solution. They need metrics and monitoring on the health of the platform itself, and will be responsible for creating and managing integrations with other infrastructure systems such as identity providers, log management and enterprise monitoring.

For each persona, identify the expected skills, experience and needs of the user, and use these when evaluating whether the capabilities of a solution are relevant. For more details on the personas involved in API management, see "How to Successfully Implement API Management."

What Is Your Target Hosting Model?

While API management is often associated with modern applications such as software as a service (SaaS) and applications running on platform as a service (PaaS), it is just as relevant to applications deployed on-premises.

An API management solution can have any of three hosting and deployment models (see Table 1).

Table 1: API Management Solution Hosting and Deployment Models

Responsibility	API Management Solution Software	Cloud-Hosted API Management Solution	Hybrid Cloud API Management Solution
Control plane hosting	You	Vendor	Vendor
Gateway hosting	You	Vendor	Vendor/You/Both
Patching	You	Vendor	Vendor/You/Both

Source: Gartner (August 2019)

There are three aspects to consider for each hosting model. The control plane is the part of the solution that provides management functionality, such as deploying new or updated API policies. The gateway is the proxy that sits in front of your services and enforces policies. The third aspect is who will apply patches to the platform.

The deployment model has a significant impact on the organizational structures and skills you will need:

- If you are selecting a platform that will run entirely on-premises (or on infrastructure you manage in the cloud), you will need an environment in which to run it, and the skills to maintain it.

- If you are selecting a cloud-hosted API solution, you will need to manage the less-predictable areas of cloud billing and vendor-driven update cycles.
- With a hybrid solution, make sure you understand whether you are responsible for patching gateway instances running on-premises or whether the vendor will be responsible for this. A hybrid solution is the most versatile, given the hybrid nature of most organizations' application landscapes, but combines the downsides of both software and service offerings.

Gartner research indicates that organizations evaluating API management solutions have a specific hosting and deployment model in mind based on where their services are hosted. You must identify your preferred hosting model upfront and ensure that your chosen solution supports this model.

For more information on deciding on a deployment model, see "How to Successfully Implement API Management."

Prepare Your Proof of Concept

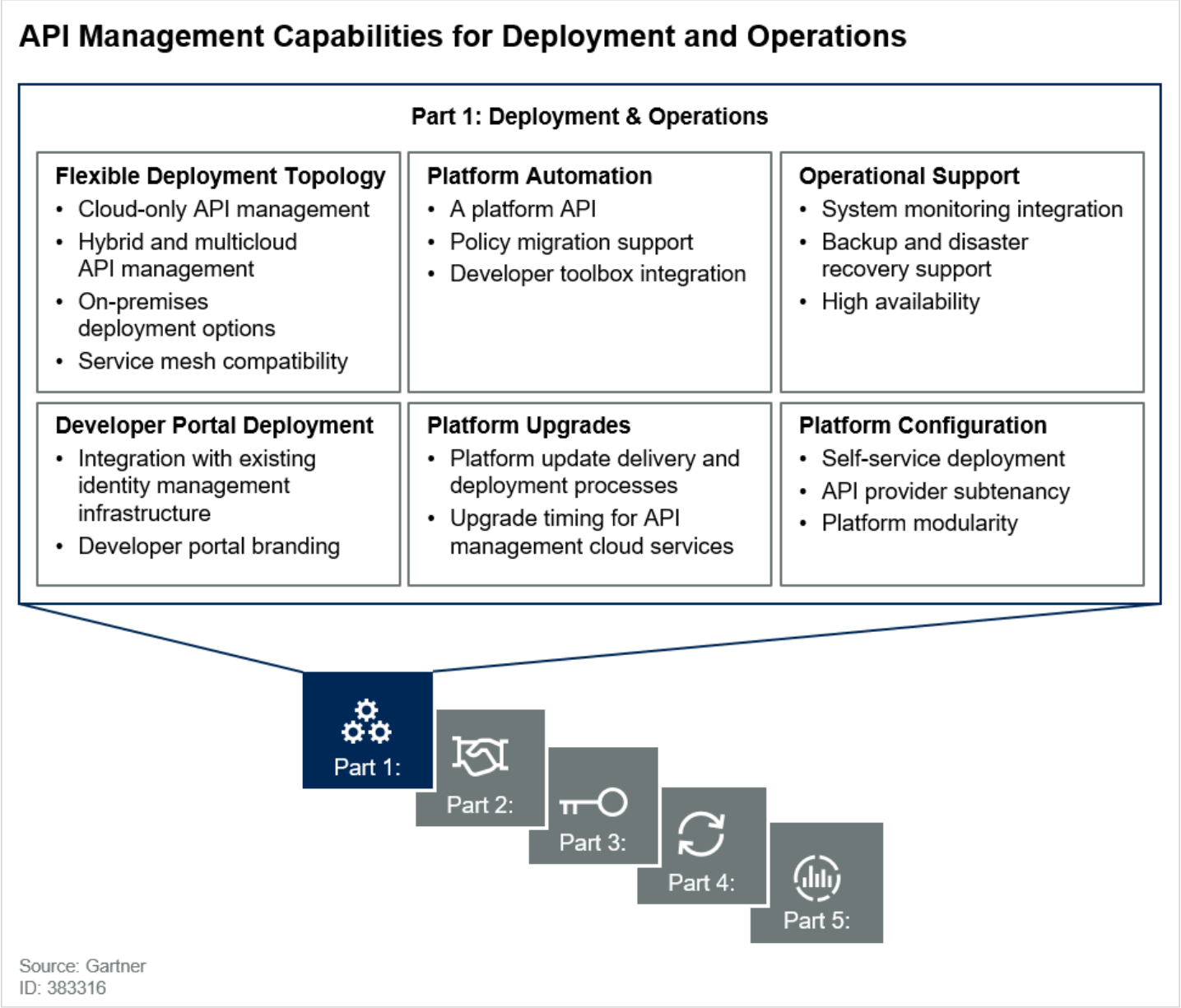
Once you've completed the research needed to answer the questions above, you can use this information to run your POC. Based on the use cases, personas and hosting model identified earlier, use your target personas to evaluate a solution's API management capabilities to see how well they address the areas you think are critical to your current needs.

Because the goal of the POC is to evaluate whether your target users can use the platform, it is important to ensure that the people performing the POC are a set of those target users. POC activities that are performed by the vendors' sales engineers, or by an external consultant that will not be the end user of the platform, create a risk that you will select a platform that is not usable by your intended audience.

Part 1: Evaluate Deployment and Operational Capabilities

An API management solution consists of multiple components (gateways, administration interfaces and portals) that you can deploy and manage in multiple ways (cloud, on-premises, hybrid). Figure 3 summarizes the key capabilities that will support your deployment and operations of an API management solution.

Figure 3. API Management Capabilities for Deployment and Operations



Flexible Deployment Topology

The key to successful API management is to be able to consistently manage and apply policies to APIs that may be implemented by multiple teams and deployed in multiple locations. Some of your services may be consumed internally, while others are intended for external consumption by B2B partners under close control, or more openly by a broader audience of developers. The components of an API management solution can be deployed in a variety of configurations.

Use your API management solution as part of a flexible deployment topology that can adapt to these different requirements, reducing the risk of unacceptable network hops, routing issues or network bottlenecks. The deployment topology will enable you to have a single portal for API discovery and developer productivity, while maintaining an optimal gateway footprint for runtime policy enforcement. The three primary topologies to consider are:

- **Cloud-only API management:** A cloud-only API management service has all three components (gateway, administrator portal and web portal) delivered as a cloud service. All API requests are routed through the cloud provider for policies to be enforced. These services are often priced on a per-transaction model and are suited to scenarios where the services you want to manage are cloud-based or where the apps consuming your APIs are consistently external, mitigating the impact of additional network hops.

- **Hybrid and multicloud API management:** In this model, one or more components (typically the administrator and developer portals) are hosted exclusively in the cloud, while the gateways used to enforce policy at runtime are deployed where needed. This can include gateways managed by the cloud provider and software gateways deployed to infrastructure that you manage on-premises or using infrastructure as a service (IaaS) from a different cloud provider. The hybrid model of API management lets you manage APIs consistently across different environments, including multiple clouds and on-premises. It's valuable to have features that allow you to have a single control plane, enabling you to manage things in the cloud.

- **On-premises deployment options:** These should be flexible to help organizations that are not ready to host some or all of their API management capabilities as a cloud service, or that are bound to on-premises solutions by industry or privacy regulations. This flexibility comes in the form of having a choice. You can deploy your API management technology as a physical or virtual appliance (often preferred by information security professionals), as a software install gateway hosted on traditional server infrastructure, or using software agents or microgateways. The software agent or microgateway model is typically used in conjunction with a primary appliance or software installation, and is beneficial when distributing API management to reduce latency and policy enforcement overheads. An important aspect of the microgateway model is that it places the policy enforcement point nearer to the service endpoints.

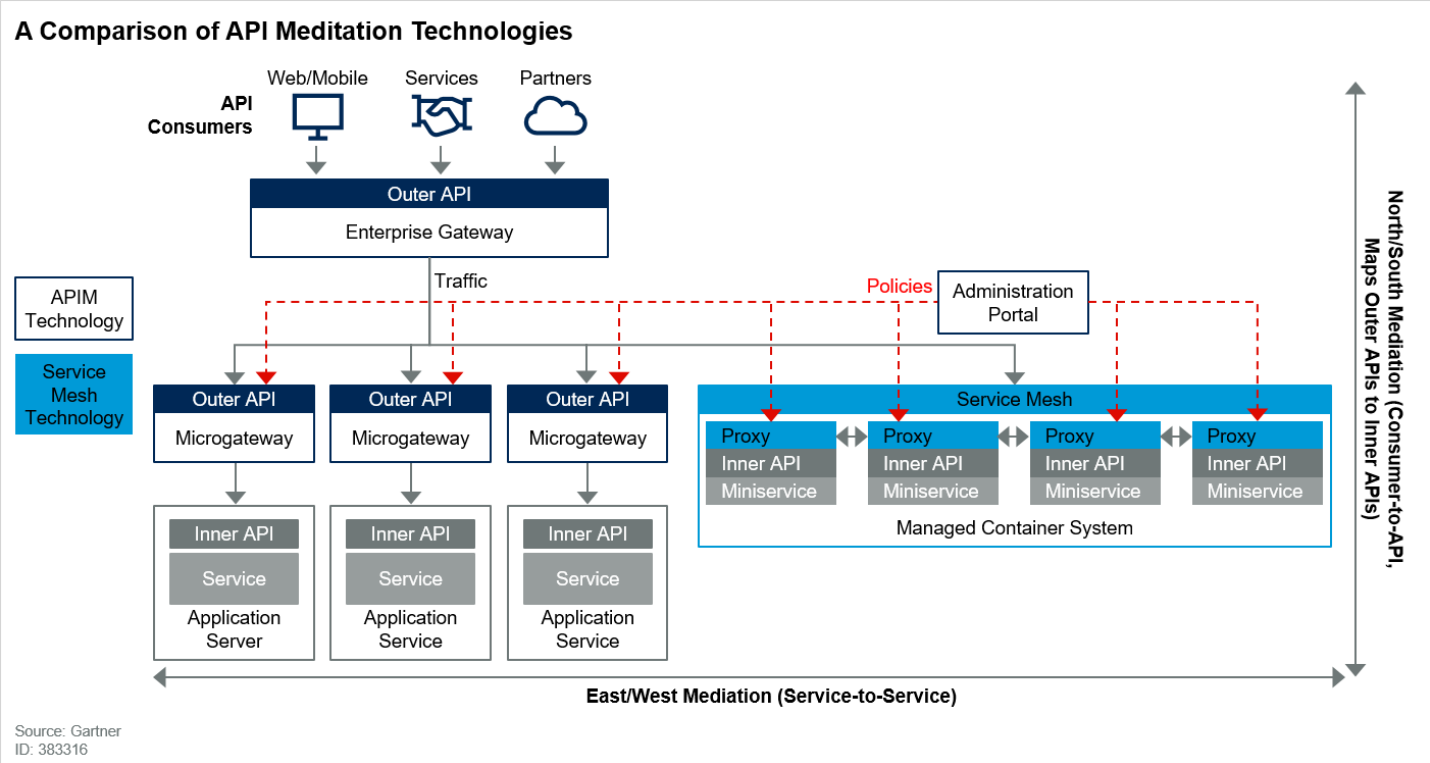
Service Mesh Compatibility

There is overlap between the capabilities of a service mesh and the capabilities of an API gateway. If you are using or planning to use a service mesh in your containerized architecture, you must consider how the API management solution and service mesh can work together to implement security policies and monitoring.

API gateways manage the interactions between API consumers and published APIs. A service mesh focuses on the management of "east-west" traffic between services deployed in a container infrastructure. Both are used to enforce security and service-to-service communication. However, a service mesh is not a replacement for an API gateway. Technical professionals should always manage their consumer-to-API interactions using API gateways (or microgateways), but evaluate whether they can manage service-to-service interactions using a service mesh, as illustrated in Figure 4.

Security best practices dictate that you should have a central policy management platform that routes and manages policies across all your relevant enforcement points. Some API gateway vendors are now delivering plug-ins for popular service meshes — for example, the Apigee adapter for Istio. [4] This enables organizations to create security policies using their API management control plane, and to have these policies enforced by the service mesh. It also allows your API gateway to observe and manage service-to-service communications occurring within the mesh. For example, a policy stipulating that Service A can talk to Service B could be wired into the service mesh, rather than you having to deploy a microgateway container into the service mesh to manage policies. For more details on service mesh capabilities, see "Assessing Service Mesh for Use in Microservices Architectures."

Figure 4. Service Mesh and API Mediation



If you are using containerization platforms such as Docker, consider a containerized deployment approach, where API instances are run in your containerized infrastructures. In this case, the ability to wrap your gateways in containers allows you deploy and manage them using the same tools and processes as the rest of your infrastructure.

Platform Automation

The configuration of your APIs and the policies applied to them is closely tied to your software delivery life cycles. When implementing continuous integration (CI), continuous deployment (CD) or release automation, you will need the ability to build deployment and modification of API policies into your processes. To support this, you must evaluate how an API management platform can be integrated for automation purposes:

- **A platform API** for an API management platform might seem recursive, but it is increasingly common for API management platforms to expose their core capabilities (such as user management, API definitions and policy definition) via RESTful APIs, which are then both used by the vendor to implement administrator portal features and published for automation and integration purposes. When you need to support CI/CD processes, ensure that you evaluate the API, its documentation and any wrappers that may assist (e.g., software development kits, plug-ins or command line interfaces).

- **Policy migration support** lets you move policy definitions from one environment to another. Your policy deployment workflow will likely involve scenarios where you must test API definitions and policies in nonproduction environments and then promote them to production environments. This might be supported by direct connection between API management environments or simply by exporting and importing a comprehensive set of definitions and policy metadata.

- **Developer tooling integration** improves developer productivity by allowing developers building or consuming APIs to access APIs directly from their development tools. For example, a number of API management vendors (including Axway, Broadcom-CA Technologies, IBM, Microsoft, TIBCO Software and WSO2) provide plug-ins for SmartBear's SoapUI and ReadyAPI test tools to allow testers to import API definitions directly from the API management platform. See "A Guidance Framework for Creating Usable REST API Specifications" for more details on API developer tooling.

Operational Support

As with any other critical part of your application infrastructure, you will need to integrate your API management capabilities with your operational processes to ensure the robustness and recoverability of the platform itself. The extent to which you manage elements of your own API management infrastructure, such as administration portals, and associated policy and data stores or gateways deployed on-premises or to IaaS, will influence the sophistication of your requirements for this capability:

- **Platform health monitoring** tracks the performance of the platform itself (e.g., resource utilization, log file sizes and other storage consumption) and supports notification based on specific events to allow you to take immediate corrective action.

- **Backup and disaster recovery (DR) capabilities** will help you select an API solution that will fit with your own SLA and operational level agreement (OLA) commitments for the APIs you will be managing. For hardware and software deployments, you must evaluate the steps required to replicate or restore configuration and data between redundant or recovery deployments. You must plan and test the DR capabilities of your API management solution because it is a critical part of your routing and security infrastructure. For cloud-hosted services, you must evaluate the SLAs

offered and satisfy yourself that the provider's own resilience and DR processes have a high probably of success.

- **High availability** is also important due to the role of your API gateway as a critical part of your application, security and network infrastructure. In a cloud environment, that should be handled by the cloud provider. In a hybrid or completely on-premises deployment, however, you'll need to plan for the instances you're managing. You can accomplish this using the usual approaches, such as active-active multisite topologies, or active-passive DR locations.

Developer Portal Deployment

The developer portal for your API is a critical discovery and support channel, and will be part of the overall customer or user experience that is critical to the success of your API program. While the content within an API portal is often configurable using content management capabilities (see Part 3: Evaluate Capabilities That Enable Developers), you may also need to integrate and customize the functionality of the portal to achieve your goals:

- **Integration with existing identity management infrastructure** allows you to support authentication and authorization of developers via single sign-on (SSO) for existing user identities, and to use existing user registration or provisioning processes. To support this, your API management solution will need the capabilities to support identity federation using technologies such as:
  - Security Assertion Markup Language (SAML)
  - OpenID Connect (OIDC)
  - Lightweight Directory Access Protocol (LDAP)
  - Java Database Connectivity (JDBC)
  - Open Database Connectivity (ODBC)

- **Developer portal branding** controls the layout, style sheets and images used to deliver a developer user experience in line with your organization's internal or external branding requirements. This may be an extension of content management capabilities for the portal, but branding is typically changed separately from content (e.g., upon first deployment and when there are new branding standards to implement). This can be particularly important for external API programs in highly brand-conscious organizations.

Platform Upgrades

Effective API management solutions are a critical part of your design, build and runtime environment, and you must evaluate their mechanisms for upgrade and patching. The significance of this will be

affected by your choice of deployment architecture: cloud, hybrid or on-premises:

- **Platform update delivery and deployment processes** must be evaluated so that you can plan your operational processes accordingly. This is particularly relevant for the components that you deploy and manage yourself, including physical and virtual appliances, and installed components, including software agents and microgateways. Vendors may have different classes of update that use different delivery channels, and installation requirements that may require downtime for some or all of the components of the solution.
- **Upgrade timing for API management cloud services** will be the responsibility of your provider. However, you should evaluate its policies on outage windows, SLAs and whether you have any control over the timing of updates to the environment. You should also understand how changes to the platform will be communicated to you both before and upon completion of the change. The level of control you can expect will vary between multitenant, shared-everything cloud services and shared-nothing managed services. Hybrid-cloud hosting arrangements create additional complications for synchronous deployment of new API management platform releases.

Platform Configuration

When planning an API management deployment, you must consider a number of configuration-related attributes or capabilities. These will affect both the short-term and long-term viability of your selection, and you should prioritize accordingly:

- **Self-service deployment:** Use API management not just for the services that you see as strategic and business-critical, but for all services exposed via APIs. Many of an organization's most-used APIs emerge from experimental prototypes. These APIs need governance too, and the ability of a team to provision and operate its own API management capabilities, whether in the cloud or on-premises, can often be critical when operating in an agile environment.
- **API provider subtenancy:** When you have more than one API, you need to be able to manage each independently. The IT staff responsible for administering and monitoring each API may be different, and your APIs may be aimed toward different audiences with different access rights and development objectives. The most common example of this is organizations publishing APIs for external use separately from APIs for internal developer use cases. In these scenarios, subtenancy allows multiple APIs to be managed within a single environment, thus maintaining the benefits of centralized oversight and platform integration while allowing some independence for each tenant API.

  This subtenancy can be supported by the following capabilities:

  - Delegating access to limited feature and policy sets to the administrators of individual APIs while retaining some overall control

- Publishing multiple API portals for different sets of APIs, potentially with different branding and access control configurations
- Rolled-up monitoring, analytics and administrative control across all subtenants (i.e., these are not simply co-tenants in a multitenant environment)

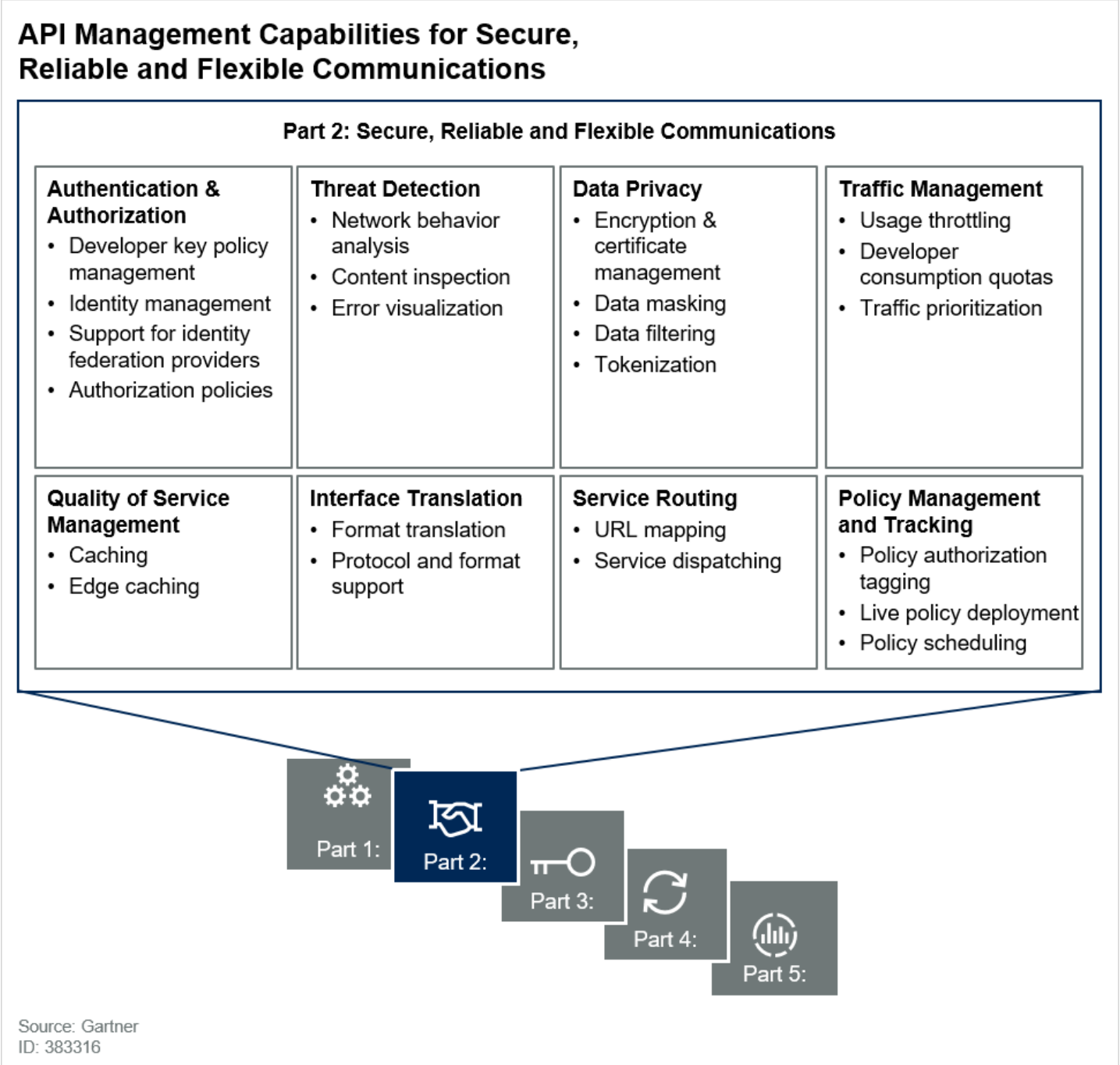
- **Platform modularity:** Evaluate platform modularity from two perspectives:
  - Best-of-brand: For a single-vendor implementation, consider whether you can start with basic features needed for essential management of your APIs, and then make additional investments to add more sophisticated features when needed.
  - Best-of-breed: You may choose to compose your API management solution from a core API gateway, supplemented by self-generated and hosted API documentation, software development kits (SDKs) and supplemental monitoring and analytics.

  The latter approach affords substantial flexibility to control the developer experience, but will add significant operational complexity and require more manual or scripted control to ensure consistency between API contracts, developer portal content and runtime policies.

Part 2: Evaluate Features for Secure, Reliable and Flexible Communications

The core of an API is communication, which provides for a consistent, protected and adaptable flow of service invocations and data between applications and application components. When applied to web APIs, communication always occurs using web technologies (e.g., HTTP). The ideal API allows consumers to interact with the service it exposes in the manner they prefer — be that REST, SOAP or other protocols, such as web sockets, MQTT, gRPC and GraphQL. The capability to communicate securely, reliably and flexibly provides control over service mediation, security and performance, as summarized in Figure 5.

Figure 5. API Management Capabilities for Secure, Reliable and Flexible Communications



Authentication and Authorization

Authentication is the process of uniquely determining and validating the identity of a user so that proper access to information resources can be granted (see "Selecting the Right API Gateway to Protect Your APIs and Microservices" and "Building Identity Into Microservices"). Controlled access to APIs is predicated on a robust authentication scheme that incorporates the full spectrum of authentication life cycle management:

- **Developer key policy management:** Unless an API is offered without regard for authentication, developers who wish to consume it must establish an identity with the provider. In public APIs, this is often done through developer API keys. These keys are typically issued and controlled via the API management platform. Your API management platform should therefore have a developer key

policy management capability that enables key issuance, tracking and revocation. This capability may be implemented through the integration of a native identity management capability or through support for third-party identity federation providers.

- **Identity management:** Identity management services enable management of user identities and control user access to applications and other services. Identity management capabilities grant or deny access in keeping with policies defined by the organization that owns and controls the requested resources. These services allow access based on identity attributes, including a user's identity, permissions and role information. If you do not already have an identity management capability in-house or through a third-party provider, you should seek an API management platform that addresses this need by providing identity provider and secure token service capabilities. Alternately, if you do have existing identity management infrastructure, your API management solution will need to integrate with it to support authentication and authorization of both API developers and the users of the apps that call your API.

- **Support for identity federation providers:** Identity federation supports access across multiple identity domains. In federated environments, domains exchange just-in-time assertions of identity attributes or events, such as whether a given user has logged into a given site and has a particular set of permissions. Identity federation may be established across internal business units, affiliated enterprises or public identity networks. Your API management platform should include support for identity federation providers and common protocols, such as SAML 2.0, OpenID Connect and OAuth 2.0 (see "Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST"). This will enable you to delegate authorization to a best-of-breed identity management platform if desired.

- **Authorization policies:** Authorization capabilities enforce policies by denying unauthorized access requests and granting authorized ones. This context can include attributes, entitlements and roles as well as external factors, including geolocation, transaction value and time of day (see "A Systematic and Practical Approach to Optimizing Authorization Architecture"). As with authentication, this capability can be implemented internally, native to the platform or through integration of a third-party authorization service.

Threat Detection

In the case of a public API, the likelihood of bad actors making attacks on your infrastructure is relatively high. Even with a private or enterprise API, such threats are not outside the realm of possibility. As a programmatic channel into your enterprise, it is critical that you identify and address any attacks or misuse of your API. Therefore, your API management platform should incorporate a threat detection capability, or integrate with your existing capabilities:

- **Network behavior analysis:** Network behavior analysis provides mechanisms to detect policy violations on the network. It is extremely useful in circumstances where consumerization has

taken hold — such as in a public API initiative — because it aids in detecting attacks or misuse from endpoints into which the enterprise has poor visibility. Your API management platform should incorporate network behavior analysis capabilities or integrate with third-party tools that provide them.

- **Content inspection:** Content inspection provides algorithmic evaluation of network traffic payloads. This is important in an API program where a nefarious API consumer could attempt to hack the system by repeatedly sending through different test values intended to discover an exploitable weakness in the API's security mechanisms. A content inspection capability and associated warning mechanisms can alert API managers to abusive consumption of the API by particular users. Your API management platform should therefore either provide support for integration with third-party content inspection tools or embed a native implementation. For further details, see "Protecting Web Applications and APIs From Exploits and Abuse."

- **Error visualization:** Repeated errors from a small number of users could indicate a problem with the API design or something more sinister, like a hacker attempting to find an exploitable weakness. Errors should be logged, and those logs should be subjected to data visualization techniques that expose patterns. API managers can use these patterns to make informed decisions about API design changes or to identify abusive users. Your API management platform should include such a capability.

Data Privacy

Your API exposes data that may contain private information. This may be acceptable for authorized users. For other users, you may desire to only expose less-sensitive information. Your API management platform should include data privacy capabilities that act in unison with authentication and authorization:

- **Encryption and certificate management:** Sensitive data should be encrypted with digital certificates in transit. Therefore, your API management platform should support Secure Sockets Layer/Transport Layer Security (SSL/TLS). In some circumstances, you may desire to apply additional encryption to data elements within the payload (going beyond the encrypted communication channel provided by SSL/TLS). If your use scenario requires this level of encryption, you should seek out the capability in your API management platform.

- **Data masking:** Data aliasing protects information while maintaining data formats. Runtime data aliasing is the reversible replacement of confidential data with similarly formatted surrogates. Data can be aliased using format-preserving encryption or (pseudo) random replacement using a lookup database (see "Protecting PII and PHI With Data Masking, Format-Preserving Encryption and Tokenization"). This capability is useful when your API converses partially in standard data formats, but you do not wish to expose the full set of data contained within that format to API consumers. If your use scenario requires this kind of runtime encryption, you should seek out the

data-masking capability in your API management platform or ensure that it will successfully integrate with a third-party data-masking technology.

- **Data filtering:** Data filtering can be used to filter or remove data from response data to prevent it being leaked through an API to consumers. Filtering rules are policy-based, so they can be used to provide different levels of data access to different classes of developer while allowing each to use the same underlying service implementation.

- **Tokenization:** Rather than removing data from a payload, tokenization replaces it with a token that can be used as a proxy for the actual data value. For example, the API consumer might receive a token in place of a Social Security number or other sensitive identifier, or the gateway might replace a credit card number submitted to the API with a token that is then passed to underlying service implementations to prevent them from requiring PCI certification. In either case, when the token is passed back through the gateway, it can (if necessary) be replaced by the original data.

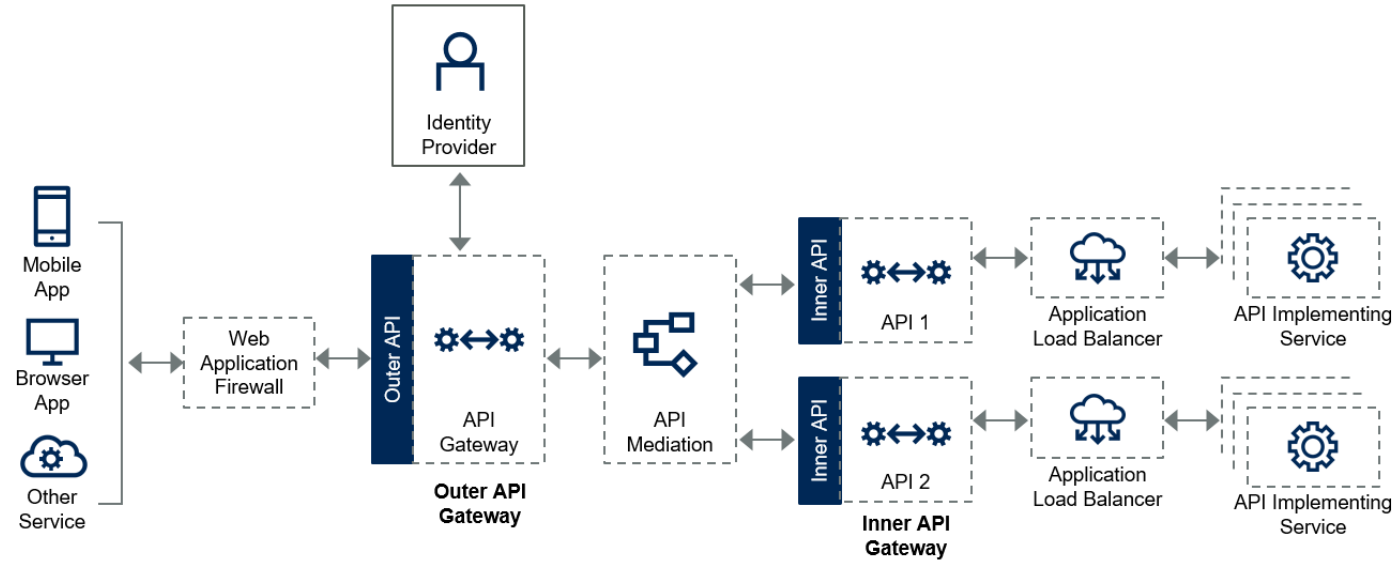
Traffic Management

Your API serves different constituencies, and each offers a unique business value to your organization. API managers link traffic management strategies to the relative value of a particular constituency or class of consumer. For example, an internal development team creating mobile apps under the corporate brand may get unfettered access to high-performing API calls, while third-party consumers of a free tier on your public API may not be entitled to anything more than a small sampling of high-performance API calls.

Note that you should not use your API gateway as a replacement for an application load balancer that splits calls across specific service instances. API management fits into your architecture in front of any load balancing for back-end services, but behind security features such as your firewall and anti-DDoS protection (see Figure 6). API management can provide security protection functions (e.g., rate limiting) in front of services, and is part of your security architecture. It also makes routing decisions and performs protocol transformation as part of your integration architecture.

Figure 6. Load Balancing and Security in the API Architecture

Load Balancing and Security in the API Architecture



Source: Gartner
ID: 383316

API management traffic management features to consider include:

- **Usage throttling and rate limiting:** Usage throttling provides a mechanism to slow down subsequent API calls to improve overall performance and reduce thrashing during peak periods. In pay-per-use APIs, throttling is a necessary feature to ensure that developers are motivated to buy the appropriate service tier. In free APIs, throttling is still needed to ensure that infrastructure is not overwhelmed with requests of a certain type or from a particular customer or consumer group. Your API management platform should include basic traffic management capabilities that enable the regulation of request volumes. Rate limiting is similar to usage throttling, but when the consumer's threshold is reached, further calls are rejected.

- **Tiered consumption quotas:** Developer consumption quotas allow the API consumer to "use up" an allotted number of API calls; subsequent calls are throttled or halted. Like throttling, quotas are a necessary feature in pay-per-use APIs. Your API management platform should include a quota capability that allows you to assign specific API call targets to different user classes or service tiers.

- **Traffic prioritization:** Even in the face of throttling and quotas, a popular API can become a major source of traffic for the back-end services it exposes. An API initially constructed to support a mission-critical mobile app can easily be overwhelmed by popular public demand. Traffic prioritization enables API managers to determine which classes of traffic or consumer groups are of utmost importance, ensuring that API calls from those applications are treated with high priority, while lower-value API calls are pushed down the pile. Unless you expect to add limitless capacity to your API infrastructure, you should seek a traffic prioritization capability in your API management platform.

Quality of Service Management

Your API may become so heavily used that quality of service (QoS) to consumers is degraded. Traffic management is a useful tool for prioritizing some API usage over others, but it may also be too coarse-grained a lever for your initiative. In this case, additional QoS management capabilities are useful:

- **Caching:** APIs that incur a heavy read-only load and for which results are relatively static are candidates for caching. For example, citizen-oriented APIs, like those provided by the U.S. Census Bureau, make heavy use of read-only interaction patterns on a static, albeit very large, dataset. This is an ideal scenario for API-management-level caching, which could offload work from underlying databases and services that otherwise might be overwhelmed by API consumer demand. Although caching can also be implemented within other layers of the service infrastructure, incorporating it into the API management capability portfolio gives API managers and designers control over how caching is implemented.

- **Edge caching:** Besides caching within the API management infrastructure, some vendors provide integration with third-party caches such as those offered by Akamai and Amazon. Pushing data to the "edges" of the network, where it is physically in close proximity to end consumers, can improve application performance while reducing load on the service delivery infrastructure. Edge caching is particularly useful in scenarios where an API will be consumed over a wide geographic area (e.g., a mobile app offered by a multinational corporation with end customers in many markets). A good example of an API that can make good use of edge caching is the Best Buy Products API. It contains a large amount of relatively static data that can be pushed to the edge of the network and only occasionally expired. If you're engaging in API management primarily to support a mobile campaign, make support for edge caching a priority.

Protocol and Format Support

API management platforms are used to expose an existing set of data and processes. API design often requires an exercise in translation to ensure that the enterprise's existing service and database portfolio is surfaced in a well-formed, intuitive manner. Interface translation is at the heart of service-oriented architecture (SOA) governance and is an important component in API management:

- **Protocol support:** While most of the APIs you expose and consume will use HTTP as the underlying transport (e.g., REST and SOAP), there is an increasing awareness that not all APIs are based on HTTP, or even on a request/response pattern. Determine whether your organization is planning to use event-driven protocols and patterns, such as Advanced Message Queuing Protocol (AMQP) or MQTT or webhooks, or emerging protocols such as GraphQL, and prioritize them accordingly. Having support for these at a protocol level, rather than just treating them as HTTP or TCP endpoints, will allow you to provide a consistent developer experience, independent of the protocol and interaction style used when accessing your data. Note that, as of the time of

publication (August 2019), there is very limited support for protocols other than REST and SOAP in API gateway implementations, so this may be a differentiating factor in your evaluation.

- **Format translation:** Data exposed by an existing service may not be formatted in the manner needed by API consumers. For example, a proprietary service interface may respond to requests using XML. This kind of data format will be unacceptable to developers who want to consume JSON data. Unless you're building your API entirely from scratch (without the benefit of existing services and data), format translation is an important capability for your API management platform to offer you at design time.

- **Protocol translation:** Your existing services may be designed in accordance with a protocol not suitable for your new API and its consumers. For example, the widely used SOAP protocol is usually not suitable for public web API initiatives, but you may have a large catalog of existing services designed with it. In such cases, you need a protocol translation capability so that you can redefine the protocol to something more appropriate, like REST. Note that, while some platforms offer the capability to automatically convert SOAP to REST, the resulting APIs suffer from all the faults that made your SOAP services hard to use and understand. Your consumer-facing APIs should be designed, rather than automatically generated. See "A Guidance Framework for Designing a Great API" for more details.

Service Routing

As an API and its underlying services evolve, it is highly likely that multiple versions of a service will be available to support a particular piece of functionality. It is also probable that, in a versioned API, you will need to support two or more ways of doing similar things to preserve backward compatibility. Service routing capabilities ensure that the correct version of a service is invoked by the correct API:

- **URL mapping:** URL mapping allows you to map or transform data into and out of the URL. For example, your API may include version information in the URL root namespace, but this must be removed before forwarding the request to the implementing service (and used to route the request to an instance that handles the correct version).

- **Service dispatching:** Service dispatching is the process by which the API management platform selects and invokes the appropriate service (or services, in the case of orchestration) and returns the result to the API consumer.

Policy Management and Tracking

Policy enforcement is central to the capabilities of any API management solution. However, there are some generic capabilities for the management and tracking of policies that need to be evaluated:

- **Policy authorization tagging:** When a policy has been applied to a given request, it may be necessary to tag the request so that downstream services can be sure both that the request has been processed by the policy enforcement point (i.e., the API gateway) and that it has passed specific policy rules. For example, this tagging may take the form of injecting headers into a request or decorating the payload of a request with the tagging information.

- **Live policy deployment:** The policies that you define need to be published to the gateway to take effect. Real-time policy enforcement ensures that there is no disruption in service as new policy is applied.

- **Policy scheduling:** The timing of deployment for a new policy version may not be immediate, and you should have the option to choose when a policy is deployed or revoked. This may extend to the ability to apply different policies at different times of day. For example, you may offer a more relaxed throttling policy outside of core business hours when your systems have spare capacity, or you may need to apply a more restrictive throttling policy for known periods of critical business processing.

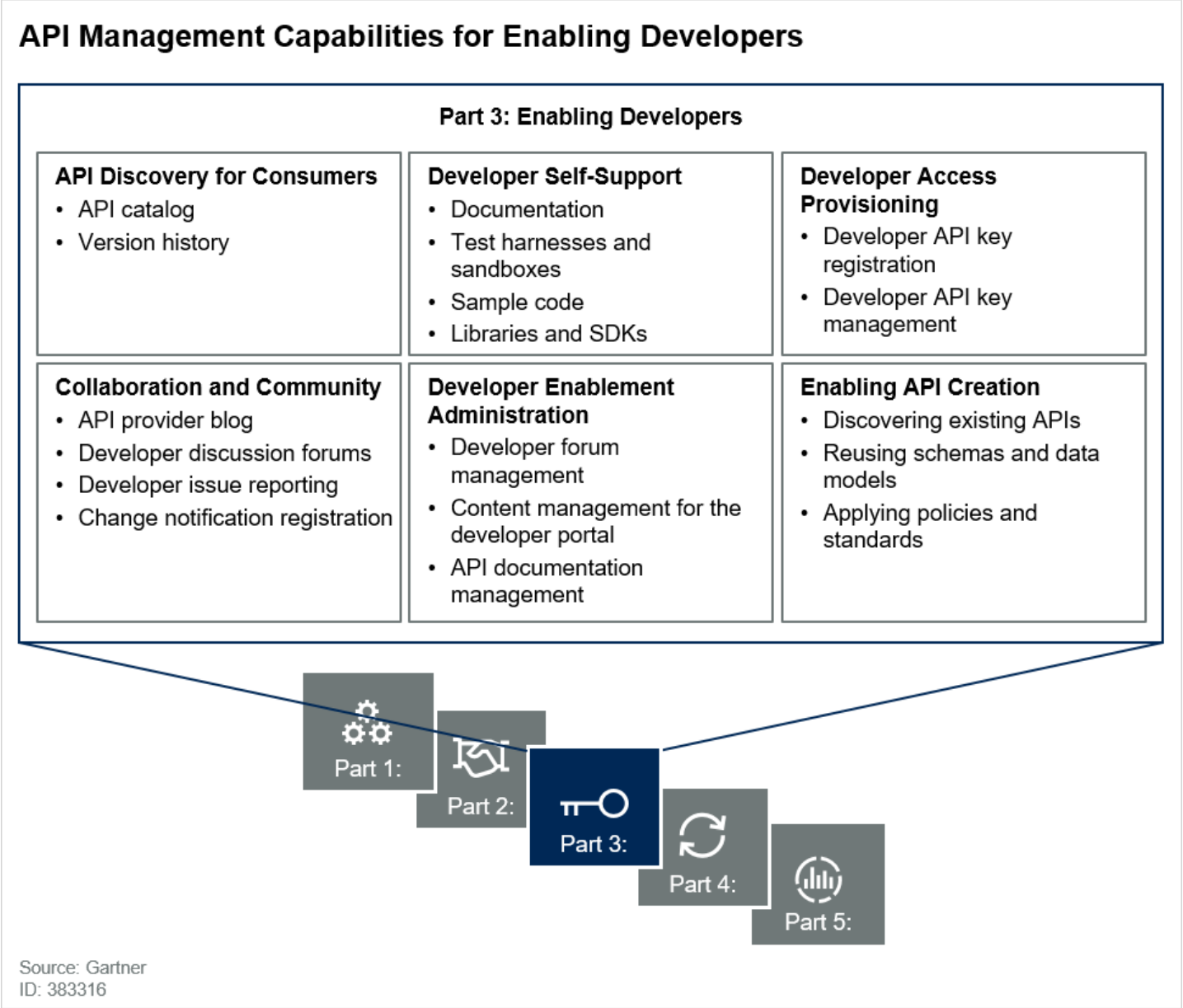
- **Policy exception guidance:** What should you do if you see threat detection or traffic management policies being broken? An advanced API management solution should provide you with guidance that will help you take the appropriate action to handle specific policy violations and protect your API and services. This can include guidance on creating policy rules for handling exceptions, and more dynamic guidance on actions to take in the event of network behavior policy or other threat detection policy being triggered.

Part 3: Evaluate Capabilities That Enable Developers

APIs are used by API consumers in programs and for integrations. Because an API is programmatically consumable, the primary consumers of an API are, by definition, software developers. Developers may use an API to support a mobile application, to integrate smart things into a digital business process, to create a custom integration between two or more unique applications, to back a web application, or in myriad other ways not readily predictable by the API provider. It is therefore important that an API offer flexible ease of use for its consumers.

To that end, the ability to “enable developers” includes several capabilities important for improving the lives of API consumers, as summarized in Figure 7.

Figure 7. API Management Capabilities for Enabling Developers



API Discovery for Consumers

API consumers use discovery metadata to find APIs so they can:

- Find the API that offers the service or data they are looking for
- Compare changes between API versions
- Identify the correct version of an API for their needs

API providers should offer discovery metadata so developers can find the services exposed through the API and put them to use. Without this, even a great API will go undiscovered (for more information, see "How to Successfully Implement API Management"):

- **API catalog:** The API catalog is the repository of the APIs you publish and their associated metadata. It provides a discoverable record for runtime information and is the first place

developers search to find an API, usually via the developer portal (if deployed). As the number of APIs under management increases, a well-structured catalog, helpful descriptions and other metadata become critical to ease of discovery and governance. The API catalog is referred to by myriad names in the industry, including “API store,” “API registry” and “API repository.”

- **Version history:** Providers may wish to support several versions of their API or, at a minimum, provide history for new developers just starting to work with their API. Version history provides this information and instructions for simultaneous use of multiple API versions.

Developer Self-Support

Part of the value of APIs is that developers can get work done without assistance. Self-support capabilities improve the opportunities for the service that the API exposes to generate business value and free up your developers from common direct requests from API consumers. Developer self-support resources are created and maintained by the API provider for the benefit of API consumers. API providers that skimp on self-support capabilities are likely to incur higher costs for technical support and help desk resources. Worse yet, without great documentation and “solution accelerators” in the form of test and development kits, developers may bypass an API and access your data store directly, or use the API of a competitor. Self-support capabilities include:

- **Documentation:** Documentation includes both API definitions and tutorials for learning how to interact with the API. It may be created and maintained by hand or generated by a tool. API specification languages such as OpenAPI Specification (formerly known as Swagger Specification) and RESTful API Modeling Language (RAML) include the ability to embed documentation within the formal specification. For a detailed analysis of these API definition formats and how they fit into the API design and development life cycle, refer to “A Guidance Framework for Creating Usable REST API Specifications.”

- **Test harnesses and sandboxes:** A test harness is a downloadable project or code snippet that a developer uses to try out or provide proof of concept for an application. In the context of an API, which is hosted by a provider, a test harness takes on a slightly different meaning as a test endpoint. Although a downloadable component may still be offered, the sandbox exposes test endpoints that are colocated with the API. Developers can experiment with different API calls, both programmatically and by hand, to better understand the different types of responses possible under various circumstances. A test harness or test endpoint is a good visual tool for helping developers better comprehend API structure and functionality. For example, it is a great idea to include an API credential test endpoint that allows an API consumer to confirm that an API key works before attempting to use it in a project. The ability to submit test transactions to an API without creating a client application is an important learning and validation tool.

- **Sample code:** Sample code, sometimes called a “quick start,” is also often provided as a downloadable project to help an API consumer get an application up and running quickly. Astute

API managers will use a combination of monitoring metrics and consumer interactions to understand the most commonly used API features and the most important usage patterns. Offering sample code in a variety of programming languages that implement the key usage scenarios is a great way to help new developers onboard quickly.

- **Libraries and SDKs:** By default, APIs are provided in stand-alone form without any downloadable client components intended for use in a production application. Although this "raw" version will be acceptable to many developers, certain audiences may prefer device-specific libraries when interacting with the services fronted by an API. For example, an iOS developer might like to use a downloadable library that could be compiled into an application. Use of this library would allow the developer to interact with the API through a well-defined object model rather than through low-level API calls. Such libraries are referred to as "SDKs" once they approach full coverage of all API features. The easier the SDK is to use, and the more complete the functionality it addresses, the more productive a developer using it will be.

  Developing an SDK from scratch can be expensive. The provider must decide which platforms to support. While iOS and Android may seem to be obvious choices for SDK initiatives targeting mobile developers, only a thorough, objective analysis of the developers using or interested in using the API will reveal where investments should be made. Once you've decided to support one or more platforms with an SDK, you will be saddled with the ongoing cost of maintenance, as with any other software product.

  Another option is to generate SDK code from the OpenAPI Specification. SDK code generation is available via stand-alone open-source tools like Swagger Code Generator, as web-based tools like APIMatic and REST United, and integrated into API management solutions like Amazon API Gateway. The code generated by these tools follows generic patterns and structures and will provide an optimal developer experience, and it may be adequate for your API consumers or be used as the basis for customization and enhancement.

Developer Access Provisioning

To use an API, API consumers must first be provisioned with access credentials of some kind. They must also have the means to manage their credentials, particularly for more advanced or data-sensitive API implementations that grant API consumers the flexibility to manage the access privileges of multiple participants under a single API consumer account. You need a self-registration process to enable developers to onboard themselves — otherwise, you'll have to do it manually, which is an expensive proposition:

- **Developer API key registration:** Developers expect that public APIs will provide a self-registration feature that allows API consumers to enroll in or modify their consumption. If you do not have these features, potential consumers may look elsewhere for APIs that are easier to get access to.

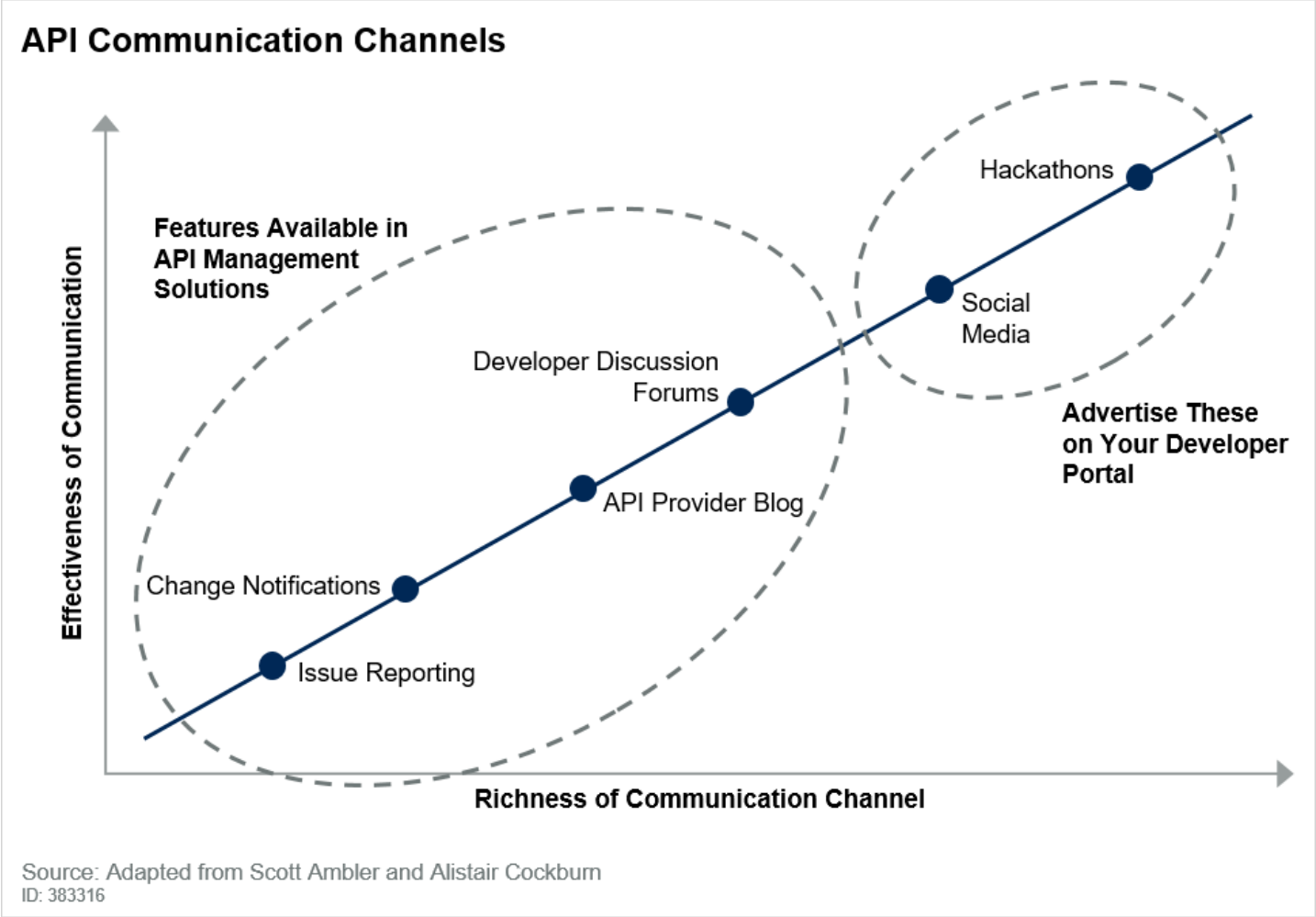
The provisioning request may be serviced automatically, or it may result in an entry in a registration queue monitored by a member of the API management team. When the request is approved, the API consumer is issued an API key that can be used for future requests. Either way, for a public API in particular, developer access provisioning via self-service is critical. Without it, every new user must be added manually by the API management team. Note that developer identities could be managed externally, and you might be using federated identity management to provide authentication using existing external credentials (see also the "integration with existing identity management infrastructure" bullet in the Developer Portal Deployment section).

- **Developer API key management:** Some APIs provide a means for API consumers to have master/detail relationships with themselves and others. For example, lead developers may grant themselves the ability to access an API in production, but may allow other developers on the team only preproduction use. This approach is of particular interest in paid APIs that generate a cost with each API call (or within banded, consumption-based tiers). For these APIs, a key management capability is important so that API consumers can add, modify or revoke API keys within their organization.

Collaboration and Community

In contrast to developer self-support, which is created and maintained by the API provider, collaboration and community capabilities turn the API provider into a curator, enabling developers to help each other. Wikis, social networks and activity streams are great vehicles for collaborative problem-solving and community interaction. Figure 8 illustrates how different communication channels between API providers and developers provide more effective and richer communication approaches.

Figure 8. API Communication Channels



The API portal serves as a hub for your API developer community. While it plays an obvious role in providing message boards, wikis and chat channels, it also acts as a signpost to more effective and valuable types of communication, such as meetups and hackathons.

A developer-friendly API program should use these and other techniques to enable an open developer discussion that advances usage and business objectives. API providers hoping to attract a diverse audience of API consumers — as is the case with most public API programs — will find collaboration and community to be a critical capability for growing knowledge and interest. Though still beneficial, API providers focused purely on internal project audiences can justifiably limit investment in this area and instead tap into existing collaboration practices and tools that are available within the organization:

- **API provider blog:** The most often overlooked aspect of the collaboration capability is the provider blog. The blog is an excellent vehicle for sharing updates and future plans with your API consumer community, for providing expert opinion and advice that leads to adoption growth via inbound marketing, and, eventually, for interacting directly with an active and engaged audience to plan the future of the API product.

- **Developer discussion forums:** Discussion boards provide a continuous and always-available conversation between interested parties. Usually organized by topic, robust discussion forums

provide a searchable means for getting up-to-speed on the conversation. They are particularly useful for developers who are coming into an established API community and are unsure which questions are the right ones to ask. Experienced developers will often spend significant time reviewing existing discussion threads before seeking help, which contributes to the self-service features of the API. For public APIs, you may choose to leverage an external platform or framework, such as Stack Overflow or Disqus, to manage developer discussion and encourage community contribution. However, your ability to moderate and administer those discussions will be constrained by the tool you choose, and it may not be possible to track discussions or issues to specific API developer accounts and keys for diagnostic processes.

- **Developer issue reporting:** Enterprise application development organizations already have change management capabilities with built-in issue management. These features are only available for internal use, however. As the audience for your API grows beyond purely internal use to include external business partners and even the general public, issue reporting becomes increasingly important. External parties using your API may encounter problems or have feature requests. Without an issue-reporting system that offers the potential for ticketing integration with your existing change management platforms, you'll only find out about these problems and potential opportunities by trolling discussion forums (or worse, through angry replies on your blog).

- **Change notification registration:** When a provider changes the API in a significant way, existing applications that use the API can be disrupted. Sometimes this disruption may be impossible to avoid. It should, however, always be possible to inform API consumers that a potential disruption is coming. Although your API provider blog should be kept up-to-date with this kind of information, it is folly to assume that every interested developer reads it regularly. Therefore, developers should be offered the ability to register for change notifications in the medium of their choice — be that email, SMS or one of the popular social networks (such as Twitter) frequented by developers.

Developer Enablement Administration

The developer enablement capabilities discussed in the previous section are important to those who intend to consume an API. The provider also needs ways to manage some of these capabilities:

- **Developer discussion forum management:** For a public API, the discussion forum is the key point of interaction between the API consumer community and the API provider. Although a small number of highly engaged API consumers may establish robust direct communications with the provider via other channels (e.g., email or phone), the vast majority of API consumers will see only the blog and discussion forum as a conduit for communicating with the API provider. It's critical, therefore, that the API discussion forum properly represents both the consumers using it and the perspectives and plans of the provider organization. The API management team should have the ability to remove posts (and users) when required, as well as the ability to delay posts via moderation when appropriate. If you provide a discussion forum, you must appoint an owner, together with a team of administrators and moderators, who will manage and respond to posts. If

you cannot find a team to take on this responsibility, you should not evaluate (or offer) discussion forum features, as doing so may damage your brand.

- **Content management for the developer portal:** There is a wide variety of content relevant to the API developer community. For example:
  - Information about test harnesses
  - Updates to the API provider blog
  - New sample code and SDK releases
  - Upcoming hackathons and community or ecosystem relevant events

  Updating this content shouldn't require manual intervention by a very technical person to achieve. It should be entirely possible for API managers to perform the updates through content management tools embedded in the developer portal. The integrated content management capabilities of an API solution function much like other content management system (CMS) tools, but there should be site templates and custom widgets (e.g., to expose API metrics and metadata) to help accelerate the creation and maintenance of your developer portal.

- **API documentation management:** API documentation should be kept up-to-date with the evolution of API functionality and design. Tools to do this should be embedded into developer enablement administration so that the API management team can easily make changes (or automate them) when needed.

Enabling API Creation

Another important priority for API management capability evaluations is to consider features that facilitate the process of API creation. These include features that help developers do a quicker and more effective job of discovering existing APIs, reusing schemas and data models, and applying policies and standards:

- **Discovering existing APIs:** When multiple teams of developers across an organization are writing services that expose APIs, they need to first discover the APIs that already exist in the organization, which is no easy task. Any API management capabilities that can effectively help them do this will pay dividends.

  For example, it will help them to avoid unnecessarily duplicating the functionality of APIs that already exist. If one team has already created an API that performs address lookup, for example, you may not need a different team to create another address lookup API.

In addition, API developers can leverage existing APIs to serve as one portion of a multi-API service they're creating. If they want to compose a new check-out process out of multiple lower-level APIs, for example, they might want to find out what already exists in terms of an order management API and a payment API.

- **Reusing schemas and data models:** Features that enable discovery and reuse can also be useful for data schemas and data models. APIs accept and return data, and the format of that data won't always be consistent across multiple APIs. For example, if an application calls two APIs that return customer info, that information will probably need to look the same, with the same field names and formats. This goal can be facilitated by enabling the people creating APIs to discover existing schemas and models. In a customer database somewhere, somebody may have already created a perfectly good address data format or a date format. Exposing such schemas and models for reuse can enable them to be leveraged across different APIs.

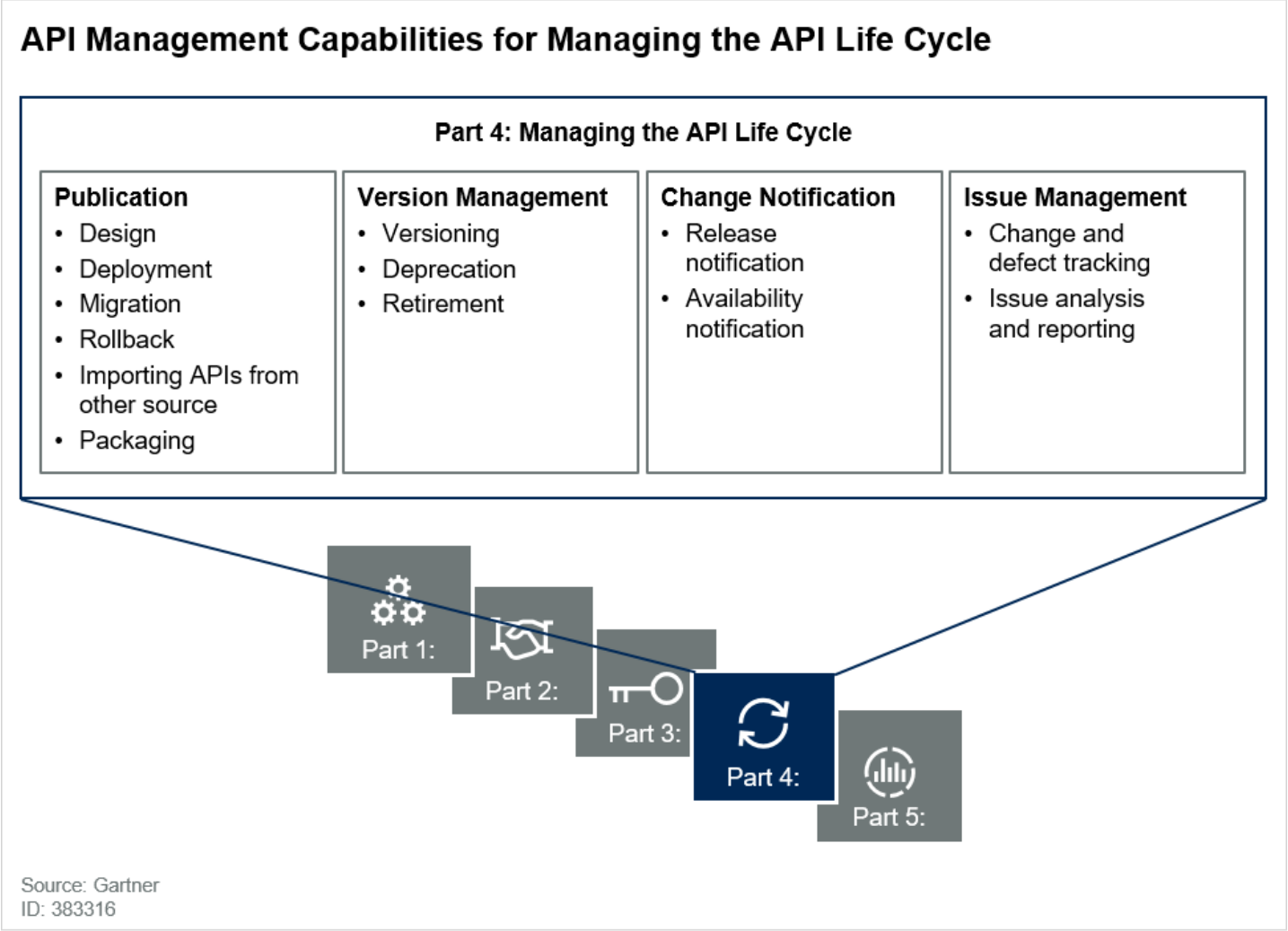
- **Applying policies and standards:** API management features can be used to ensure that developers have access to the various policies and guidelines they need to create APIs, such as standards for error handling, versioning and resource naming. The solution can act as a "one-stop shop" for people creating APIs so they can find all the information they need. For more details on the policies you should define when creating APIs, see "A Guidance Framework for Creating Usable REST API Specifications."

Part 4: Evaluate Features for Managing the API Life Cycle

Application life cycle management (ALM) is a set of practices that govern and support application architecture, design and programming. ALM provides tangible benefits in the transparent coordination of development activities, including visible reminders of project status and health. Although an API doesn't require a full-blown set of ALM governance activities, the fact that it is consumed by a wide audience in potentially unforeseen ways increases the need for certain aspects of ALM. The "manage the API life cycle" discipline controls how APIs are released to consumers, and it provides the feedback mechanism for consumers to report problems or request new API features.

Figure 9 describes the key capabilities for life cycle management.

Figure 9. API Management Capabilities for Managing the API Life Cycle



Publication

Before an API can be consumed, it must be published to an environment where the desired audience can find it:

- **Design:** As discussed in "A Guidance Framework for Designing a Great API," API creators should use a consumer-driven API design process. The conceptual design is often established using API definition tools, as discussed in "A Guidance Framework for Creating Usable REST API Specifications." A number of API management solutions embed these API definition tools.
- **Importing APIs:** Another important feature is the capability to import externally created API definitions. This includes taking SaaS or other services that expose APIs and rehosting them in your API gateway to provide more control over who is consuming existing services. If you create API definitions in API design tools such as SmartBear SwaggerHub, you need the ability to import these API definitions into your API gateway and management portal, rather than having to re-create them.

  A significant benefit of this capability is that it avoids forcing API designers to use multiple documentation portals for different APIs, simply because you're using multiple API management

technologies. By importing data and API definitions from other platforms into a single platform that has all your documentation, these resources are available centrally. This is particularly beneficial in multicloud and hybrid cloud environments, where organizations are using platforms from multiple cloud providers such as Amazon or Microsoft together with traditional data centers.

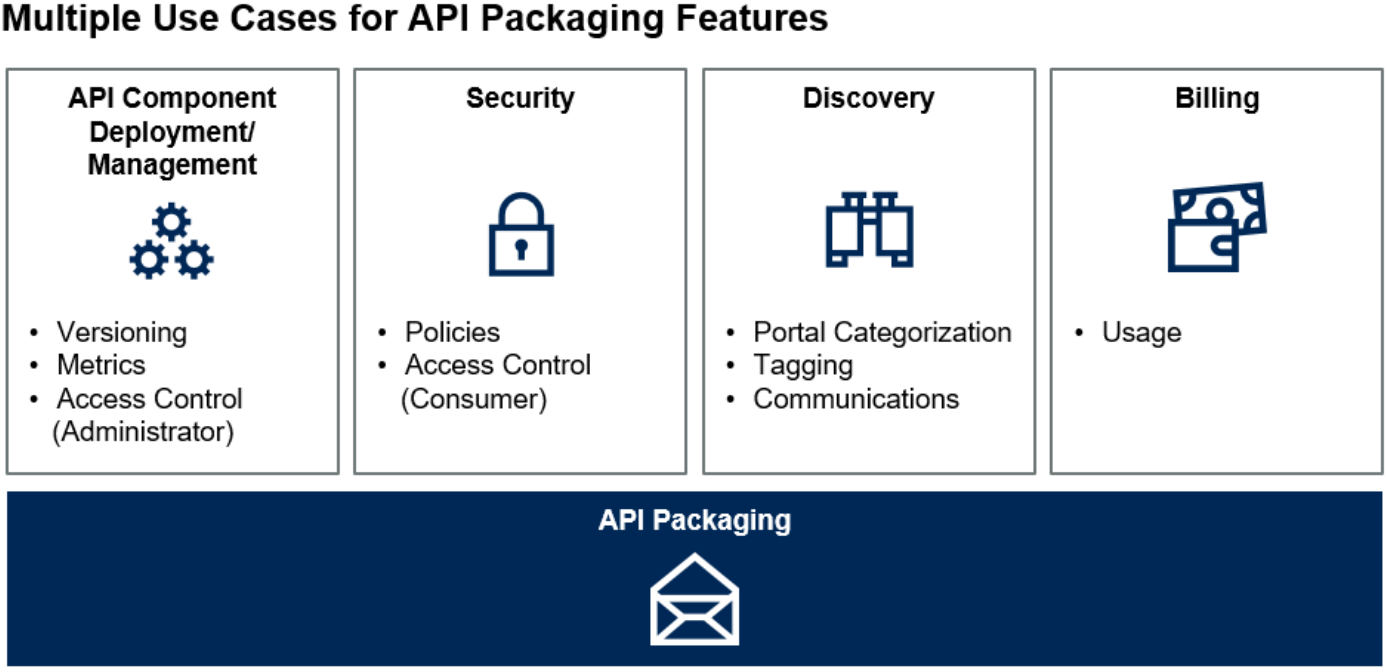
- **Deployment:** Once the conceptual design is complete, API designers can use a wide variety of approaches to implement the API. This might include mapping the API to an existing service implementation, or creating a new service from scratch or based on existing data or application resources (see "Decision Point for API and Service Implementation Architecture" for more details). API management tools provide metadata-driven capabilities for surfacing these new or existing services and data using a RESTful paradigm. Once the API is designed, developers can deploy the metadata-driven API into an environment where it can be consumed, often by promoting the API definition through a number of life cycle stages or phases.
- **Migration:** Besides deployment into a production state, your API might require various preproduction environments for development and testing. Therefore, it is important that an API management capability include the ability to smoothly migrate API versions from one staging area to another.
- **Rollback:** When an incorrect or buggy version is deployed or migrated, it must be removed. Without a rollback capability, reverting to a prior API version will be an error-prone, heavily manual affair. API managers should have the ability to roll back to earlier versions from the management console. Note that you should version your APIs independently of the implementing service, and this feature refers to rolling back versions of API definitions and endpoints.
- **Packaging:** How will your APIs be organized? You may choose to bundle certain types of features (e.g., transactional access supporting read/write functionality) into a single package for deployment as part of a new version or as distinct tiers of service. Your API management capability should include a way to isolate or bundle different components into discrete units that can be published, consumed and managed separately.

API packaging can be used for multiple purposes (see Figure 10). For example, it can be used to group and expose related APIs together, in order to:

- Make certain API groups discoverable together, so clients seeking APIs related to certain sets of features or lines of business can find them.
- Apply security policies to specific groups of APIs, which could be packaged for that purpose.
- Group APIs for billing reasons, for a product offered as a group of APIs.

There are a number of features within an API management solution, such as bundling and tagging features, that can enable these and other packaging use cases.

Figure 10. Multiple Use Cases for API Packaging Features



Version Management

Your API will evolve over time. As it does, you will be faced with decisions around versioning:

- What aspect of your API needs to be versioned?
- Should you run one version or offer several?

Whatever you choose, you will need your API management capabilities to provide for versioning, packaging, deprecation and retirement. Although versioning an API should always be a last resort, you should plan for when the time comes to do so (see "Choosing the Right Web API Versioning Model" for more detailed analysis of web API versioning):

- **Versioning:** How many versions will you run? Even if you think you will only run a single version, you'll most likely still extend that version repeatedly over time. Your API management capability should include a mechanism for viewing current and prior versions, determining when a change was made and making the differences between versions visible through documentation. Versioning can also benefit from service-mapping capabilities to help assess the impact of a versioning event.

- **Deprecation:** How long will an older API be supported? When you introduce a new version or a change to your single version that obviates an existing feature, you need a way to tell the consumer that the old version is near end of life. Your API management capability should include a way to clearly deprecate and flag out-of-date functionality while providing for continued functionality until the feature is truly taken out to pasture.

- **Retirement:** How will you remove retired API versions and functionality? Once a feature has been deprecated, you will want to remove it so that you no longer incur associated maintenance costs and activities. However, you shouldn't hide the fact that it ever existed, and you may want to provide clear guidance on how developers suffering broken applications at the hands of retired functionality can fix their programs. Your API management capability should provide a vehicle for managing the retirement of your API, including communication to consumers still using the API, and monitoring usage of the version to assess the impact (if any) when the retirement is finalized.

Change Notification

Changes to an API may adversely affect its consumers. Developers subscribed to an API may not work inside your organization, and even if they do, they may not be aware of every change you implement. Change notifications provide a self-service mechanism that keeps consumers apprised of API evolution and service outages. They should be provided through the channel most appropriate to the API consumer — be that email, social media, SMS or a webhook integration:

- **Release notification:** Release notifications tell consumers that a new version of the API has been deployed. These new versions may be major versions (breaking versions that will require client changes to use) or minor versions that do not require client changes, but make additional functionality available if the consumer wants it. Consumers subscribe to release notifications from the developer portal. API managers create release notifications through the management console.

- **Availability notification:** Availability notifications inform API consumers that one or more features of the system are not available due to a planned or unplanned outage. They also inform API consumers when the availability disruption has passed. Consumers subscribe to availability notifications from the developer portal. API managers create availability notifications from the management console (in the case of planned outages), or they may be generated automatically out of the monitoring aspects of the API management platform.

Issue Management

Issue management systems allow testers, developers, users, operations staff and others to document and manage changes and defects from the time those issues initiate to their moment of resolution. These features may be part of your API management platform, but will more likely be provided in the form of integration with external issue tracking services or software. The system may even be tied into your continuous delivery practice:

- **Change and defect tracking:** An externally available issue management capability lets API consumers working outside your organization report problems or shortcomings in your API. Your API management platform should provide such a capability. It should also allow you to automate integration of its discrete issue management and defect-reporting capabilities with your existing systems.

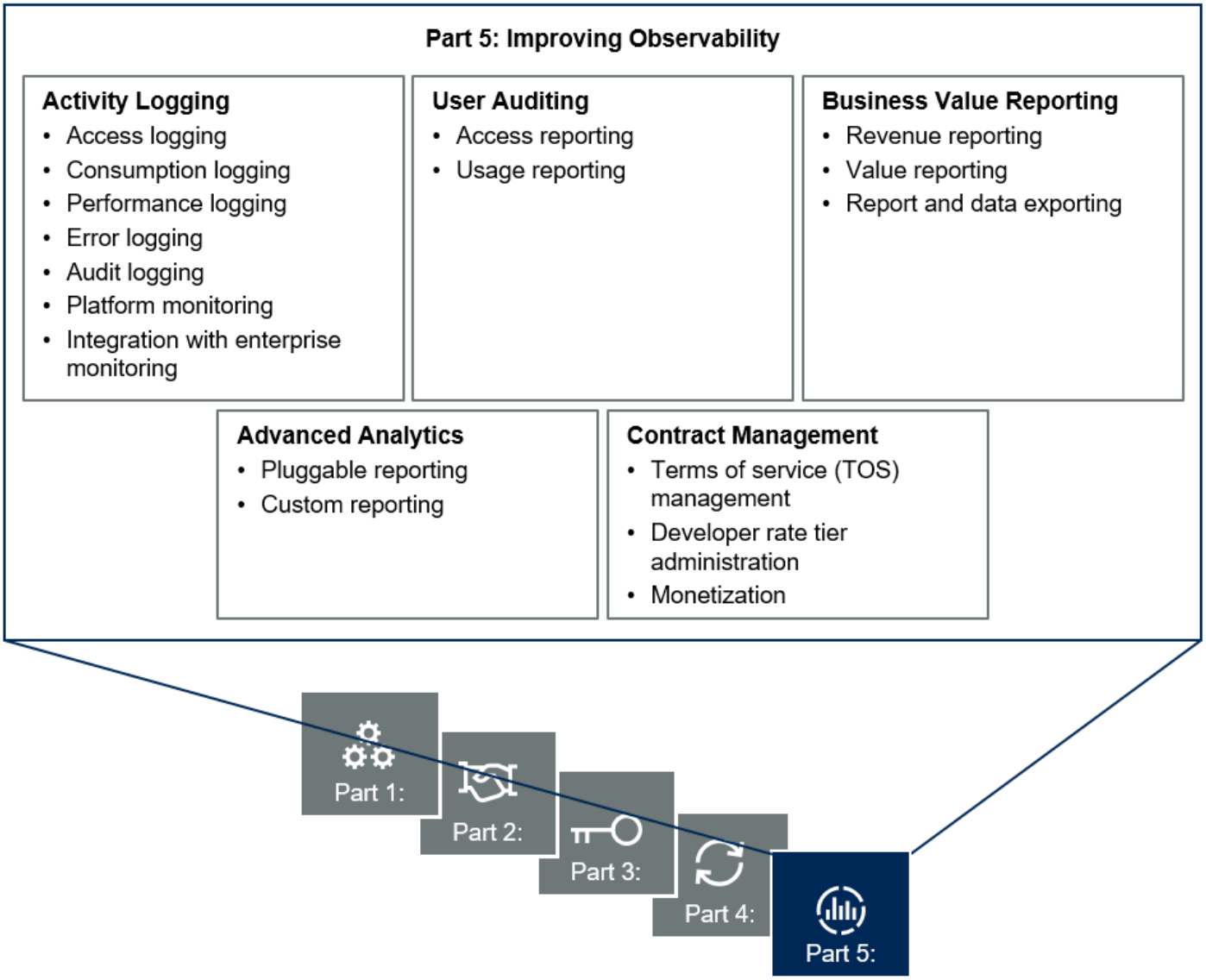
- **Issue analysis and reporting:** Issue analysis and reporting capabilities allow the enterprise to answer questions like, “How many API consumers are having this problem?” and “How many external change requests name a particular feature as desirable?” Through both canned and custom ad hoc reporting, API managers can make more effective data-driven decisions about fixes or improvements most likely to resonate with API consumers.

Part 5: Evaluate Features for Observability and Monitoring

Providers offer APIs for use by service consumers, but these APIs exist to further the business objectives and mission of the provider. In the case of a private enterprise, objectives are profit-oriented or market-growing in nature. In the case of a public institution, objectives are more likely to revolve around measurable benefits in serving citizens or communities. In either case, the enterprise requires tools to understand how the API is used, who is using it, what kind of value is being created and how that value can be enhanced, generally by changing the API or enriching it. The capability to measure and improve business value provides monitoring, reporting, analytics and control over API monetization (if desired), as summarized in Figure 11.

Figure 11. API Management Capabilities for Improving Observability

API Management Capabilities for Improving Observability



Source: Gartner
ID: 383316

Activity Logging

Activity logging includes basic tracking of access, consumption, performance and exceptions. This is critical data for an API management platform to track because of the potentially diverse use scenarios and developer constituencies an API may support. Access logging is particularly important if you expect an external audience to consume the API, but it can be very useful even for internally oriented APIs because of the wealth of debugging and performance data available. Logged data may be accessible directly through the API administration or developer portal, or exportable for analysis using third-party tools:

- **Access logging:** Access logging provides records of each consumer API invocation attempt, and may include the request and response content if required. At a minimum, identity, client, Internet

Protocol (IP) and request URL information is saved for each API request. The data is stored for later reporting and analysis.

- **Consumption logging:** Consumption logging tracks the popularity of specific API functionality, the usage behaviors of individual API subscribers and aggregate information about API utilization. This data is stored for downstream reporting and analytics.
- **Performance logging:** Performance logging tracks the performance characteristics of individual and aggregate API invocations. This data is stored for reporting, analytics, service-level monitoring, quality-of-service management and performance alerting.
- **Error logging:** Error-logging record exceptions are sometimes encountered during API invocation. These exceptions may be intentionally thrown by the API manager (such as when an API consumer makes an invalid request), or they may result from a runtime problem with the API itself (such as an orchestrated service becoming unavailable). Error logging is important because it can expose a problem with the API definition as well as a potential attack vector by an unscrupulous API consumer. It should capture detailed snapshots of erroneous transactions for triage and diagnosis.
- **Audit logging:** If API management logs are to be used for audit purposes, you will need to evaluate the ability of the platform to implement an appropriate log correlation mechanism to ensure that transactions can be audited from beginning to end. You will also need to evaluate security features to ensure the nonrepudiation of log entries and log sanitization to filter sensitive data from events before logging.
- **Platform monitoring:** Features in this category are used to monitor factors such as whether the API gateway instances are healthy, are using too much CPU or memory, or are low on disk space.
- **Integration with enterprise monitoring:** This criterion considers whether the solution can raise an alert up to your enterprise monitoring platform if, for example, an API gateway becomes unhealthy, the API platform goes down or specific services running on it become unavailable.

User Auditing

User auditing features allow inspection of previously recorded activity, consumption and error logging to identify opportunities and challenges in the current API design and its use. For example, an API administrator might audit the user base for frequency or depth of use to identify defunct accounts that can be culled from the system. Or an API administrator might audit usage to identify behaviors that place an undue burden on the API infrastructure:

- **Access reporting:** Access reporting allows the API administrator to review who accesses the API, how they access it and when they access it. This reporting allows for auditing of user access history.

- **Usage reporting:** Usage reporting allows the API administrator to review how often a consumer accesses the API, how much the consumer uses the API and how broad the consumer's use is.

Business Value Reporting

Useful, basic metrics about access and usage are insufficient for API product managers to understand the value the API is delivering to the organization. Business value reporting incorporates additional metrics and dashboards that enable API product managers to make decisions about feature roadmaps, pricing and packaging, and other considerations that drive the evolution of an enterprise API strategy:

- **Revenue reporting:** Revenue reporting provides a direct measure of API monetization. Revenue information is critical for APIs that are consumed on a paid basis, but many APIs will never be directly monetized, so additional value-based reporting capabilities should also be considered.

- **Value reporting:** Value reporting measures engagement as a proxy for indirect monetization. Engagement can be measured in a variety of ways within an API management platform, including the number of unique users, the number of developer registrations, the number of developers actively using the API and the number of applications of a particular type (e.g., mobile or web-based) built atop the API. API product managers should be able to define the correlations between business value and metrics observable within the API platform. This is how they can begin to identify the relative value of one subscriber over another without relying on revenue metrics (which are unavailable unless the API is pay-to-use). Demonstrating and driving the business value delivered by an API is critical to its long-term success and sustainability.

- **Report and data exporting:** Even if a wide variety of business value metrics is available in the API management platform, API product managers may wish to use external data visualization tools to better understand the data they're working with. API management platforms offering advanced business value reporting capabilities should therefore offer full programmatic access to the underlying data.

Contract Management

Contract management is an obvious requirement for APIs that you will monetize directly, but it is valuable even for free-to-use APIs, especially if you are providing different service levels to different API consumer constituencies. For example, in a public API, subscribers who self-provision a developer credential via the developer portal might be given free access of up to several thousand API invocations per month, while business partners might be given unlimited use. Even internal APIs have SLAs that you will need to publish, monitor and administer. Contract management enables the API product managers to tune utilization, performance, packaging and cost for different portions of the audience:

- **Terms of service (TOS) management:** An API should have clearly defined TOS. These terms may need to be updated from time to time for one reason or another. Not only should the API management platform allow the publication of TOS, but it should also support versioning and change tracking so that existing and prospective subscribers alike can see the track record of support from the API provider. This is often supported via an embedded CMS.

- **Developer rate tier administration:** Different developers may need access to different levels of API consumption (utilization), different levels of responsiveness (performance), different kinds of API calls (packaging) and different levels of price (cost). API product managers can configure these as rate plans or tiers of service, enabling developers to choose from among them. If internal control is desired, managers can associate a developer with a particular role that has access to a particular rate plan. Furthermore, how the provider chooses to charge and bill for API consumption is likely to change over time with the popularity of the API and the business objectives of the organization. Central rate plan administration is necessary so that API product managers can implement business decisions on pricing and packaging in one place.

- **Monetization:** If your goal is to monetize your APIs by charging people to use them (or, conversely, to pay people for transactions performed through your APIs), you need capabilities that enable monetization. Most importantly, you require the ability to gather information about who used which APIs, and how much they used them. In terms of billing-related information, two capabilities are key:

  - *Generating information needed for billing:* In this case, you'll want to consider functionality required to collect all the API usage information you'll need to accurately bill customers. For example, if your free tier of API usage is capped at 1,000 calls per day, you'll need to be able to monitor when a customer hits 1,001 calls — and then take appropriate steps, such as limiting them and asking if they want to upgrade their service. Complications in this case may include:

    - If the 1,000-call limit spans multiple APIs, how will that be detected and calculated?

    - Or, if two or more groups of APIs are billed separately, how can the usage information be split across the products you're billing for?

  - *Generating the information needed to inform customers where they stand in terms of usage that impacts billing.* You may need the capability to expose information to API consumers so they can see where their usage levels stand. This usually involves providing them with a portal they can log into and see how many API calls they've made in a given time period that will count toward a specific billing tier.

  - *Exposing customer billing information via an API.* Providing an API that exposes customer billing data allows you to provide that data to billing systems (to generate invoices for customers). It

can also be used to allow customers access to their own invoice data programmatically, rather than just through a webpage.

Advanced Analytics

Beyond the basic reporting functionality described so far, API product managers and API administrators may require additional reporting functionality and extensibility:

- **Pluggable reporting:** An advanced API management platform may incorporate a pluggable reporting system to enable the addition of proprietary reporting widgets into the API administration portal. Pluggable reporting could also include extensibility points that allow the integration of a third-party reporting system into the API management platform.
- **Custom reporting:** Custom reporting supports the creation, versioning and execution of proprietary scheduled and ad hoc reports authored by an API administrator or API product manager. These custom reports are authored within the API management platform itself and do not require additional external tools.
- **Data preparation and ingestion:** To support business value reporting and analytics, you may need to include data from other sources to augment the API-specific metrics gathered by the API management solution. This capability ensures that you can integrate external data sources to ingest external data and prepare it for use by the reporting and analytics capabilities of the solution.

Risks and Pitfalls

API management capabilities are often viewed as a set of technology that is relatively easy to select and deploy. Once their capabilities are understood, the gateway-centric architecture they use can make them very painless to adopt and, if used carefully, ensure that they are one of the “least sticky” application platform technologies in your portfolio. However, there are still risks and pitfalls to be aware of as you apply this guidance framework to your own selection activities.

Overbuying Your API Management Solution

While API management capabilities provide a critical set of governance, security, monitoring and API delivery capabilities, it is still important that you source these capabilities within your means to procure, configure and operate the API management platform itself. “Overbuying” refers to becoming enamored by advanced features that add marginal benefit to your API deployment scenarios.

How to mitigate: Ensure that your process for evaluating API management solutions clearly separates the required features from the preferred and optional ones.

Treating API Management as an Integration Platform

API management features are very closely related to integration activities. The act of consuming an API is an integration, and the act of exposing an API often involves integration with a back-end resource of some kind (see "CIO Call to Action: Shake Up Your Integration Strategy to Enable Digital Transformation"). However, API management is about management and governance of those APIs, and not all of the services that you publish as APIs will require integration features (see "Decision Point for API and Service Implementation Architecture"). While you might source your API management capabilities from the same vendor as your integration platform, it is important not to conflate the responsibilities of the two technologies. API management solutions used to implement complex integration scenarios are the most common source of scalability and management problems.

How to mitigate: Separate the appropriate usage scenarios for different platforms. For example, use an integration platform for creating integration-centric services and an API management platform to govern access to services.

Failing to Account for Security and Identity Integration Complexities

Security capabilities are frequently cited as a driver for deploying API management capabilities. However, the complexities of integrating the solution within your existing (or planned) security and identity infrastructure can surface very late in the evaluation process unless you engage your information security and/or identity and access management (IAM) teams at an early stage. Examples include constraints on deployment topology and connectivity to connect to enterprise identity stores, and the need to integrate multiple identity systems to authenticate and authorize different users in the API ecosystem.

How to mitigate: Engage with your organization's security architects and security team to gain a better understanding of the security and IAM environment and tools you'll need to integrate with.

Buying a Solution That You Can't Expand to Adapt to Future Needs

While overbuying is a risk, it is also very easy to be too tactical about your selection of API management solutions and focus on very near-term objectives and deployment scenarios. A common example of this is selecting a cloud-hosted API management solution to address a mobile integration requirement, only to discover that the next major API opportunity involves an internally hosted and consumed API or a set of sensitive data that your compliance team precludes (still) from routing through a cloud provider. Another example is being caught out by unexpected success. APIs, by their nature, encourage innovation, and you should have a plan for when one or more of those ideas is a "killer app."

How to mitigate: To address these scenarios, think ahead, assess the likelihood that you will need to adopt alternative deployment patterns or add new capacity or operational capabilities, and favor modularity and flexible deployment architectures accordingly.

Failing to Prioritize a Great API "Customer Experience"

The developers who will consume your API are your most precious customers, and they deserve a great customer experience. When selecting API management capabilities, do not underestimate the importance of the developer portal. This involves more than just portal features that provide a nice user experience. It is also important to focus on related self-service, collaboration and support capabilities.

How to mitigate: Ensure that your API solution supports effective and easy-to-follow developer workflows for registration, evaluation of the API, support and problem resolution.

Failing to Clearly Define When to Use Different Integration Technologies

After buying a new technology that can perform integration — such as an API management platform — many organizations fall into the trap of trying to use it for too many things, including uses for which it is unsuitable. As a result, it doesn't perform well in those misapplied use cases and gets a bad reputation among users. The key to successfully managing your hybrid integration platform (i.e., your portfolio of integration technologies) is to provide clear advice on when to use one integration technology versus another.

It's also important to articulate the right users of multiple integration or API management technologies. For example, one API management solution may be best for external, customer-facing APIs, and another for internal APIs. The users of each type have different feature needs, expectations and communication channels. Internal users may be a small community of developers who are well known to you, while external ones may be a large community of unknown users of an API you've provided publicly on the internet.

How to mitigate: Provide clear guidance on when to use differing integration and API management technologies, targeted to different user communities.

Related Guidance

This guidance applies to selecting tools for managing the creation and operation of APIs through their life cycles. To deliver successful API programs, these governance activities need to be supported by strong API design and development practices and tooling, which are covered in the following guidance frameworks:

- "A Guidance Framework for Designing a Great API"
- "A Guidance Framework for Creating Usable REST API Specifications"

Evidence

[1] "Payment Services (PSD)," European Commission.

[2] "Data Sharing and Open Data for Banks," Open Data Institute.

[3] "Jobs to Be Done," Clayton Christensen Institute.

[4] "Apigee Adapter for Istio Concepts," Google (Apigee).

Document Revision History

A Guidance Framework for Evaluating API Management Solutions - 22 August 2017

Guidance Framework for Evaluating API Management Solutions - 16 November 2015

Run and Evolve a Great Web API With API Management Capabilities - 8 October 2013

Recommended by the Authors

Assessing Service Mesh for Use in Microservices Architectures

Decision Point for API and Service Implementation Architecture

Selecting the Right API Gateway to Protect Your APIs and Microservices

Solution Comparison for OAuth-Based Access Management Capabilities of Nine Vendors

Modern Identity and APIs: Mobile, OpenID Connect, OAuth, JSON and REST

Decision Point for User Authentication

Recommended For You

Comparing Architectures for Hybrid and Multicloud API Management

Assessing GraphQL for Modern API Delivery

How to Successfully Implement API Management

A Guidance Framework for Evaluating Application Integration Platforms

Guidance Framework for Modernizing Microsoft .NET Applications

research is produced independently by its research organization without input or influence from any third party. For further information, see "Guiding Principles on Independence and Objectivity."