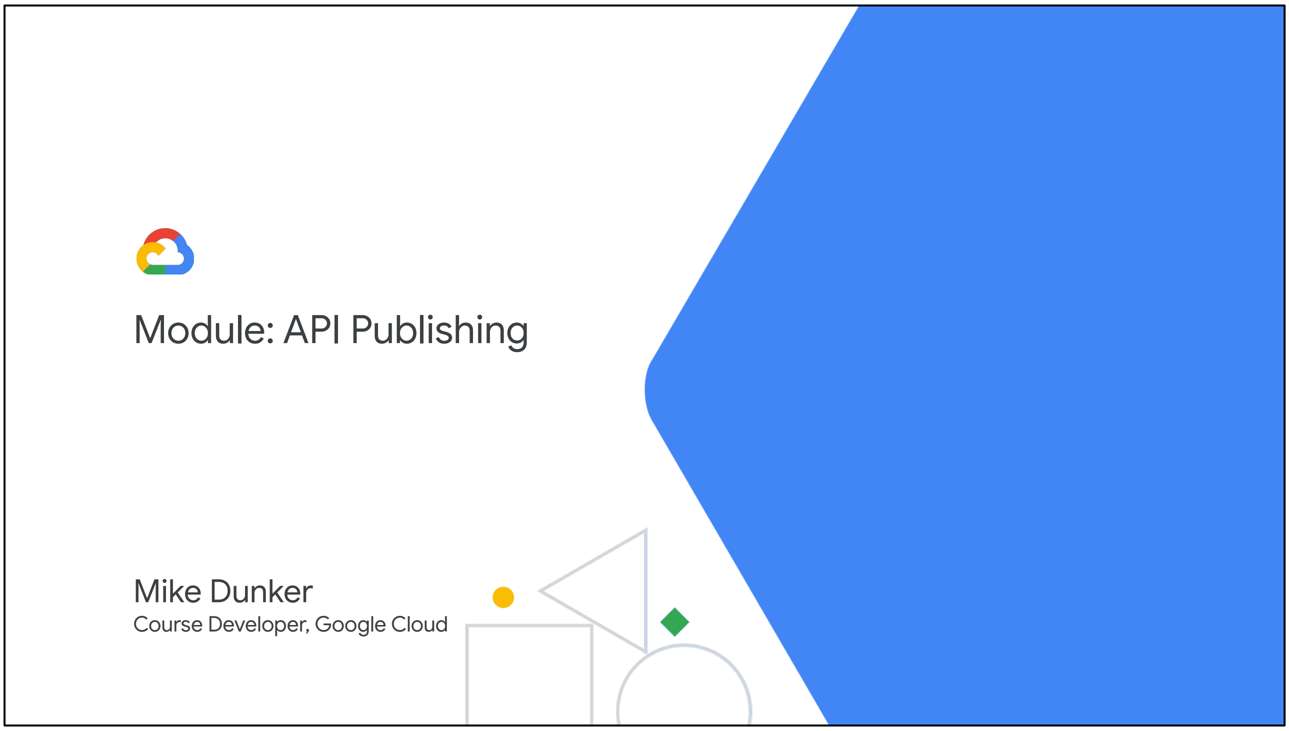


Module: API Publishing

Mike Dunker
Course Developer, Google Cloud

In this module you'll learn about API publishing and developer portals.

We use developer portals to publish API products and allow app developers to discover APIs and register apps to use them.

During a lab, you will create an integrated developer portal and publish your retail API product to the portal.

You will also learn about versioning REST APIs.

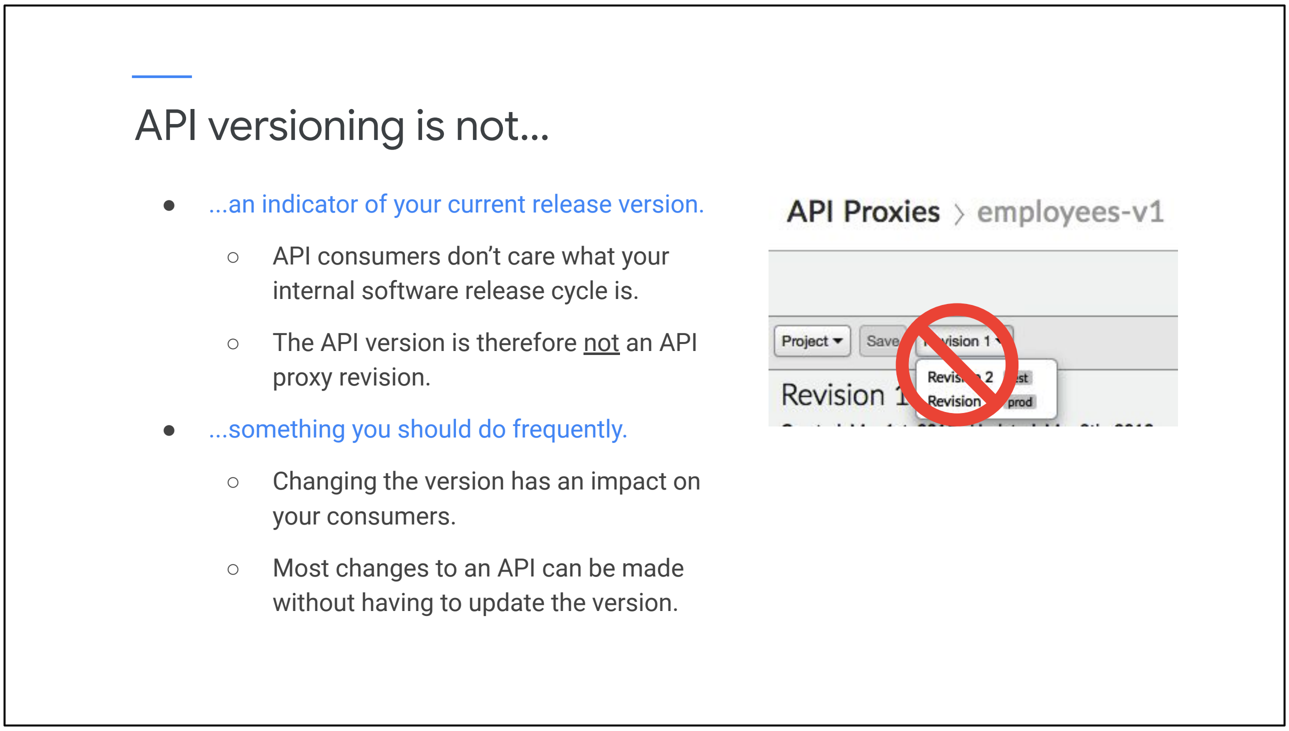
REST API Design Part III: Versioning

This lecture is the third of three lectures on REST API Design.

The first lecture introduced REST APIs and explained how to design RESTful APIs.

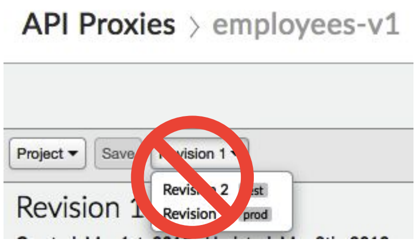
In the second lecture we discussed REST API status codes and responses.

In this lecture you will learn about API versioning.



API versioning is not...

- ...an indicator of your current release version.
  - API consumers don't care what your internal software release cycle is.
  - The API version is therefore not an API proxy revision.
- ...something you should do frequently.
  - Changing the version has an impact on your consumers.
  - Most changes to an API can be made without having to update the version.



Before we discuss what API versioning is, let's clear up some misconceptions by explaining what API versioning is NOT.

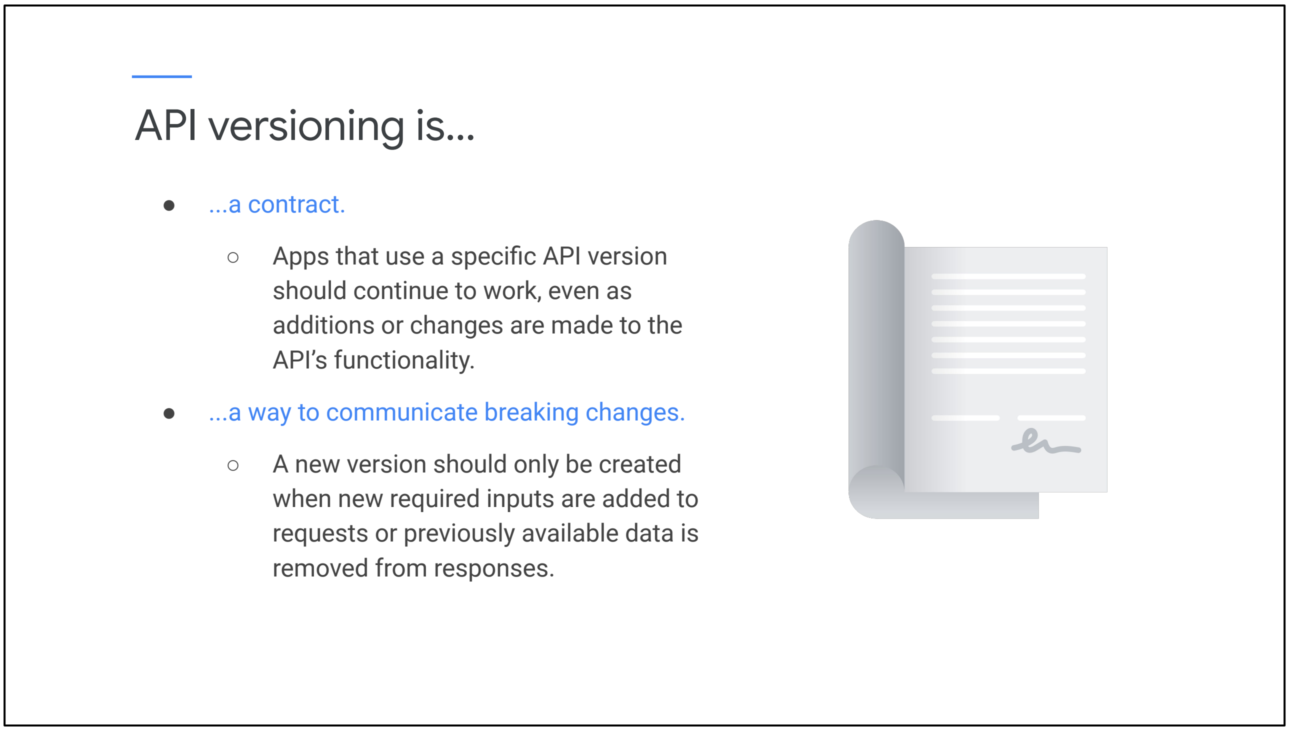
API versioning is not an indicator of your current release version.A release version tracks the software development or release cycle. This typically has no effect on the consumers of an API, if care is taken to make changes backward-compatible.

Therefore, the version of an API is not tied in any way to the proxy revision.

API versioning is not something you should do frequently.

Changing an API's version often requires that changes be made in the apps that consume your API. This means extra work for the app developer.

The good news is that most changes to an API can be made without changing the API version.



API versioning is...

- ...a contract.
  - Apps that use a specific API version should continue to work, even as additions or changes are made to the API's functionality.
- ...a way to communicate breaking changes.
  - A new version should only be created when new required inputs are added to requests or previously available data is removed from responses.

Here's what API versioning is.

API versioning is a contract.

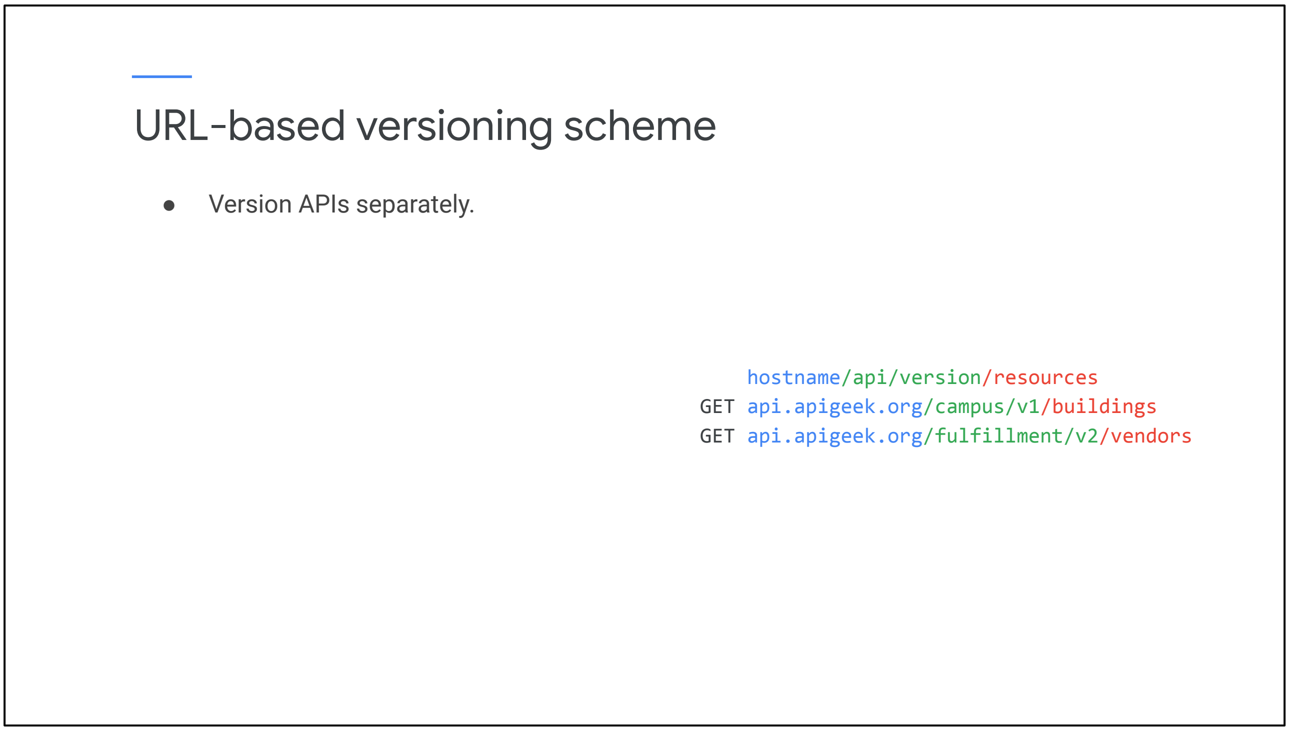
Apps written against a specific API version should continue to work, even if changes are being made to the API. This is especially important for mobile apps, because many users do not regularly update their apps.

API versioning is used to communicate breaking changes.

Breaking changes may be necessary when you add new required inputs to a request, or when data that was previously available is removed from responses. If these changes are being made due to a security issue with the previous design, the change is definitely necessary.

You can often find a way to add other functionality to an API without breaking the contract.

Remember, changing the version has an impact on your app developers, so only do it when necessary.



URL-based versioning scheme

- Version APIs separately.

```
    hostname/api/version/resources
GET api.apigeek.org/campus/v1/buildings
GET api.apigeek.org/fulfillment/v2/vendors
```

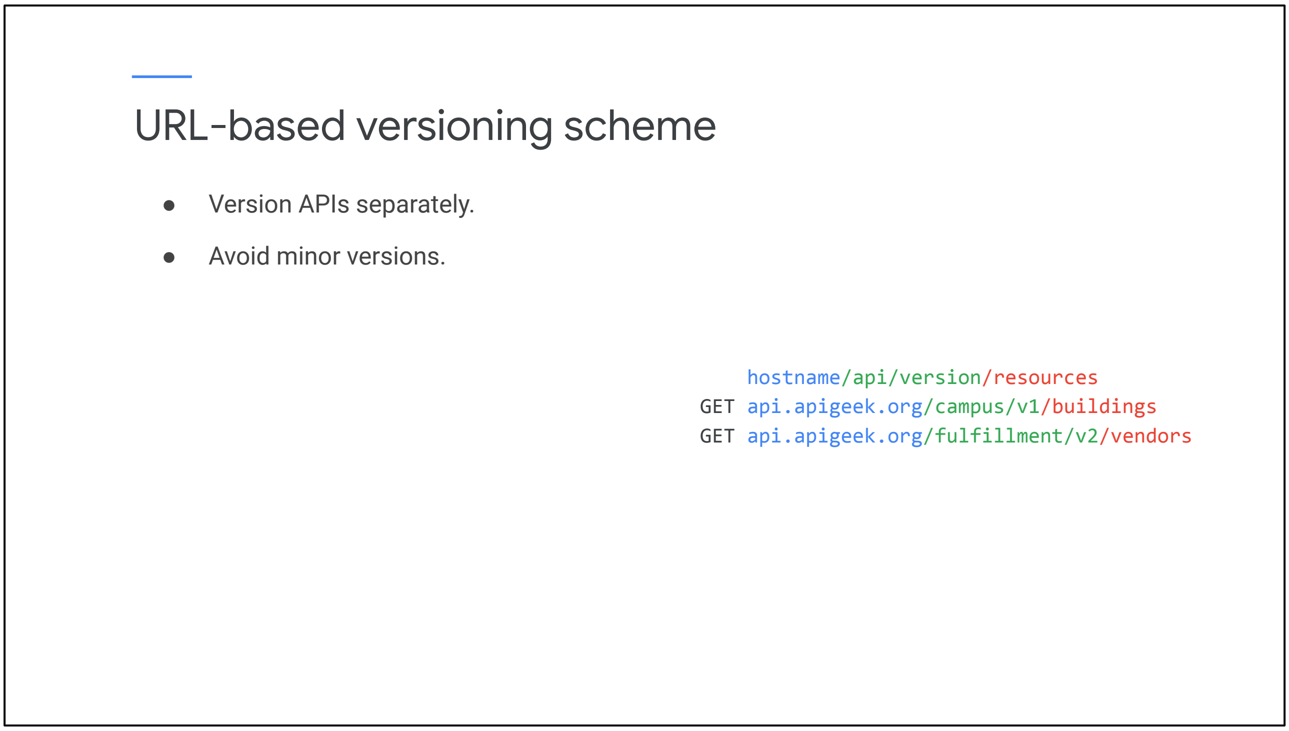
Let's take a look at a URL-based versioning scheme.

In this case, the API and version are specified immediately after the host name, and resources for the API follow the version number.

Versioning is typically done at an API level.

If you have multiple APIs, you should generally version them separately.

In this example, the campus API is version 1, and the fulfillment API is version 2.



URL-based versioning scheme

- Version APIs separately.
- Avoid minor versions.

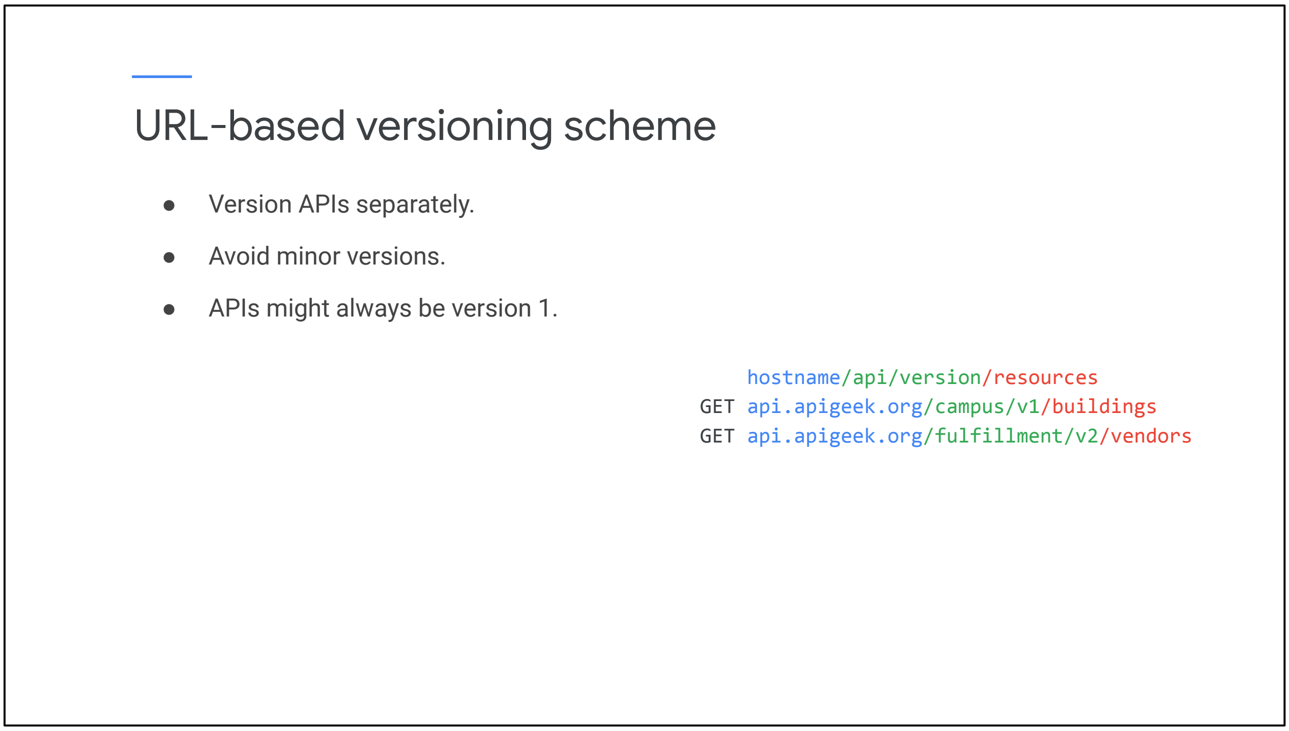
```
    hostname/api/version/resources
GET api.apigeek.org/campus/v1/buildings
GET api.apigeek.org/fulfillment/v2/vendors
```

You should avoid minor versions for your APIs.

Version changes are supposed to happen infrequently, and they typically have a major impact on applications.

You may be used to seeing version numbers that include both major and minor versions. For example, version 1.2 would have a major version of one, and a minor version of two.

Including minor versions in your version numbers implies that versions change frequently, or that version changes are made for small or incremental updates to the API.



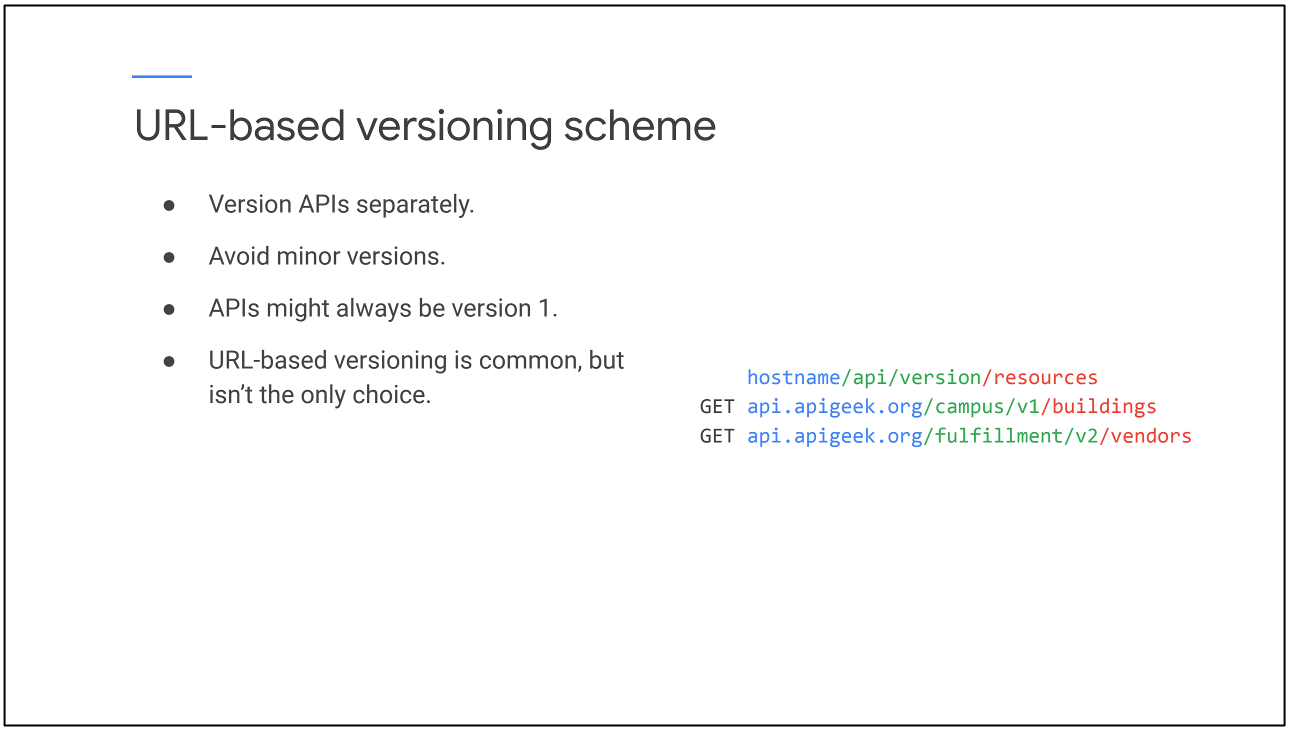
URL-based versioning scheme

- Version APIs separately.
- Avoid minor versions.
- APIs might always be version 1.

```
    hostname/api/version/resources
GET api.apigeek.org/campus/v1/buildings
GET api.apigeek.org/fulfillment/v2/vendors
```

APIs may never need to have version changes. For example, Apigee's management API is still version 1.

This doesn't mean that the management API hasn't changed. It means that the changes were made without breaking backward-compatibility.



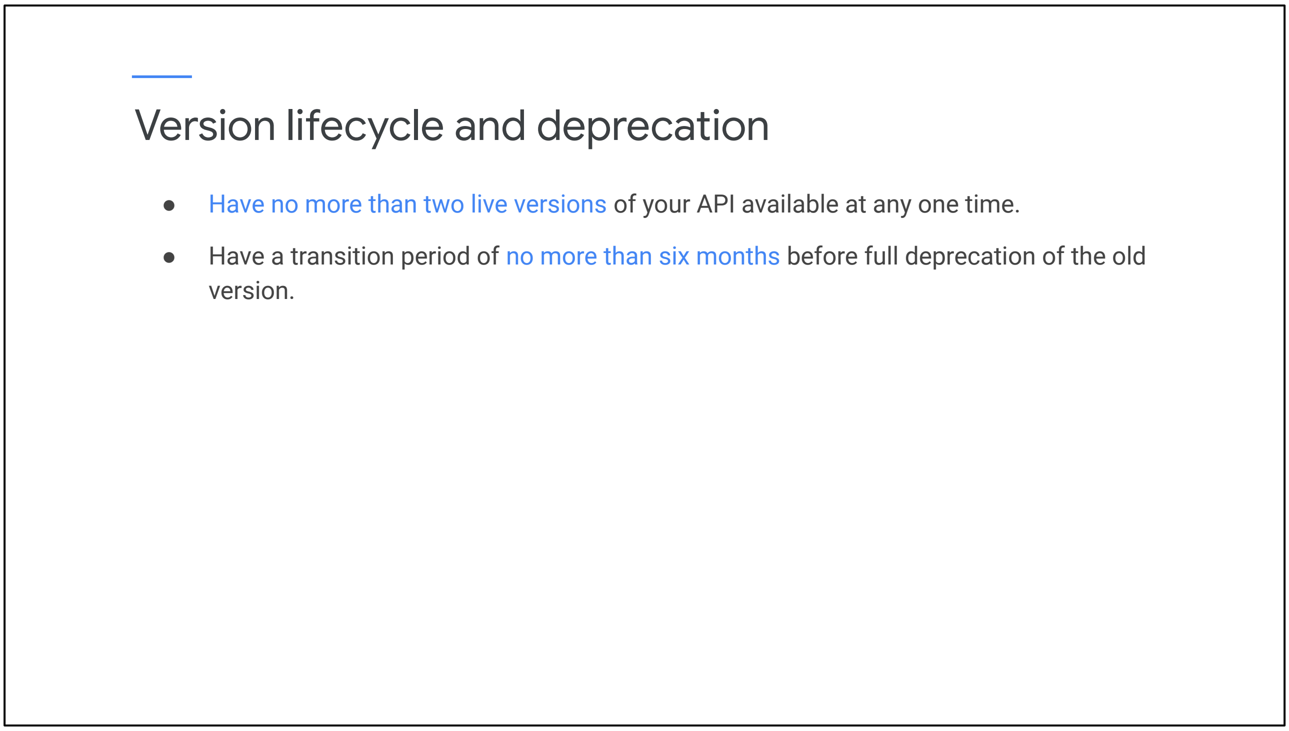
URL-based versioning scheme

- Version APIs separately.
- Avoid minor versions.
- APIs might always be version 1.
- URL-based versioning is common, but isn't the only choice.

```
    hostname/api/version/resources
GET api.apigeek.org/campus/v1/buildings
GET api.apigeek.org/fulfillment/v2/vendors
```

URL-based versioning is a common scheme. It is easy to see which version of an API you are calling, because it is specified in the URL.

This is not the only scheme, though. Whatever you decide, be consistent across your APIs.



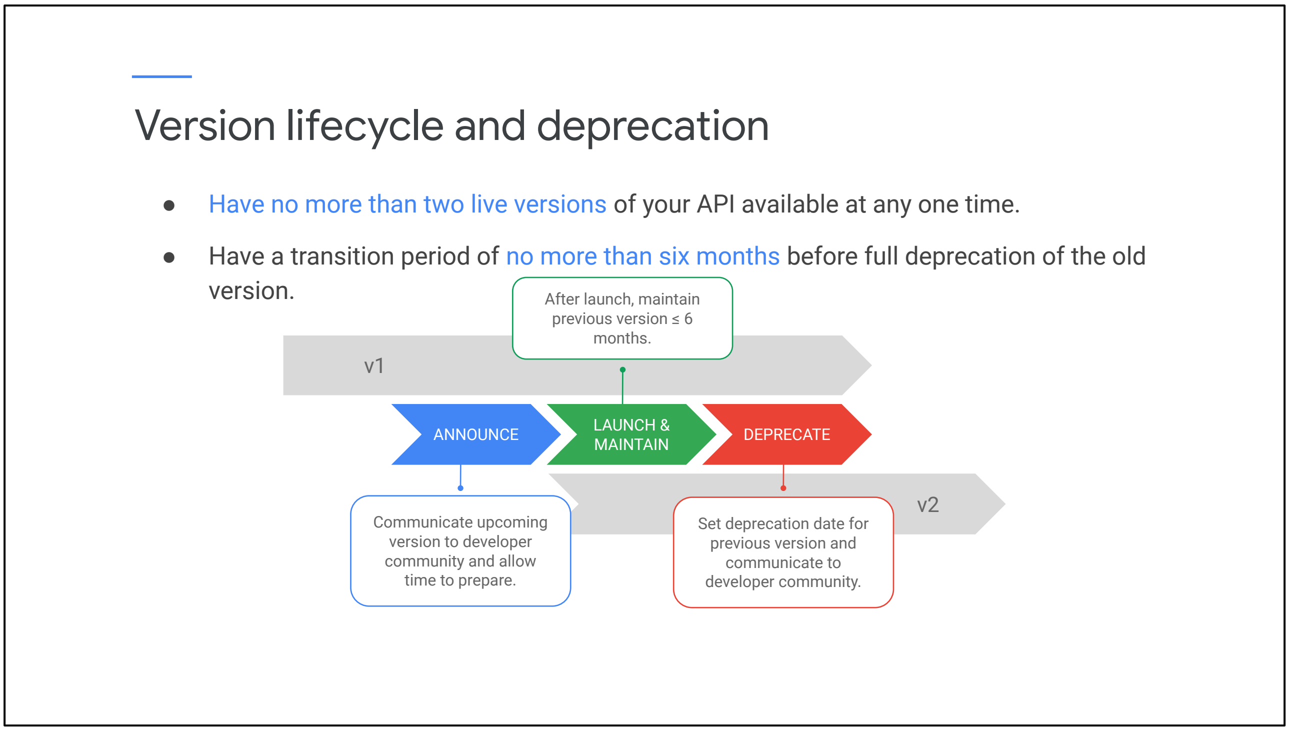
Version lifecycle and deprecation

- Have no more than two live versions of your API available at any one time.
- Have a transition period of no more than six months before full deprecation of the old version.

So how do we manage multiple versions of an API, and when should APIs be deprecated?

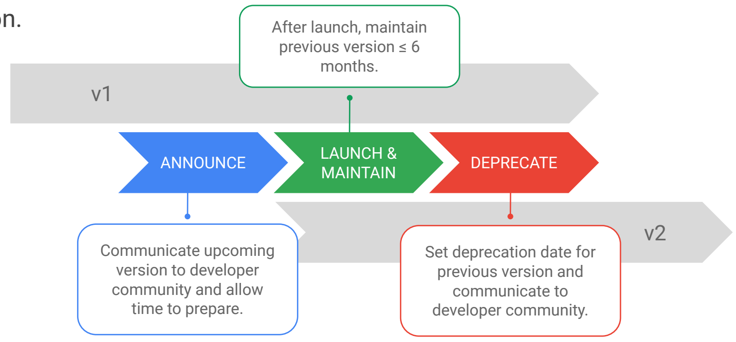
It is generally recommended to only have two live versions of your API available at any one time. Having too many active versions can be difficult to maintain, and gives a developer less incentive to upgrade to the latest version.

You should also try to have a transition period of no more than six months before the old version is deprecated. This should be enough time for the developer to make any necessary changes.



Version lifecycle and deprecation

- Have no more than two live versions of your API available at any one time.
- Have a transition period of no more than six months before full deprecation of the old version.


After launch, maintain previous version ≤ 6 months.
v1
ANNOUNCE
LAUNCH & MAINTAIN
DEPRECATE
v2
Communicate upcoming version to developer community and allow time to prepare.
Set deprecation date for previous version and communicate to developer community.

Here's how you might handle moving from API version 1 to version 2. The upgrade and deprecation process typically has 3 phases.

The *Announce* phase is when you communicate version 2 to the app developer community. You specify all of the changes for version 2, why the changes are being made, and when the new version will be available. If there are benefits for app developers in the new version, make sure to publicize those. You should also explain the process for upgrading to the new version.

Next is the *Launch and Maintain* phase. The new API version can be launched with a blog post. You should continue to market the new version, because some app developers may not have made any progress in upgrading to the new version. Any issues that arise when using the new version, or upgrading to the new version, should be handled quickly. Try to make the upgrade process as easy as possible. Developers should be confident that upgrading to version 2 is safe.

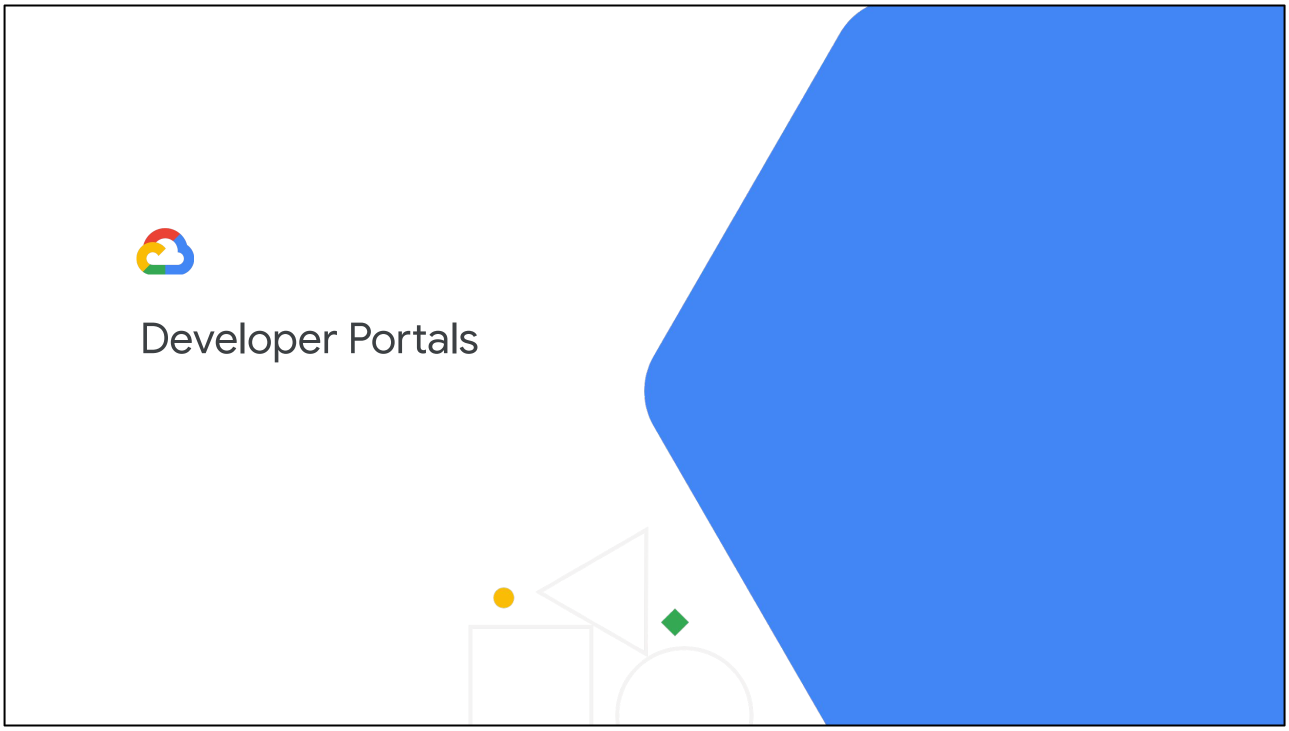
The third phase is *Deprecate*. When you determine the deprecation date for the previous version, it needs to be clearly communicated to the app developer community. Any significant issues should have been resolved during the Launch and Maintain phase.

As the deprecation date gets closer, you should use analytics to see which apps are still sending traffic to the old version, and reach out to developers to help them understand that their apps will stop working after the deprecation date if they are not upgraded.

When the deprecation date finally arrives, you can completely turn off version 1 of the API and change error messages to explain that calls should use version 2. You may instead choose to capture information about all apps that are still sending to version 1, and give the app developers a short grace period to fix their apps.

Eventually, you'll only have to maintain version 2.

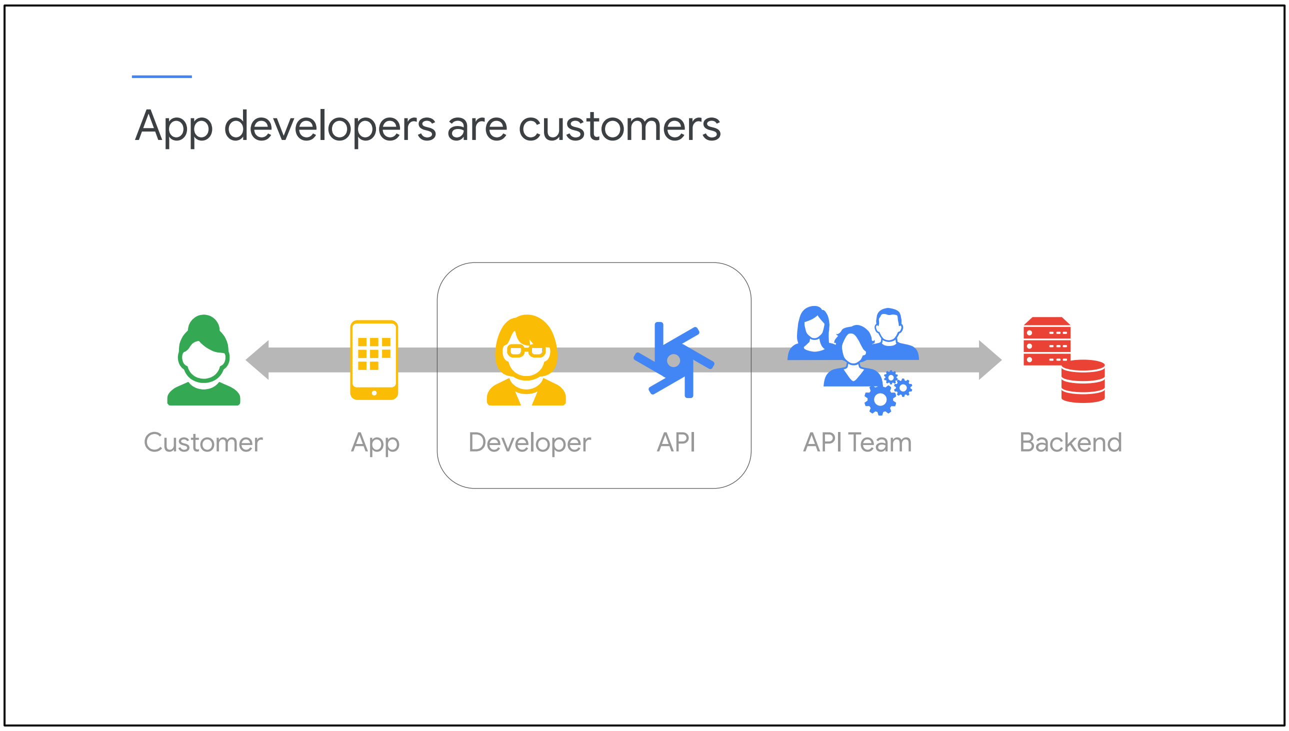
Remember, you should avoid changing the version when possible; most features can be added to an API without breaking backward-compatibility.



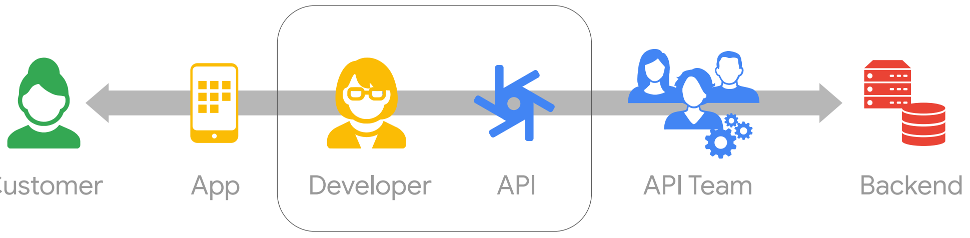
Developer Portals

An API is only as useful as the apps that use it.

Developer portals help app developers make apps that use your APIs.

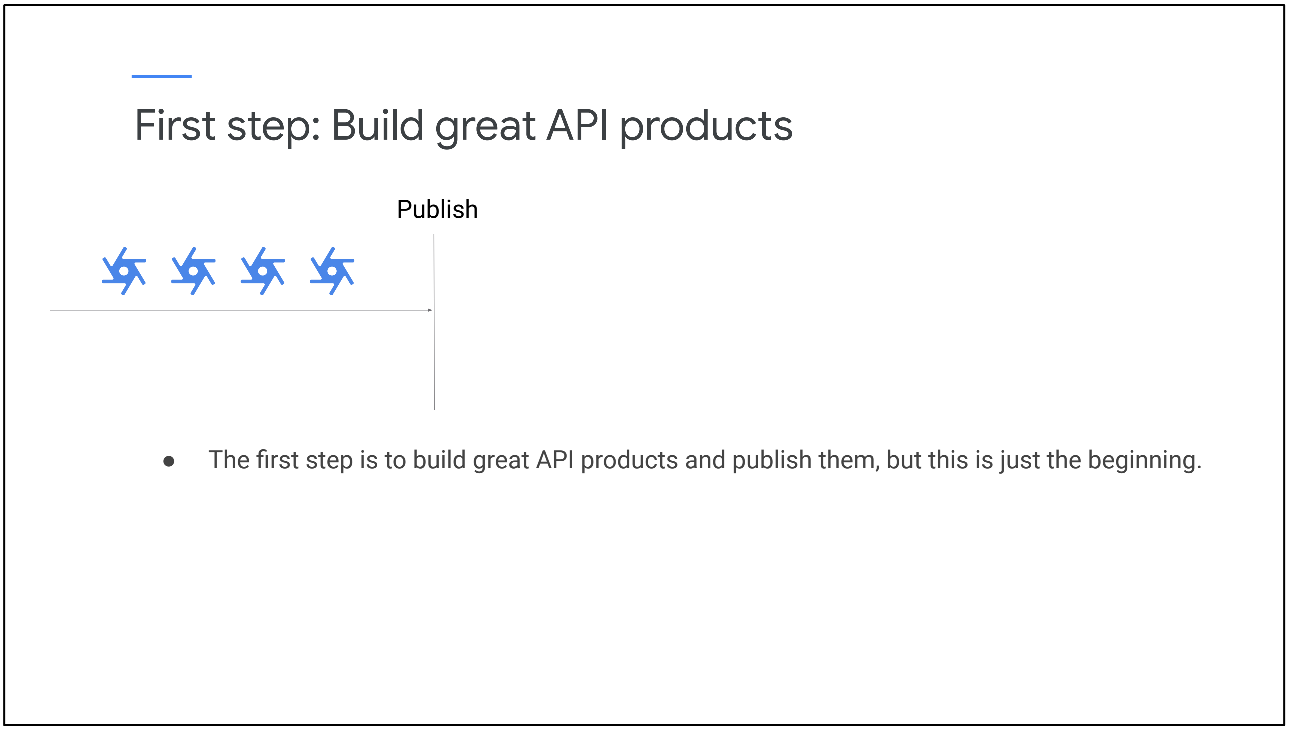


App developers are customers



Remember: we design our APIs with the app developer in mind. App developers are customers of our APIs.

Whether developers are internal, external, or partners, we want to help app developers use our APIs to build great apps.

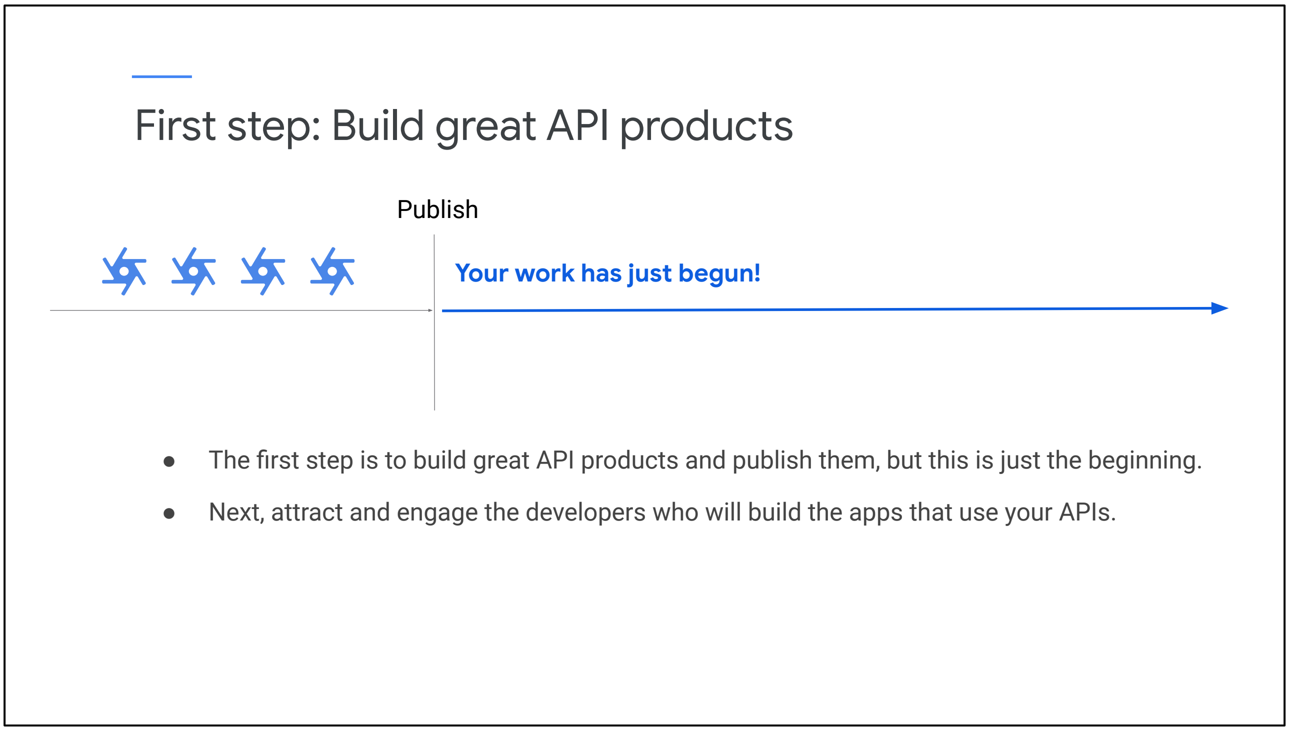


First step: Build great API products

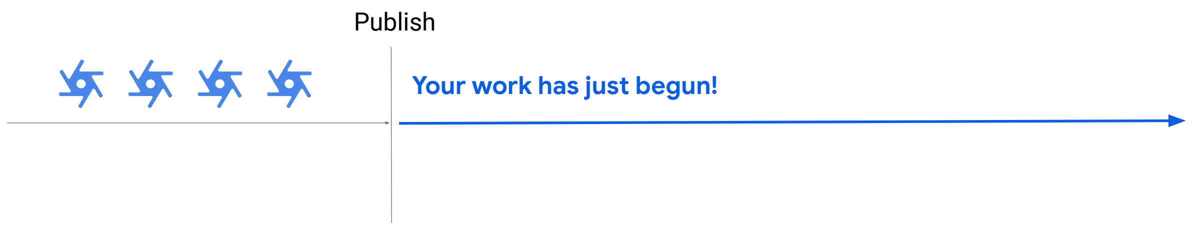
- The first step is to build great API products and publish them, but this is just the beginning.

The first step is to build great API products and publish them.

After your APIs are published, you still have more work to do.



First step: Build great API products

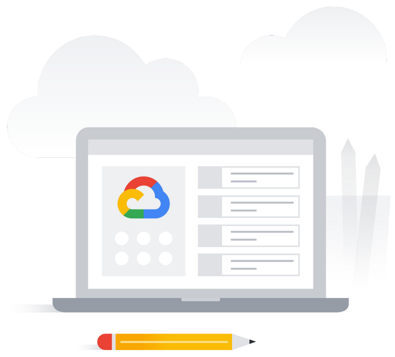


- The first step is to build great API products and publish them, but this is just the beginning.
- Next, attract and engage the developers who will build the apps that use your APIs.

Next, you have to attract and engage app developers.

This happens in the developer portal.

Developer portals enable app developers to...



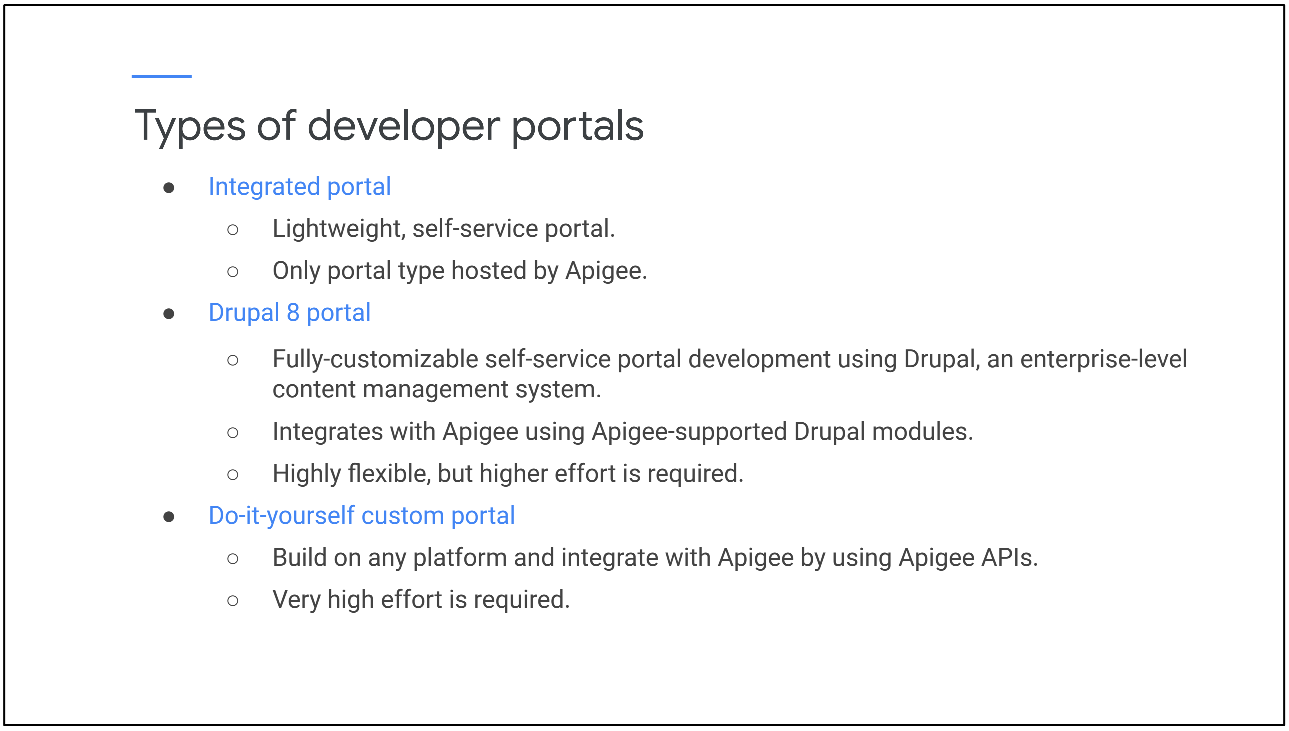
- Learn about your service offerings.
- Learn how to use your APIs by using comprehensive, live documentation.
- Register apps that will use your APIs.
- Minimize time to first app using the API.

The developer portal is where an app developer can learn about the APIs you are providing and any service levels for your APIs.

An app developer can use the live documentation in the developer portal to learn how to use your APIs, and even try them out. This live documentation is created by linking an OpenAPI specification to an API product that is published on the developer portal.

App developers can also use a self-service process to register their apps to be able to use your API products. Self-service allows developers to get started with your APIs as quickly as possible.

The rich documentation and self-signup minimizes the time it takes a developer to build an app using the API.



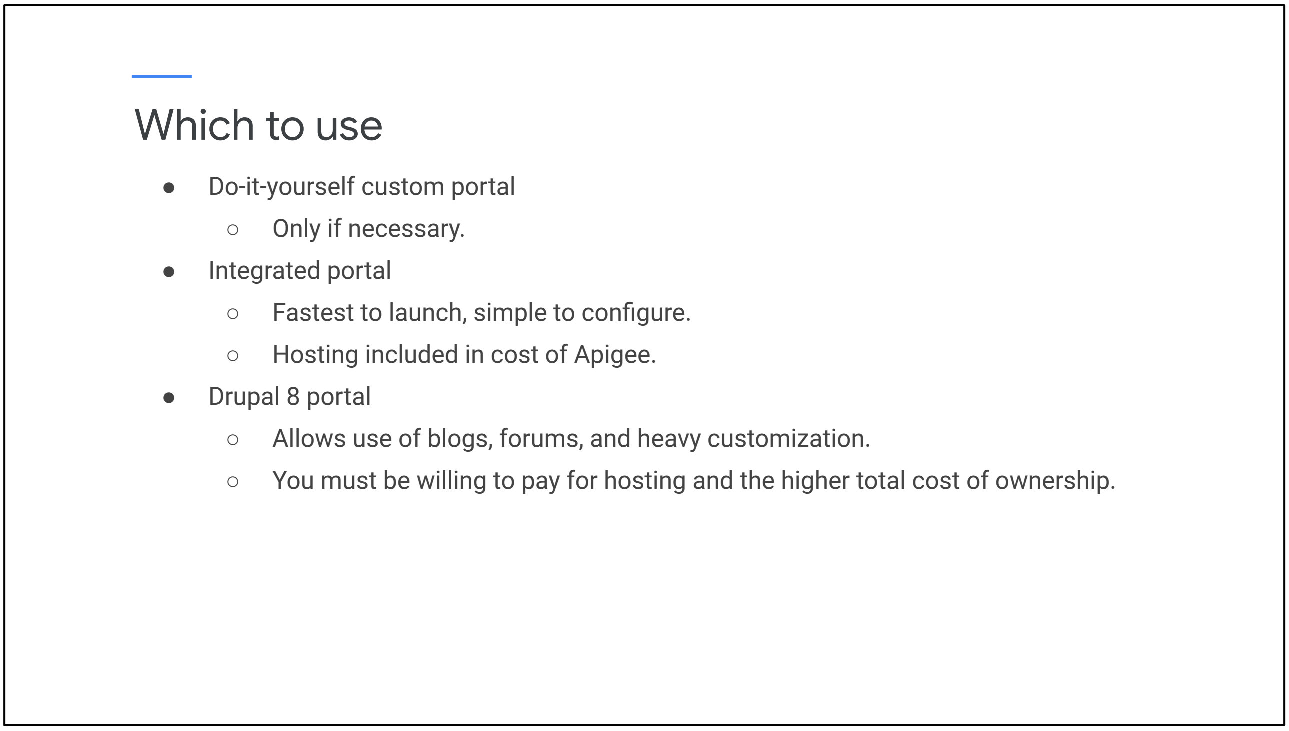
Types of developer portals

- Integrated portal
  - Lightweight, self-service portal.
  - Only portal type hosted by Apigee.
- Drupal 8 portal
  - Fully-customizable self-service portal development using Drupal, an enterprise-level content management system.
  - Integrates with Apigee using Apigee-supported Drupal modules.
  - Highly flexible, but higher effort is required.
- Do-it-yourself custom portal
  - Build on any platform and integrate with Apigee by using Apigee APIs.
  - Very high effort is required.

The first type of developer portal is the integrated portal. This is a lightweight, self-service portal that can be created and deployed quickly.

The second type of portal is the Drupal 8 portal. Drupal is an enterprise-level content management system. Apigee has created Drupal modules that integrate with Apigee and support a Drupal-based developer portal. The Drupal 8 portal allows self-service design, creation, and management of the portal. Drupal 8 portals are highly flexible, but there is significantly more effort required to develop and maintain them than for the integrated portal.

The integrated and Drupal 8 portals integrate with Apigee by using Apigee management API calls. If you need to, you can implement your own custom portal on any platform by using management APIs. However, creating a developer portal from scratch is a large, complex project, and requires a very high level of effort.



Which to use

- Do-it-yourself custom portal
  - Only if necessary.
- Integrated portal
  - Fastest to launch, simple to configure.
  - Hosting included in cost of Apigee.
- Drupal 8 portal
  - Allows use of blogs, forums, and heavy customization.
  - You must be willing to pay for hosting and the higher total cost of ownership.

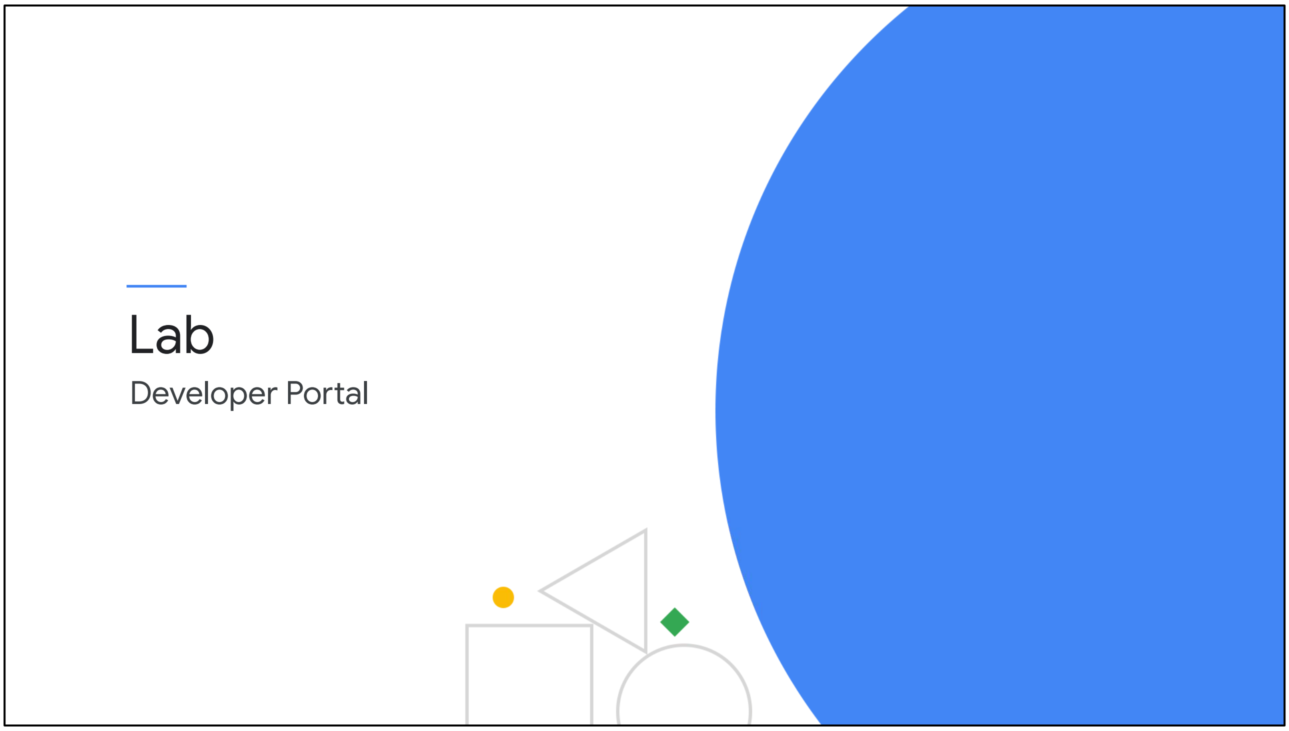
How do you decide which type of portal to use?

Typically, you won't choose to build your own portal. It requires a huge effort. You should generally only choose the custom portal if you absolutely have to integrate the portal with an existing site that is not based on Drupal.

The decision usually comes down to integrated versus Drupal.

The integrated portal may be used if speed to launch or cost are most important. The integrated portal is simple to configure, is the fastest to launch, and requires the least effort. Hosting of the integrated portal is included in the cost of Apigee.

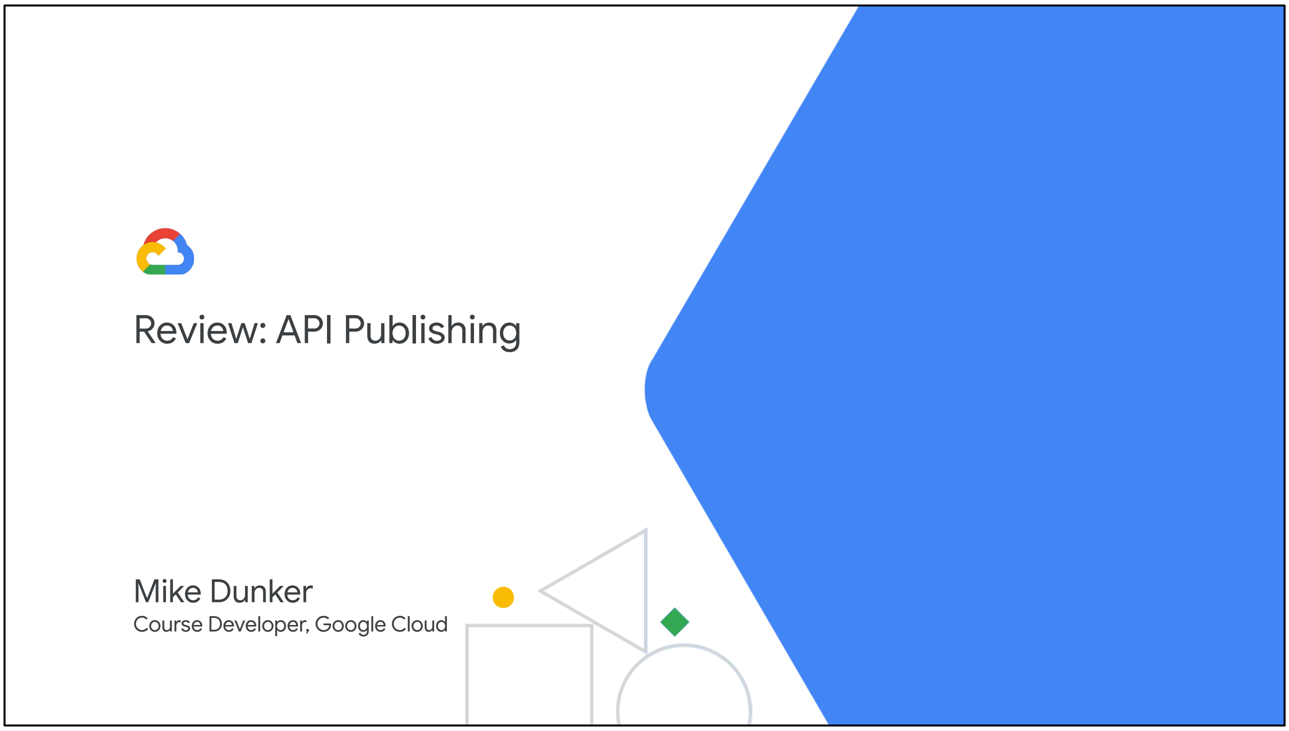
Use cases that are complex or require features like blogs, forums, or heavy customization typically require a Drupal 8 portal. You must be willing to pay for hosting the portal, and, more importantly, to absorb the higher total cost of ownership.



Lab

Developer Portal

In this lab, you take your retail API and publish it to a new developer portal. You will also use the OpenAPI specification to provide live documentation within the portal, which allows you to make API calls directly from the portal into your API.



Review: API Publishing

Mike Dunker
Course Developer, Google Cloud

During this module you learned about considerations when versioning your APIs.

You learned why developer portals are important, and created an integrated developer portal during a lab.

During this lab you added CORS to your API, and published your API product to the developer portal.

Finally, you were able to try out your API using live documentation in the developer portal.