

What's the Difference Between OpenAPI 2.0, 3.0, and 3.1?

There are currently two major OpenAPI releases, 2.0 and 3.0. Version 3.1 was announced on June 18, 2020 as an initial release candidate. In this post, we'll look at some of the key differences between OpenAPI 2.0, 3.0, and 3.1.



Janet Wagner

Sep 29, 2020

The [OpenAPI Specification](#) (OAS) is a widely adopted, standardized format for describing REST APIs. You can use OpenAPI to detail every part of your API, including endpoints, operation parameters, request responses, and authentication flows. The OpenAPI format is easy for both developers and machines to read and understand. What's harder to immediately decipher is how the versions differ.

At the time of publication, almost everyone will likely want to use OpenAPI v3.0, the latest official major version that was released in 2017.

CONTENTS

Differences Between OpenAPI 2.0 and 3.0

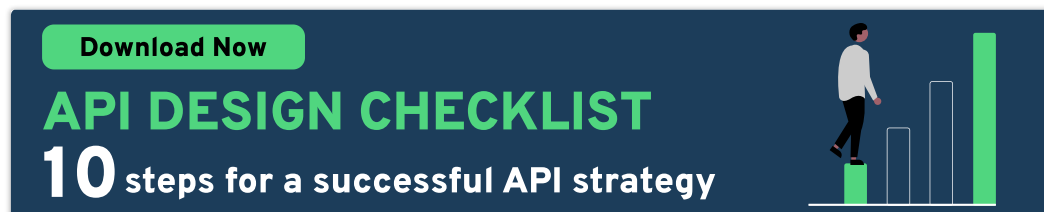
I. Specification Restructured to Increase Reusability

II. Extended JSON Schema Support

Some folks are on OpenAPI 2.0 due to being stuck there with tooling that still has not added support for OpenAPI v3.0. Some people might talk about OpenAPI v3.1, but whilst it is still in the “Release Candidate” stage there is essentially no tooling support for it.

However, for those digging into the nuances between versions, it's worth looking into the details.

There are currently two major OpenAPI releases, 2.0 and 3.0 (the latest release is 3.0.3). Version 3.1 was [announced](#) on June 18, 2020, as an initial release candidate. While it takes some important steps forward, it adds to the confusion over which version to use. In this post, we'll look at some of the key differences between OpenAPI 2.0, 3.0, and 3.1.



Download Now

API DESIGN CHECKLIST

10 steps for a successful API strategy

The banner features a dark blue background with a white silhouette of a person climbing a bar chart with four bars of increasing height. The text is in white and green.

Differences Between OpenAPI 2.0 and 3.0

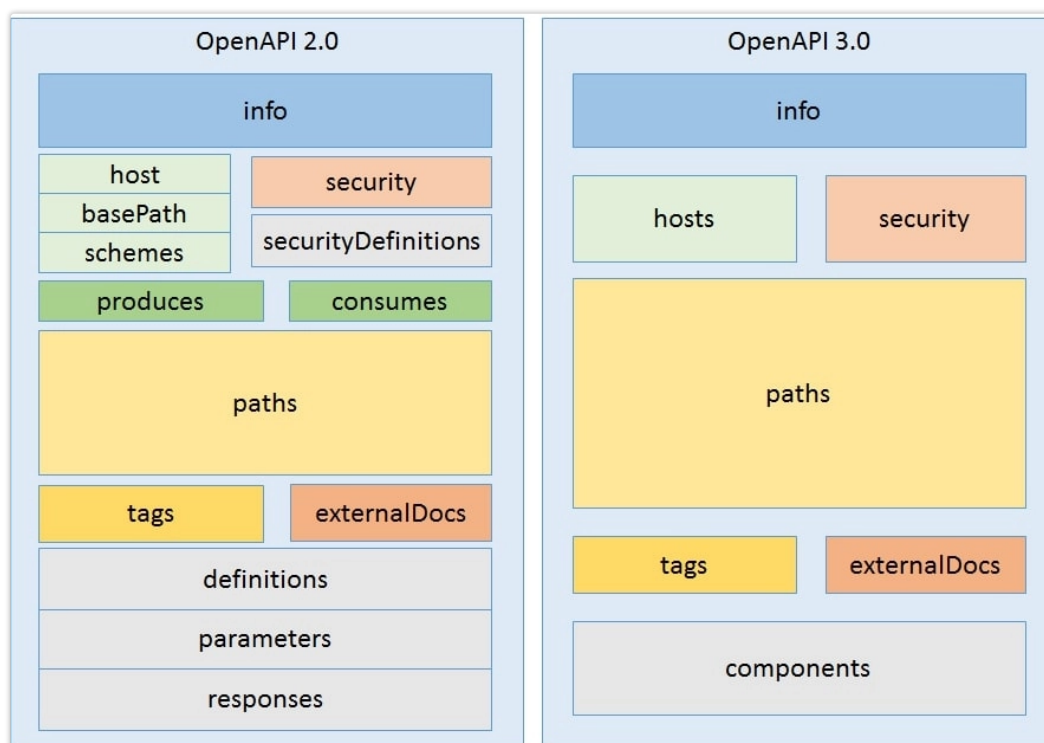
The two major versions of OpenAPI have the most significant differences, which come from their history. OpenAPI 2.0 was previously known as Swagger and is intended to replace it with backward compatibility. Once adopted as an open format, the community began working on OpenAPI 3.0, released in 2017. Let's

- III. Examples Overhauled for Easy Reusability
 - IV. Changes to Security Flows
 - V. Improved Parameter Descriptions
 - VI. Content Type Negotiation Support Added
 - VII. Support to Describe Callbacks
 - VIII. New OpenAPI Links Feature
- Differences Between OpenAPI 3.0 and 3.1
- I. Full JSON Schema Support
 - II. Semantic Versioning Dropped
 - III. Paths Are Now Optional
 - IV. Single “Example” Keyword Has Been Deprecated
 - V. Type Can Now Be An Array
 - VI. Nullable is No More
- One Platform to Explore OpenAPI

highlight some of the significant changes made to OpenAPI components in version 3.0.

Specification Restructured to Increase Reusability

OpenAPI 3.0 introduces a reorganized document structure so that it's easier to write and reuse API definitions. This simplified document structure moves securityDefinitions, definitions, parameters, and responses to **components**.



Version 3.0 includes the new *Components Object*, which contains reusable objects, such as examples, parameters, responses, request bodies, links, security schemes, and schemas. The *Components Object* also contains headers and callbacks. In version 2.0, you could describe headers but couldn't reuse them as easily as in version 3.0. Support for describing callbacks is a new feature in version 3.0, which we cover later in this section.

Extended JSON Schema Support

The 3.0 release includes extended support for JSON Schema, which means you can use more JSON Schema keywords than with version 2.0. Some keywords supported in version 3.0 are handled slightly differently than in JSON Schema, including:

- `oneOf`
- `anyOf`
- `allOf`

OpenAPI 2.0 does not support the `oneOf` or `anyOf` keywords, but you *can* use these keywords with version 3.0. For example, you might want to use `oneOf` or `anyOf` to describe an API request or response with a few alternative schemas (built on the concept of [polymorphism](#)).

Examples Overhauled for Easy Reusability

Examples are a handy OpenAPI feature you can use for a wide range of purposes. For instance, you could create a [mock API server tool](#) that uses `examples` to generate mock requests. In version 3.0, `examples` have been completely overhauled so that you can easily reuse them.

In previous versions, you could only describe `examples` using a JSON or YAML object. With version 3.0, you can describe multiple `examples` of any format using a JSON string. Also, `examples` can be placed within objects, properties, parameter descriptions, request

bodies, and response descriptions. You can also now reference external examples.

Changes to Security Flows

OpenAPI 3.0 supports more security schemes and bearer formats than version 2.0. Plus, the way you describe security flows has changed. Version 3.0 includes a new `http` type, which covers *all* HTTP security schemes. Additionally, the `basic` type has been renamed to `http`.

Support for OpenID Connect Discovery was added to the specification in the 3.0 release, and the type is `openIdConnect`.

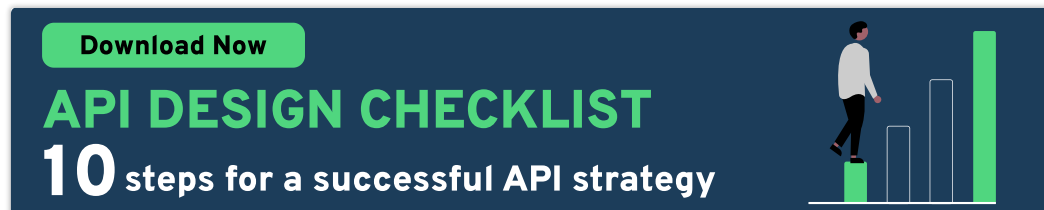
To better match the OAuth 2.0 specification, OAuth 2.0 flows in OpenAPI 3.0 have been renamed and restructured slightly. For example, “application” has been renamed as “clientCredentials,” and “accessCode” has been renamed as “authorizationCode.”

Improved Parameter Descriptions

Version 3.0 includes improvements to parameter descriptions. The `body` and `formData` parameter types have been removed and replaced with `requestBody`. The specification supports arrays and objects in operation parameters, and you can specify the method of serialization. Operation parameters include path, query, header, and cookie.

A new `cookie` parameter type was added in version 3.0 so that you could describe APIs that use cookies. Cookie authentication is *not supported* in version 2.0, so you have to find workarounds for the

limitation, usually through request header objects. With OpenAPI 3.0, you can send API keys via cookie, although this is not a best practice. You have better options for transmitting API keys, such as authentication with API tokens and OAuth.



Content Type Negotiation Support Added

The 3.0 release features a `requestBody` that supports content negotiation so that you can define different schemas and examples for various media types. The `requestBody` allows web services to accept and return data in different formats, such as images, plain text, XML, and JSON. It also allows you to provide one or multiple examples.

In OpenAPI 2.0, mimetypes supported by the API are defined in the “consumes” and “produces” sections. Consumes and produces have been eliminated in version 3.0. They’ve been replaced by a content object that maps the supported media types to their schemas. In OpenAPI 3.0, you can use wildcards for media types. For example, you could apply a wildcard that represents all image types: `image/*`

Support to Describe Callbacks

Callbacks allow you to describe asynchronous APIs where communication happens in real-time. An [example of callbacks in](#)

real-life would be shipment alerts—when you’ve subscribed to receive alerts about a package shipped by a logistics company. You get notifications in real-time from the logistics API informing you “the package is in transit” or “the package has been delivered.”

```
callbacks:
  notification:
    '{$request.body#/url}?token={$request.body#/token}':
      post:
        summary: 'Receive single notification about a parcel'
        requestBody:
          required: true
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/notification'
        responses:
          200:
            description: 'Notification successfully processed'
            content:
              text/plain:
                examples:
                  ok:
                    value: 'ok'
```

OpenAPI 2.0 doesn’t provide a way to describe callbacks, but in version 3.0, a new callback object was added that allows you to define asynchronous APIs and webhooks. You can define the format of the API operation as well as the callback messages and their expected responses. With OpenAPI 3.0, you can describe callback messages like in our logistics API example above.

New OpenAPI Links Feature

Links are another new feature in version 3.0. With Links, you can describe the relationships between API responses and operations, and then provide a traversal mechanism between them. Links allow you to define what will happen next with each API endpoint. The concept of Links is similar to hypermedia APIs (HATEOAS). However, this feature was not explicitly created for describing hypermedia APIs.

One way you could use Links would be to describe API endpoints that retrieve product details for different `productIDs`. For each `productID`, you would use Links to describe how to expand the information, and clients could follow each link automatically. You could also use Links for pagination. For example, if you are designing an API that will retrieve an extensive list of products, you would want a way to paginate the list automatically. You could use Links to explain how to navigate the page results, e.g. 50-99, 100-149.

Differences Between OpenAPI 3.0 and 3.1

Moving from the legacy Swagger description format of OpenAPI 2.0 to 3.0 obviously brought sweeping changes. While OpenAPI 3.1 is considered a minor release, there are some significant differences between version 3.1 and 3.0, especially related to [JSON Schema in OpenAPI](#).

The changes were big enough that the community debated whether to refer to this release candidate as OpenAPI 4.0. In this section, we'll highlight some of the differences in what will become OpenAPI 3.1.

Full JSON Schema Support

Version 3.1 is *100% compatible* with the latest draft of [JSON Schema, version 2019-09](#). OpenAPI 3.1 supports **ALL** JSON Schema Keywords, so if the keyword exists in the JSON Schema vocabulary, then you can use it with OpenAPI 3.1. The move to full JSON Schema support has created some minor [breaking changes](#).

Semantic Versioning Dropped

Semantic versioning (semver) was introduced in OpenAPI 3.0. The goal of semver is to communicate when API updates include breaking changes and when updates are backward compatible. Because of the minor breaking changes in version 3.1, there was some debate among the members of the OpenAPI Technical Steering Committee (TSC) about the continued use of semver. Continuing to use semver would require that the next OpenAPI release number be 4.0.0 instead of 3.1. So, in the end, the TSC decided to drop semver.

Paths Are Now Optional

Paths are resources that an API exposes. They include relative endpoints, operations, and responses. In 3.1, **paths** became optional to allow API descriptions that only include webhooks. The

change allows for flexibility, which fits OpenAPI's goal to define any REST API. This feature builds on the callback support of OpenAPI 3.0, which helped describe event-driven APIs. Most OpenAPI files will still have paths, but now they won't be required.

Single "Example" Keyword Has Been Deprecated

OpenAPI 3.0 includes a single Schema Object `example` keyword:

```
type: string
example: squirtle
```

The single `example` keyword has been deprecated in OpenAPI 3.1. This change resolves a [discrepancy](#) between OpenAPI and JSON Schema. JSON Schema includes an `examples` keyword which is an array, so you can list `examples` as a string:

```
type: string
examples:
- squirtle
- charmander
```

OpenAPI 3.0 *does* allow you to add an "example" to an array. But arrays and items with arrays don't support multiple `examples`; they only support the single `example` keyword.

Type Can Now Be An Array

In previous versions of OpenAPI, `type` could only be a **single string**. But in JSON Schema, `type` can be an **array of strings**.

There is no workaround for this in version 2.0, but in version 3.0, you can select multiple types using `oneOf` :

```
oneOf:  
  - type: string  
  - type: integer
```

With OpenAPI 3.1, the specification now supports `type` as an array:

```
type: [string, integer]
```

Nullable is No More

Neither OpenAPI 2.0 nor 3.0 support `null` as a type, but JSON Schema *does* support type `null` . OpenAPI 3.0 includes the field name `nullable` , which you can set to `true` if you want the value to be `null` :

```
type: string  
nullable: true
```


However, support for type `null` has been added in version 3.1, and `nullable` *has been removed*.

```
type: [string, "null"]
```

[Download Now](#)

API DESIGN CHECKLIST

10 steps for a successful API strategy



One Platform to Explore OpenAPI

We've only scratched the surface of the differences between OpenAPI versions 2.0, 3.0, and 3.1. You can use Stoplight to build quality APIs faster, without remembering the OpenAPI format details. Get started with our [API design tools](#), then connect your Git repositories to gain visibility into all the elements of your API program.

Read how world's leading API first companies are solving API Design Management at Scale.

[Get the API Design Guide](#)