Google Cloud

Edge REST Design Fundamentals

API Versioning Strategy

API Versioning

What ***isn't*** API Versioning?

It is ***NOT*** an indicator of the current release version.

Consumers don't care what your internal software release is. This information doesn't help a consumer build or maintain APIs.

Because it's internally and centrally managed, on any given date your operations team already knows what version

What *IS* API Versioning?

Versioning is how you communicate when existing software is likely to break because of an update.

Versioning only needs to happen when new required fields are added to queries, OR previously available data are removed from payloads.

The Implications

- Versioning happens infrequently
- Versioning is a very impactful change, and may cause breakage of software that you were unaware even existed
- Communications about versioning should be proactive and complete

Some Approaches to Versioning

Twilio: A date parameter used to look up the version of an API that was valid at that time.

`GET /2010-04-01/Accounts`

SalesForce.com: a version indicator (v20.0) somewhere in the URL.

GET /services/data/**v20.0**/sobjects/Account

Twitter: A custom header.

GET /1/accounts
x-api-version: **1.1**

Approach Pros & Cons

Approach	Pro	Con
Twilio's "Date" based filter	Clever - always resolves to whatever version was active	Unfamiliar to most developers. May be hard to implement.
SalesForce's version in URI	Generally clear, familiar to developers. Easy to work with in browsers and tools.	Sometimes confused for a version of the resource, not a version of the API.
Twitter's Version in header.	Semantically beautiful. Adheres best to Fielding's REST ideals.	Not as easy to explore with common tools.

Our Favorite Approach

GET /<domain>/**v1**/<resource>/{id}

Simple: Easily understandable, non-magical, and can be easily passed in in testing tools.

Unambiguous: When a change happens, the URL changes.

Intuitive: You can see that the domain has updated its resource model

Best Practice:

Avoid making breaking changes.

Changes made to an API that will cause downstream breakage will reduce the utility of your API.

Reduction in utility will reduce usage and increase time to market, as well as increase the cost associated with updates.

Reduction in utility will also reduce use, which impacts value directly when clients pay for usage or indirectly when the organization can't easily leverage the solution.

Best Practice:

Pick something and ***be consistent***!

Consistency leads to **predictability**.

Predictability leads to **intuitive usability**.

Usability leads to **usage**.

Usage leads to **innovation**.

THANK YOU