Google Cloud

Edge

API Versioning

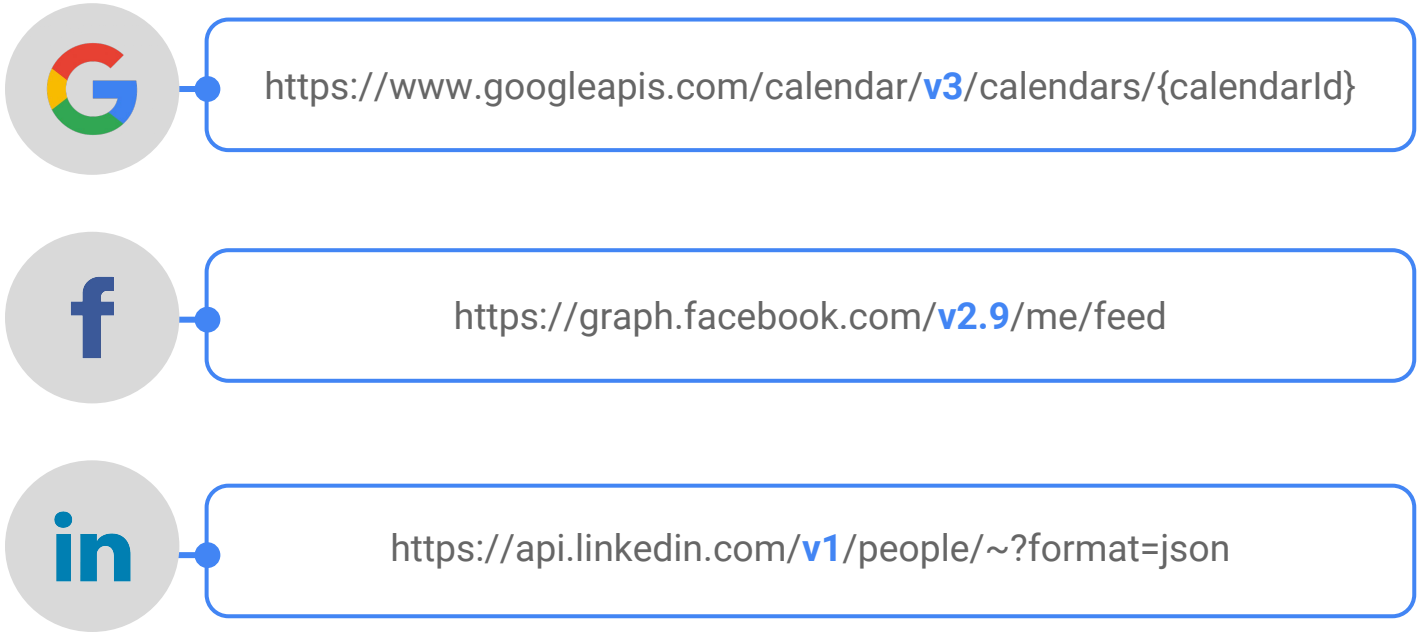
An API is a Contract

The API is a contract which forms the basis of communication between your systems and consuming applications.

- Changes can have a significant impact.
- Your versioning strategy is essential to your partner's businesses success.
- A well managed API versioning strategy could be reassuring for potential consumers of your API.
- Poorly managed changes can have a negative impact in your developer community.

API Versioning Examples

Common to see versioning in URL



API Versioning Options

	/v1	/grouping/v2	Accept Header	Custom Header
Definition	Version in URI right after domain name	Version in URI after a particular grouping	Version in Accept header as vendor mime type	Version appears in a custom header
Example	LinkedIn API https://api.linkedin.com/v1/people/~	Gmail API https://www.googleapis.com/gmail/v1/users	Github API Accept: application/vnd.github.v3+json	Azure API x-ms-version: 2015-02-21
Scope	When a resource changes a new version of the whole API is created	When a resource changes, the group version is incremented	When a resource changes, only the version for that resource is incremented	
Discovery	✔	✘	✔	✔
Caching	✔	✔	✔	! Vary Header
Browser friendly	✔	✔	✘	✘
Links to other resources	✔	✔	✔	✔

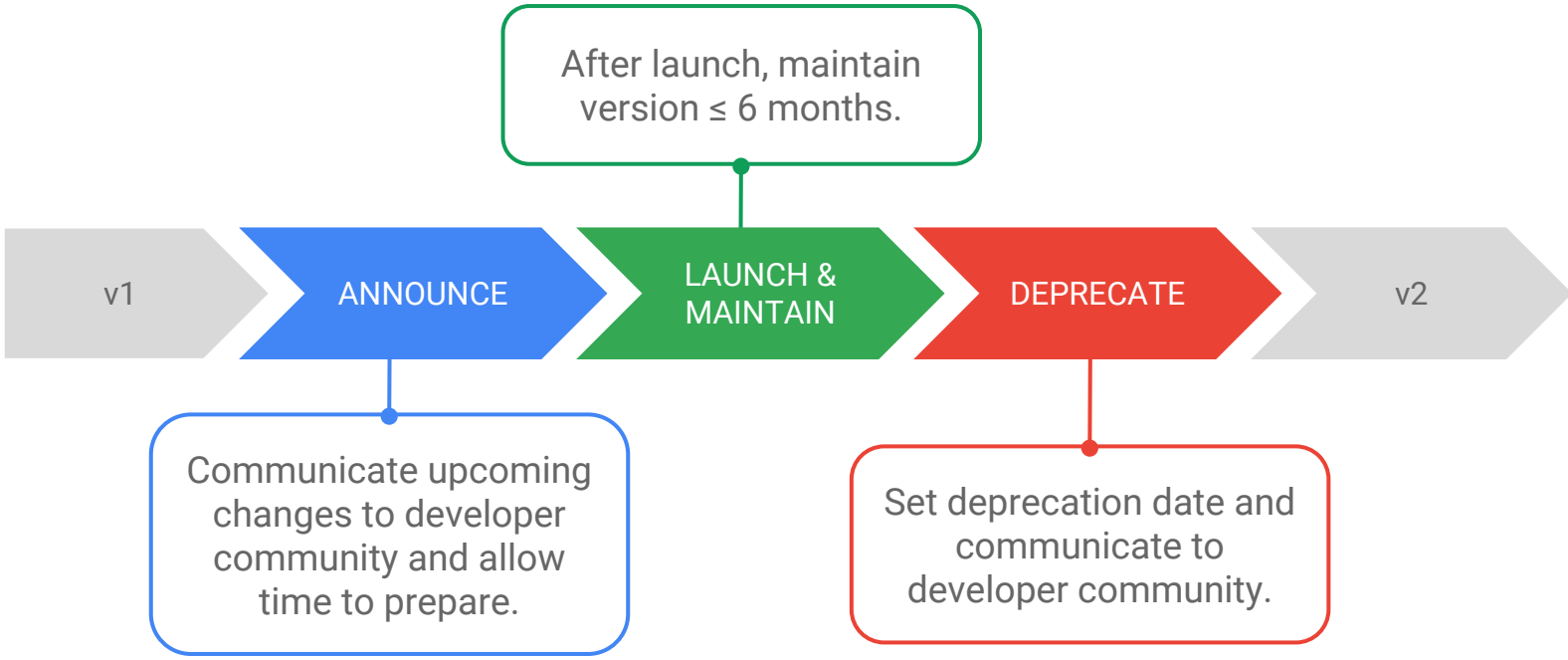
When to Increment a Version?

You increment to a new version of your API when you introduce significant non-breaking changes, or you introduce a breaking change.

No Version Change Recommended	
Minor Non-breaking Change	**Example:** Adding a “non essential” new field
Version Change Recommended	
Major Non-breaking Change	**Example:** Adding an “essential” new field, or a significant number of new fields
Version Change Required	
Breaking Change	**Example:** Changing an ID from an “int” to a “uuid” Userid (int): 12345 Userid (uuid): 111111-aaaaa-22222-bbbbb-33333

How Many Versions?

The general recommendation is to have **no more than two live versions** of your API available at any one time, and to have a transition period of **no more than 6 months** before the full deprecation of the old version of the API.



Good API Versioning Practices

- Never release an API without a version.
- Use major versions, no minor versions.
- Give clients at least one cycle to react before deprecating a version.
- Once a new version is created, deploy all API proxies at once (a mismatch across APIs will create a lot of confusion).
- If a client tries to call a deprecated API then the HTTP error code should be 410 Gone (make this the blanket response for all old API versions).
- Ensure your versioning strategy works with your branching strategy.