**Apigee API Management**

Apigee is a platform for developing and managing APIs. By fronting services with a proxy layer, Apigee provides an abstraction or facade for your backend service APIs and provides security, rate limiting, quotas, analytics, and more.

**High-level architecture**

The following image shows the high-level architecture of Apigee:

High level overview
  of the Apigee architecture.

As the image shows, Apigee consists of the following primary components:

* **Apigee services:** The APIs that you use to create, manage, and deploy your API proxies.
* **Apigee runtime:** A set of containerized runtime services in a Kubernetes cluster that Google maintains. All API traffic passes through and is processed by these services.

In addition, Apigee uses other components including:

* **GCP services:** Provides identity management, logging, analytics, metrics, and project management functions.
* **Back-end services:** Used by your apps to provide runtime access to data for your API proxies.

For more detailed explanations, see [Components of Apigee](https://cloud.google.com/apigee/docs/api-platform/get-started/what-apigee#components).

Here's a more granular image showing the connection between your Cloud project and Google services over a private peering network:

Architecture
  diagram showing the VPC connections.

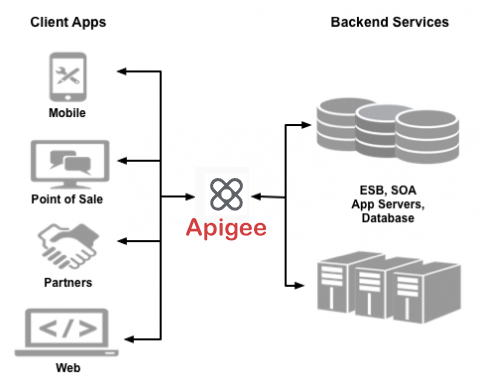
For an example of how to use Apigee, check out [this webcast about how Walgreens uses APIs and Apigee](https://www.youtube.com/watch?v=_Bb7UXb7ds4) to provide a rich app ecosystem around photo printing, prescriptions, and other services.

**Make services available through Apigee**

Apigee enables you to provide secure access to your services with a well-defined API that is consistent across all of your services, regardless of service implementation. A consistent API:

* Makes it easy for app developers to consume your services.
* Enables you to change the backend service implementation without affecting the public API.
* Enables you to take advantage of the analytics, developer portal, and other features built into Apigee.

The following image shows an architecture with Apigee handling the requests from client apps to your backend services:



Rather than having app developers consume your services directly, they access an API proxy created on Apigee. The API proxy functions as a mapping of a publicly available HTTP endpoint to your backend service. By creating an API proxy you let Apigee handle the security and authorization tasks required to protect your services, as well as to analyze and monitor those services.

Because app developers make HTTP requests to an API proxy, rather than directly to your services, developers do not need to know anything about the implementation of your services. All the developer needs to know is:

* The URL of the API proxy endpoint.
* Any query parameters, headers, or body parameters passed in a request.
* Any required authentication and authorization credentials.
* The format of the response, including the response data format, such as XML or JSON.

The API proxy isolates the app developer from your backend service. Therefore you are free to change the service implementation as long as the public API remains consistent. For example, you can change a database implementation, move your services to a new host, or make any other changes to the service implementation. By maintaining a consistent frontend API, existing client apps will continue to work regardless of changes on the backend.

You can use policies on the API proxy to add functionality to a service without having to make any changes to the backend service. For example, you can add policies to your proxy to perform data transformations and filtering, add security, execute conditional logic or custom code, and to perform many other actions. The important thing to remember is you implement policies on Apigee, not on your backend server.