API Best Practices

Managing the API Lifecycle:

Design, Delivery, and Everything In Between

#DigitalKnowHow

by apigee

Table of Contents

Executive Summary

Digital is disrupting every industry. From drug store chains to banks to telcos, businesses are becoming software companies and adopting modern software practices. Why? If they don't adapt to a new market reality they will fail.

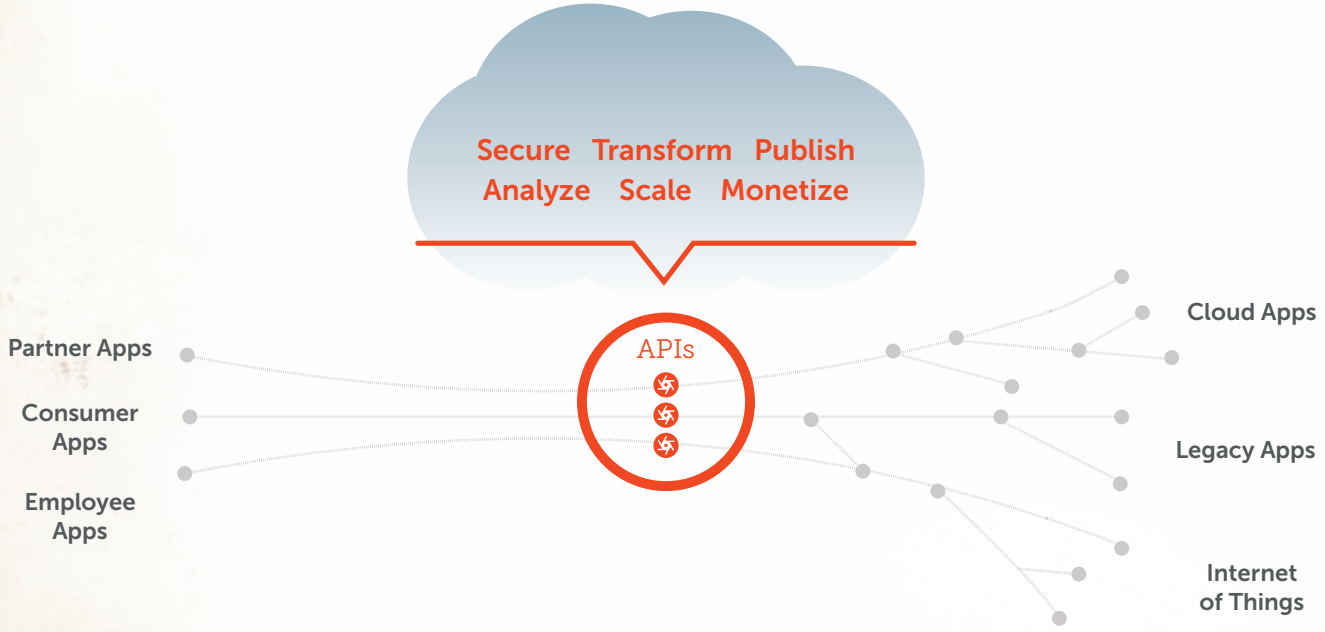
As the business context is changing so is the technology stack. Enterprise application architectures are evolving from integration-centric enterprise service bus (ESB) architectures to application-centric, microservices, platform-as-a-service (PaaS), multi-cloud, and API-driven architectures.

APIs are the lynchpin to the success of these digital businesses.

All applications use APIs to access application services and data through APIs. These services can be microservices or cloud workloads or legacy SOAP services or IoT.

To ensure that applications and developers can effectively use these services to build partner, consumer, and internal apps, companies need to deliver secure, scalable, easy-to-use modern APIs.

Over the last few years, we've participated in hundreds of enterprises' API-led digital transformation initiatives. This guide distills our learnings from these customer engagements and shares best practices about managing APIs across the lifecycle.

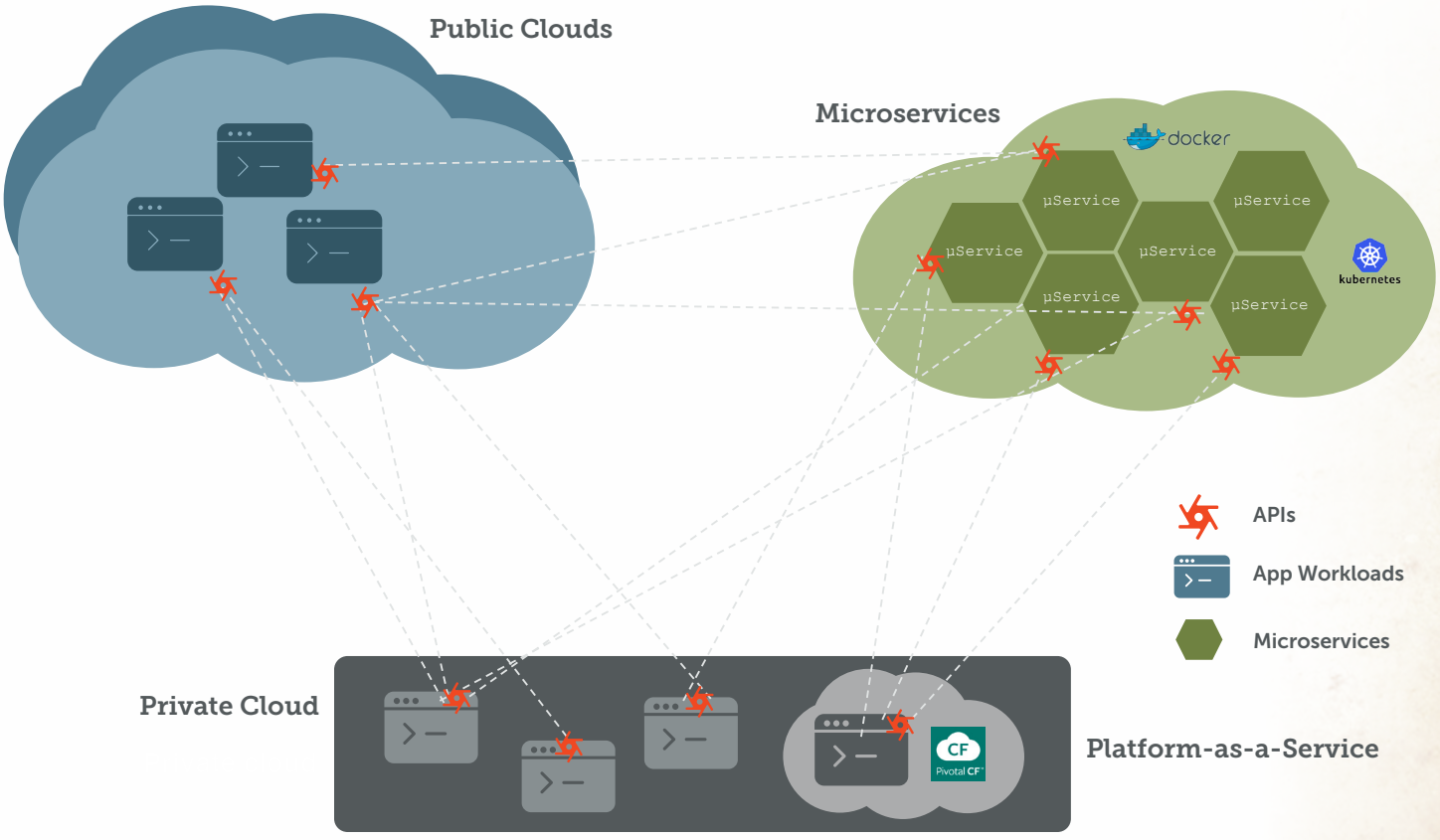


APIs in a Modern Software Development Context

Gartner found that 77% of app development supporting digital business will occur in-house. Seventy percent of organizations claim to be either using or investigating microservices, and nearly one-third currently use them in production, according to a report from NGINX. Multi-cloud strategies will jump from just 10% in 2015 to more than 70% in three years, according to Gartner.

Why? Because microservices, cloud, and platform-as-a-service (PaaS) technologies enable organizations to innovate fast: development teams can independently develop, deploy, and scale applications. The adoption of the cloud, containers, and continuous integration/continuous deployment (CI/CD) tools has made new apps implementation easier, leading to more modern software being built as microservices in the cloud.

Moreover, because different workloads have different needs that may be best delivered by a particular cloud vendor (or with a private cloud deployment), organizations are adopting multi-cloud strategies. App development teams implement microservices using a variety of stacks like Kubernetes, Netflix OSS, and Mesos, depending on their needs. All these microservices and cloud workloads use web APIs as the mechanism to communicate with one another.



There's a challenging side effect, however. As app development teams rush to implement microservices across multiple clouds or in a PaaS, they inadvertently create silos with inconsistencies in security, visibility, documentation, and governance. For example, many APIs are not secured, or are secured inconsistently, across organizations.

They might not be accompanied by standardized documentation, or access control mechanisms. These microservices and cloud APIs are often difficult to reuse, analyze, or even to discover for use by other teams.

How to:
Eliminate silos created with inconsistencies in security, visibility, and discoverability

Enforce consistent security and governance

In the world of microservices and public clouds, there should be no distinction between internal and external APIs.

Developers are building APIs and microservices without the kind of centralized oversight that once existed. They are deploying them more widely than ever before. They implement inconsistent and varying levels of security—or no security at all.

When developers deploy microservices in the public cloud and neglect to deploy common API security standards or consistent global policies, they expose the enterprise to potential security breaches. A microservice could be used by another app in the cloud today and by an external partner tomorrow.

Companies must assume a zero-trust environment. An API platform enables enterprises to implement security and governance policies like OAuth2 across all of their microservices APIs.

Drive end-to-end visibility

Without usage and performance data about your microservices and cloud workloads, it's difficult to understand the extent to which these services are reused, or potential performance bottlenecks. The problem is more acute when these APIs are used by multiple teams in a large enterprise or as external APIs for partners and customers.

API platforms provide fine-grained analytics and reporting capabilities to measure API usage, developer and partner engagement, microservices adoption, and traffic composition. Organizations use API analytics to monitor API performance by analyzing total traffic, throughput, latency, and errors.

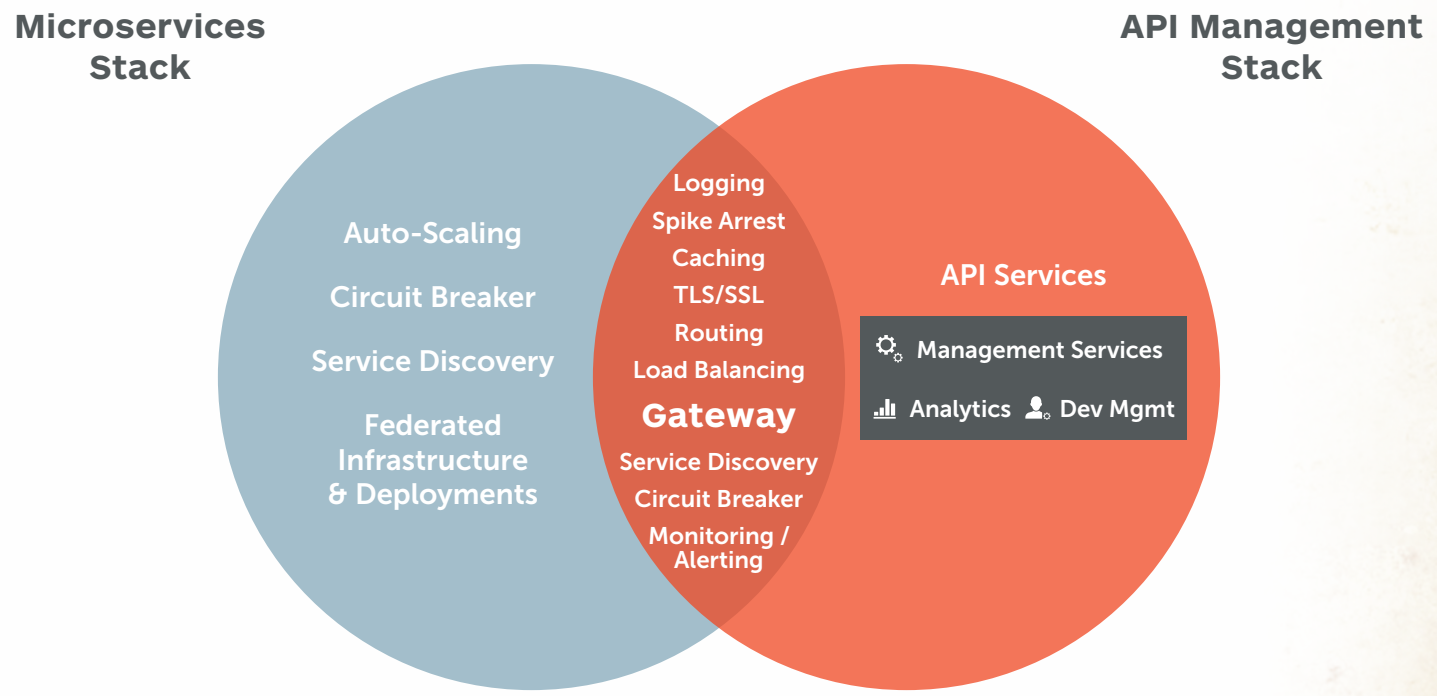
Make services easily discoverable and reusable

As enterprises employ multiple and disparate microservices stacks, clouds, and PaaS environments, it can be difficult for teams to discover and reuse microservices and APIs.

Internal and external app developers use developer portals to discover services, register to use APIs, access interactive and community documentation, register their apps, and view data about their app usage on a dashboard.

Co-exist with microservices orchestration frameworks

App teams are adopting a variety of microservices orchestration tools including Kubernetes, Docker, and Mesos. Although there is some overlap in the gateway functionality between the microservices stack and API management, most organizations colocate their API management with their microservices and PaaS gateway/routers to ensure consistent visibility and discoverability across the organization.



Don’t adopt cloud-specific gateways

All public cloud providers offer their own captive API gateways (AWS API gateway, for example) that provide basic functionality like traffic management, simple meditation, and caching. For the needs of most enterprises, the functionality of these cloud-specific gateways is too limited; they don’t solve the problem of silos across the various clouds that many organizations employ.

Companies like Autodesk have adopted an enterprise API platform to support its application workloads, including serverless apps being built on AWS. Instead of using the AWS gateway, they realized the value of having a single pane of glass for all APIs across their organizations, regardless of which cloud these APIs might operate in.

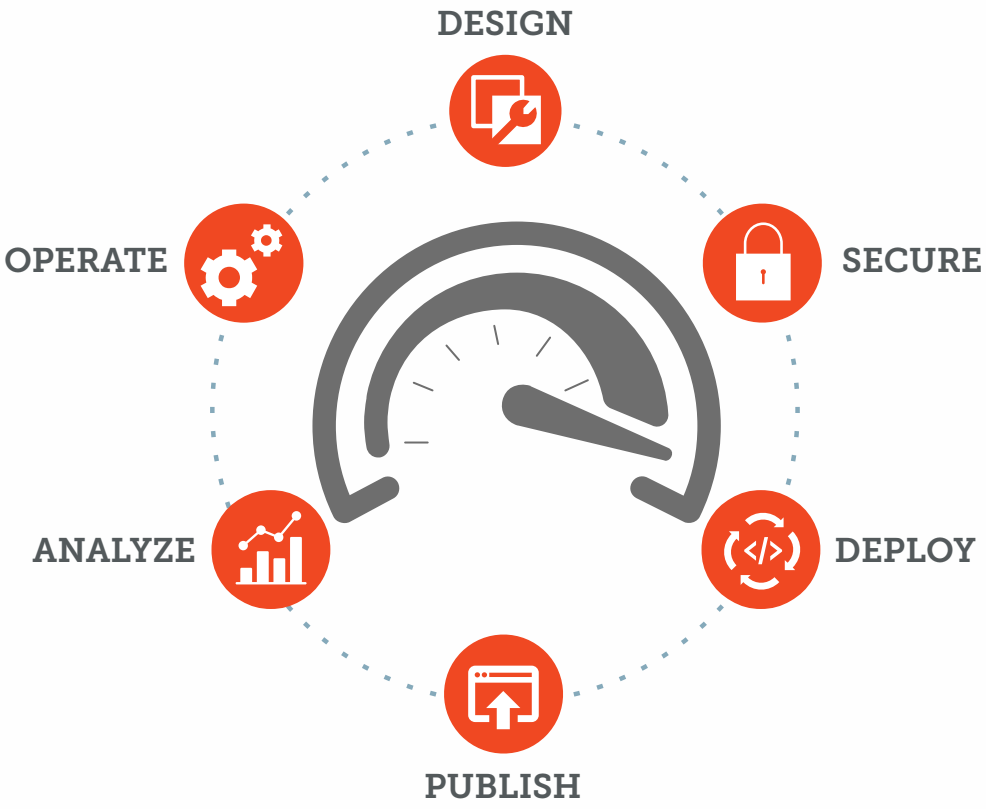
“A central unified API platform across the company has been a game changer for us.”

- Alan Williams, Autodesk

Managing the API Lifecycle

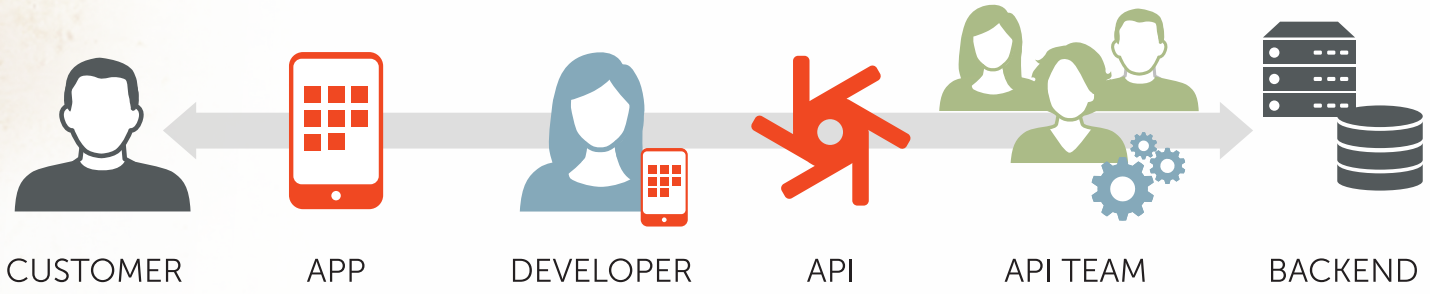
As organizations adopt modern software practices, holistic API management has become critical. Enterprises put in place people, processes, and technologies to manage APIs across the entire API lifecycle—from design through to analysis and operation. By adopting best practices in each stage of the API lifecycle, teams become more agile and can deliver on the promise of digital transformation.

Different organizations have slightly different API lifecycles, but for simplicity's sake we're using the API lifecycle below to share best practices.



A well-crafted API should make an app developer as successful as possible. When building APIs, it's critical to think about design choices from the app developer's perspective.

Why? Look at the value chain below. The app developer is the linchpin of the entire API strategy. The primary design principle when building an API should be to maximize app developer productivity and success.



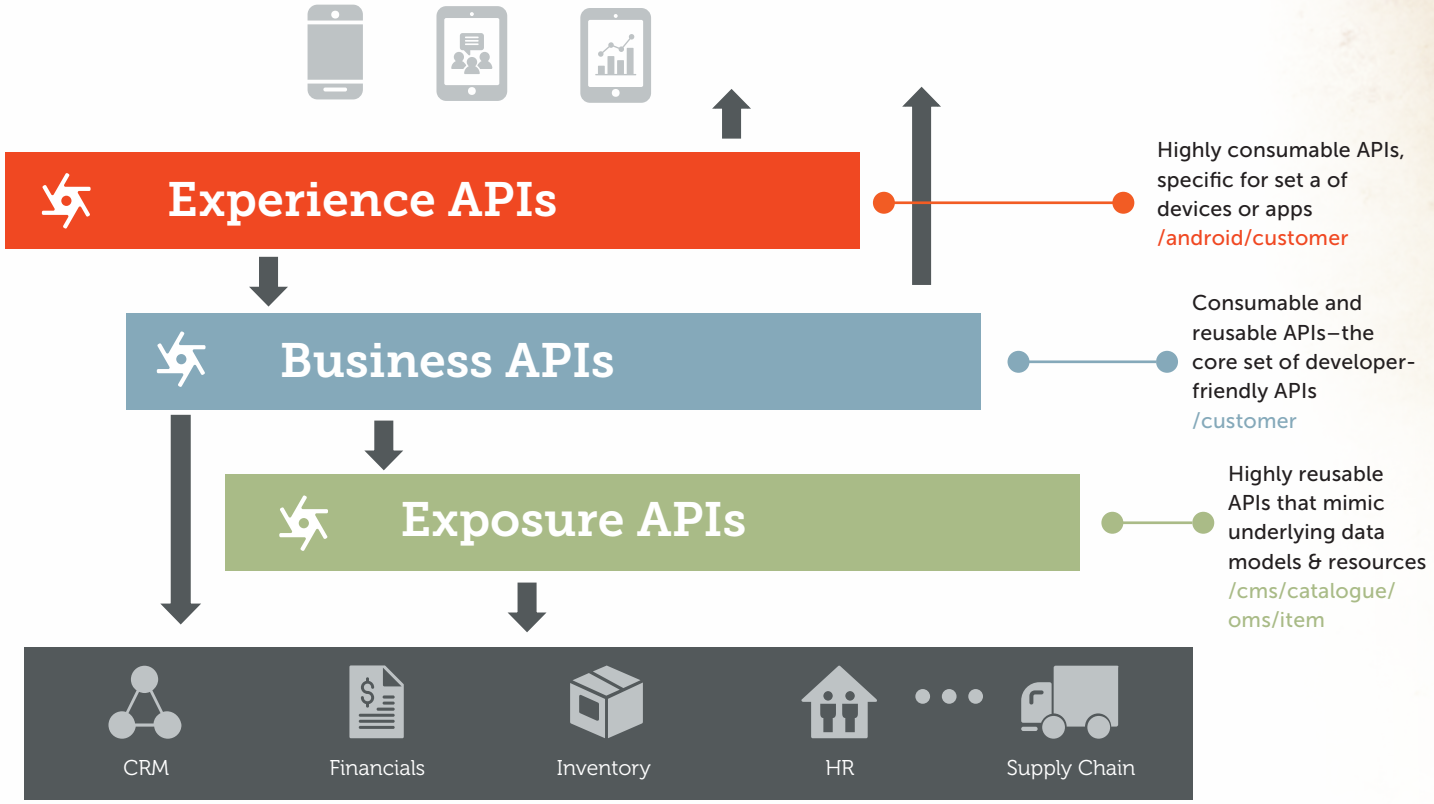
Modern organizations use the API-first strategy, in which they start by documenting the API using the Open API specification before implementing code. Documenting and mocking the API straight away enables greater collaboration and accelerates overall software delivery.

With the app developer in mind, there are several best practices to ensure agility and ease of consumption of APIs.

How to:
Design and build your APIs to ensure ease-of-consumption and fast delivery for developers

Adopt a layered API strategy

Organizations are embracing a layered API approach to make it easier for developers to consume SOAP and other legacy services from systems of record. This approach enables developers to deliver new software faster across a plethora of mobile devices and form factors.



In a layered API approach, you have various flavors of APIs:

Exposure APIs, which are RESTful, cached, secure modern APIs to your legacy SOAP services (/crm/catalog , /cms/item, for example). These are tied closely to your backend systems.

Business or domain APIs, which are abstractions of business entities as consumable resources for app developers. An API-first design approach focuses on delivering these consumable resources and hides implementation details from developers by providing a consistent interaction across disparate backends (/catalog, for example).

Experience APIs are tightly coupled to a particular set of consuming applications/devices.

Specializing to a particular UI, API developers provide an optimized experience to the app developer focusing on things like aggregation and optimizing payload for performance (/android/catalog, for example).

This layered API approach abstracts the underlying complexities and dependencies of APIs and systems below and accelerates delivery of new apps. Organizations use the concept of proxy chaining (connecting one or more proxies to connect to a remote host) available in API management platforms to build layered APIs.

Design easy-to-consume APIs

A good API design makes the API easy to consume by the app developer. Below are a set of design best practices that have enabled many API designers with SOAP design experience to build the right set of easy-to-consume RESTful APIs.

Using a data-centric model

APIs should focus on the underlying entities/resources they expose, rather than a set of functions that manipulate those entities. In other words, the URLs should have nouns, not verbs. For example, a collection of dogs could have a URL https://dogtracker.com/dogs. And, individual dogs would each have a unique URL like http://dogtracker.com/dogs/9876543. With this approach, you can retrieve the details of the dog using the GET method, delete the dog using the DELETE method, and modify properties of the dog using the PATCH or PUT methods.

By contrast, in a function-oriented API, there is much more variability, and much more detail a developer has to learn. And there is no clear structure or pattern you can use to help them with the next API.

Building simple JSON

Due to its simplicity, JavaScript Object Notation (JSON) has become the de facto standard for web APIs. When JSON is used well, it is simple and intuitive. If your JSON doesn't look as straightforward as the example below, you may be doing something wrong.

```
{ "kind": "Dog"
  "name": "Lassie",
  "furColor": "brown",
   ...
}
```

Your JSON API will be simpler and easier to understand if you stick to the principle that the names in your JSON are always property names, and the JSON objects always correspond to entities in your API's data model.

Expressing relationships as links

If your web APIs do not include links today, a first step is simply to add some links without making other changes, like this:

```
{ "id": "12345678",
  "kind": "Dog"
  "name": "Lassie",
  "furColor": "brown",
  "ownerID": "98765432",
  "ownerLink": "https://dogtracker.com/persons/98765432"
}
```

Using links makes it easier for app developers to consume resources, with less to learn and no need to hunt for documentation. Moreover, links can be plugged into templates to produce the right URL.

Designing URLs

A good way to make APIs human-friendly involves the creation of entity URLs that have the entity type in them when fetching a specific resource. Thus, instead of https://dogtracker.com/ZG9n;a8098c1a, it is more desirable to have https://dogtracker.com/dogs/a8098c1a. Also, it is not recommended to code a hierarchy of entities into an URL. Hierarchies are not as stable as they might seem; encoding them in your URLs could prevent you from reorganizing your hierarchies in the future.

For query URLs, it is recommended to use the format

https://dogtracker.com/persons/{personId}/dogs rather than https://dogtracker.com/search?type=Dog&owner={personId}

Many app developer prefer the first format because it is more readable, more intuitive, and easier for API developers to implement.

Handling errors

A good error design is important for API designers, as it provides context and visibility into how app developers use APIs. Developers learn to write code through errors and they depend on well-designed errors for troubleshooting, and resolving issues after the applications they've built using your API are in the hands of their users.

Thus, it is important to use standard HTTP status codes and complete HTTP response messages to communicate with app developers. For example, the `201 Created` status code should always be paired with a `Location` header that provides the URL of the newly-created resource.

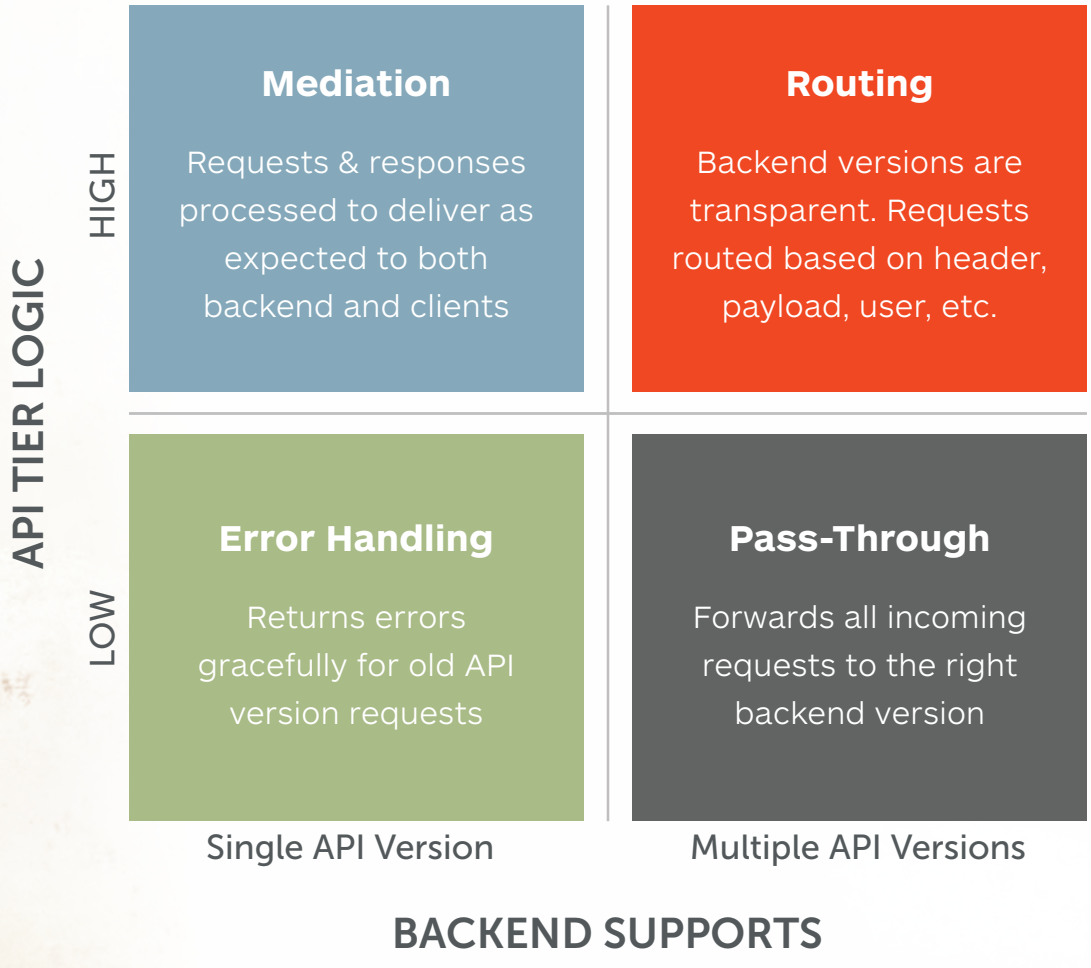
```
HTTP/1.1 201 Created
Location: https://dogtracker.com/dogs/1234567
```

Use plain language in the payload messages, as there usually is a programmatic user agent between the user and the message.

For further details on the above recommendations and many more best practices in API Design, refer to the eBook, *Web API Design: The Missing Link*.

Pick the right API versioning approach

A common problem in managing APIs is versioning. When APIs are used by many developers and partners, getting everyone to upgrade to the latest API version is challenging, if not impossible.



Organizations take different approaches to versioning, depending on their willingness to add business logic in the API tier and the limitations of maintaining multiple versions on backend systems. On the northbound side, most organizations typically include version numbers in the base path of the the API call (for example, api.company.com/v1/… , api.company.com/v2/…).

Error handling

If the backend systems can support only one API version and an organization wants to limit business logic in the API proxy layer, the API proxy can send an error for older versions. It can be gradually phased out by adding an error message to inform the developer, followed by the addition of strict rate limits on older API versions and, finally, blocking requests and sending errors.

Pass-through

If the backend supports multiple API versions, the API proxy layer can be kept lean and simply enable the northbound requests to pass through to the right soundbound APIs. Organizations use analytics in API platforms to understand the usage of API versions, deprecate low-traffic versions, and communicate with targeted developers using older versions.

Routing

Many organizations use an API platform to decouple the northbound and southbound APIs for agility and maintenance purposes. In these cases, depending on the variety of parameters like request path, query parameters, or payload, the API platform can route the request to the right backend versions based on business logic in the API proxy layer.

Mediation

If the backend system can't support multiple API versions, organizations can add business logic in the API proxy layer to mediate the request/response of various northbound API versions. The API platform will process the incoming requests and create a southbound request that's supported by the backend. It does the same on the response by processing the backend response and sending the response in the right format/data back to the client. As long as the mediation rules are relatively simple, this option is attractive.

"An API-first vision and approach is what is needed to take advantage of social, mobile, analytics, and cloud. It's the perfect storm for innovative CIOs."

- Brian Lillie, Equinix

Three-quarters of mobile apps fail standard security tests—and most cyber attacks target the app layer, according to Gartner.

Organizations have used web application firewalls and DDoS (distributed denial of service) protection solutions to secure their web apps. However, in the world of mobile, cloud, and microservices, where enterprise data is accessed with APIs in a zero-trust environment, what's needed is security at all points of engagement. After all, hundreds of thousands of sensitive customer records or millions of dollars are at risk.

There are known threats. The Open Web Application Security Project (OWASP), an online community of application security researchers, publishes an annual list of the top 10 security threats enterprises face. But there are also potential attacks from "unknown" threats—software that constantly scans for vulnerabilities in application infrastructures.

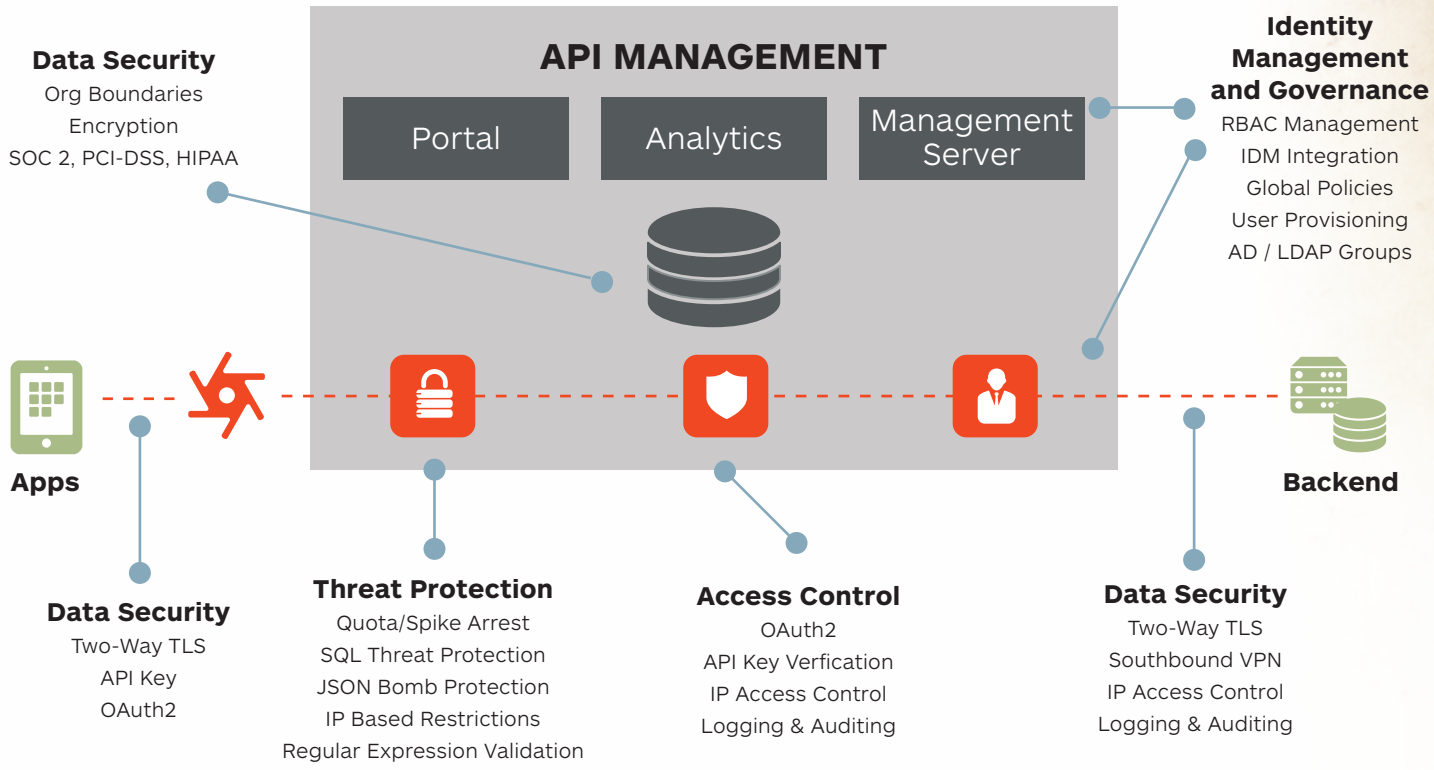
For protection against all kinds of external threats, organizations should create proxies in front of their APIs with an API management platform and enforce a set of consistent security policies at the API proxy layer.

How to:
Enforce a consistent set of security policies across all APIs

Mitigate OWASP threats

OWASP threats include XML/JSON injection threats, cross-site scripting attacks, broken authentication, insecure direct object reference, and several others. Seventy-two percent of organizations using an API management platform employ out-of-the-box security policies to protect their external-facing APIs from these threats.

To prevent data leakage in transit, organizations also implement out-of-the-box OAuth2 and two-way TLS on all critical APIs. Using OAuth2 with the right set of scopes ensures minimization of the attack surface available for threats. Using a PCI- and HIPAA-compliant API platform, organizations can create secure proxies and ensure APIs keys are stored in encrypted mode.



Prevent volumetric attacks

Application layer volumetric attacks comprise 17% of all reported DDoS attacks. In these attacks, an application is flooded with HTTP requests, tying up application servers. Eighty percent of organizations use API management platforms with out-of-the-box spike arrest and rate limiting policies to mitigate risk from such attacks. Applying quota and rate limits to APIs provides protection from sophisticated DDoS attacks that mimic human behavior.

Protect against adaptive threats

Unlike web apps, APIs are programmable, making it easier for attackers to target APIs using bots. Bot traffic can probe for weaknesses in APIs, abuse guest accounts with brute force attacks, use customer API keys to access private APIs, abuse loyalty programs, or scrape pricing data for competitors via APIs. Bad bots comprise 10% to 15% of internet traffic today.

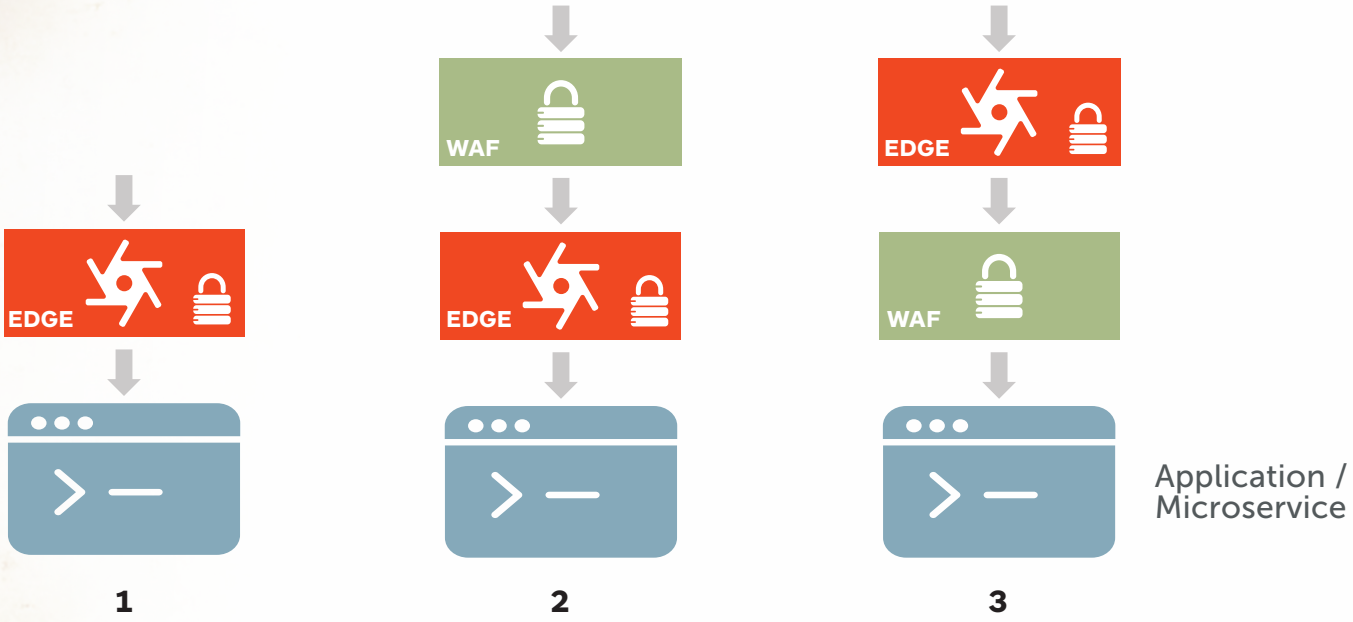
An advanced API platform uses sophisticated machine learning algorithms on data aggregated across multiple customers. Because it analyzes billions of API calls, it can distinguish legitimate human traffic more effectively than it would from a single data source.

When the API platform identifies a bot-like signature in an call pattern, it flags that signature. An organization can then specify the action to take for each such identified bot signature. The API platform then automatically takes appropriate actions like blocking, throttling, or honeypotting.

Don’t rely on WAFs for API security

Many organizations use web application firewalls (WAF) to secure their web apps. But how does an API platform, with its API security capabilities, fit with WAFs?

Typically, WAFs treat applications as black boxes and apply IP-based (or some light content-based traffic blocking). However, API platforms have the ability to fully inspect API calls—and, as a consequence, can do a better job of API security than a WAF. API platforms are “smarter,” as they have an execution engine built right into the proxy layer. Data persistence in API platforms enables you to do more (stop bots, for example) than stateless operations.



In the potential options above, as the sophistication of API platforms continue, most organizations’ application security needs would be covered by API platforms (option 1 above). It has the benefit of lower overall latencies and consistent management of security policies.

If your existing WAF is built into your CDN, option 2 might be the right approach. The API platform sits behind the WAF/CDN. In situations where applications can only be accessed through a WAF gateway, option 3 might the right approach.

> **“We wanted to secure our APIs, as we don’t distinguish between internal or external use.”**
>
> - Ole Dallerup, TrustPilot

Once API proxies are designed and built, they need to be tested and deployed into production. But these API proxies don't live in isolation. They're closely linked to the target APIs/backend applications they front. Thus, one needs to coordinate the API lifecycle and the software development lifecycle (SDLC) of these applications.

One of the critical phases of the API lifecycle is testing. Organizations have adopted test-driven development (TDD) and behavior-driven development (BDD) approaches to application testing; they're also using TDD and BDD to test their APIs. Organizations can also leverage their application testing framework and tools to automate API testing and reduce the maintenance costs of their API catalog.

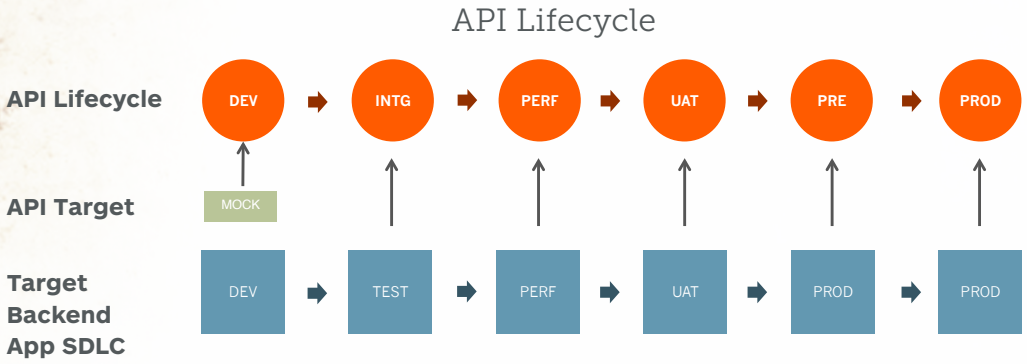
How to:
Sync the API lifecycle with the SDLC and automate testing and deployment of APIs.

Align the API lifecycle with the SDLC

Every organization has its own SDLC with various stages (dev, test, and prod, for example). The API lifecycle should be aligned with the SDLC process.

To ensure separation of concerns between production and non-production API usage, organizations use the concept of organization available in an API management platform to keep users, APIs, and API traffic distinct from one another.

API management platforms also employ the concept of environment, which provides the runtime execution context for the API proxies (target endpoints, for example). It is recommended to have two organizations (production and non-production) and six different environments for your API proxies in an API platform as shown below. Typically, the pre and prod environments are setup in a production organization, while the rest of the environments are setup in a separate, non-production organization.



In the example above, if the target backend app has fewer stages, you can either reduce the number of API lifecycle stages/environments or point a number of different API environments to the same target environment (API PERF and API UAT point to APP PERF, for example).

As the API proxy goes from one stage to another, different types of testing and target backends are used. An API lifecycle can be automated by employing the same continuous integration (CI) framework and tools used for applications.

Apart from the target app SDLC above, the apps consuming the APIs will have their own SDLC and the API lifecycle needs to be in-sync with that SDLC. Typically, there is also a sandbox environment provided for app developers to test their apps with a mock backend. After the initial app development, developers typically use prod APIs for the rest of their SDLC.

Test APIs using TDD and BDD approaches

Organizations use TDD or BDD approaches to create the right set of tests to validate the API being built. In the dev environment, API proxies are created from Open API specifications; alternatively, template patterns are used to generate proxies with yeoman generators or common proxy templates. The API proxies are then attached a set of out-of-the-box policies like XML/JSON threat protection or OAuth2. Most organizations store the API proxies in the same repository as their application code (typically GitHub).

Organizations do behavioral testing of their APIs in this stage, using mock targets to ensure wide testing coverage. Static code analysis can be run on custom policies (Java or JavaScript policies) with tools like JSLint and CheckStyle, followed by unit testing and code coverage. Typically, organizations use tools like Junit and Mocha for unit testing and Istanbul and Cobertura for code coverage.

In the INTEGRATION (INTG) phase, the APIs use a real TEST target. You can use tools like apickli for functional and integration testing. In the PERF stage, performance is tested using a non-production southbound target to determine performance degradation and volume capacity limits.

In UAT, PRE and PROD, a smaller subset of tests are run to test basic API setup and functionality. The PRE environment is a mirror of prod and is used for final testing before deployment of production APIs. When moving an API proxy into production, an organization should also publish the specifications and associated documentation to the developer portal.

Once an API proxy is ready for production deployment, an organization needs to decide where to deploy it.

Deploy APIs depending on type of workload

The target backends of API proxies can be microservices, legacy SOAP services, cloud workloads, or services in a PaaS. Several attributes of the target application/service determine where and how API proxies should be deployed. They include:

- **Type of service** Microservices or a serverless or monolith app
- **Existing interface** RESTful API or legacy API, such as a SOAP API
- **Application environment** On premises, in the public cloud, or in a PaaS
- **Use case** Internal consumption only or for external use (partners, for example)

An organization can either deploy production proxies into a centralized runtime environment or deploy proxies in a distributed mode.

Centralized deployment

In this mode, the API is deployed to a central set of gateways that are either hosted in the public cloud or in an organization's private data centers, depending on where the API management solution is deployed. Typically, for external use cases or monolith apps running in the public cloud, this is the preferred approach to minimize operational overhead. For applications with legacy interfaces that require complex transformations and processing, centralized API deployment may be the only option.

Distributed deployment

For internal use cases, microservices, or serverless apps, it's best to have the API closer to the application environment to minimize latency. In these cases, the API is deployed to a lightweight API gateway that is collocated with the app or microservices, while the rest of API management services like analytics and developer services are centrally located.

Automate your testing and deployment lifecycle

In many organizations, for compliance reasons and to reduce manual errors, the deployment lifecycle is automated. Most organizations use continuous integration (CI) methodology and tools to automate their software development lifecycle. The API testing and deployment lifecycle can be automated using the same set of tools. Using the management APIs of your API platform, organizations can programmatically migrate API proxies from one environment to another.

Automating the testing and deployment of APIs speeds up finding and fixing proxy code and integration errors, enabling more frequent and successful API proxy releases.

A typical approach entails writing scripts that programmatically deploy proxies into an environment, as part of a larger SDLC automated process that also deploys or migrates the application code. Organizations might use tools like Apache Grunt or Maven to automate their SDLC. By using API management plugins, such as the Apigee deploy maven plugin, to these popular devOps tools, it's easier to automate the API testing and deployment lifecycle and fit into an organization's SDLC environment.

"Moving to a platform-based approach, where APIs wrap our core services, allows us more business agility."

- Paul Clark, ITV

A key aim for API providers should be the prolific adoption of APIs by app developers. As organizations build developer portals, the key questions to answer include:

- How do we make APIs easily discoverable and what API-related content and packaging do we need to make app developers successful?
- How do we manage app developer onboarding?
- How do we manage identities and access permissions of our API team members, internal app developers, and partner app developers for the developer portal?

How to:
Publish easy-to-use APIs with interactive documentation and self-service capabilities

Publish automated, interactive documentation

A big part of making APIs easy to use for developers is making them easily discoverable. Interactive API documentation, sample request/response patterns, and easy tracking of API usage help, too.

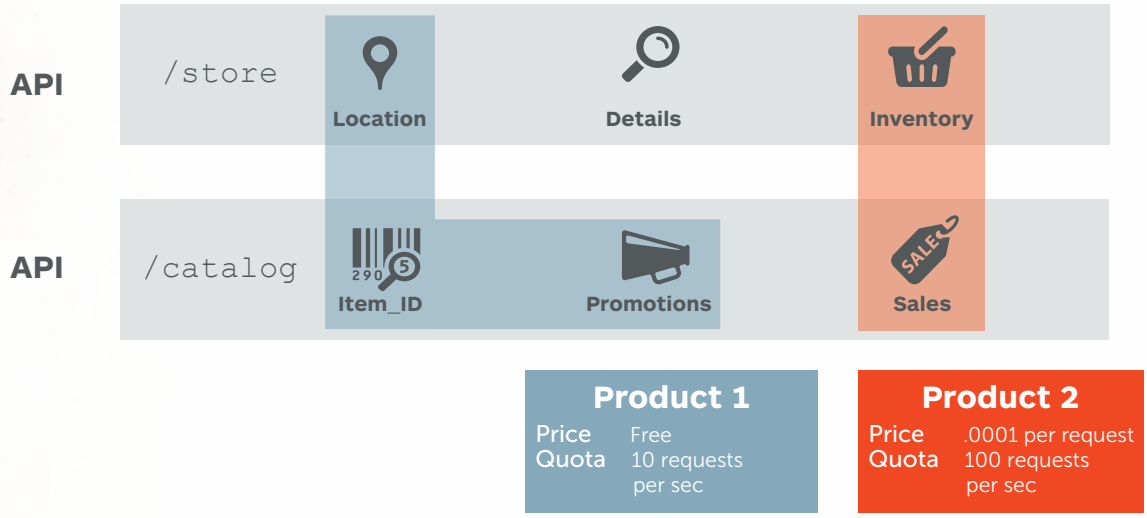
Most organizations use the Open API specification to create interactive API documentation. It enables developers to not only understand the API specifications, but also try, test, and debug API calls.

API management platforms enable the automatic generation of API documentation and publishing to the developer portal. The automatically generated, up-to-date documentation ensures app developers can easily discover the right APIs and get started.

Enabling a vibrant developer community also entails providing feedback, making it easy to request support and features, and enabling the submission of content that can be accessed by other developers.

Package your APIs for consumption

Organizations also simplify API consumption and enforce API best practices by promoting the concept of an API product, which is a way to package APIs and associated resources into bundles and attach rate limits and pricing. Using the API product feature available in API platforms, organizations can create different tiers of offerings for different sets of users using the same set of APIs and resources.



An API product enables organizations to achieve two objectives: differentiated access to APIs to various groups of users, and the ability to quickly try out new API-based business models.

In terms of differentiated access, organizations can configure API products to provide access to APIs only to API developers, or for certain group of registered app developers like partner developers, or to all registered app developers (in the case of public APIs).

Organizations can also easily experiment with different out-of-the-box business models for their APIs, including usage-based pricing ($0.00005 per API call, for example), a flat fee ($100 per month) or a revenue-sharing model (2% of API-related revenue).

Automate developer onboarding

There are several ways to onboard app developers to a developer program and get them started building apps using an organization's APIs. The appropriate mode depends on the business model, target developers, and a company's compliance policies.

Fully self-service

Here, app developers can sign up, register their app, get their app keys, and get started—all without any approvals from the portal administrator. Most internal developer portals are set up in this mode, and they're typically integrated with corporate active directory systems.

Organizations that have public APIs and want to engage large developer communities use this method as well. In most cases, the administrator is notified by email about the signups to track usage.

Admin-approved

In this mode, app developers register on their own, but there's an admin approval step in the process. Upon admin approval, the app developer can register apps, get keys, and access all the documentation. Organizations that want to expose APIs to strategic partners tend to use this method.

Admin-led

Here, the portal administrator signs up the app developers. This is an uncommon approach, but in certain, highly sensitive cases, organizations will use this method to restrict access to APIs.

Tie user identities to existing enterprise IDMs

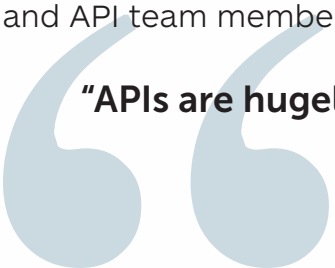
Several types of developers use a developer portal, including an organization's API developers, as well as internal and external app developers. For each type of user, an organization chooses one of two approaches to manage users' identities in the developer portal.

Built-in directory

Most developer portals provided by API management vendors have a built-in module to store and manage user identities. Typically, organizations use this capability to store external app developers' information.

Single sign-on integration

Most enterprises have an active directory (AD) and use an active directory federated service (ADFS) provider like Okta, CA Siteminder, or Ping Identity. To enable single sign-on using ADFS, API management developer portals enable easy integration with an enterprise ADFS through SAML protocol. This is the most common implementation for internal app developers and API team members.

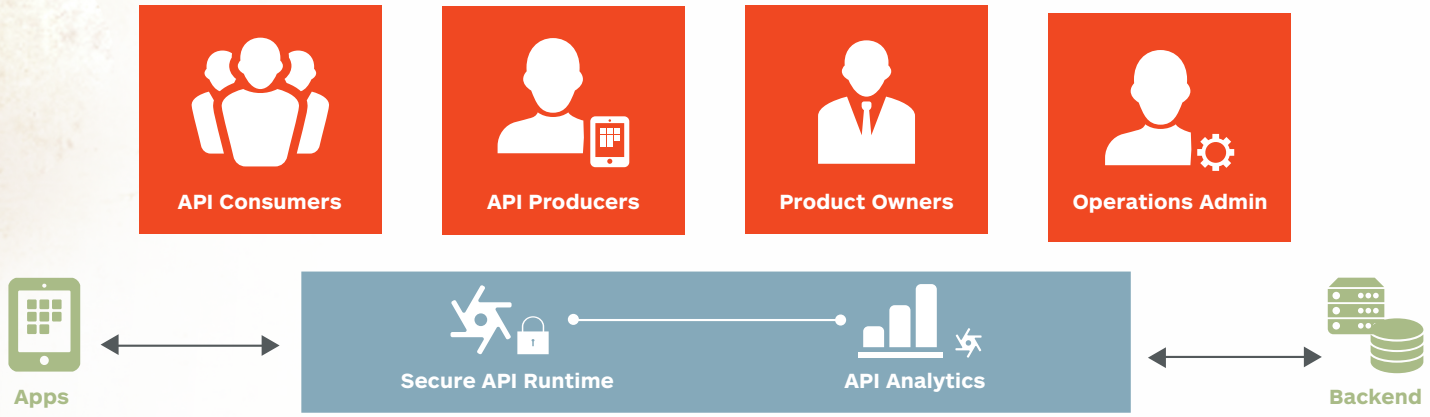


"APIs are hugely important for today's business ... we see APIs as critical to our ability to keep pace with customers."

- Charlotte Yarkoni, Telstra

API Analytics

API providers need to measure, analyze, and act on metrics associated with their APIs and API programs. API programs typically involve four types of users with unique needs when it comes to analyzing API metrics.



API consumers

App developers want to understand the volume of API traffic and the quality of service (success rates, response times, and response codes, for example) for the APIs that they build their apps against. App developers also need to track business metrics (including money exchanged with the API producer), based on the API product pricing plans.

API producers

API developers care about building APIs using best practices based on learnings derived from other API developers who are doing similar things (such as applying specific types of policies to their API proxies). In addition, API developers need visibility into the step-by-step behavior of all the APIs they build in order to diagnose latency problems and improve performance of those APIs.

Product owners

Product managers are responsible for the success of API programs; they need to measure the adoption and usage of the published APIs across various dimensions including products, developers, apps, channels, and locations. Product managers also want to measure the business impact and financial value of those APIs by capturing transaction or business metrics related to them.

Operations admins

Operations teams care about maintaining peak performance and availability of their APIs. They want to see the throughput, latency, and errors associated with those APIs. In addition, they expect to get alerted in near real-time to quickly identify and resolve any issues that affect the quality of service of their APIs. The same teams also care about protecting their APIs against malicious bots that may compromise their data and services.

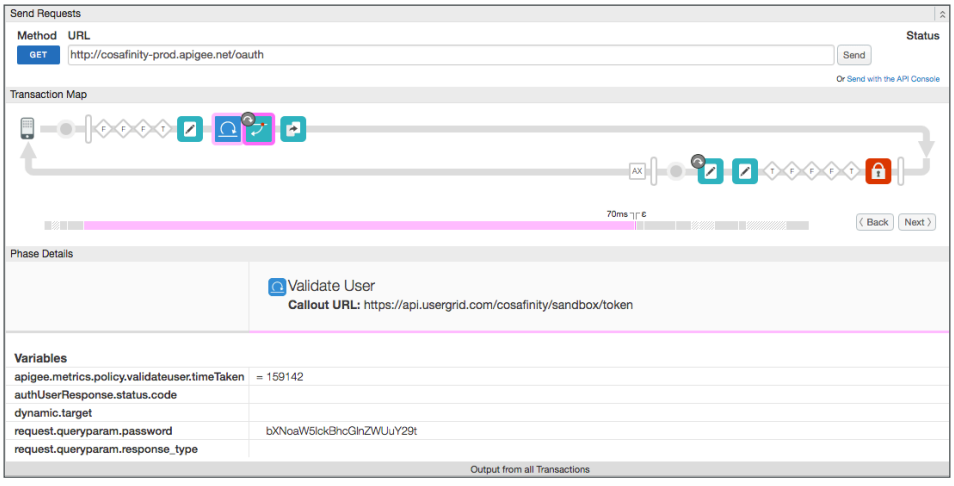
Analytics in API management platforms solves different problems for each of these user types and leverages data related to APIs, app developers, applications, and end users.

How to:
Use analytics to gain better insights into your APIs and API program

Optimize API functionality by tracing API calls

API developers apply a set of policies on APIs to ensure seamless and robust app functionality while protecting their backend systems. API developers must ensure that once implemented, their APIs are functioning as expected and performing with minimal latencies. This is enabled by visibility into the step-by-step flow with timing information for each API request as it flows through the API proxy.

Here's an example of a real-time trace capability that can help API developers diagnose their APIs.



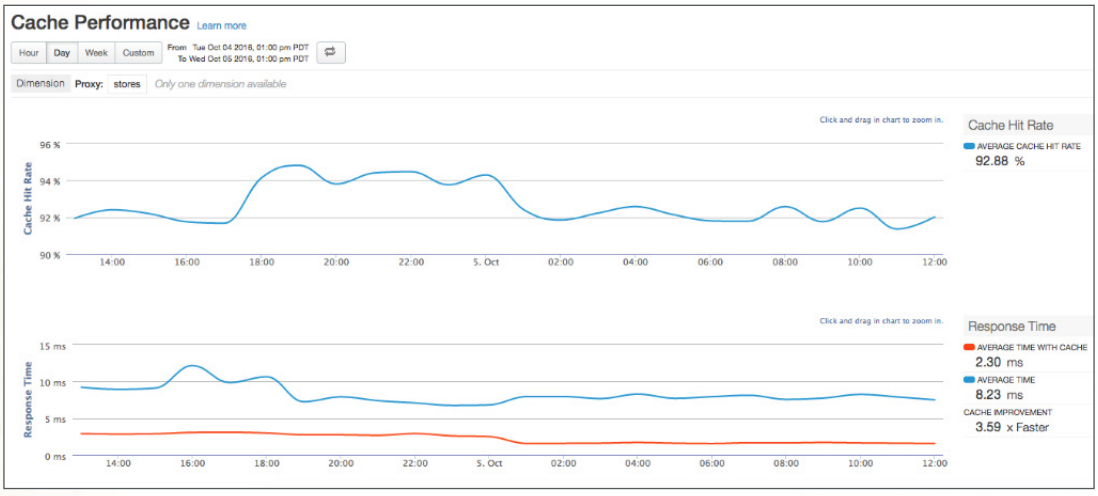
Implement the wrong policy, and the API will never get used by app developers. For example, putting an OAuth policy in a product catalog API will force a user to log in to the mobile app before getting generic information about a company's products. This adds friction to that API's adoption. By anonymously analyzing APIs across a wide population of customers, the analytics platform can provide API developers insights into best practices on the most common policies implemented across a cross-section of APIs.

Monitor peak performance and availability

Once deployed, APIs become the conduit—and potentially the gating factor—for all user experience that depends on information exchanged via those APIs. Operations teams need the ability to monitor various traffic metrics in near real-time. In addition to keeping track of total traffic volume and throughput for each of the APIs, the following metrics serve as first-level indicators for the overall health of the published APIs:

- Response times for both the API proxy as well as the backend systems at multiple call distribution levels (median, TP95, and TP99, for example)
- Availability measurements based on error rates at each of the various tiers (client tier, API proxy, and the backend systems)
- Cache performance for measuring response times and hit rates for each API enabled with local cache

The diagram below shows the benefit of using a caching policy as part of the API where over 90% of the API calls were addressed from that cache. This resulted in a net improvement of over 3.5x in response time.



Another concern is identifying and blocking malicious users (typically automated bots) from hitting APIs to either steal valuable information or consume resources. Analyzing incoming traffic for patterns associated with API call frequency, location, and sequences can give operations teams the power to maintain optimal operation of their APIs for all their consumers.

Measure API program success with the right metrics

To measure the success of any API program, product managers must be able to analyze the following types of metrics and reports:

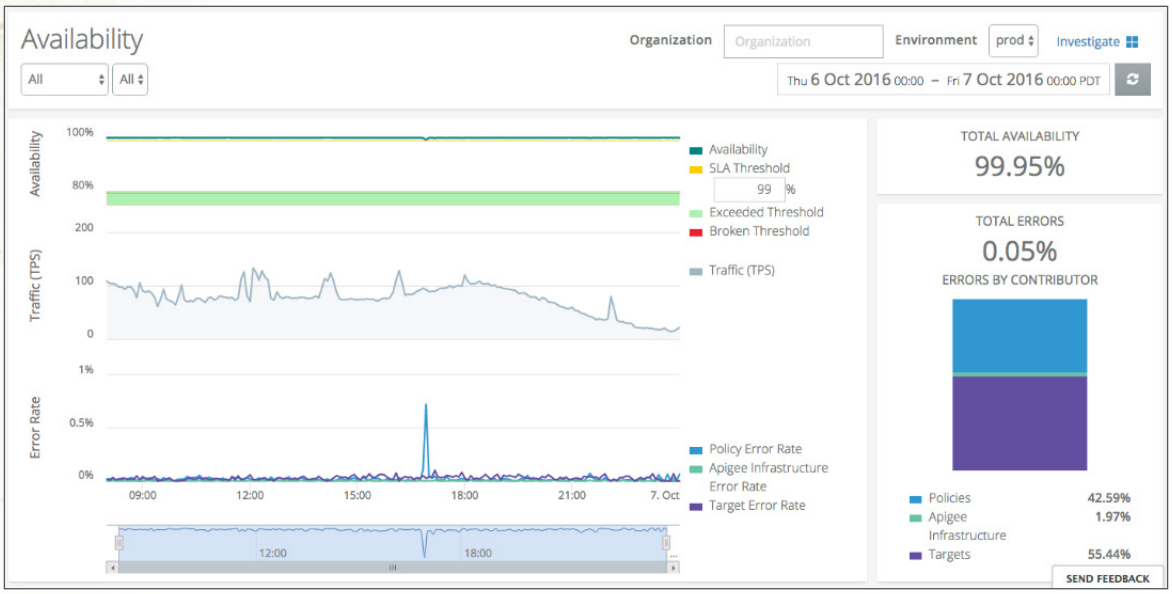
- API traffic trends broken down by products, app developers, and apps
- Trends in signups of new app developers and apps registered for each of their products
- Revenue or business value delivered for each of their published APIs
- Revenue generated from app developers for subscribing to their published APIs
- Most prolific or highest-value developers
- Developers who are consistently exceeding their quotas
- Developers who use APIs for free and are candidates for paid offering

Empower app developers with usage and performance data

The organizations that best engage developers provide them with insight into their specific API usage, performance metrics, and revenue measures.

App developers who subscribe to API products through the company's developer portal get visibility into their usage and quality of service for each of those APIs. Some of the metrics that app developers care about include:

- Traffic volume, response times, and errors for each of the APIs called over time
- Breakdown of API calls by the various registered apps
- Distribution of clients (location, device type, OS platform) making those API calls
- Overall availability for each of the APIs for valid calls that don't contain client-side errors



In addition, if the app developer has subscribed to specific pricing plans for using those APIs, then it's necessary to provide some of the following reports for those developers as part of the developer portal:

- Traffic volume that applies to each of the various pricing tiers
- Monthly payment breakdown and overage charges (if applicable) per pricing tier
- Revenue shared (if applicable) by the API publisher for calls made by the API subscriber's apps

"Analytics helps us identify partner applications that aren't delivering the best GoToMeeting experience for our users so we can help those developers improve their apps."

- API program manager, Citrix

So you've decided to purchase an API management platform. Now you need to decide where you deploy your API management solution: in the public cloud or in your own private cloud? Or is there a hybrid approach that's appropriate?

Organizations need to have a plan to integrate API management into their existing infrastructure, including monitoring and logging systems. For on-premises deployments, there should be process in place to scale up and scale down the API management infrastructure as the business needs change.

How to:
Integrate API management into your enterprise operations

Deploy API management in the cloud or on premises

When evaluating deployment alternatives, there are several considerations: time to success, total cost of ownership, security, performance, scalability, and reliability.

Time to success

Deploying API management in a private cloud requires time to acquire, provision, and deploy hardware; configure software; and train employees to manage the software. Typically, cloud deployment is the fastest way to launch your API program. For example, the Dutch consumer review site, TrustPilot, went live in production on Apigee's API platform in only four hours.

With readily accessible infrastructure and the right people, however, a private cloud is a viable alternative, as it grants more control over an organization's infrastructure.

Total cost of ownership

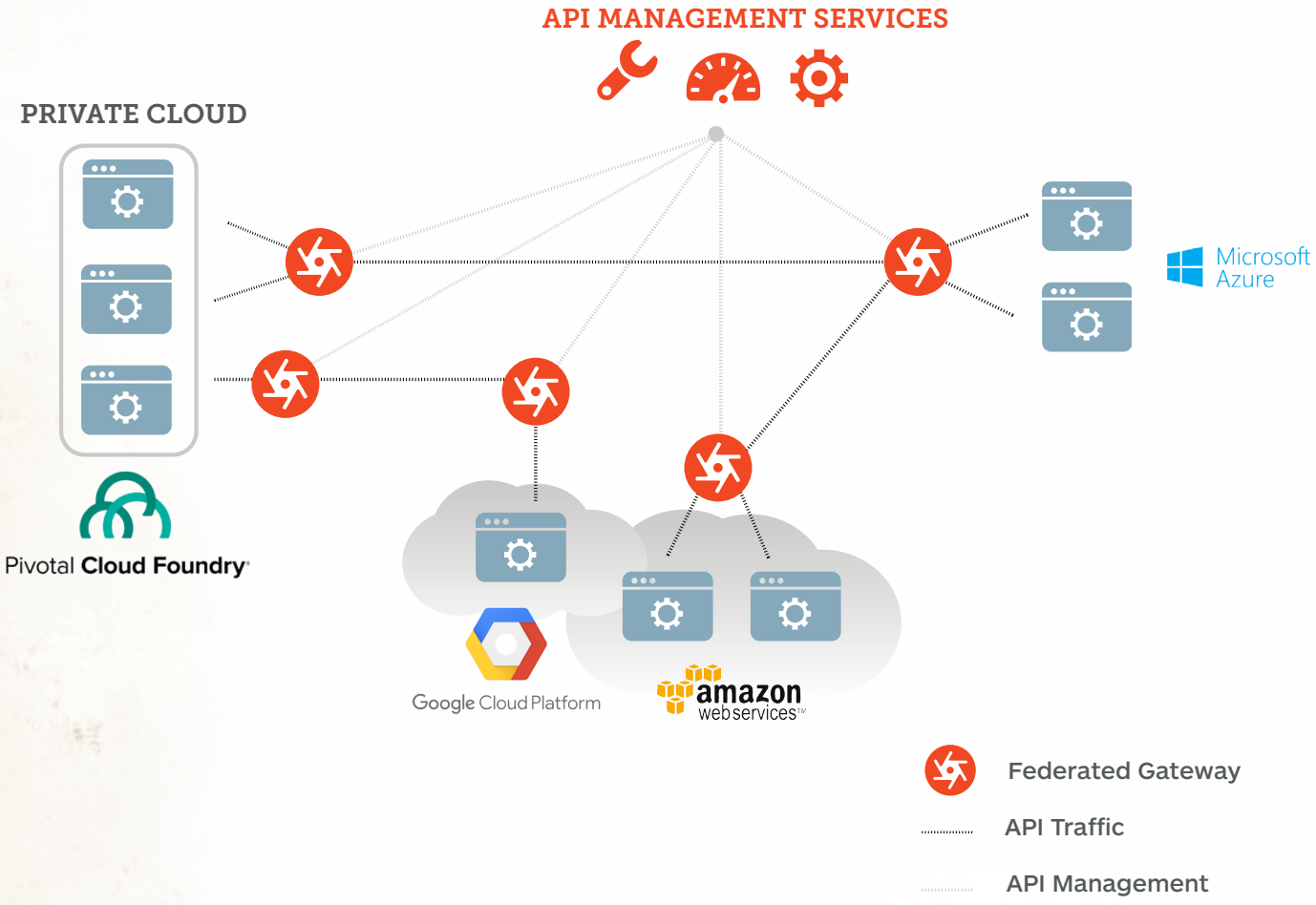
Typically, private cloud deployments tend to have lower software license costs compared to API management cloud subscription fees. To do an apples-to-apples comparison, organizations typically look at the total cost of ownership of each option over three years.

After factoring in infrastructure and people costs to deploy, manage, monitor, and support the API management infrastructure around the clock, the cloud option typically carries a lower total cost of ownership than the private cloud option. API management vendors can distribute infrastructure and operational costs across a large set of customers—and pass the savings on.

Performance

In most use cases, performance doesn't differ much between private or public cloud options. There can be an exception, however. In internal use cases, where the target backends and API users are both in the private cloud, public cloud deployment can sometimes add additional round-trip latency to the API call. In this instance, one can pursue either the private cloud or a hybrid cloud solution.

In the hybrid approach, federated gateways are co-located with the application environment, while the rest of API management is in the public cloud.



Security and compliance

Depending on an organization's specific security and compliance requirements, private cloud deployment might be the only option. In some organizations, certain workloads like payment transactions cannot be on the public network. In these cases, organizations pursue a hybrid API management deployment, where some workloads remain on-premises while the rest are in the public cloud.

API management vendors have put in place many security processes, and have employed third parties to audit, certify, and enhance the security of their public cloud offerings.

Scale and reliability

An organization's peak API traffic volume and uptime requirements can also help determine the right deployment option, as the public cloud option might or might not be available from your API management vendor. For example, Apigee in the public cloud processes over 300 billion API calls per year and hit a peak traffic of over 50,000 requests per second over Thanksgiving weekend 2015. Apigee also delivered 99.99% availability to its customers over the past year.

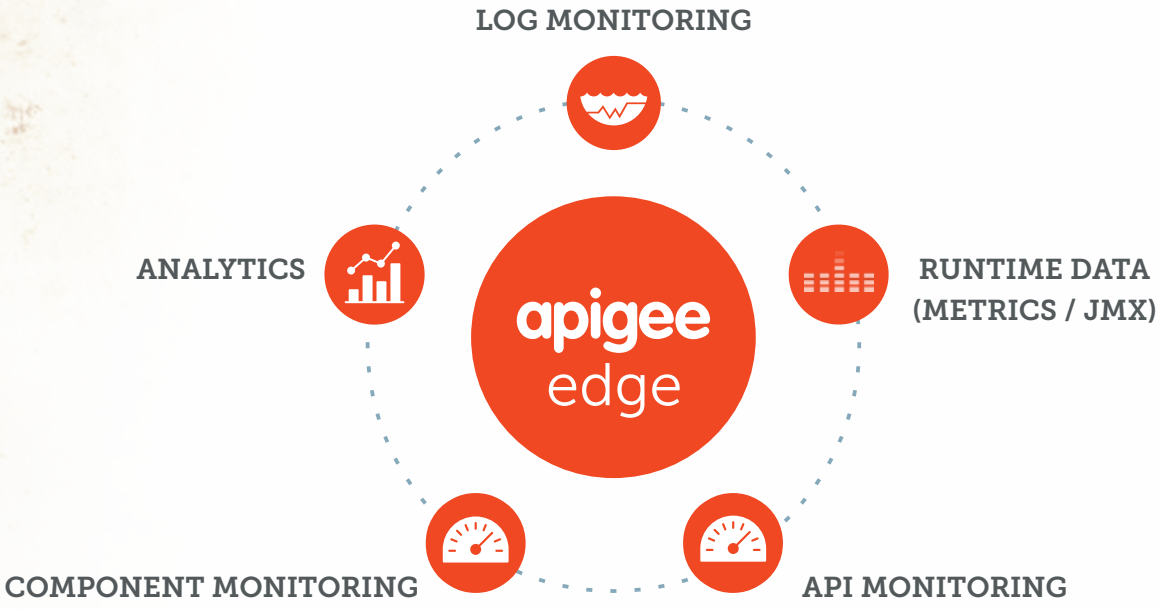
Integrate with existing monitoring infrastructure

Once an organization has deployed API management, the API operations team will want to monitor various types of data.

Built-in mesage logging policies in an API platform enable organizations to generate logs and process messages in an API proxy. Organizations then use logging tools like Splunk to collect and analyze log data. One can inject a correlation ID in the API tier to correlate events in the logging platform.

Typically, API platforms enable the collection of runtime data (including API response time, error rates, and target latency data) using JMX MBeans. Organizations can then use any JMX compliant APM tool to access runtime data using JMX for on-premises installations.

The loose coupling of an API platform and the APM tool using JMX grants the flexibility to easily switch APM platform in the future, if needed.



API monitoring Many API platforms provide this capability to do stress testing of APIs and target systems. It enables organizations to deploy health check APIs that hit target systems through the API platform and monitor the performance of these APIs.

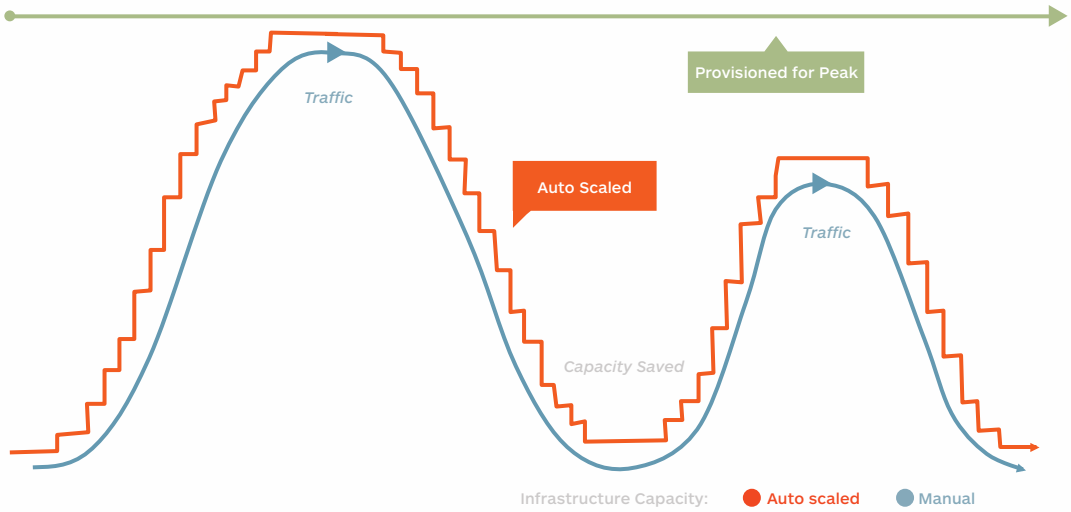
Component monitoring Organizations can conduct system level checks (e.g., CPU, memory, network, Disk) and JVM checks (e.g., Thread statistics, heap, GC) by invoking API platform APIs and using existing monitoring tools.

API analytics Get visibility into variety of usage (developer, API traffic) and performance data (response time, error rate) data. API platforms provide out-of-the-box API analytics that can be used to track this data.

If API management is in the public cloud, the infrastructure is maintained by the vendor so organizations typically have access to only API monitoring and API analytics data.

Scale API platform infrastructure

Organizations that deploy API management in the public cloud take advantage of auto-scaling and blue-green deployments; both are available out-of-the-box from API platform vendors.



Scaling private cloud deployments

There are two aspects of infrastructure scaling: API gateways (routers and message processors) and distributed databases that store API keys, users, and policies.

Most vendors provide a private cloud estimator (PCE) that automates sizing and topology requirements analysis and the creation of topology design. The tools take as input a collection of requirements, and perform analysis based on the vendor's recommended topology design patterns and practices. PCE is typically available in two interfaces: a Web UI that makes it simple to submit requirements and visualize results as well as a REST API, which supports all available functionality on the Web UI.

An API gateway typically has a router and a message processor that processes and routes incoming API requests and outgoing API responses. There's a variety of factors that determine how the API gateways should be scaled up or down, including the average transactions per second, peak transactions per second, complexity of the proxies (processing done by message processors), geographic distribution, and resiliency requirements.

Most API platforms use distributed databases to store API proxy bundles, API keys, and app developer profiles. An organization might need to scale up its distributed database if its data outgrows the capacity of a cluster or node, or to improve latency when API traffic increases.

In distributed databases like Cassandra, capacity can be added to an existing cluster by adding one node at a time or doubling the capacity.

Generally, automating the scaling of the infrastructure to eliminate manual provisioning errors is recommended.

Conclusion

APIs play an increasingly critical role in evolving application architectures, with organizations adopting modern software development practices like microservices, multiple clouds, and PaaS. APIs are the connecting tissue.

But as APIs pervade enterprises, a host of new challenges arise, requiring a holistic approach to managing APIs across the organization.

A sophisticated API management platform is the answer; it provides consistent security, visibility, discovery, and reuse of all an organization's APIs. An API management platform facilitates operational agility, enabling companies to treat monolithic legacy systems as modular microservices that can be modified without impacting overall system health.

It enables the speedy delivery of APIs that are easy for developers to consume, thanks to interactive documentation and self-service capabilities. It provides insights into an organization's APIs and API program, thanks to advanced analytics dashboards. It enables the management of APIs to be tightly integrated with enterprise operations, thanks to public and private cloud deployment options.

And an API management platform provides enterprises a single pane of glass to securely manage all APIs across the organization.

In today's digital economy, organizations must either adopt best practices in each stage of the API lifecycle—or risk becoming cautionary tales. Enterprises increasingly are relying on API management platforms to meet this challenge.

About Apigee

Apigee® powers the APIs that make every business a digital business. Apigee provides a leading API platform that helps companies—from disruptive start-ups to the Fortune 100—rapidly adapt to the business and technology requirements of the connected, digital world. Many of the world's largest organizations select Apigee to enable their digital business, including over 30 percent of the Fortune 100, four of the top five Global 2000 retail companies, and five of the top 10 global telecommunications companies.

For more information, visit apigee.com

Share this eBook