Crafting Effective Microservices in Python

The API-first approach combined with Connexion are powerful tools to create effective microservices.

Rafael Caricio
Software Engineer

Posted on Dec 01, 2016

Tags: APIs, Connexion, Microservices, Open Source, OpenAPI, Python

TL;DR: The API-first approach combined with Connexion are powerful tools to create effective microservices. The use of API-first brings the benefit of creating APIs that fulfil the expectations of your clients. Besides that, using Connexion will help you develop APIs in Python in a smooth manner.

Google's acquisition of Apigee emphasises how important APIs are in today's architecture of applications. Using microservices, with well crafted APIs, is crucial for maintaining a successful business, as it simplifies the development of complex software solutions.

Growing companies experience a natural rise of their businesses' complexity. This complicatedness comes from market changes and catering to customer demands. Companies such as Zalando, which are faced with such challenges, have chosen to adopt microservices. This aims to make it easier to build and maintain their applications.

Microservices are a style of breaking up and organizing a complex software solution into smaller composable services. Those smaller services can be independently maintained and deployed. Each service is developed around a business capability that usually provides a REST API. Clearly stating what capability each service provides can have a huge impact on the productivity of the teams working together to build and maintain their microservices.

Microservices developers are expected to provide a description to client-side developers, who build and maintain the web, mobile or other services, of how to use their REST APIs. A failure to describe an API in a clear manner or maintaining updated documentation leads to broken system designs and the frustration of team members. The API-first approach is being established as a solution for those problems.

API-first approach

The API-first approach has been recently added to the original 12 factors of app development methodology—a set of rules and guidelines for developing robust applications that a group of experienced developers came up with while building the Heroku platform. It is an approach that focuses on having API specifications as a first-class artifact of the development process and using them as a contract to be shared among software developers and teams. Building the specification of a service upfront facilitates discussions among possible consumers of the API, creation of mocks of the API, and generates documentation, even before you write the first line of code.

Google, IBM, Microsoft, and other companies have come together to create the Open API initiative (OAI), to support the definition and establishment of a vendor independent format for describing REST APIs known as Open API Specification, formerly named Swagger 2.0. There are other formats available for describing APIs that can be used with an API-first approach, the most well-known being API Blueprint and RAML. However, the Open API format currently holds a larger community of users and supporters. From a specification written using the Open API format, the initial code for the implementation can be easily generated. There is support for many languages and frameworks: Ruby, Java, Node, C#, etc. For Python, Connexion is the best choice in API First development, since it relies neither on code generation nor needs boilerplate code.

Connexion is an open source framework built on top of Flask that facilitates the development of microservices in Python following the API-first approach. It was created by Zalando to meet the in-house demand for such solutions. Connexion is in active development; I am one of the primary maintainers of the project, along with Joao Santos and Henning Jacobs.

Building a simple service in Python

The first step to building an effective microservice in Python is to describe the resources that are going to be available in our API using the OpenAPI Specification. We will focus on describing what routes, parameters, payloads, and which response codes our API produce. We are starting with a simple example of an endpoint that responds with the string “Hello API!”.

```
# `my_api.yaml` file contents
swagger: '2.0'
info:
  title: Hello API
  version: "0.1"
paths:
  /greeting:
    get:
      operationId: api.say_hello
```

```
      summary: Returns a greeting.
      responses:
        200:
          description: Successful response.
          schema:
            type: object
            properties:
              message:
                type: string
                description: Message greeting
```

In the code snippet above, we specified that our API has an endpoint “/greeting” that accepts requests with method “GET” and returns a status code 200, which represents success. Notice that the business logic is not defined in our specification – this part is left to the implementation of our endpoint, which will be done in Python. The “operationId” is what defines what Python function should be executed when the API call is made to your endpoint.

```
# `api.py` file contents
def say_hello():
    return {"message": "Hello API!"}
```

As you can see, Connexion API handlers are free of boilerplate code. Normal Python functions that return a simple data structure are used as handlers for the API calls. Data structures returned by the handlers may be validated against the API specification by Connexion (if specified so). This validation is disabled by default to grant flexibility during development. Now we can use Connexion to glue our code to the API specification and get a server running. The easiest way to run our API is using the Connexion CLI tool:

```
# run this command in the same directory where you saved the previous files
$ pip install connexion # installs connexion, run only once.
$ connexion run my_api.yaml -v
```

Now we can go to the browser and access the address http://localhost:5000/greeting, where we should be able to see the message:

```
{"message": "Hello API!"}
```

To make it more dynamic, we can change our specification to add a HTTP parameter for the user’s name. For that our specification should look like this:

```
# `my_api.yaml` file contents
swagger: '2.0'
info:
  title: Hello API
  version: "0.1"
paths:
  /greeting:
```

```
    get:
      operationId: api.say_hello
      summary: Returns a greeting.
      parameters:
        - name: name
          in: query
          type: string
      responses:
        200:
          description: Successful response.
          schema:
            type: object
            properties:
              message:
                type: string
                description: Message greeting
```

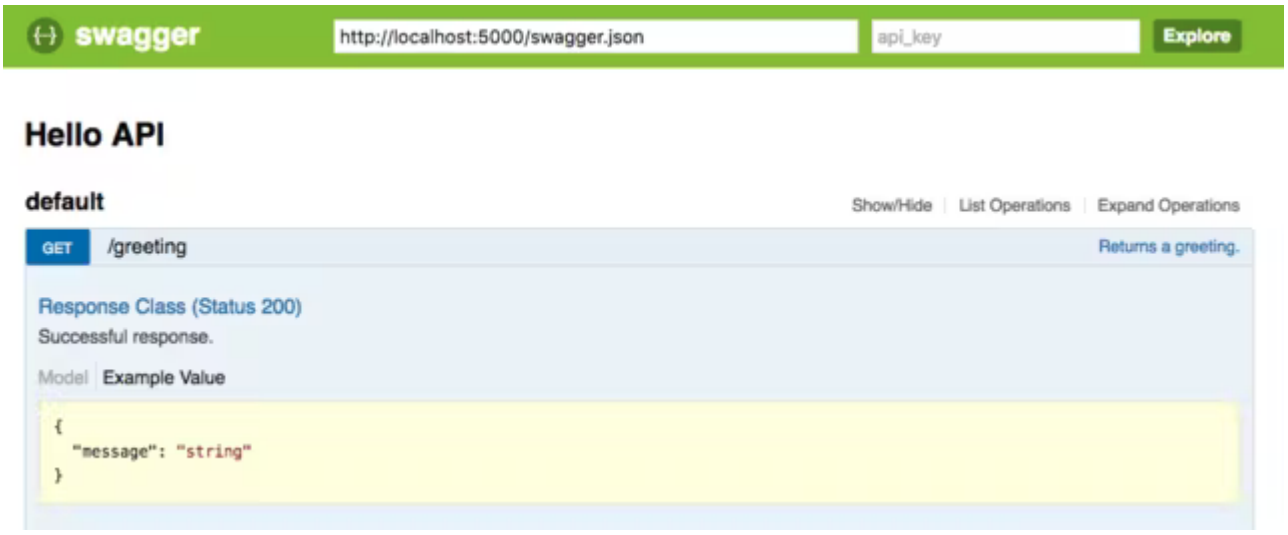
Now Connexion will pass an optional parameter to our function. To handle that we also have to change our Python function.

```
# `api.py` file contents
def say_hello(name=None):
    return {"message": "Hello {}, from API!".format(name or "")}
```

Our Python code now matches the API specification. Now we can restart our server, press Ctrl+C, and run the following command again to see these changes in play:

```
# run this command in the same directory where you saved the previous files
$ connexion run my_api.yaml -v
```

Once again our server will be listening on http://localhost:5000/greeting. Additionally, it is now possible to pass an optional parameter “name”. Connexion includes an API console interface, which is available at http://localhost:5000/ui. This console UI is enabled by default and makes it easy to call the endpoints of our API. It also serves as documentation of our microservice.



Now we know the basics to create a Connexion application. For a more detailed example, please check the Connexion example project available on GitHub. The official Connexion documentation is a complete resource of information regarding its functionalities. This documentation should be used as a reference when developing RESTful microservices using Connexion.

Moving forward

Companies are starting to realize the benefits of the API-first approach. Etsy has published a blog post describing how the API-first approach is helping them tackle challenges with providing a consistent API. At Zalando, API-first is at the core of our software development lifecycle, with a process for peer review feedback and guidelines for creating RESTful microservices, available as open source.

We can find many open source tools that support API-first approaches available for diverse languages. Connexion is the perfect framework for a clean implementation of the API-first approach in Python and is in active development. Connexion does not rely on code generation. This leaves you free to evolve your API specification without breaking the code you have already implemented.

If you have any issues with Connexion, please feel free to post your problem to our issue tracker. You are also very welcome to send a pull request. We already have many contributors and you can join by improving the documentation or the code. If you'd like to get in touch, you can find me on Twitter at @rafaelcaricio.
