

API Strategy: Best Practices for Platform Engineering Leaders

Scale Operations, Fight Sprawl, and Build a Resilient API Platform

By Andrew Stiefel and Akash Ananthanarayanan, F5 NGINX



Table of Contents

Introduction: API Sprawl	4
API Adoption Continues to Increase	5
Factors Driving API Sprawl	6
Consequences of API Sprawl	7
How Can Platform Engineering Leaders Respond?	8
Prerequisite: Install and Configure API Connectivity Manager	9
1. Create a Single Source of Truth for Your APIs	10
Build an Inventory of Your APIs	10
Streamline API Discovery with an API Developer Portal	11
Tutorial: Create a Developer Portal with NGINX	11
Automatically Generate API Documentation	12
Ensure Proper Versioning	13
Generate API Credentials	14
Try Out APIs on the Developer Portal	16
2. Put an API Governance Plan in Place	17
The Importance of API Governance	17
Why Do You Need API Governance?	17
What Types of APIs Do You Need to Govern?	18
Common API Governance Models	19
Implement Adaptive Governance to Empower Developers	20
Tutorial: Govern Your APIs with NGINX	20
Provide Shared Infrastructure	21
Give Teams Agency	22
Balance Global Policies and Local Control	23
3. Adopt an API-First Approach to Building Microservices	26
What Is API-First?	26
The Value of API-First for Organizations	27
The Importance of Adopting a Common API Specification	28
The OpenAPI Specification	30
How to Use NGINX for API-First Software Development	31
Publish APIs to the API Gateway	32
Generate API Documentation for the Developer Portal	32
Apply Positive Security to Protect API Endpoints	32

4. Manage APIs Across Multi-Cloud and Hybrid Architectures	33
Common Multi-Cloud and Hybrid API Deployment Patterns	33
Tutorial: Enable High Availability for API Gateways in Multi-Cloud and Hybrid Environments	34
Deploy NGINX Plus Instances as API Gateways	35
Set Up an Infrastructure Workspace	36
Create an Environment and API Gateway Clusters	37
Deploy an Environment with One API Gateway Cluster	38
Create an Environment and API Gateway Cluster	38
Assign API Gateway Instances to an API Gateway Cluster	39
Deploy an Environment with Multiple API Gateway Clusters	41
Create an Environment and API Gateway Cluster	42
Assign API Gateway Instances to an API Gateway Cluster	43
Apply Global Policies	45
5. Protect APIs Across Every Touchpoint	47
The Rise of API-First Software Development	47
The Attack Surface Grows as APIs Proliferate	48
Thwarting API Attacks Requires the Right Strategy and Tools	49
What Is API Security Posture Management?	50
What Is API Security Testing?	51
What Is API Runtime Protection?	51
API Security Best Practices	52
Conclusion	53
6. Identify and Track Important API Metrics	54
Operational Metrics	54
Infrastructure Teams	55
Application Teams	55
Adoption Metrics	55
Product Metrics	56
Conclusion	57

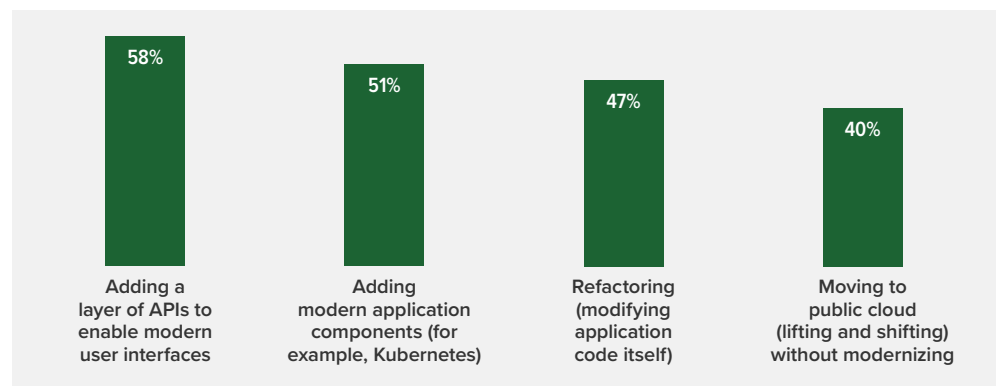
Introduction: API Sprawl

APIs power modern business. Enterprises use APIs to connect across internal teams, integrate with business partners, and deliver customer experiences. APIs are also one of the most popular methods that organizations use to modernize their existing software applications.

According to F5's [2021 State of Application Strategy Report](#):

- 58% of organizations are adding a layer of APIs to enable modern user interfaces
- 51% are adding modern application components (for example, Kubernetes)
- 47% are refactoring (modifying application code itself)
- 40% are moving to public cloud (lifting and shifting) without modernizing

Figure 1: API Usage in Organizations



Source: F5's [2021 State of Application Strategy Report](#)

NEW TRENDS IN SOFTWARE ARCHITECTURE AND DESIGN ARE INCREASING COMPLEXITY

As businesses and applications scale, so does the number of APIs they have in production. According to [TechRadar](#), the average enterprise today is leveraging a total of 15,564 APIs, up 201% year-on-year.

New trends in software architecture and design are increasing complexity. Cloud-native applications are more often distributed and decentralized by design – composed of dozens, hundreds, or even thousands of APIs deployed across cloud, on-premises, and edge environments.

This proliferation of APIs across teams, infrastructure, and applications is called *API sprawl*. Recent research from F5's Office of the CTO identified [continuous API sprawl](#) as a significant threat to businesses undergoing digital transformation. But what does that mean?

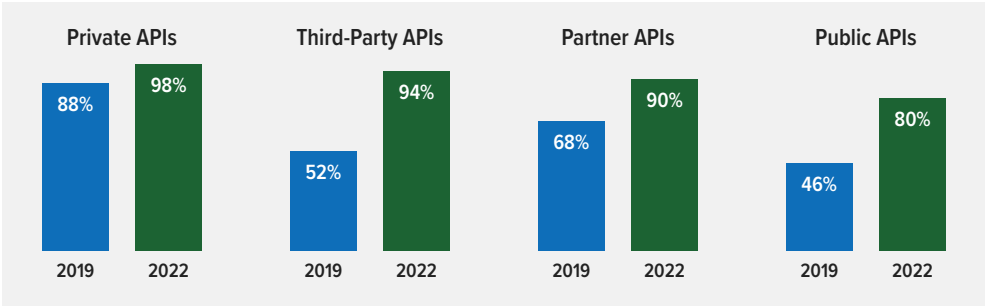
API sprawl describes two intertwined challenges that arise as organizations implement digital transformation: exponential growth in the number of APIs and the physical distribution of APIs across multiple architectures and teams.

API ADOPTION CONTINUES TO INCREASE

According to a recent survey by Gartner, enterprise API adoption has increased dramatically since 2019:

- Private APIs – 98%, up from 88% in 2019
- Third-Party APIs – 94%, up from 52% in 2019
- Partner APIs –90%, up from 68% in 2019
- Public APIs – 80%, up from 46% in 2019

Figure 2: API Adoptions Continues to Increase



Source: Mark O'Neill, Chief of Research for Software Engineering, Gartner (2022)

FACTORS DRIVING API SPRAWL

Many factors contribute to API sprawl. Some of the most common include:

- **Microservices architectures** – The growing adoption of microservices architectures leads to the proliferation of API endpoints as new services come online.
- **Multi-cloud and hybrid infrastructures** – Today **81% of enterprises operate across three or more architectures**, including public clouds, on-premises data centers, and edge infrastructures.
- **DevOps practices** – Developers can rapidly churn out dozens of APIs, or many versions of a single API, over a short period of time.
- **Abandoned APIs** – As developers move on to support and work on other projects, they stop managing and maintaining the APIs they created.
- **Business complexity** – Globally distributed teams and different lines of business create ad hoc APIs for different processes, often with little oversight and security.

Unless preventive measures are put into place, these factors can quickly lead to API sprawl becoming endemic across your platform infrastructure. This leads to many significant consequences.

Factors Driving API Sprawl

Many factors contribute to API sprawl. Some of the most common include:

- 01 Microservices Architectures**


The growing adoption of microservices architectures leads to the proliferation of API endpoints as new services come online.
- 02 Multi-cloud and Hybrid Infrastructures**

Today 81% of enterprises operate across three or more architectures, including public clouds, on-premises data centers, and edge infrastructures.
- 03 DevOps Practices**

Developers can rapidly churn out dozens of APIs, or many versions of a single API, over a short period of time.
- 04 Abandoned APIs**

As developers move on to support and work on other projects, they stop managing and maintaining the APIs they created.
- 05 Business Complexity**

Globally distributed teams and different lines of business create ad hoc APIs for different processes, often with little oversight and security.



API SPRAWL CREATES BOTH OPERATIONAL AND SECURITY CHALLENGES

CONSEQUENCES OF API SPRAWL

Many businesses don't yet recognize API sprawl as a significant problem – but it is. Organizations that understand and address the root causes of API sprawl are the ones that will thrive in the coming decade.

API sprawl creates both operational and security challenges. As API endpoints proliferate across multiple teams and environments, securing and governing APIs becomes a monumental task. For enterprises, API sprawl often results in hidden costs – lower developer productivity, more rework, slower reviews – that are not easily measurable until it is too late.

Some of the most common consequences of API sprawl include:

- **Increased complexity** – Keeping track of changes, or who is responsible for a particular API, creates a lack of clarity and can lead to delays in releases and increased costs.
- **Diminished visibility** – Multi-cloud and hybrid architectures often use different tooling, making it difficult to get a unified, consistent view of API traffic and vulnerabilities.
- **Decreased reliability** – With increased complexity and lack of visibility comes an increase in misconfigurations, which can result in partial or full outages.
- **Elevated threats** – It is challenging to monitor who has access to APIs and what they can do. This in turn heightens the risk of data breaches and other security issues.
- **Lower productivity** – With too many APIs, it becomes difficult for developers to find them. Time is wasted on searches, and teams might end up recreating APIs that already exist.
- **Less agility** – It is risky to make changes to an API when there are dependencies on it that might be unknown. The inability to react quickly results in missed opportunities and a competitive disadvantage.
- **Higher costs** – A higher numbers of APIs usually corresponds to higher development and maintenance costs.

Ultimately, API sprawl indicates a lack of software development, governance, and security practices that are adequate for the growing demands of complex engineering organizations. Without the proper practices and tooling in place, it is hard to have a clear picture of what is happening within your infrastructure.

HOW CAN PLATFORM ENGINEERING LEADERS RESPOND?

To start, identify whether you currently face a problem with API sprawl by answering these questions:

- Do you have an up-to-date inventory of your APIs?
- Do you have clear, up-to-date documentation for all your APIs?
- Do you know where each API is running?
- Do you have an established owner for each API?
- Does each API have a clear purpose that doesn't overlap with another API?
- Do security tests on your APIs return a clean bill of health with no vulnerabilities?

If you answered 'no' to any of these questions, you either are experiencing API sprawl in your organization or are at high risk of developing problems in the future.

The first step in building a resilient API infrastructure is getting a handle on API sprawl. This usually starts with the creation of holistic API strategy that incorporates best practices around persistent API ownership and the creation of a central API or service catalog. Next, overlay API governance and security to streamline API lifecycle management in a practical and scalable manner.

How Can Platform Engineering Leaders Respond?

To start, identify whether you currently face a problem with API sprawl by answering these questions:

- 01 Do you have an up-to-date inventory of your APIs?
- 02 Do you have clear, up-to-date documentation for all your APIs?
- 03 Do you know where each API is running?
- 04 Do you have an established owner for each API?
- 05 Does each API have a clear purpose that doesn't overlap with another API?
- 06 Do security tests on your APIs return a clean bill of health with no vulnerabilities?



ORGANIZATIONS FARTHER
ALONG THEIR API JOURNEYS
TYPICALLY EXPERIENCE
A REDUCED RISK OF
DATA BREACHES

Organizations farther along their API journeys typically experience a reduced risk of data breaches, better agility to deliver new capabilities to market, improved developer satisfaction, and ultimately, reduced costs when operating their API platform.

The rest of this guide explores some of the best practices your platform teams can put into place to start managing continuous API sprawl in your organization. Each section concludes with a short tutorial demonstrating how to quickly implement each best practice with tools from NGINX.

PREREQUISITE: INSTALL AND CONFIGURE API CONNECTIVITY MANAGER

If you plan to do the tutorials in this eBook, you need to install and configure [API Connectivity Manager](#), part of [F5 NGINX Management Suite](#).

1. Obtain a trial or paid subscription for your APIs. Start a [free 30-day trial of the NGINX API Connectivity Stack](#) to get started.
2. Install [NGINX Management Suite](#). In the [Install Management Suite Modules](#) section, follow the instructions for API Connectivity Manager (and optionally other modules).
3. [Add the license](#) for each installed module.
4. (Optional.) [Set up TLS termination and mTLS](#) to secure client connections to NGINX Management Suite and traffic between API Connectivity Manager and NGINX Plus instances on the data plane, respectively.

APIs CAN PROLIFERATE
FASTER THAN PLATFORM OPS
TEAMS CAN MANAGE THEM

1. Create a Single Source of Truth for Your APIs

As the number of APIs continues to grow, the complexity of managing your API portfolio increases. It gets harder to discover and track what APIs are available and where they are located, as well as find documentation about how to use them. Without a holistic API strategy in place, APIs can proliferate faster than Platform Ops teams can manage them.

Ultimately, APIs can't be useful until they are used – which means API consumers need a way to find them. Without the proper systems in place, API sprawl makes it difficult for developers to find the APIs they need for their applications. At worst, lists of APIs are kept by different lines of business and knowledge is shared across teams only through informal networks of engineers.

BUILD AN INVENTORY OF YOUR APIs

One of the first steps toward fighting API sprawl is creating a single source of truth for your APIs. That process starts with building an inventory of your APIs. An accurate inventory is a challenge, though – it's a constantly moving target as new APIs are introduced and old ones are deprecated. You also need to find any "shadow APIs" across your environments – APIs that have been forgotten over time, were improperly deprecated, or were built outside of your standard processes.

UNMANAGED APIs ARE ONE
OF THE MOST INSIDIOUS
SYMPTOMS OF API SPRAWL

Unmanaged APIs are one of the most insidious symptoms of API sprawl, with both obvious security implications and hidden costs. Without an accurate inventory of available APIs, your API teams must spend time hunting down documentation. There's significant risk of wasteful, duplicated effort as various teams build similar functionalities. And without proper version control, changes to a given API can lead to costly cascades of rework or even outages.

Techniques like automated API discovery can help you identify and treat the symptom of unmanaged APIs. But to solve the problem, you need to eliminate the root causes: broken processes and lack of ownership. In practice, integrating API inventory and documentation into your CI/CD pipelines is the only approach that ensures visibility across your API portfolio in the long term. Instead of having to manually track every API as it comes online, you only need to identify and remediate exceptions.

STREAMLINE API DISCOVERY WITH AN API DEVELOPER PORTAL

Streamlining API discovery is one area where an API developer portal can help. It provides a central location for API consumers to discover APIs, read documentation, and try out APIs before integrating them into their applications. Your API developer portal can also serve as the central API catalog, complete with ownership and contact info, so everyone knows who is responsible for maintaining APIs for different services.

A core component of our [API reference architecture](#), an effective API developer portal enables a few key use cases:

- **Streamline API discovery** – Publish your APIs in an accessible location so developers can easily find and use your APIs in their projects.
- **Provide clear, up-to-date documentation** – Ensure developers always have access to the most up-to-date documentation about how an API functions.
- **Ensure proper versioning** – Introduce new versions of an API without creating outages for downstream applications, with support for future versioning.
- **Generate API credentials** – Streamline the onboarding process so developers can sign in and generate credentials to use for API access.
- **Try out APIs** – Enable developers to try out APIs on the portal before they integrate them into their projects.

YOU NEED TO FIGURE OUT
HOW TO MAINTAIN YOUR API
DEVELOPER PORTAL

As part of your API strategy, you need to figure out how to maintain your API developer portal. It's crucial to have an automated, low-touch approach that seamlessly integrates publishing, versioning, and documenting APIs without creating more work for your API teams.

TUTORIAL: CREATE A DEVELOPER PORTAL WITH NGINX

To enable seamless API discovery, you need to create a single source of truth where developers can find your APIs, learn how to use them, and onboard them into their projects. That means you'll need a developer portal – and up-to-date documentation.

[API Connectivity Manager](#), part of [F5 NGINX Management Suite](#), helps you integrate publication, versioning, and documentation of APIs directly into your development workflows, so your API developer portal is never out of date. API Connectivity Manager not only makes it easy to create API developer portals to host your APIs and documentation, it lets you add custom pages and completely customize the developer portal to match your branding.

Let's look at how API Connectivity Manager helps you address some specific use cases. Refer to the API Connectivity Manager documentation for detailed instructions about [setting up a developer portal cluster](#) and [publishing a developer portal](#).

Automatically Generate API Documentation

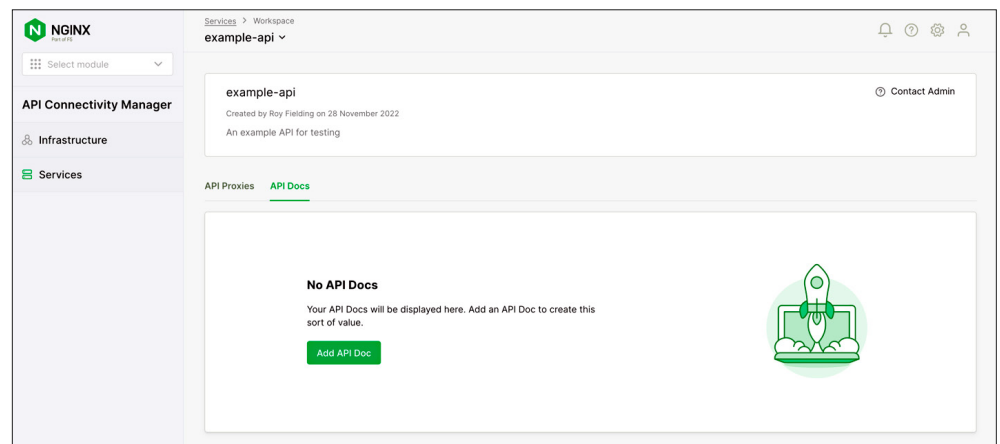
There is often a wide gulf between the level of quality and detail your API consumers expect from documentation and what your busy API developers can realistically deliver with limited time and resources. Many homegrown documentation tools fail to integrate with the development lifecycle or other engineering systems. This doesn't have to be the case.

How NGINX can help: API Connectivity Manager uses the [OpenAPI Specification](#) to publish APIs to the API gateway and automatically generate the accompanying documentation on the developer portal, saving API developers time and ensuring API consumers can always find what they need. You can upload OpenAPI Specification files directly via the API Connectivity Manager user interface, or by sending a call via the REST API. This makes it easy to automate the documentation process via your CI/CD pipeline.

To publish documentation in the API Connectivity Manager user interface, click **Services** in the left navigation column to open the **Services** tab as shown in Figure 1. Click the name of your Workspace or [create a new one](#).

Once you are in the Workspace, click **API Docs** below the box that has the name and description of your Workspace (**example-api** in the screenshot). Simply click the **Add API Doc** button to upload your OpenAPI Specification file. Click the **Save** button to publish the documentation to the developer portal.

Figure 3: Creating Documentation by Uploading an OpenAPI Specification File to API Connectivity Manager



VERSION CHANGES MUST
ALWAYS BE HANDLED
WITH CARE

Ensure Proper Versioning

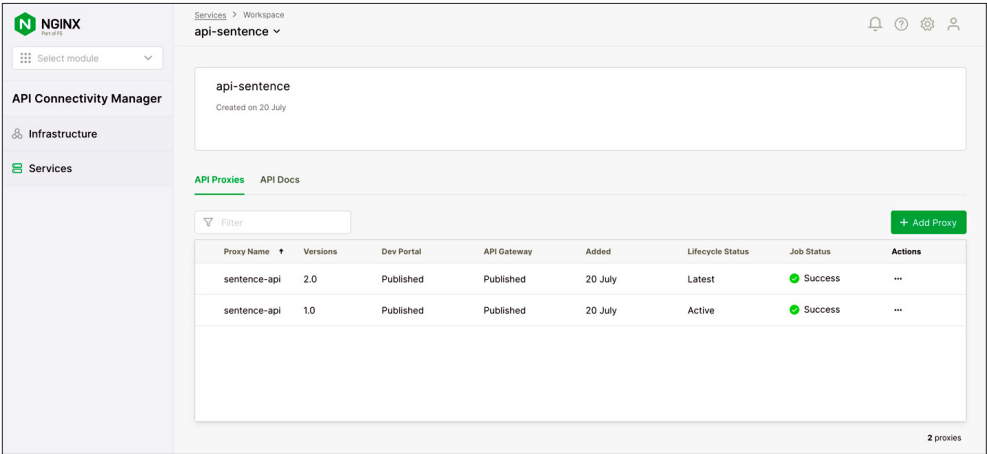
Version changes must always be handled with care, and this is especially true in microservices environments where many services might be interacting with a single API. Without a careful approach to introducing new versions and retiring old ones, a single breaking change can lead to a cascading outage across dozens of microservices.

How NGINX can help: Using OpenAPI Specification files with API Connectivity Manager enables easy version control for your APIs. In addition to setting the version number, you can provide documentation for each version and manage its status (latest, active, retired, or deprecated).

To publish a new version of an API, click **Services** in the left navigation column. Click the name of your Workspace in the table, then click the name of your Environment on the page that opens. Next, click the **+ Add Proxy** button. From here you can upload the OpenAPI Specification, set the base path and version to create the URI (for example, `/api/v2/`), and input other important metadata. Click the **Publish** button to save and publish your API proxy.

The original version of the API remains available alongside your new version. This gives your users time to gradually migrate their applications or services to the most recent version. When you are ready, you can fully deprecate the original version of your API. Figure 4 shows two versions of the Sentence Generator API: published and in production.

Figure 4: Creating Documentation by Uploading an OpenAPI Specification File to API Connectivity Manager



TO DRIVE ADOPTION OF YOUR APIs, YOU NEED TO MAKE THE ONBOARDING PROCESS AS SIMPLE AS POSSIBLE

Generate API Credentials

To drive adoption of your APIs, you need to make the onboarding process as simple as possible for your API consumers. Once users find their APIs, they need a method to securely sign in to the developer portal and generate the credentials that grant them access to your API. Most often, you'll want to implement a self-managed workflow so users can sign up on their own.

How NGINX can help: API Connectivity Manager supports self-managed API workflows on the developer portal so users can generate their own credentials for accessing APIs. Credentials can be managed on the portal using [API keys](#) or HTTP [Basic authentication](#). You can also enable single sign-on (SSO) on the developer portal to secure access and allow authenticated API consumers to manage credentials.

To quickly enable SSO on the developer portal, click **Infrastructure** in the left navigation column. Click the name of your Workspace in the table (in Figure 5, it's **team-sentence**).

Figure 5: List of Workspaces on the Infrastructure Tab

Workspace N...	Description	Environments	Owner	Contact Info	Last Updated	Date Created	Actions
team-sentence	-	1	-	--	20 July	20 July	...

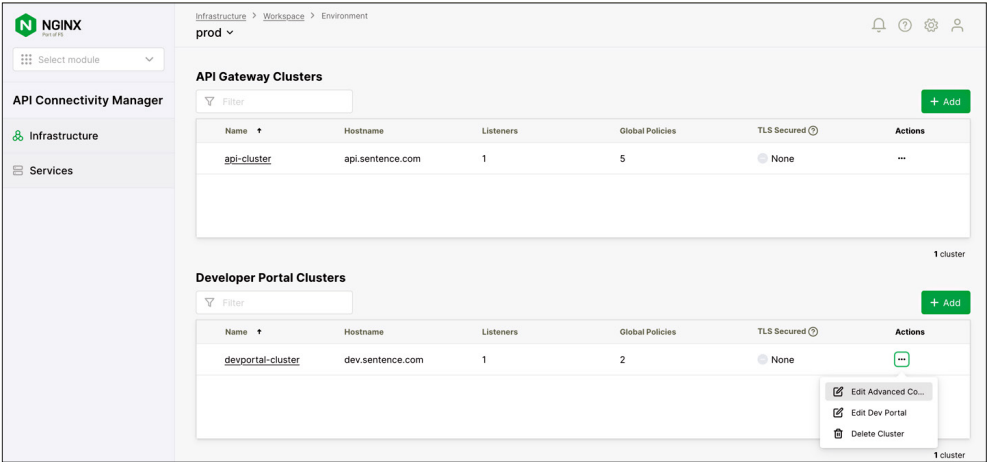
In the table on the Workspace page, click the name of the Environment you want to configure (in Figure 6, it's **prod**).

Figure 6: List of Environments in a Workspace

Name	Description	API Gateway (# of)	Dev Portal (# of)	Last Updated	Date Created	Job Status	Actions
prod	-	1	1	20 July	20 July	Success	...

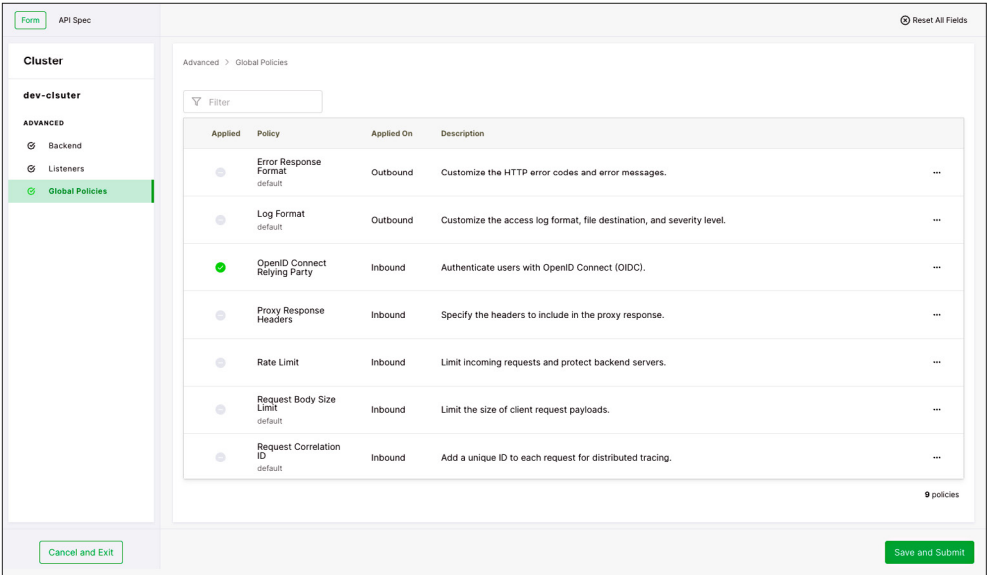
In the **Developer Portal Clusters** section, click the ... icon in the **Actions** column for the developer portal you are working on and select **Edit Advanced Config** from the drop-down menu. In Figure 7, the single Developer Portal Cluster is **devportal-cluster**.

Figure 7: Selecting Edit Advanced Config Option for a Developer Portal Cluster



Next, click **Global Policies** in the left navigation column (as shown in Figure 8). Configure the **OpenID Connect Relying Party** policy by clicking on the ... icon in the rightmost column of its row and selecting **Add Policy** from the drop-down menu. For more information, see the [API Connectivity Manager documentation](#).

Figure 8: Configuring the OpenID Connect Relying Party Global Policy to Enable Single Sign-On



Try Out APIs on the Developer Portal

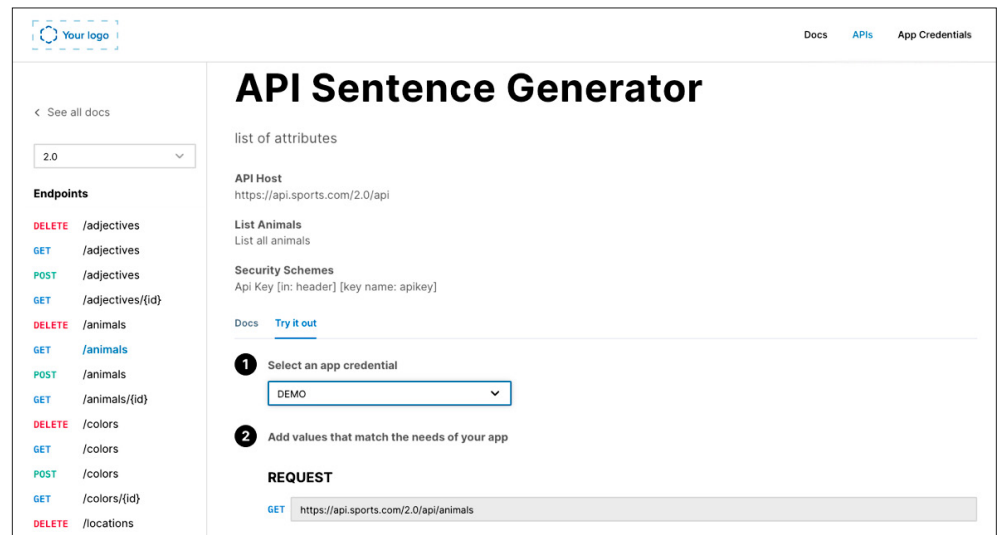
One way you might measure the success of your API strategy is to track the “time to first API call” metric, which reveals how long it takes a developer to send a basic request with your API.

We’ve established that clear, concise documentation is essential as the first entry point for your API. It’s where your users get a basic understanding of how to work with an API. Usually, developers must then write new code to integrate the API into their application before they can test API requests. You can help developers get started much faster by providing a way to directly interact with an API on the developer portal using real data – effectively making their first API call without writing a single line of new code in their application!

How NGINX can help: Once you enable SSO for your API Connectivity Manager developer portals, API consumers can use the API Explorer to try out API calls on your documentation pages. They can use API Explorer to explore the API’s endpoints, parameters, responses, and data models, and test API calls directly with their browsers.

Figure 9 shows the API Explorer in action – in this case, trying out the API Sentence Generator. The user selects the appropriate credentials, creates the request, and receives a response with actual data from the API.

Figure 9: Testing an API on the Developer Portal



ENSURING THE RELIABILITY
AND SECURITY OF THESE
COMPLEX, MULTI-CLOUD
ARCHITECTURES IS A
MAJOR CHALLENGE

2. Put an API Governance Plan in Place

Today's enterprise is often made up of globally distributed teams building and deploying APIs and microservices, usually across more than one deployment environment. According to F5's [State of Application Strategy in 2022](#) report, 81% of organizations operate across three or more environments ranging across public cloud, private cloud, on-premises, and edge.

Ensuring the reliability and security of these complex, multi-cloud architectures is a major challenge. According to software engineering leaders surveyed in the F5 report, visibility (45%) and consistent security (44%) top the list of multi-cloud challenges faced by [Platform Ops](#) teams.

With the growing number of APIs and microservices today, API governance is quickly becoming one of the most important topics for planning and implementing an enterprise-wide API strategy. But what is API governance, and why is it so important for your API strategy?

THE IMPORTANCE OF API GOVERNANCE

At the most basic level, API governance involves creating policies and running checks and validations to ensure APIs are discoverable, reliable, observable, and secure. It provides visibility into the state of the complex systems and business processes powering your modern applications, which you can use to guide the evolution of your API infrastructure over time.

Why Do You Need API Governance?

The strategic importance of API governance cannot be overestimated – it's the means by which you realize your organization's overall API strategy. Without proper governance, you can never achieve consistency across the design, operation, and productization of your APIs.

When done poorly, governance imposes burdensome requirements that slow teams down. When done well, however, API governance reduces work, streamlines approvals, and allows different teams in your organization to function independently while delivering on the overall goals of your API strategy.

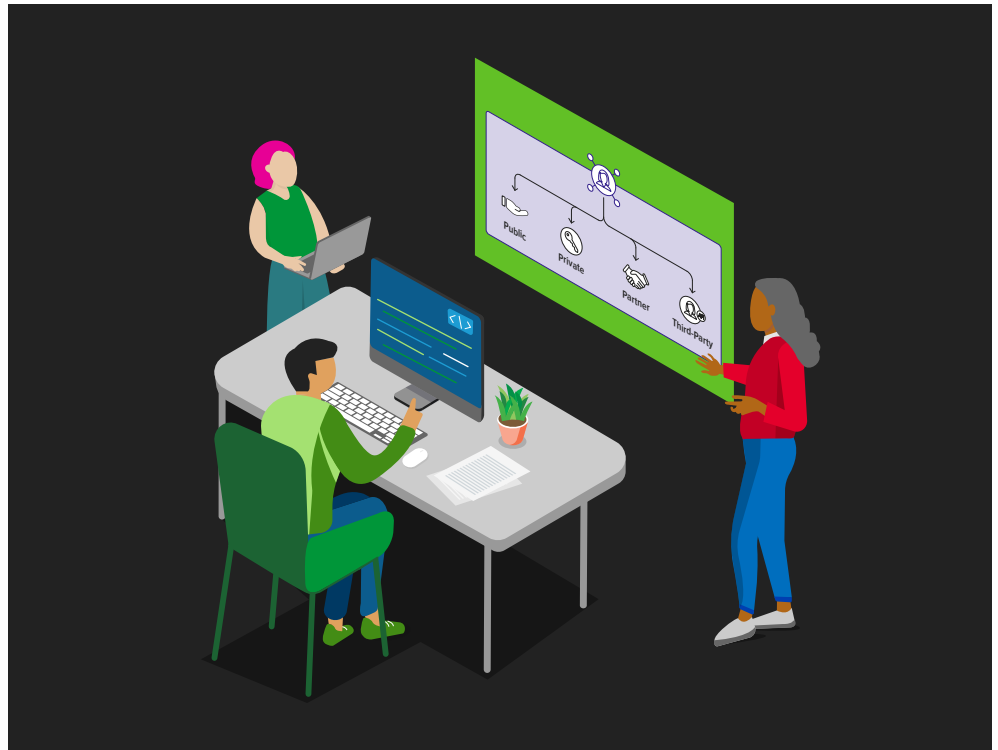
BUILDING AN EFFECTIVE API
GOVERNANCE PLAN STARTS
WITH IDENTIFYING THE
TYPES OF APIs YOU HAVE

What Types of APIs Do You Need to Govern?

Building an effective API governance plan starts with identifying the types of APIs you have in production, and the tools, policies, and guidance you need to manage them. Today, most enterprise teams are working with four primary types of APIs:

- **Public APIs** – Delivered to external consumers and developers to enable self-service integrations with data and capabilities
- **Private APIs** – Used for connecting internal applications and microservices and only available to your organization’s developers
- **Partner APIs** – Facilitate strategic business relationships by sharing access to your data or applications with developers from partner organizations
- **Third-Party APIs** – Consumed from third-party vendors as a service, for example for handling payments or enabling access to data or applications

Each type of API in the enterprise must be governed to ensure it is secure, reliable, and accessible to the teams and users who need to access it.



THERE ARE MANY WAYS
TO DEFINE AND APPLY
API GOVERNANCE

COMMON API GOVERNANCE MODELS

There are many ways to define and apply API governance. At NGINX, we typically see customers applying one of two models:

- **Centralized** – A central team reviews and approves changes; depending on the scale of operations, this team can become a bottleneck that slows progress.
- **Decentralized** – Individual teams have autonomy to build and manage APIs; this increases time to market but sacrifices overall security and reliability.

As companies progress in their **API-first** journeys, however, both models start to break down as the number of APIs in production grows. Centralized models often try to implement a one-size-fits-all approach that requires various reviews and sign-offs along the way. This slows down development teams and creates friction – and, in their frustration, developers sometimes even find ways to work around the requirements (the dreaded “shadow IT”).

The other model, decentralized governance, works well for API developers at first, but over time complexity increases. Unless the different teams deploying APIs communicate frequently, the overall experience becomes inconsistent across APIs. Each API is designed and functions differently, version changes result in outages between services, and security is enforced inconsistently across teams and services. For the teams building APIs, the additional work and complexity eventually slows development to a crawl, just like the centralized model.

Cloud-native applications rely on APIs for the individual microservices to communicate with each other, and to deliver responses back to the source of the request. As companies continue to embrace microservices for their flexibility and agility, **API sprawl will not be going away**. Instead, you need a different approach to governing APIs in these complex, constantly changing environments.

API SPRAWL WILL NOT
BE GOING AWAY

ADAPTIVE GOVERNANCE
OFFERS AN ALTERNATIVE
MODEL THAT EMPOWERS
API DEVELOPERS

IMPLEMENT ADAPTIVE GOVERNANCE TO EMPOWER DEVELOPERS

Fortunately, there is a better way. Adaptive governance offers an alternative model that empowers API developers while giving Platform Ops teams the control they need to ensure the reliability and security of APIs across the enterprise.

At the heart of adaptive governance is balancing control (the need for consistency) with autonomy (the ability to make local decisions) to enable agility across the enterprise. In practice, the adaptive governance model unbundles and distributes decision making across teams.

Platform Ops teams manage shared infrastructure (API gateways and developer portals) and set global policies to ensure consistency across APIs. Teams building APIs, however, act as the subject matter experts for their services or line of business. They are empowered to set and apply local policies for their APIs – role-based access control (RBAC), rate limiting for their service, etc. – to meet requirements for their individual business contexts.

Adaptive governance allows each team or line of business to define its workflows and balance the level of control required, while using the organization's shared infrastructure.

TUTORIAL: GOVERN YOUR APIs WITH NGINX

FOLLOW THESE BEST
PRACTICES TO IMPLEMENT
ADAPTIVE GOVERNANCE
IN YOUR ORGANIZATION

As you start to plan and implement your API strategy, follow these best practices to implement adaptive governance in your organization:

- **Provide shared infrastructure** – Provide teams with access to API gateways and developer portals for publishing API proxies and documentation.
- **Give teams agency** – Help teams onboard and manage the lifecycle of their APIs within a shared workspace.
- **Balance global policies with local control** – Set global policies across shared infrastructure while giving teams granular controls for their services.

Let's look at how you can accomplish these use cases with API Connectivity Manager, part of F5 NGINX Management Suite.

PLATFORM OPS TEAMS
CAN PROVIDE ACCESS TO
SHARED INFRASTRUCTURE

Provide Shared Infrastructure

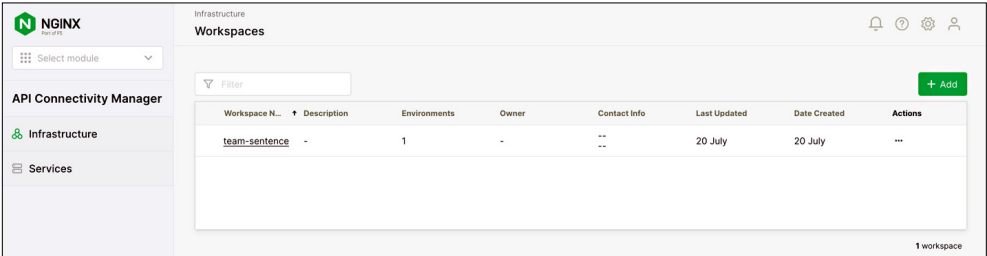
Teams across your organization are building APIs, and they need to include similar functionality in their microservices: authentication and authorization, mTLS encryption, and more. They also need to make documentation and versioning available to their API consumers, be those internal teams, business partners, or external developers.

Rather than requiring teams to build their own solutions, Platform Ops teams can provide access to shared infrastructure. As with all actions in API Connectivity Manager, you can set this up in just a few minutes using either the UI or the fully declarative REST API, which enables you to integrate API Connectivity Manager into your CI/CD pipelines. In this chapter we use the UI to illustrate some common workflows.

API Connectivity Manager supports two types of Workspaces: infrastructure and services. Infrastructure Workspaces are used by Platform Ops teams to onboard and manage shared infrastructure in the form of API Gateway Clusters and Developer Portal Clusters. Services Workspaces are used by API developers to publish and manage APIs and documentation.

To set up a shared infrastructure, first add an infrastructure Workspace. Click **Infrastructure** in the left navigation column and then the **+ Add** button in the upper right corner of the tab. Give your Workspace a name (in Figure 10 it's **team-sentence** – an imaginary team building a simple “Hello, World!” API).

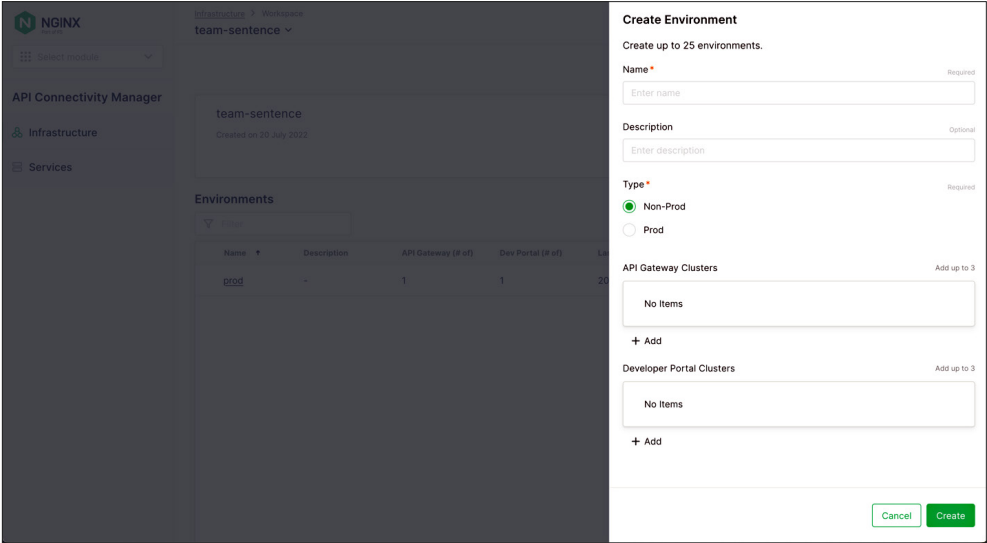
Figure 10: Add Infrastructure Workspaces



Next, add an Environment to the Workspace. Environments contain API Gateway Clusters and Developer Portal Clusters. Click the name of your Workspace and then the ... icon in the **Actions** column. Select **Add** from the drop-down menu.

The **Create Environment** drawer opens, as shown in Figure 11. Fill in the **Name** (and optionally, **Description**) field, select the **Type** of Environment (**Non-Prod** or **Prod**), and click + **Add** for the infrastructure you want to add (API Gateway Clusters, Developer Portal Clusters, or both). Click the **Create** button to finish setting up your Environment. For complete instructions, see the [API Connectivity Manager documentation](#).

Figure 11: Create an Environment and Onboard Infrastructure



Give Teams Agency

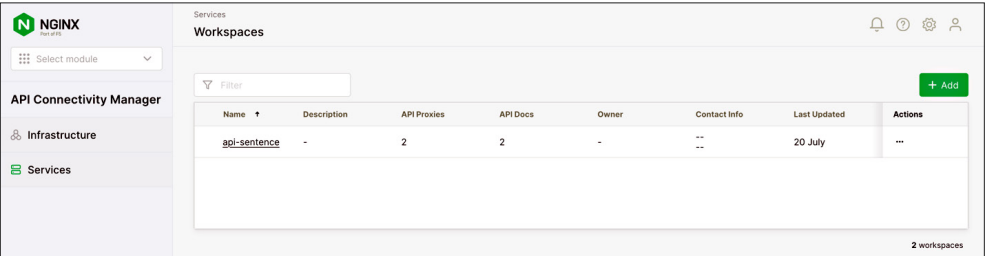
TEAMS NEED ACCESS TO
SHARED INFRASTRUCTURE
WITHOUT HAVING TO WORRY
ABOUT ACTIVITIES AT
THE GLOBAL LEVEL

It makes sense to provide clear separation for teams by line of business, geographic region, or another logical boundary – if that doesn't deprive them of access to the tools they need to succeed. Teams need access to shared infrastructure without having to worry about activities at the global level. Instead, you want them to focus on defining their own requirements, charting a roadmap, and building their microservices.

To help Platform Ops can provide Workspaces where teams can organize and operate their services and documentation. Workspaces create logical boundaries and provide access to different environments (e.g., development, testing, and production) that are used while developing services. The process is similar to creating the infrastructure Workspace we made in the [previous section](#).

Figure 12: Create a Services Workspace

First, click **Services** in the left navigation column and then the **+ Add** button in the upper right corner of the tab. Give your Workspace a name (in Figure 12, **api-sentence** for the “Hello, World” service) and optionally provide a description and contact information.



At this point, you can invite API developers to start publishing proxies and documentation in the Workspace you’ve created for them. For complete instructions on publishing API proxies and documentation, see the [API Connectivity Manager documentation](#).

Balance Global Policies and Local Control

ADAPTIVE GOVERNANCE
REQUIRES A BALANCE
BETWEEN ENFORCING GLOBAL
POLICIES AND EMPOWERING
TEAMS TO MAKE DECISIONS
THAT BOOST AGILITY

Adaptive governance requires a balance between enforcing global policies and empowering teams to make decisions that boost agility. You need to establish a clear separation of responsibilities by defining the global settings enforced by Platform Ops and setting “guardrails” that define the tools API developers use and the decisions they can make.

API Connectivity Manager provides a mix of global policies (applied to shared infrastructure) and granular controls managed at the API proxy level.

Global policies available in API Connectivity Manager include:

- **Error Response Format** – Customize the API gateway’s error code and response structure
- **Log Format** – Enable access logging and customize the format of log entries
- **OpenID Connect** – Secure access to APIs with an OpenID Connect policy
- **Response Headers** – Include or exclude headers in the response
- **Request Body Size** – Limit the size of incoming API payloads
- **Inbound TLS** – Set the policy for TLS connections with API clients
- **Backend TLS** – Secure the connection to backend services with TLS

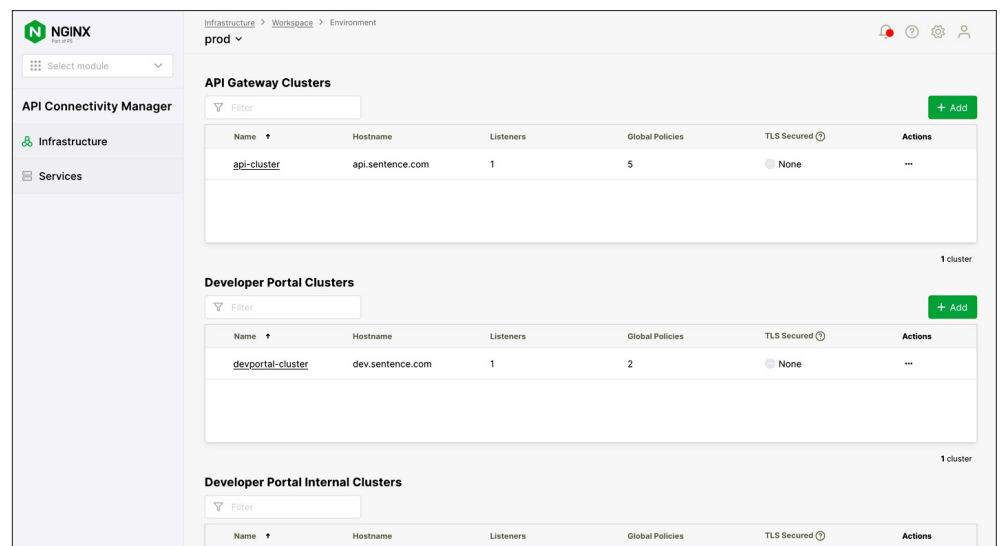
API proxy policies available in API Connectivity Manager include:

- **Allowed HTTP Methods** – Define which request methods (GET, POST, PUT, etc.) can be used
- **Access Control** – Secure access to APIs using different authentication and authorization techniques (API keys, HTTP Basic authentication, JSON Web Tokens)
- **Backend Health Checks** – Run continuous health checks to avoid failed requests to backend services
- **CORS** – Enable controlled access to resources by clients from external domains
- **Caching** – Improve API proxy performance with caching policies
- **Proxy Request Headers** – Pass select headers to backend services
- **Rate Limiting** – Limit incoming requests and secure API workloads

In the following example, we use the UI to define a policy that secures communication between an API Gateway Proxy and backend services.

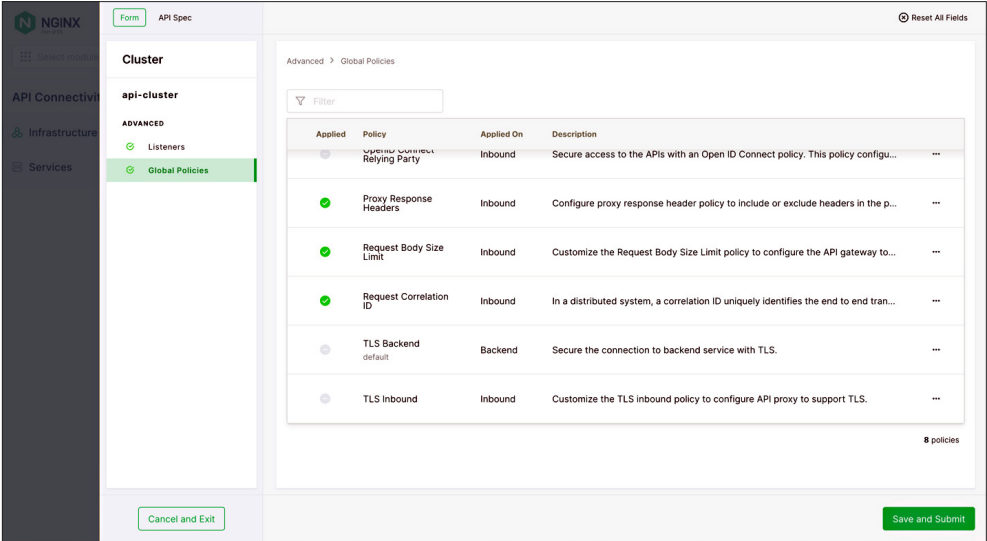
Click **Infrastructure** in the left navigation column. After you click the name of the Environment containing the API Gateway Cluster you want to edit, the tab displays the API Gateway Clusters and Developer Portal Clusters in that Environment, as shown in Figure 13.

Figure 13: Configure Global Policies for API Gateway Clusters and Developer Portal Clusters



In the row for the API Gateway Cluster to which you want to apply a policy, click the ... icon in the Actions column and select Edit Advanced Configuration from the drop-down menu. In the drawer that opens, click Global Policies in the left column to display a list of all the global policies you can configure, as shown in Figure 14.

Figure 14: Configure Policies for an API Gateway Cluster



3. Adopt an API-First Approach to Building Microservices

AS APPLICATIONS GROW AND SCALE, SO DOES THE NUMBER OF MICROSERVICES AND APIs

As applications grow and scale, so does the number of microservices and APIs. While this is an unavoidable outcome in most cases, it creates significant challenges for the **Platform Ops** teams responsible for ensuring the reliability, scalability, and security of modern applications.

As a first attempt to solve **API sprawl**, many organizations try to use a top-down approach by implementing tools for automated API discovery and remediation. While this is effective in the near term, it often imposes an undue burden on the teams responsible for building and operating APIs and microservices. They either must rework existing microservices and APIs to address security and compliance issues or go through an arduous review process to obtain the required approvals.

Rather than putting in last-minute safeguards, a bottom-up approach to the problem is more effective over the long term. The teams building and operating APIs for different microservices and applications are the first to be involved, and often begin by adopting an **API-first** approach to software development in your organization.

WHAT IS API-FIRST?

APIs have been around for decades. But they are no longer simply “application programming interfaces”. At their heart, APIs are developer interfaces. Like any user interface, APIs need planning, design, and testing. API-first is about acknowledging and prioritizing the importance of connectivity and simplicity across all the teams operating and using APIs. It prioritizes communication, reuseability, and functionality for API consumers, who are almost always developers.

A DESIGN-LED APPROACH TO SOFTWARE DEVELOPMENT IS THE END GOAL FOR MOST COMPANIES EMBARKING ON AN API-FIRST JOURNEY

There are many paths to **API-first**, but a design-led approach to software development is the end goal for most companies embarking on an API-first journey. In practice, this approach means APIs are completely defined before implementation. Work begins with designing and documenting how the API will function. The team relies on the resulting artifact, often referred to as the API contract, to inform how they implement the application’s functionality.

THE VALUE OF API-FIRST FOR ORGANIZATIONS

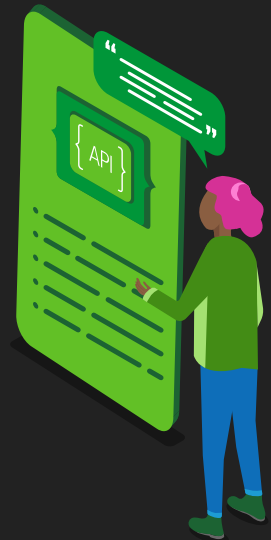
An API-first strategy is often ideal for microservices architectures because it ensures application ecosystems begin life as modular and reusable systems. Adopting an API-first software development model provides significant benefits for both developers and organizations, including:

- **Increased developer productivity** – Development teams can work in parallel, able to update backend applications without impacting the teams working on other microservices which depend on the applications' APIs. Collaboration is often easier across the API lifecycle since every team can refer to the established API contract.
- **Enhanced developer experience** – API-first design prioritizes the developer experience by ensuring that an API is logical and well-documented. This creates a seamless experience for developers when they interact with an API. Learn why it's so important for Platform Ops teams to take the API developer experience into consideration.
- **Consistent governance and security** – Cloud and platform architects can organize the API ecosystem in a consistent way by incorporating security and governance rules during the API design phase. This avoids the costly reviews required when issues are discovered later in the software process.

The Value of API-First for Organizations

Adopting an API-first software development model provides significant benefits for both developers and organizations, including:

- 01 Increased Developer Productivity
- 02 Enhanced Developer Experience
- 03 Consistent Governance and Security
- 04 Improved Software Quality
- 05 Faster Time to Market



- **Improved software quality** – Designing APIs first ensures security and compliance requirements are met early in the development process, well before the API is ready to be deployed to production. With less need to fix security flaws in production, your operations, quality, and security engineering teams have more time to work directly with the development teams to ensure quality and security standards are met in the design phase.
- **Faster time to market** – With fewer dependencies and a consistent framework for interservice communication, different teams can build and improve their services much more efficiently. A consistent, machine-readable API specification is one tool that can help developers and Platform Ops teams to work better together.

Overall, adopting an API-first approach can help a company build a more flexible, scalable, and secure microservices architecture.

THE IMPORTANCE OF ADOPTING A COMMON API SPECIFICATION

THERE ARE MORE COMPONENTS
IN PLAY THAN A PLATFORM
OPS TEAM CAN KEEP TRACK
OF DAY-TO-DAY

In the typical enterprise microservices and API landscape, there are more components in play than a Platform Ops team can keep track of day-to-day. Embracing and adopting a standard, machine-readable API specification helps teams understand, monitor, and make decisions about the APIs currently operating in their environments.

Adopting a common API specification can also help improve collaboration with stakeholders during the API design phase. By producing an API contract and formalizing it into a standard specification, you can ensure that all stakeholders are on the same page about how an API works. It also makes it easier to share reusable definitions and capabilities across teams.

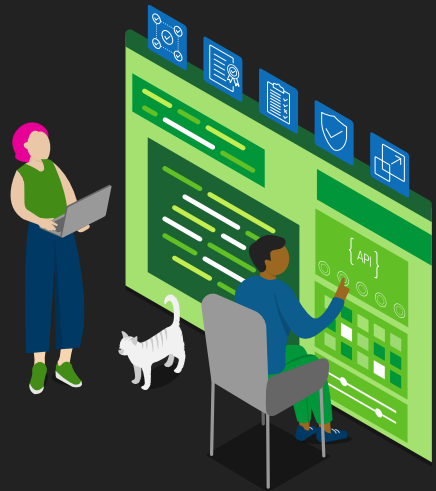
Using a common API specification has several benefits:

- **Improved interoperability** – A common, machine-readable specification means different systems and clients can consume and use the API contract. This makes it easier for Platform Ops teams to integrate, manage, and monitor complex architectures.
- **Consistent documentation** – The API contract is documented in a standard format, including the endpoints, request and response formats, and other relevant details. Many systems can use the contract to generate comprehensive documentation, providing clarity and making it easier for developers to understand how to use the API.

The Importance of Adopting a Common API Specification

Using a common API specification has several benefits:

- 01 Improved Interoperability
- 02 Consistent Documentation
- 03 Better Testing
- 04 Improved Security
- 05 Easier Evolution



- **Better testing** – API specifications can be used to automatically generate and run tests, which can help ensure the API implementation adheres to the contract and is working as expected. This can help identify issues with an API before it is published to production.
- **Improved security** – Advanced security tools can use the OpenAPI Specification to analyze API traffic and user behavior. They can **apply positive security** by verifying that API requests comply with the methods, endpoints, parameters supported by the API endpoint supports. Non-conforming traffic is blocked by default, reducing the number of calls your microservices have to process.
- **Easier evolution** – API specifications can help facilitate the evolution of the API contract and application itself over time by providing a clear and standard way to document and communicate changes in both machine- and human-readable format. When coupled with proper versioning practices, this helps minimize the impacts of API changes on API consumers and ensures that an API remains backward compatible.

THE OpenAPI SPECIFICATION

Today there are three common API specifications, each supporting most types of APIs:

- **OpenAPI** – JSON or YAML descriptions of all web APIs and webhooks
- **AsyncAPI** – JSON or YAML descriptions of event-driven APIs
- **JSON Schema** – JSON descriptions of the schema objects used for APIs

REST APIs make up the bulk of APIs in production today, and the OpenAPI Specification is the standard way to write the API definition for a REST API. It provides a machine-readable contract that describes how a given API functions. The OpenAPI Specification is widely supported by a variety of API management and API gateway tools, including NGINX. The rest of this chapter focuses on how you can use the OpenAPI Specification to accomplish a few important use cases.

The OpenAPI Specification is an open-source format for defining APIs in either JSON or YAML. You can include a wide range of API characteristics, as illustrated by the following API example – a simple HTTP GET request that returns a list of items on an imaginary grocery list.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Grocery List API
  description: An example API to illustrate the OpenAPI Specification

servers:
  - url: https://api.example.io/v1

paths:
  /list:
    get:
      description: Returns a list of stuff on your grocery list
      responses:
        '200':
          description: Successfully returned a list
          content:
            schema:
              type: array
              items:
                type: object
                properties:
                  item_name:
                    type: string
```

Definitions that follow the OpenAPI Specification are both machine- and human-readable. This means there is a single source of truth that documents how each API functions, which is especially important in organizations with many teams building and operating APIs. Of course, to manage, govern, and secure APIs at scale you need to make sure that the rest of the tools in your API platform – API gateways, developer portals, and advanced security – also support the OpenAPI Specification.

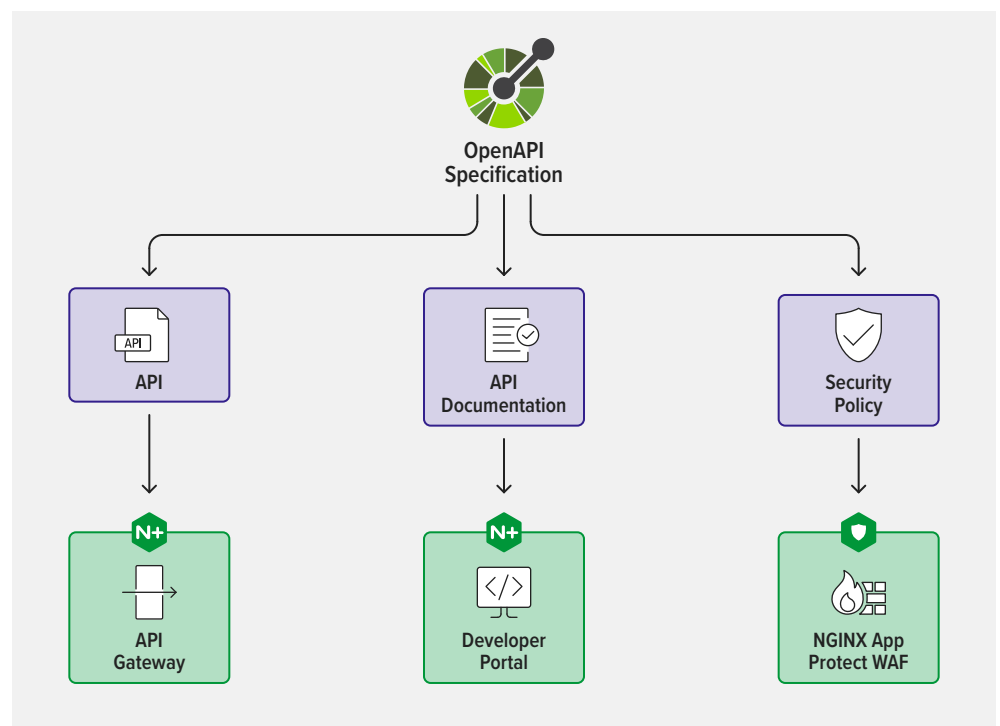
HOW TO USE NGINX FOR API-FIRST SOFTWARE DEVELOPMENT

NGINX provides a set of lightweight, cloud-native tools that makes it easy to operate, monitor, govern, and secure APIs at scale. For example, [API Connectivity Manager](#), part of [F5 NGINX Management Suite](#), provides a single management plane for your API operations. With it you can configure and manage API gateways and developer portals. As an API-first tool itself, every function is accessible via REST API, making CI/CD automation and integration easy for [DevOps](#) teams.

Using the OpenAPI Specification, you can use NGINX products to:

- [Publish APIs to the API gateway](#)
- [Generate API documentation](#) for the developer portal
- [Apply positive security](#) to protect API endpoints

Figure 15: Use the OpenAPI Specification to Publish an API to the API Gateway, Documentation to the Developer Portal, and to Set Security Policies for the WAF via CI/CD Pipelines or the User Interface



Publish APIs to the API Gateway

API Connectivity Manager uses the OpenAPI Specification to streamline API publication and management. API developers can publish APIs to the API gateway using either the user interface or the fully declarative REST API. APIs are added to the gateway as API proxies, which contain all the ingress, backend, and routing configurations the API gateway needs to direct incoming API requests to the backend microservice. You can use the REST API to deploy and manage APIs as code by creating simple CI/CD automation scripts with tools like [Ansible](#).

For a full tutorial on using the OpenAPI Specification to publish an API, see the [API Connectivity Manager documentation](#).

Generate API Documentation for the Developer Portal

Maintaining documentation is often a headache for API teams. But out-of-date documentation on developer portals is also a major symptom of API sprawl. API Connectivity Manager uses the OpenAPI Specification to automatically generate documentation and publish it to the developer portal, saving API developers time and ensuring API consumers can always find what they need. You can upload OpenAPI Specification files directly via the API Connectivity Manager user interface or REST API.

For a full tutorial on publishing API documentation to the developer portal, see the [API Connectivity Manager documentation](#).

Apply Positive Security to Protect API Endpoints

You can also use the OpenAPI Specification to verify that API requests to the NGINX Plus API gateway comply with what an API supports. By applying positive security (a security model that defines what is allowed and blocks everything else), you can prevent malicious requests from probing your backend services for potential vulnerabilities. With API Connectivity Manager, you can upload the OpenAPI Specification and configure [NGINX App Protect WAF](#) to verify and allow requests that conform to the API contract.

For additional information, see the [API Connectivity Manager documentation](#).

77% OF ENTERPRISES
OPERATE APPLICATIONS
ACROSS MULTIPLE CLOUDS

4. Manage APIs Across Multi-Cloud and Hybrid Architectures

According to F5's [State of Application Strategy in 2022](#) report, 77% of enterprises operate applications across multiple clouds. The adoption of multi-cloud and hybrid architectures unlocks important benefits, like improved efficiency, reduced risk of outages, and avoidance of vendor lock-in. But these complex architectures also present unique challenges.

The software and IT leaders surveyed by F5 named these as their top multi-cloud challenges:

- Visibility (45% of respondents)
- Security (44%)
- Migrating apps (41%)
- Optimizing performance (40%)

Most cloud providers offer platform-native API gateways that provide a degree of visibility and security for managing and governing APIs. These tools don't work so well when you need to govern and secure APIs across multiple clouds. Differences in the ways each vendor implements security or governance policies require custom updates for each platform, or even completely different user workflows. They also provide varying degrees of observability.

For most teams operating in multi-cloud and hybrid environments, adopting a consistent data plane across all API gateway instances helps ensure that every API is managed, governed, and secured in a uniform way across the enterprise. This in turn frees teams to deploy and manage applications where it makes the most sense, whether in different data centers or with different cloud providers.

COMMON MULTI-CLOUD AND HYBRID API DEPLOYMENT PATTERNS

You need a multi-cloud API strategy so you can implement a thoughtful approach to unifying your microservices – now distributed across multiple clouds – to ensure end-to-end connectivity. Two of the common scenarios for multi-cloud and hybrid deployments are:

- **Different services in multi-cloud/hybrid environments** – You need to operate different applications and APIs in different locations, perhaps for cost efficiency or because different services are relevant to different groups of users.
- **Same services in multi-cloud/hybrid environments** – You need to ensure high availability for the same applications deployed in different locations.

In the following tutorial, we show step-by-step how to use **API Connectivity Manager**, part of **F5 NGINX Management Suite**, in the second scenario: deploying the same services in multiple environments for high availability. This eliminates single points of failure in your multi-cloud or hybrid production environment – if a gateway instance fails, another gateway instance takes over and your customers don't experience an outage. Services can remain available even if one of the clouds goes down completely.

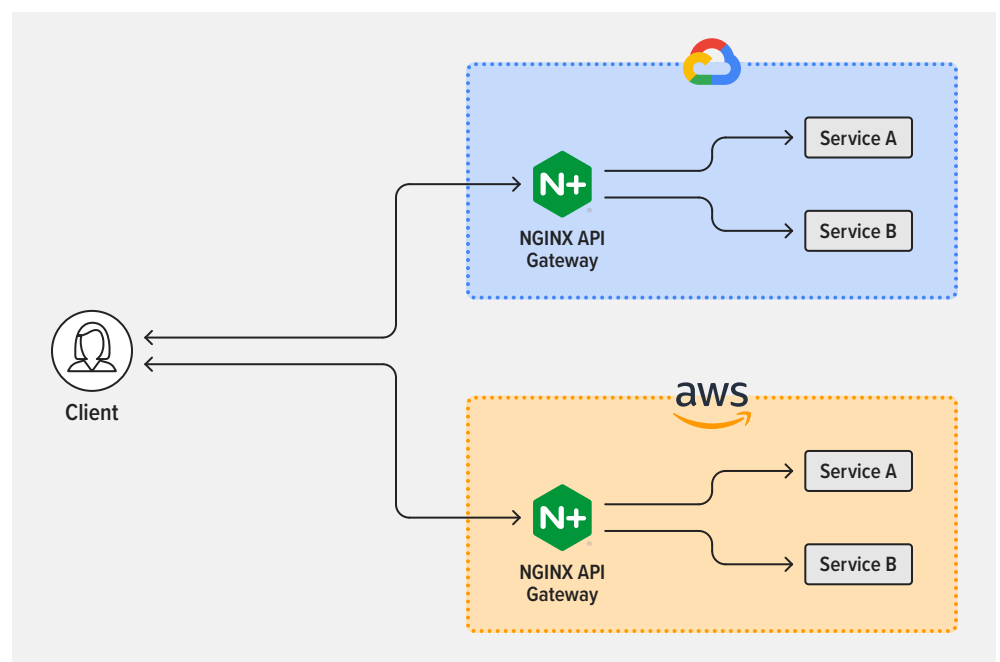
API Connectivity Manager is a cloud-native, platform-agnostic solution for deploying, managing, and securing APIs. From a single pane of glass, you can manage all your API operations for NGINX Plus API gateways and developer portals deployed across public cloud, on-premises, and edge environments. This gives your Platform Ops teams full visibility into API traffic and makes it easy to apply consistent governance and security policies for every environment.

TUTORIAL: ENABLE HIGH AVAILABILITY FOR API GATEWAYS IN MULTI-CLOUD AND HYBRID ENVIRONMENTS

As mentioned in the previous section, in this tutorial we're configuring API Connectivity Manager for high availability of services running in multiple deployment environments. Specifically, we're deploying NGINX Plus as an API gateway routing traffic to two services, Service A and Service B, which are running in two public clouds, Google Cloud Platform (GCP) and Amazon Web Services (AWS). (The setup applies equally to any mix of deployment environments, including Microsoft Azure and on-premises data centers.)

Figure 16 depicts the topology used in the tutorial.

Figure 16: API Connectivity Manager Enables Multi-Cloud Deployment of API Gateways and Services



Follow the steps in these sections to complete the tutorial:

- [Deploy NGINX Plus Instances as API Gateways](#)
- [Set Up an Infrastructure Workspace](#)
- [Create an Environment and API Gateway Clusters](#)
- [Apply Global Policies](#)

Deploy NGINX Plus Instances as API Gateways

Select the environments that make up your multi-cloud or hybrid infrastructure. For the tutorial we've chosen AWS and GCP and are installing one NGINX Plus instance in each cloud. In each environment, perform these steps on each data plane host that will act as an API gateway:

1. [Install NGINX Plus](#) on a [supported operating system](#).
2. [Install the NGINX JavaScript module](#) (njs).
3. Add the following directives in the main (top-level) `_context` in `/etc/nginx/nginx.conf`:

```
load_module modules/ngx_http_js_module.so;  
load_module modules/ngx_stream_js_module.so;
```

4. Restart NGINX Plus, for example by running this command:

```
$ nginx -s reload
```

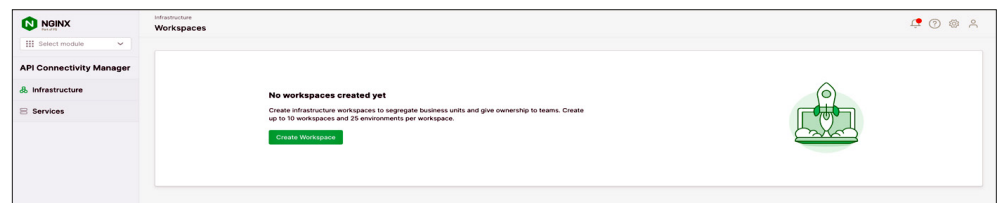
Set Up an Infrastructure Workspace

You can create multiple Infrastructure Workspaces (up to 10 at the time of writing) in API Connectivity Manager. With segregated Workspaces, you can apply policies and authentication/authorization requirements that are specific to different lines of business, teams of developers, external partners, clouds, and so on.

Working in the API Connectivity Manager GUI, create a new Workspace:

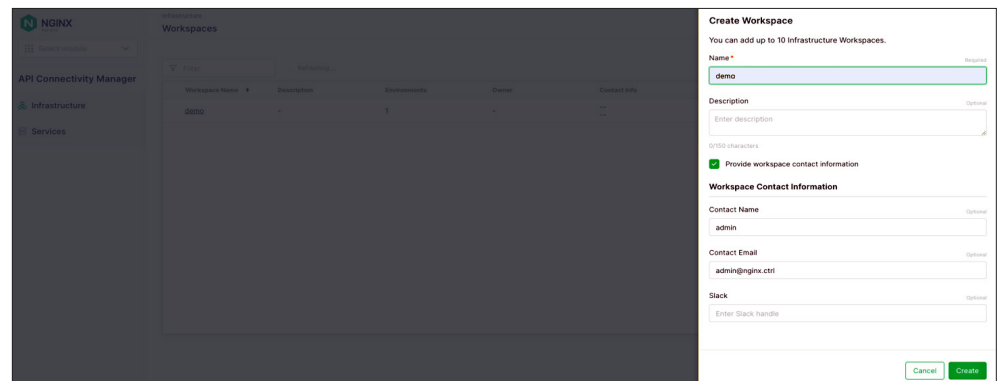
1. Click **Infrastructure** in the left navigation column.
2. Click the **Create Workspace** button to create a new workspace, as shown in Figure 17.

Figure 17: Creating a New Infrastructure Workspace



3. In the **Create Workspace** drawer that opens, fill in the **Name** field (demo in Figure 18). Optionally, fill in the **Description** field and the fields in the **Workspace Contact Information** section. The infrastructure admin (your Platform Ops team, for example) can use the contact information to provide updates about status or issues to the users of the Workspace.

Figure 18: Naming a New Infrastructure Workspace and Adding Contact Information



4. Click the **Create** button.

Create an Environment and API Gateway Clusters

In API Connectivity Manager, an Environment is a logical grouping of dedicated resources (such as API gateways or API developer portals). You can create multiple Environments per Workspace (up to 25 as of this writing). They usually correspond to different stages of app development and deployment such as coding, testing, and production, but can serve any purpose you want.

Within an Environment, an API Gateway Cluster is a logical grouping of NGINX Plus instances acting as API gateways. With an Environment, you can create multiple API gateway clusters with the same hostname (for example, **api.nginx.com**). The NGINX Plus instances in an API Gateway Cluster can also be located in more than one type of infrastructure (e.g., in multiple clouds).

There are two ways to configure an Environment in API Connectivity Manager for active-active high availability of API gateways:

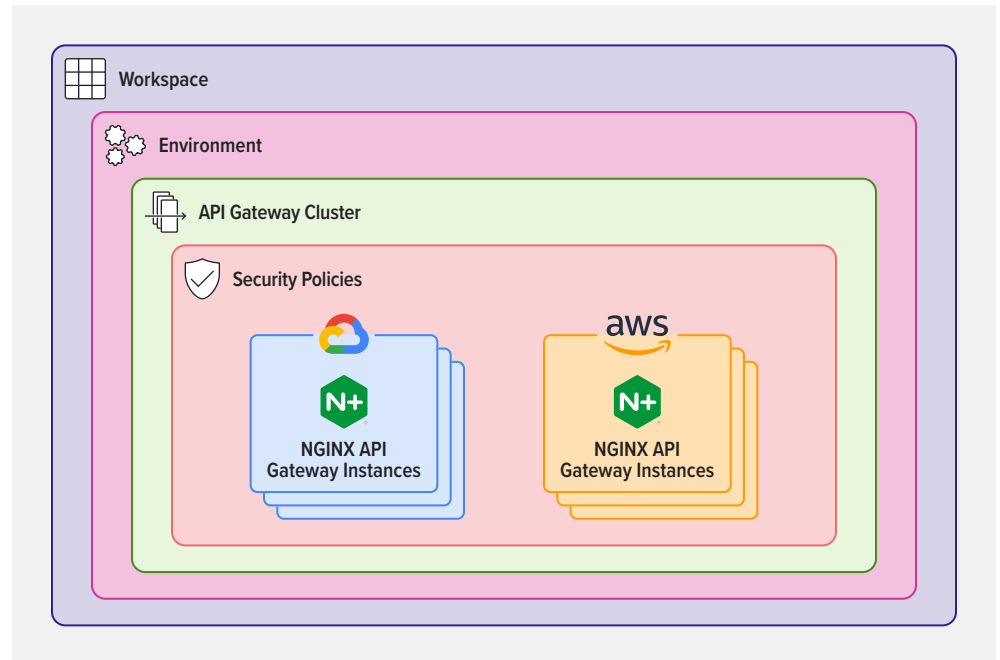
- [With one API Gateway Cluster](#)
- [With multiple API Gateway Clusters](#)

In the previous section, we deployed two NGINX Plus instances – one in AWS and the other in GCP. The tutorial uses the same instances in both types of Environment; in an actual deployment, the more common approach is either to deploy only one type of Environment or create additional NGINX Plus instances for the second Environment.

Deploy an Environment with One API Gateway Cluster

With one API Gateway Cluster, the same security policies apply to all API gateway instances, as shown in Figure 19.

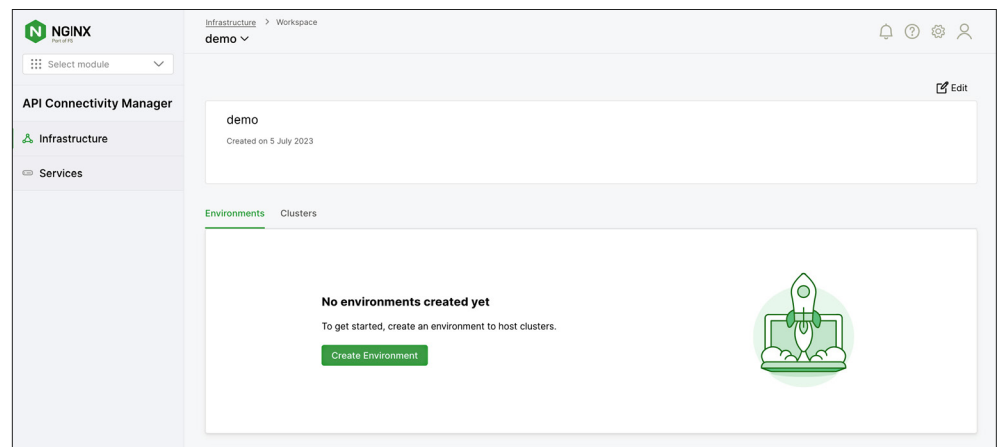
Figure 19: The Same Security Policies Apply to API Gateways Deployed in One API Gateway Cluster



Create an Environment and API Gateway Cluster

1. Navigate to your Workspace and click the **Create Environment** button, as shown in Figure 20.

Figure 20: Creating a New Environment in an Infrastructure Workspace



2. In the **Create Environment** drawer that opens, fill in the **Name** field (**prod** in Figure 21) and optionally the **Description** field, and select the Environment **Type** (here we're choosing Production).

Figure 21: Naming a New Environment and Assigning an API Gateway Cluster to It

The screenshot shows the NGINX API Gateway console interface. On the left, there's a sidebar with 'API Connectivity Manager' and 'demo' selected. The main area shows 'demo' with a 'Created on 14 December 2022' timestamp. A 'Create Environment' button is visible. On the right, the 'Create Environment' drawer is open, showing fields for Name (prod), Description, Type (Prod selected), API Gateway Clusters (api-cluster, api.nginx.com), and Developer Portal Clusters. The 'Create Environment' button is highlighted in green.

3. In the **API Gateway Clusters** section, fill in the **Name** and **Hostname** fields (**api-cluster** and **api.nginx.com** in Figure 21).
4. Click the **Create** button.

The **Environment Created** panel opens to display the command you need to run on each NGINX Plus instance to assign it to the API Gateway Cluster. For convenience, we show the commands in Step 7 below.

Assign API Gateway Instances to an API Gateway Cluster

Repeat on each NGINX Plus instance:

5. Use ssh to connect and log in to the instance.
6. If NGINX Agent is already running, stop it:

```
$ systemctl stop nginx-agent
```

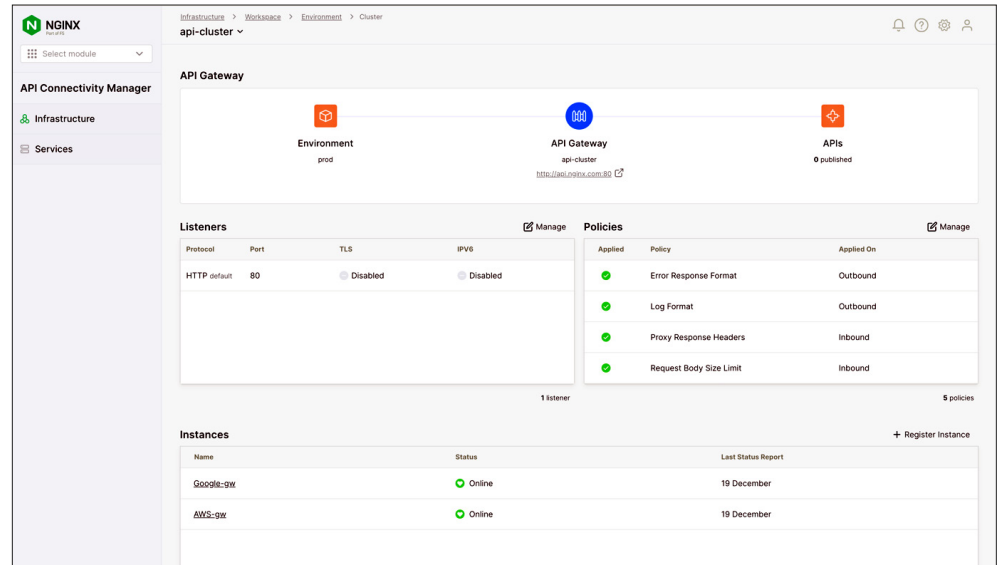
7. Run the command of your choice (either `curl` or `wget`) to download and install the NGINX Agent package:
 - If you didn't enable mTLS in [Install and Configure API Connectivity Manager](#), add:
 - The `-k` flag to the `curl` command
 - The `--no-check-certificate` flag to the `wget` command
 - For `<NMS_FQDN>`, substitute the IP address or fully qualified domain name of your NGINX Management Suite server.
 - For `<cluster_name>`, substitute the name of the API Gateway Cluster (`api-cluster` in this tutorial).

```
$ curl [-k] https://<NMS_FQDN>/install/nginx-agent > install.sh  
&& sudo sh -install.sh -g <cluster_name> && sudo systemctl  
start nginx-agent
```

```
$ wget [--no-check-certificate] https://<NMS_FQDN>/install/  
nginx-agent --no-check-certificate -O install.sh && sudo  
sh install.sh -g <clusterName> && sudo systemctl start  
nginx-agent
```


The NGINX Plus instances now appear in the **Instances** section of the **Cluster** window for **api-cluster**, as shown in Figure 22.

Figure 22: A Single API Gateway Cluster Groups NGINX Plus Instances Deployed in Multiple Clouds

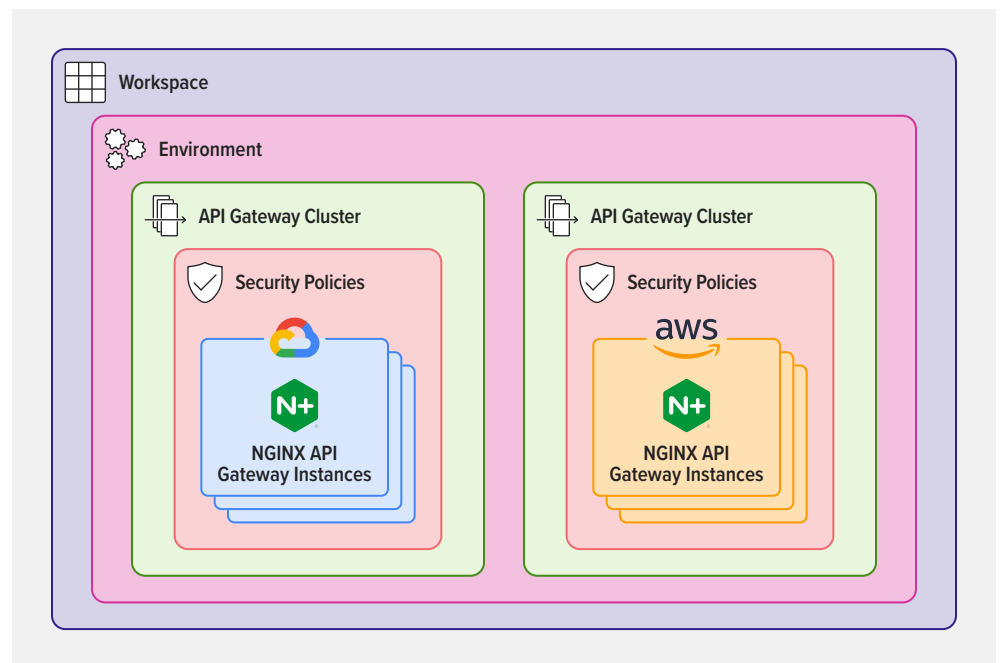


8. Proceed to [Apply Global Policies](#).

Deploy an Environment with Multiple API Gateway Clusters

With multiple API Gateway Clusters, we can apply different security policies to the API gateway instances in different clouds, as shown in Figure 23.

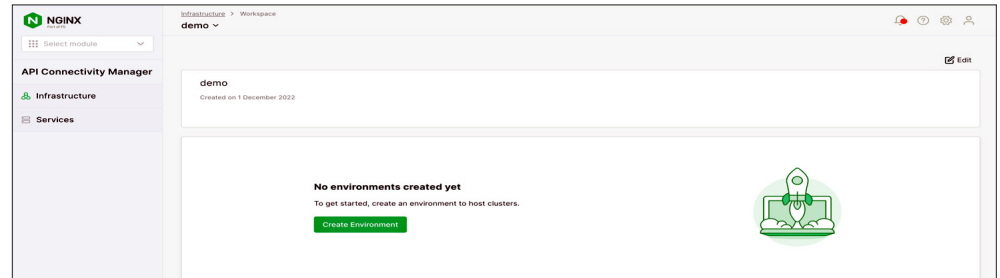
Figure 23: API Gateway Instances in Different Clouds



Create an Environment and API Gateway Cluster

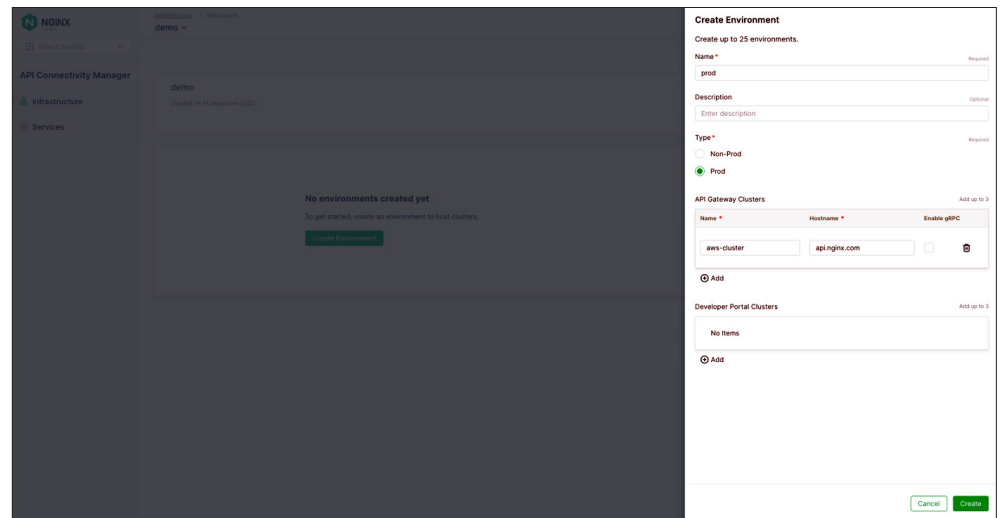
1. Navigate to your Workspace and click the **Create Environment** button, as shown in Figure 24.

Figure 24: Creating a New Environment in an Infrastructure Workspace



2. In the **Create Environment** drawer that opens, fill in the **Name** field (**prod** in Figure 25) and optionally the **Description** field, and select the Environment **Type** (here we're choosing Production).

Figure 25: Naming a New Environment and Assigning the First API Gateway Cluster to It

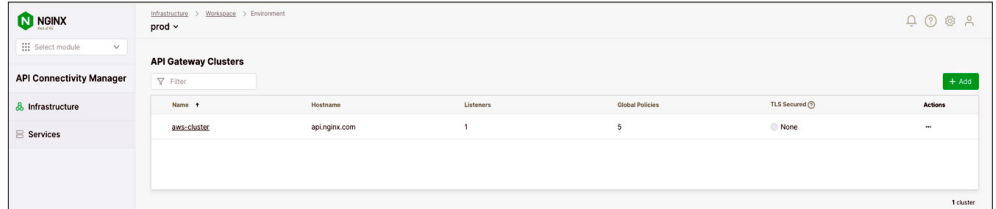


3. In the API Gateway Clusters section, fill in the **Name** and **Hostname** fields (in Figure 25, they are **aws-cluster** and **api.nginx.com**).
4. Click the **Create** button.

The **Environment Created** panel opens to display the command you need to run on each NGINX Plus instance to assign it to the API Gateway Cluster. For convenience, we show the commands in Step 10 below.

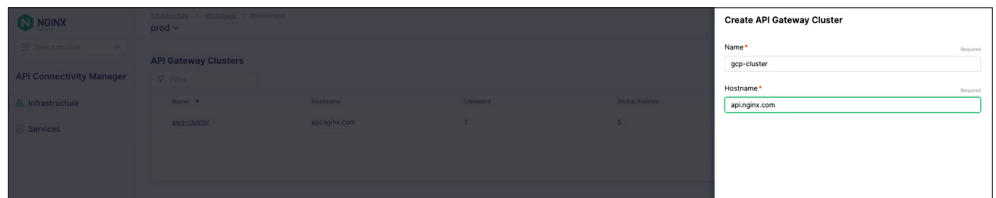
5. Navigate back to the Environment tab and click the **+ Add** button in the upper right corner of the **API Gateway Clusters** section, as shown in Figure 26.

Figure 26: Adding Another API Gateway Cluster to an Environment



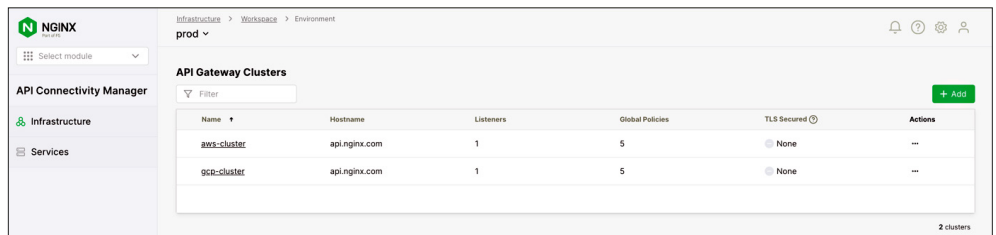
6. On the **Create API Gateway Cluster** panel, fill in the **Name** field with the second cluster name (**gcp-cluster** in Figure 27) and the **Hostname** field with the same hostname as for the first cluster (**api.nginx.com**).

Figure 27: Adding the Second API Gateway Cluster to an Environment



The two API Gateway Clusters now appear on the API Gateway Clusters for the production Environment, as shown in Figure 28.

Figure 28: List of NGINX Plus Instances Deployed in Multiple Clouds and Separate API Gateway Clusters



Assign API Gateway Instances to an API Gateway Cluster

Repeat on each NGINX Plus instance:

7. Use ssh to connect and log in to the instance.
8. If NGINX Agent is already running, stop it:

```
$ systemctl stop nginx-agent
```

9. Run the command of your choice (either `curl` or `wget`) to download and install the NGINX Agent package:
 - If you didn't enable mTLS in [Install and Configure API Connectivity Manager](#), add:
 - The `-k` flag to the `curl` command
 - The `--no-check-certificate` flag to the `wget` command
 - For `<NMS_FQDN>`, substitute the IP address or fully qualified domain name of your NGINX Management Suite server.
 - For `<cluster_name>`, substitute the name of the appropriate API Gateway Cluster (in this tutorial, `aws-cluster` for the instance deployed in AWS and `gcp-cluster` for the instance deployed in GCP).

```
$ curl [-k] https://<NMS_FQDN>/install/nginx-agent > install.sh
&& sudo sh -install.sh -g <cluster_name> && sudo systemctl
start nginx-agent

$ wget [--no-check-certificate] https://<NMS_FQDN>/install/
nginx-agent --no-check-certificate -O install.sh && sudo
sh install.sh -g <clusterName> && sudo systemctl start
nginx-agent
```

The appropriate NGINX Plus instance now appears in the **Instances** section of the **Cluster** windows for `aws-cluster` (Figure 29) and `gcp-cluster` (Figure 30).

Figure 29: The First of Two API Gateway Clusters in an Environment That Spans Multiple Clouds

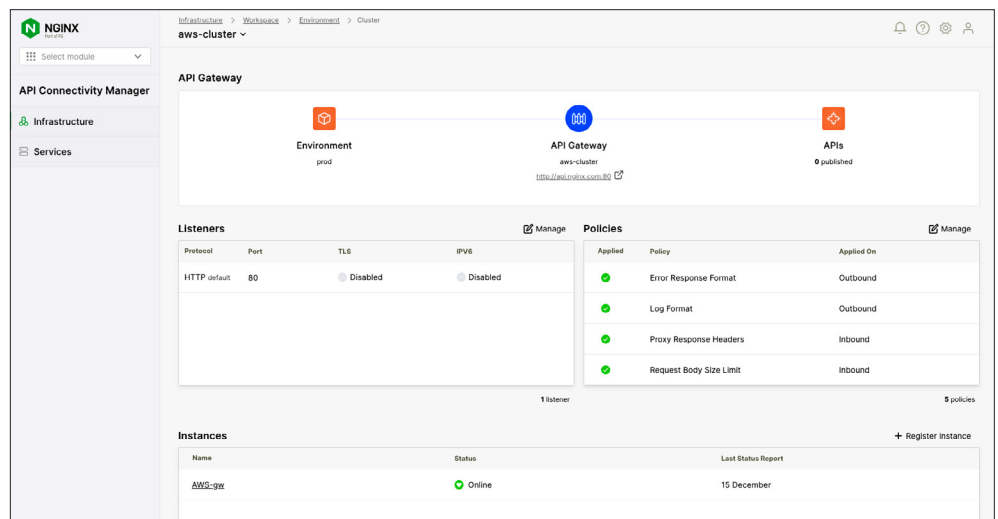
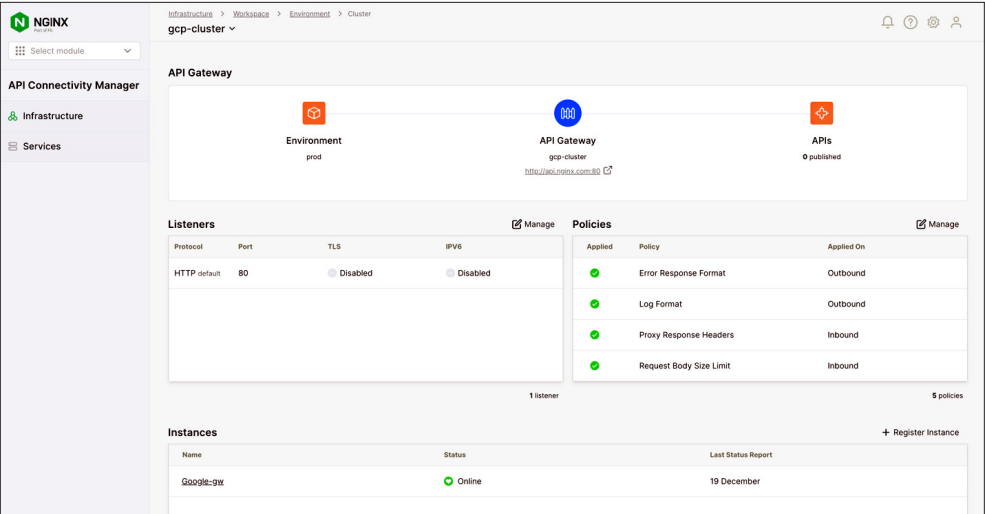


Figure 30: The Second of Two API Gateway Clusters in an Environment That Spans Multiple Clouds

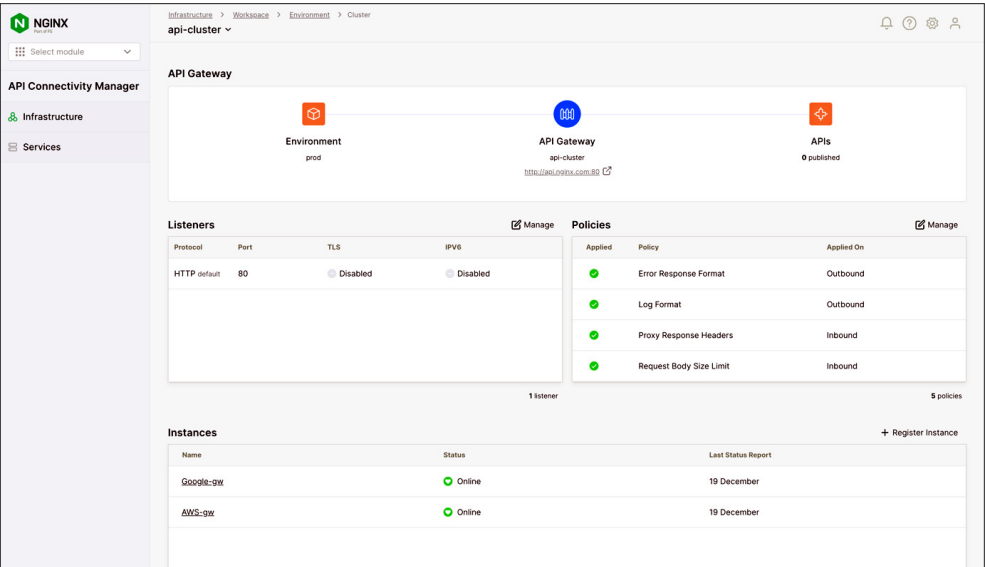


Apply Global Policies

Now you can add global policies, which apply to all the NGINX Plus instances in an API Gateway Cluster. For example, to secure client access to your APIs you can apply the **OpenID Connect Relying Party** or **TLS Inbound** policy. To secure the connection between an API gateway and the backend service which exposes the API, apply the **TLS Backend** policy. For more information about TLS policies, see the [API Connectivity Manager documentation](#).

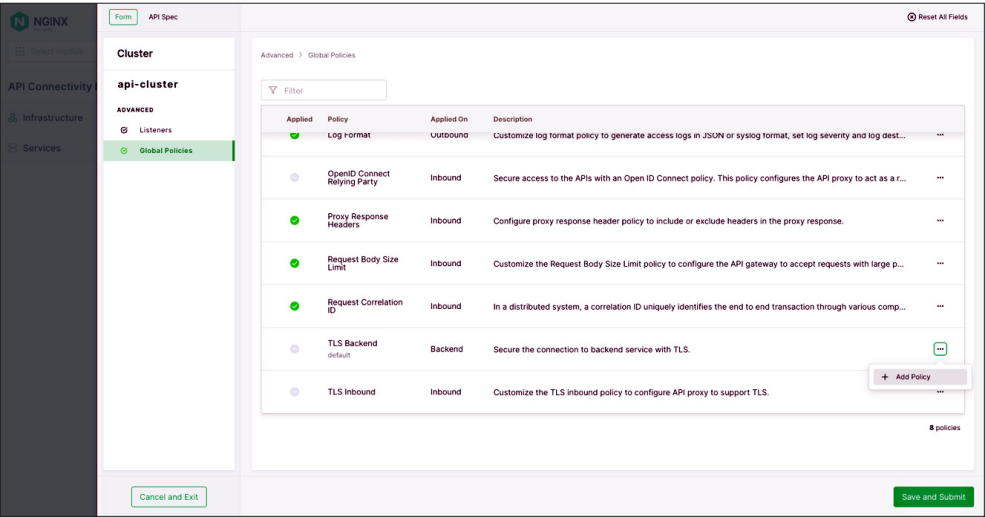
- 1. Navigate to the **Cluster** tab for the API Gateway where you want to apply a policy (**api-cluster** in Figure 31). Click the **Manage** button that’s above the upper right corner of the **Policies** table.

Figure 31: Managing Policies for an API Gateway Cluster



2. Click **Global Policies** in the left navigation column, and then the ... icon in the rightmost column of the row for the policy (**TLS Backend** in Figure 32). Select **+ Add Policy** from the drop-down menu.

Figure 32: Adding a Global Policy to an API Gateway Cluster



IN RECENT YEARS, THE
PROLIFERATION OF APIs HAS
SIGNIFICANTLY CHANGED THE
WAY ENTERPRISES OPERATE

5. Protect APIs Across Every Touchpoint

In recent years, the proliferation of APIs has significantly changed the way enterprises operate. APIs enable different applications to communicate and exchange data with each other, allowing for more efficient and effective business processes and software development.

However, with the increased use of APIs comes the risk of **API sprawl**, where APIs are created and deployed across distributed teams and architectures, often without proper oversight and management. This can create a new set of security risks for enterprises, as each API represents a potential entry point for attackers to gain unauthorized access to sensitive data and systems.

THE RISE OF API-FIRST SOFTWARE DEVELOPMENT

One of the main drivers of API sprawl is the proliferation of microservices. A microservices architecture breaks a larger application into smaller, individual applications that communicate with each other via API. This breaks complex applications into discrete parts that can be managed by individual teams and scaled independently of each other to meet traffic demands.

Microservices offer many advantages for developers, including increased flexibility and scalability. However, these benefits come with tradeoffs, including additional complexity. As a result, many enterprises adopt an **API-first approach to building microservices**. In this strategy, the design process for applications and services starts with an API contract that outlines how an API works, down to the format of requests and responses.

MICROSERVICES OFFER
MANY ADVANTAGES
FOR DEVELOPERS

THE BENEFITS OF API-FIRST
SOFTWARE DEVELOPMENT
CAN BE EASILY UNDERMINED
BY A FAILURE TO TAKE API
SECURITY SERIOUSLY

THE ATTACK SURFACE GROWS AS APIs PROLIFERATE

The benefits of API-first software development can be easily undermined by a failure to take API security seriously, especially during design and deployment. At the most basic level, more APIs mean more attack surface. While APIs play a vital role in modern software development, they are simultaneously becoming easier to exploit.

In 2018, [Gartner predicted](#) that APIs would become the most common attack vector for applications by 2022. If anything, their prediction of that much delay was overly optimistic. High-profile API breaches at major companies that affected millions of users were already occurring and have only become more common:

- In 2018, Facebook reported that at least [50 million users' data was at risk](#) after attackers exploited the company's developer API to obtain personally identifiable information (PII) linked to users' profile pages, including name, gender, and hometown.
- In 2019, LinkedIn reported that a hacker used data scraping techniques by exploiting APIs to collect over [700 million users' information](#), which was posted for sale on the dark web.
- In 2021, [an API maintained by Peloton](#) allowed a malicious actor to request PII, including age, gender, city, weight, and birthdate.
- In 2022, Twitter addressed an API breach that [exposed data from 5.4 million user accounts](#), including phone numbers and email addresses.
- In 2023, T-Mobile reported that an API breach had resulted in the [theft of data for 37 million customers](#), including names, emails, addresses, phone numbers, dates of birth, and more.

The breadth and variety of these attacks reveals the challenges faced by security and engineering leaders. Some attacks exploit APIs that were incorrectly exposed to the internet. Others use API keys or other authentication methods that were incorrectly exposed in code repositories. Or attackers get access to internal environments through VPN exploits and use internal APIs to exfiltrate data.

TRADITIONAL STRATEGIES
OFTEN FALL SHORT IN THE
FACE OF TODAY'S VARIED
API THREATS

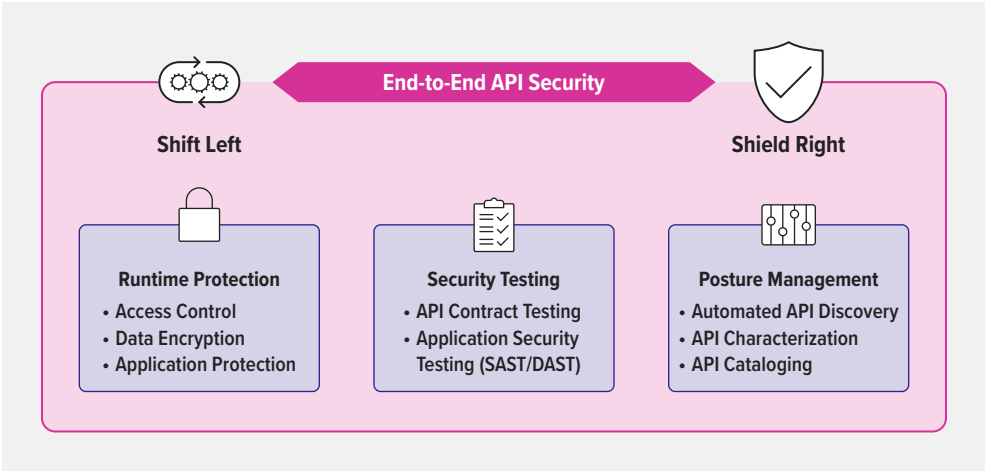
THWARTING API ATTACKS REQUIRES
THE RIGHT STRATEGY AND TOOLS

The most common way to protect against API threats is to combine traditional web application security strategies with modern API security techniques. Traditional strategies often fall short in the face of today's varied API threats. Modern techniques like automated API discovery and API contrast testing attempt to close these gaps.

It is critical for enterprises to *shield right* (implement global controls and security policies to protect deployed apps and APIs) and *shift left* (build security into code to eliminate vulnerabilities before apps and APIs go into production). Neither strategy can provide comprehensive API security on its own, so the key to preventing breaches is a holistic approach that spans three categories of API security practices:

- **API security posture management** – Provides visibility into the security state of a collection of APIs, including types of data exposed and request methods
- **API security testing** – Evaluates the security of an API across key points in its lifecycle to identify potential vulnerabilities
- **API runtime protection** – Detects and prevents malicious requests from reaching APIs during operation

Figure 33: Essential API
Security Requirements



By combining the right strategy with the right tools, organizations can better protect their APIs from attacks and ensure the security of their software systems. Let's look at the important functionality and tools that platform engineering leaders need to implement to protect APIs across their lifecycle.

What Is API Security Posture Management?

API security posture management creates visibility into the number, types, locations, and data exposed by your APIs. This information helps you understand the risks associated with each API so you can take appropriate actions to protect it.

Key functionality:

- **Automated API discovery** – Automatic and continuous API discovery for comprehensive visibility into APIs deployed in an environment
- **API characterization** – Identify and categorize APIs by protocol or architecture (REST, GraphQL, SOAP, etc.) and map sensitive data flows to understand your risk exposure
- **API cataloging** – Maintain a complete list of APIs to encourage software teams to reuse existing APIs, and to help SecOps teams build a complete view of your security posture

Representative technologies:

- **Web application and API protection (WAAP)** – Leverages a privileged global position in the API infrastructure to analyze traffic entering and leaving environments, identify APIs, and build a view of your risk exposure
- **Inline or agent-based discovery** – Attaches an agent to existing API gateways, load balancers, or Kubernetes Ingress controllers to mirror and analyze API traffic
- **Out-of-band or agentless discovery** – Uses traffic mirroring or exported logs and metrics to analyze API traffic; usually offers less visibility into APIs and threats than other technologies
- **Domain crawlers** – API security providers may offer crawlers that probe your domain for exposed API endpoints that are allowing traffic to bypass your API gateways and load balancers where security policies can be enforced

It's important to keep in mind that **no technology can reliably find every API in your architecture**. Most discovery techniques rely on visibility provided by existing load balancers, API gateways, and Ingress controllers, and are not likely to catch misconfigurations that bypass these architectural components.

Ultimately, code review and following API-first best practices offers more effective long-term prevention. But automated API discovery tools are still useful for rapidly building a view of your security posture and for catching APIs that might otherwise go unmanaged and unsecured.

API SECURITY TESTING IS VERY MUCH ABOUT INDIVIDUAL APIs

What Is API Security Testing?

While API security posture management is concerned with enterprise-wide security, API security testing is very much about individual APIs. At its most basic, API security testing helps identify and prevent vulnerabilities and their associated risks by testing the API runtime – the application running behind the API. It helps ensure that basic security requirements have been met, including conditions for authentication, authorization, rate limiting, and encryption.

Key functionality:

- **API contract testing** – Uses an API's OpenAPI Specification to verify that it is performing as designed, by comparing client requests and server responses, and an “inside out” approach to discover whether APIs are vulnerable prior to deployment
- **Dynamic application security testing (DAST)** – Simulates attacks against an API runtime to find vulnerabilities, evaluating the API from the “outside in” like a malicious user

Representative technologies:

- **API contract testing software** – Specialized tools for running tests that segment API requests and responses to verify that client and server behavior complies with the API contract
- **Application security testing (SAST/DAST)** – Tools that analyze and test applications, including APIs, by simulating attacks

There are both open source contract-testing tools and commercial products from dedicated API security vendors. The application security testing (AST) market has existed for decades, and increasingly many vendors offer dedicated scanning and testing tools for APIs.

What Is API Runtime Protection?

API runtime protection refers to securing APIs as they operate and manage requests. It prioritizes building security into the platform infrastructure as well as the code of the APIs themselves. The objective is to identify and prevent malicious API requests that emerge after deployment.

Key functionality:

- **Access control** – Enforce authentication (AuthN) and authorization (AuthZ) policies
- **Data encryption** – Encrypt and protect communications across the network
- **Application protection** – Protect API runtimes from malicious API requests and attacks
- **Real-time monitoring** – Visualize, trace, and mitigate attacks across API infrastructure

API RUNTIME PROTECTION REFERS TO SECURING APIs AS THEY OPERATE AND MANAGE REQUESTS

Representative technologies:

- **API gateway** – Applies and enforces security policies, including authentication, authorization, rate limiting, access control lists, and encryption
- **Web application firewall (WAF)** – Protects APIs and applications against sophisticated Layer 7 attacks by actively monitoring and filtering traffic based on attack signatures
- **Identity provider (IdP)** – Service that stores and verifies user identity, and typically works with single sign-on (SSO) providers to authenticate users

Not all API gateways and WAFs/WAAPs are created equal. Some services, particularly the native solutions available on cloud and other platforms, lack the global visibility and standardization required in multi-cloud and hybrid architectures.

API SECURITY BEST PRACTICES

Given the importance of securing APIs, it is essential to approach API security in an organized way. Platform engineering and security leaders must work together to address security requirements across the API lifecycle. As we explored earlier, this roughly aligns to three main areas of practice: API security posture management, API security testing, and API runtime protection. In other words, you need to focus on knowing how many APIs you have, how to test them for errors, and how to build security into your code.

The following best practices are based on well-established security controls for APIs, whether you intend to expose them publicly, to a limited number of partners, or only to internal teams:

- **Inventory and manage your APIs.** Automated API discovery tools can help you build an initial view of every API, whether public or private. Meet with engineering leaders to determine if other APIs may exist in your infrastructure that did not show up in automated scans. Ensure every API has a functional owner who is responsible for managing it.
- **Implement API security testing throughout the API lifecycle.** Verify that APIs are built and deployed with appropriate security policies, and identify potential misconfigurations.
- **Enforce authentication and authorization.** Poor or non-existent authentication and authorization is one of the most common vulnerabilities for APIs. Since APIs provide an entry point into an organization's systems and data, it's critical to enforce access control.
- **Practice the principle of least privilege.** Limit which users, groups, and roles can access specific API resources and grant the minimum access necessary to complete a task. Consider using API gateways for each team or line of business to customize security and compliance policies for their unique requirements.

- **Remove information that's not meant to be shared.** Even though APIs are developer tools, they often contain information that must not be exposed or made publicly available, like personally identifiable information (PII) that includes contact or payment information. Ensure each API only returns as much information as necessary, and obfuscate responses that contain confidential data.
- **Rate limit API requests.** Rate limiting is another form of access control and limits the number of requests the API gateway passes to the API during a defined period of time, sometimes with different limits for different clients. This ensures API resources are not overwhelmed by requests and provides additional protection against DoS attacks.
- **Validate requests and responses.** Apply positive security by using schema validation to match API requests against the API contract. If validation fails, the API call is blocked, protecting the API runtime from malicious requests or payloads.
- **Monitor API traffic for anomalies.** Monitor API traffic and flag anomalies in real time to identify when an attack might be underway. A mix of traffic metrics, logs, and other sources like a WAF is typically used to monitor and inspect API requests and responses.

CONCLUSION

Like all cybersecurity, API security is an ongoing process that requires collaboration with many stakeholders, including network engineers, security operations leaders, platform engineering leaders, and software development engineers. The good news is that it's no great mystery how to secure APIs. Most organizations already have measures in place to combat well-known attacks like cross-site scripting (XSS), injection, distributed denial-of-service (DDoS), and others that can target APIs. And many of the best practices described above are likely quite familiar to seasoned security professionals. No matter how many APIs your organization operates, your goal is to establish solid API security policies and manage them proactively over time.

6. Identify and Track Important API Metrics

As companies adopt **API-first** design practices to build modern applications, measuring the operational performance and value of those APIs becomes a top priority. Establishing a framework that clearly defines and connects API metrics with key performance indicators (KPIs) is one of the most important steps to ensure a successful API strategy.

Typically, KPIs are tied to specific goals. They have a defined time frame and are aligned to the outcomes your API strategy needs to deliver. API metrics, in contrast, are significant data points. Not every metric is a KPI, but every KPI begins as a metric.

So, how do you start? First, you need to be clear – at the outset – about the goal of your API strategy. Then choose the metrics that align with that goal. Remember that different teams might need to measure and track different metrics depending on what is important to them and what is essential for the business.

Broadly, there are three categories of API metrics that companies can track, each answering a different question:

- **Operational metrics** – Are APIs delivering the stability, reliability, and performance you need?
- **Adoption metrics** – Are developers adopting and using your APIs?
- **Product metrics** – How are APIs supporting your business objectives?

Imagine these overarching metrics as a pyramid. At the bottom, operational metrics measure the tactical performance of individual APIs and the infrastructure supporting them. At the top, product metrics measure the business value created by your APIs. The two are connected by adoption metrics, which track the growth of the API program with end users (developers). Generally, product metrics and adoption metrics align to the business outcomes you need to measure, while operational metrics align with the technical standards you need to maintain.

Next, we detail specific metrics that are critical to measure, how they enable infrastructure and application teams, and ways these metrics connect with KPIs.

OPERATIONAL METRICS

When you are just getting started, operational metrics are often the first thing to measure. They are tactical and provide insights into how APIs are functioning. Operational metrics are not usually KPIs themselves. Instead, they help you measure the quality and performance of the software your teams are building. They can provide early indicators of emerging problems, or help you drill down and discover issues that might be impacting your critical KPIs.

The operational metrics you track vary by team and responsibility.

Infrastructure Teams

Platform Ops is the team responsible for maintaining, connecting, and securing the collection of infrastructure and technologies used by different teams to deliver applications. For API programs, this often includes API gateways and API developer portals.

Key metrics for infrastructure teams like Platform Ops include:

- **Uptime** – Even as one of the most basic metrics, uptime is the gold standard for measuring the availability of a service. This is often tied to a service-level agreement (SLA).
- **CPU and memory usage** – Tracking resource utilization at the API gateway is critical to identifying when you might need to scale out your instances. It also acts as an early warning when something is starting to break or errors are causing CPU and memory usage to spike.
- **Total pass and error rates** – Measuring how often APIs trigger non-200 status codes helps you understand how error-prone your APIs may be. This aggregate measure provides information about the overall quality of the APIs your teams are putting into production.

Application Teams

Application teams, made up of API developers and service owners, are responsible for building and operating individual services or applications. These might be used as part of a larger product, to integrate with a partner, or when delivering an API as a Service (APIaaS) to developers.

The following metrics are important for application teams to measure:

- **Requests per second** – This is a performance metric that indicates how often your backend application server is being accessed. You typically want a lower number to improve efficiency and create ensure the best experience for API users.
- **Average and maximum latency** – Tracking the average time it takes your API to receive a request and return a response is crucial. A single slow API can negatively impact the user experience and therefore the business.
- **Errors per minute** – API calls inevitably fail at some point. It's not a matter of if, but when. Monitoring errors and seeing when they start to spike means it's time to plan a course of action to restore service.

Adoption Metrics

To understand how developers are interacting with your APIs, it is essential for an API-first business to look beyond engineering metrics. You also need to measure and monitor the API developer experience to ensure developers are adopting and getting value from your APIs.

A few examples of adoption metrics include:

- **Unique API consumers** – Often time-bound into monthly users, this metric measures how many developers are adopting and using your APIs. Ideally this metric grows over time as more developers integrate your API into their applications.
- **API usage growth** – This metric also measures API adoption, and is often the preferred metric for doing so. Ideally, API traffic grows monthly the number of applications and developers using the API increases.
- **Time to first call** – This metric records how long it takes a developer to create an account, generate API credentials, and run the first API call. Enabling developers to get up and running as fast as possible is a high priority, making this metric the most important in measuring the overall API developer experience.

Note: You need to devote at least one of your KPIs to tracking the adoption of your APIs. This helps calculate the overall growth of your API program. For example, you might set a KPI to increase the number of developers who have created an ongoing integration or app using your API.

Product Metrics

API product metrics play a major role in understanding the value of an API. Although only a small subset of APIs may directly contribute to revenue, every API needs to provide value to the business.

Key product metrics to measure include:

- **Direct and indirect revenue** – These metrics target the different ways APIs contribute to revenue. While some APIs are directly monetized, others support integrations with business partners. Tracking this indirect value, like adoption of your APIs, helps developers build revenue-generating apps for partners.
- **Applications per API** – APIs need to be reusable. This metric measures how many applications integrate with an API, revealing which APIs provide the most value.
- **Number of partners** – APIs often enable business relationships and enable deeper integrations into an ecosystem of applications. Tracking the number of partners using your APIs can help demonstrate the indirect value provided by your API.

Note: These product metrics align closely with business impact, and you might choose to turn some into KPIs depending on your business goals. For example, if one goal of your API strategy is to integrate your application deeper into an ecosystem of tools used by your customers, you might want to track both the number of partners integrating with your API, and any indirect revenue generated through those integrations.

CONCLUSION

Aligning API metrics and business KPIs is one of the principal ways to make data-driven decisions and ensure your API strategy delivers the value your organization requires. And not only that – gaining visibility into your APIs can also empower infrastructure and application teams to measure the operational metrics that matter most to them.