### Comparison between API gateway technologies

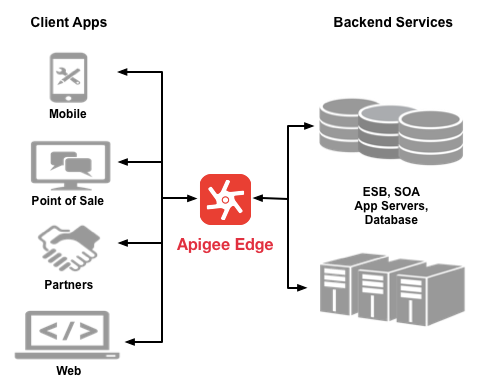
* **APIGEE Edge**

Apigee Edge is a platform for developing and managing API proxies.

Apigee Edge, which is built on Java, 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, monetization, developer portal, and other features built into Edge.

The following image shows an architecture with Edge 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 Edge. The API proxy functions as a mapping of a publicly available HTTP endpoint to your backend service.  By creating an API proxy, you let Edge handle the security and authorization tasks required to protect your services, as well as to analyze, monitor, and monetize those services.

* **Spring Rest**

Definition: Representational State Transfer (**REST**) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web **service** induce desirable properties, such as performance, scalability, and modifiability, that enable **services** to work best on the Web. (Wiki)

Why REST? "The notion that the web is an existence proof of a massively scalable distributed system that works really well, and we can take ideas from that to build integrated systems more easily." There is a pretty good reason: REST embraces the precepts of the web itself, and embraces its architecture, benefits and all.

What benefits? Principally all those that come for free with HTTP as a platform itself. Application security (encryption and authentication) are known quantities today for which there are known solutions. Caching is built into the protocol. Service routing, through DNS, is a resilient and well-known system already ubiquitously support.

REST, however ubiquitous, is not a standard, per se, but an approach, a style, a constraint on the HTTP protocol. Its implementation may vary in style, approach. As an API consumer, this can be a frustrating experience. The quality of REST services varies wildly.



Figure 1. Leonard Richardson’s Maturity Model

(source: <http://spring.io/guides/tutorials/bookmarks/>)