APIGEE

INDEX:

What is Apigee Edge?

Apigee Edge is a platform for developing and managing API proxies. Think of a proxy as an abstraction layer that "fronts" for your backend service APIs and provides value-added features like security, rate limiting, quotas, analytics, and more. The primary consumers of Edge API proxies are app developers who want to use your backend services.

Basic terminologies you should know from APIGEE perspective:

API

An 'application programming interface'—an interface that makes it easy for one application to 'consume' capabilities or data from another application.

By defining stable, simplified entry points to application logic and data, APIs enable developers to easily access and reuse application logic built by other developers. In the case of 'Web APIs', that logic and data is exposed over the network.

API proxy

A facade on Edge for one or more APIs, generic HTTP services, or applications (such as Node.js).

An API proxy is implemented as a set of configuration files, policies, and code that rely on a set of resources provided by Apigee Edge. API proxies can be generated and configured using the Apigee Edge management UI, or they can be implemented locally in a text editor or IDE.

API base path and resources

APIs defined by network addresses and URIs. An API is made up of a 'base path' and a set of 'API resources'. Every API proxy defines a base path and, optionally, multiple API resource paths. You can think of an API simply as a set of URIs, all of which share a common base path.

To make it easier to manage your APIs, Apigee augments these raw URIs with display names and descriptions. Edge enables you to attach policies and code to URIs, enabling fine-grained control and management of the behavior of your APIs.

-API product

A collection of API resources (URIs) combined with a quota, or 'service plan', which is published to app developers at design time. API products can in turn be bundled into API packages for monetization.

-API package

A collection of API products that are presented to developers as a bundle, and typically associated with a rate plan defined in monetization.

-environment

By default, organizations are provisioned with two environments: 'test' and 'prod'.

The 'test' environment is typically used for deploying API proxies during development.

The 'prod' environment is typically used for promoting API proxies from the test environment after they have been fully developed and tested.

-organization

A container for all the objects in an Apigee Edge account, including API proxies, API products, API packages, apps, and developers.

A user account is required for each organization for which you are a member. (Most users will have an account in only one organization.)

-policy

A processing step that executes as an atomic, reusable unit of logic within an API proxy processing flow.

Typical policy-based functionality includes transforming message formats, enforcing access control, calling remote services for additional information, masking sensitive data from external users, examining message content for potential threats, caching common responses to improve performance, and so on.

-API resource path

A RESTful concept, a resource path is a uniform resource identifier (URI) that identifies the network path to a given resource.

version-

The version of the developer-facing API interface.

For example, pivotaltracker.com/services/v3, or api.enterprise.apigee.com/v1.

-revision

A numbered, version-controlled package of configuration and policies bundled into an API Proxy. This term is distinguished from 'version', which is the developer-facing API interface. See version above.

Develop with Apigee Edge

As a service provider, you develop APIs for consumption by client apps. You can use two different development methods to create, configure, and maintain API proxies and API products:

1. Use the Edge management UI at https://enterprise.apigee.com.
2. Make HTTP requests to the Apigee RESTful APIs to access Edge services.

Using the management UI

The Edge management UI is a browser-based tool you can use to create, configure, and manage API proxies and API products. There are a few tasks that require you to use the management API, though.

In the Edge management UI, you can:

1. Create API proxies by editing code and tracing flow of requests through your proxies.
2. Create API products that bundle proxies for exposure to client requests.
3. Manage developers and developer apps.
4. Configure your test and production environments.
5. Implement JavaScript and Node.js applications.

Using the RESTful management API

You can use Apigee Edge RESTful management APIs to create, configure, and manage API proxies and API products, policies for logic in your API proxies, apps and app developers, caches.

These API endpoints often take data containing configuration information and require you to pass authentication information, such as username and password, to access them. Following RESTful principles, you can call HTTP GET, POST, PUT, and DELETE methods on any of the API resources.

GET: Retrieves a list of resources, or the profile of a specific resource.

POST: Creates a resource, or passed in a parameter, performed an action on resource. For example, when you revoke OAuth access tokens, you use the POST method along with the parameter action=revoke.

PUT: Updates an existing resource.

DELETE: Deletes an existing resource.

For a complete list of resources and their supported methods, see the Apigee Management API Reference.

Authenticating access

You must authenticate yourself to the management API server when you call the APIs. You can do this in one of the following ways:

- **OAuth2**
- **SAML**
- **Basic Auth (not recommended)**

What is Learn Edge?

The Learn Edge series is a Git-based, hands-on, learn-by-doing experience for beginning Edge developers. Each example is designed to be quick and easy to do and teaches a core Apigee Edge concept or technique. Each Learn Edge example follows three basic steps:

- Deploy it.
- Run it.
- Trace it.

The best way to learn Apigee Edge is by doing! You can find Learn Edge in the public Apigee GitHub repository:

https://github.com/apigee/api-platform-samples/tree/master/learn-edge

Command-line tools for Apigee Edge

You can use command-line tools to perform many of the tasks that the Apigee Edge console UI exposes. Using these tools, you can create scripts to automate many tasks.

apigeetool

A Node.js module for deploying API proxies and Node.js applications to Apigee Edge.

apigee-access

Gives Node.js applications a way to access Apigee-specific functionality.

Migrating data from an Apigee Evaluation org

You can migrate code in your Apigee Edge organization from one Evaluation organization to another, or from an Evaluation organization to a paid plan organization. Using the Apigee migrate tool, you can migrate most of the code, leaving just a few pieces to copy over manually.

The Apigee migrate tool is an open source tool that uses a JavaScript task runner, Grunt.

Data that CAN be migrated:

With the tool, you can import and export data about:

- developers
- proxies (latest version)
- products
- apps
- app keys
- KVMs (org and env)

You can also import the following kinds of data from a CSV file to an Apigee org:

- developers
- apps
- app keys

- KVMs (org and env)

Data that CANNOT be migrated:

- Cache resources and cached values.
- Load balancing across backend servers
- Keystores and Truststores
- Organization or environment level resources such as .jar files, .js files, and so on.
- Analytics data. This data can't be migrated.
- KVM entries for "encrypted" key-value maps.

Understanding API and API Proxies

What is an API?

An API is an interface that makes it easy for one application to 'consume' capabilities or data from another application. By defining stable, simplified entry points to application logic and data, APIs enable developers to easily access and reuse application logic built by other developers. In the case of 'Web APIs', that logic and data is exposed over the network.

Apigee Edge enables you to build APIs and if you have APIs already, expose them directly, while adding a management and visibility layer. If you have HTTP enabled services, such as SOA-based Web services, they can also be exposed as APIs via Apigee Edge.

What is an API proxy?

You expose APIs on Apigee Edge by implementing API proxies. API proxies decouple the app-facing API from your backend services, shielding those apps from backend code changes. As you make backend changes to your services, apps continue to call the same API without any interruption.

In an API proxy configuration, there are two types of endpoints:

ProxyEndpoint: Defines the way client apps consume your APIs. You configure the ProxyEndpoint to define the URL of your API proxy. The proxy endpoint also determines whether apps access the API proxy over HTTP or HTTPS. You usually attach policies to the ProxyEndpoint to enforce security.

TargetEndpoint: Defines the way the API proxy interacts with your backend services. You configure the TargetEndpoint to forward requests to the proper backend service, including defining any security

settings, HTTP or HTTPS protocol.

What's a policy?

Apigee Edge enables you to 'program' API behavior without writing any code, by using 'policies'. A policy is like a module that implements a specific, limited management function. Policies are designed to let you add common types of management capabilities to an API easily and reliably. Policies provide features like security, rate-limiting, transformation, and mediation capabilities, saving you from having to code and maintain this functionality on your own.

Content-based security

JSON threat protection

JSON attacks attempt to use structures that overwhelm JSON parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the JSONThreatProtection Policy type.

XML threat protection

XML attacks attempt to use structures that overwhelm XML parsers to crash a service and induce application-level denial-of-service attacks.

Such attacks can be mitigated using the XMLThreatProtection Policy type.

General content protection

Some content-based attacks use specific constructs in HTTP headers, query parameters, or payload content to attempt to execute code. An example is SQL-injection attacks.

Such attacks can be mitigated using the RegularExpressionProtection Policy type.

Data masking and hiding

When you debug APIs calls in Edge, the content can sometimes contain sensitive data, such credit cards

or personally identifiable health information (PHI) that needs to be masked.

Edge provides different ways of hiding or masking sensitive data from Trace and debug sessions.

Hiding sensitive data:

You can prevent sensitive data from appearing in the Trace tool and debug sessions by creating custom variables prefixed with "private.".

For example, when using the Key Value Map Operations policy to retrieve values from an encrypted key value map, format the variable names as follows to ensure the values don't appear in Trace or debug sessions:

<Get assignTo="private.hiddenData">

Hiding sensitive variables is an alternative to using data masking, described next. The difference between hiding and masking is that hidden variables don't appear at all, and masked values are replaced with asterisks in Trace and debug sessions.

Masking sensitive data:

Edge lets you define 'mask configurations' to mask specific data in trace and debug sessions. Masking configurations can be set globally (at the organization-level) or locally (at the API proxy level). Role-based capabilities govern which users have access to the data that is defined as sensitive.

When data is masked, it is replaced with asterisks in the trace output. For example:

<description>**********</description>

Warning: Data masking is enabled only when a trace session or debug session is enabled for an API proxy. If no trace or debug sessions are enabled on an API proxy, the data will not be masked.

Using Mask Configurations:

Mask configurations enable you to identify sensitive data in these sources:

XML payloads: Using XPath, you identify XML elements to be filtered from request or response message payloads.

JSON payloads: Using JSONPath, you identify JSON properties to be filtered from request or response message payloads.

Flow variables: You can specify a list of variables that should be masked in debug output. When you specify the request.content, response.content, or message.content flow variables, the request/response

body is also masked.

Rate-limiting

Apigee Edge provides **three** mechanisms that enable you to optimize traffic management to minimize latency for apps while maintaining the health of backend services.

1) Spike Arrest

This policy smooths traffic spikes by dividing a limit that you define into smaller intervals. For example, if you define a limit of 100 messages per second, the Spike Arrest policy enforces a limit of about 1 request every 10 milliseconds (1000 / 100); and 30 messages per minute is smoothed into about 1 request every 2 seconds (60 / 30. The . This policy should be used to prevent denial-of-service (DOS) attack.

2) Quota

This policy enforces consumption limits on client apps by maintaining a distributed 'counter' that tallies incoming requests. The counter can tally API calls for apps, developers, API keys, access tokens. Usually, API keys are used to identify client apps. This policy is computationally expensive so, for high-traffic APIs. This policy should be used to enforce business contracts or SLAs with developers and partners, rather than for operational traffic management.

3) ConcurrentRateLimiting

This policy enables traffic management between API Services and your backend services. Some backend services, such as legacy applications, may have strict limits on the number of simultaneous connections they can support. This policy enforces a limit on the number of requests that can be sent at any given time from API services to your backend service.