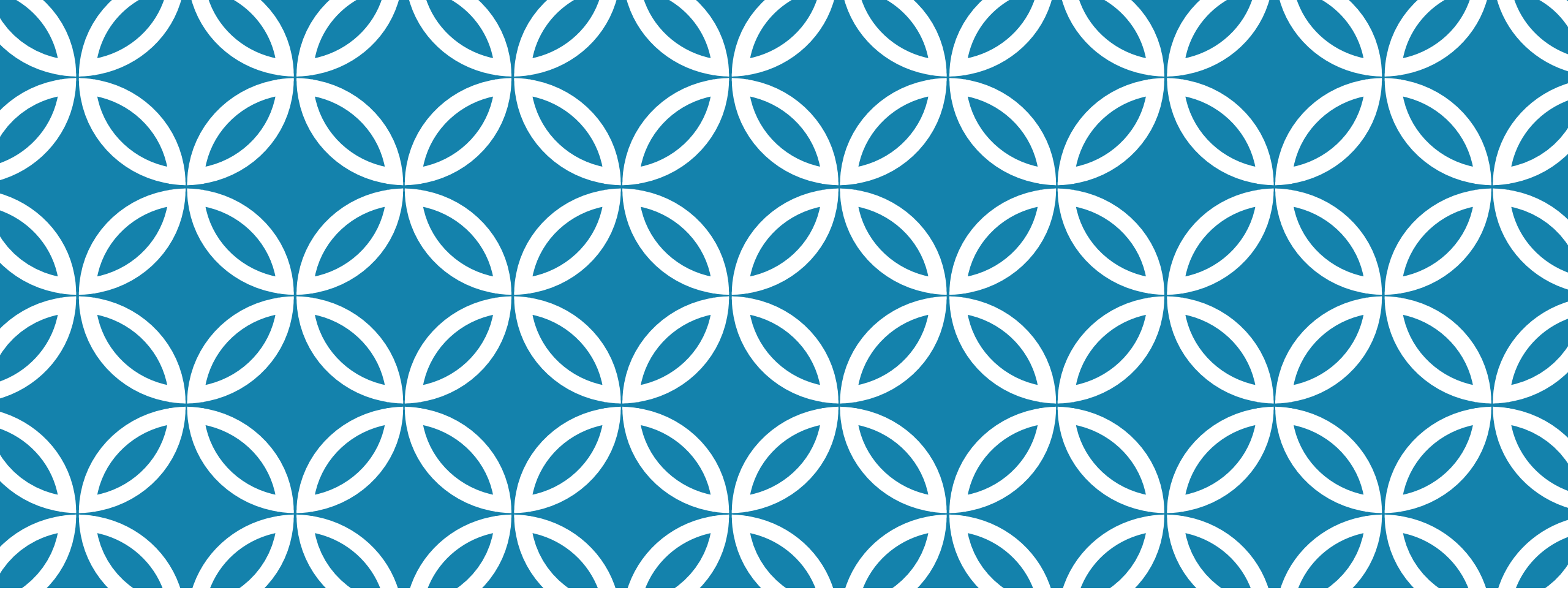


PANORAMIC VIEW OF WEB API

- Karen Immanuel

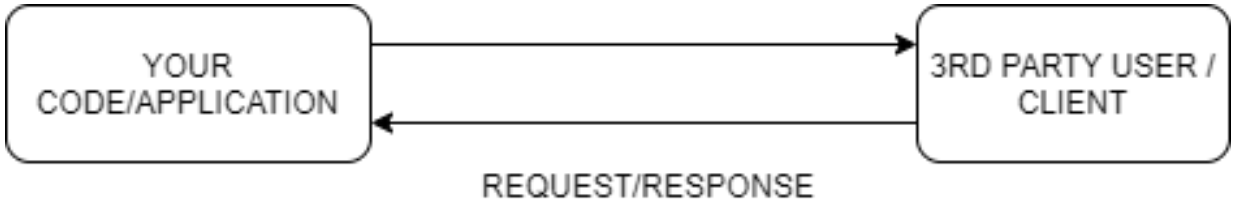
WHAT IS AN API?

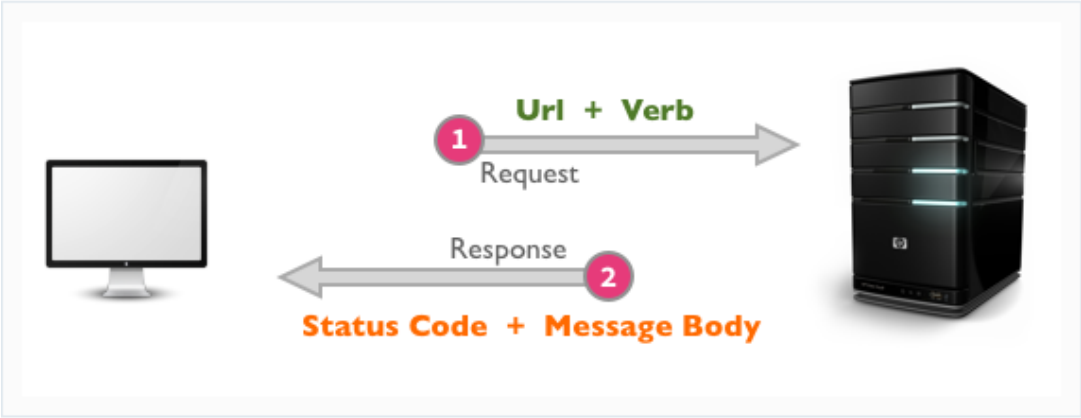
Application Programming Interface

It provides the conventions to interact with applications without knowing the internal logic of the code

Examples:

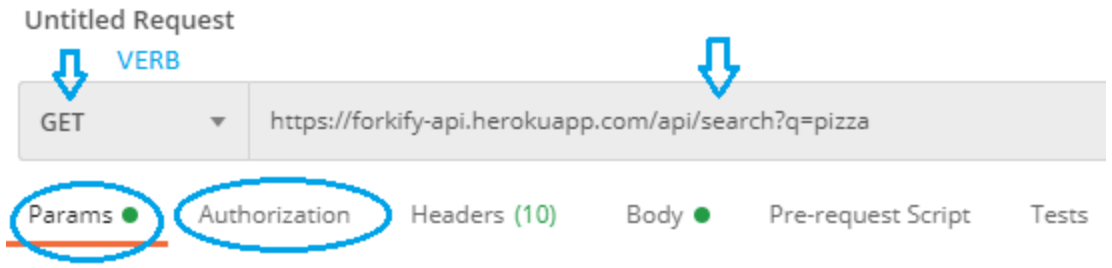
BUILD AN API FOR YOUR CODE





Source : Google images

REQUEST PARAMETERS OF API



URL

Request parameters

https://forkify-api.herokuapp.com/api/search?q=pizza

Http verb:

- GET
- POST
- PUT
- DELETE

Authorization:

- No auth
- Password based
- API Key based
- Token based

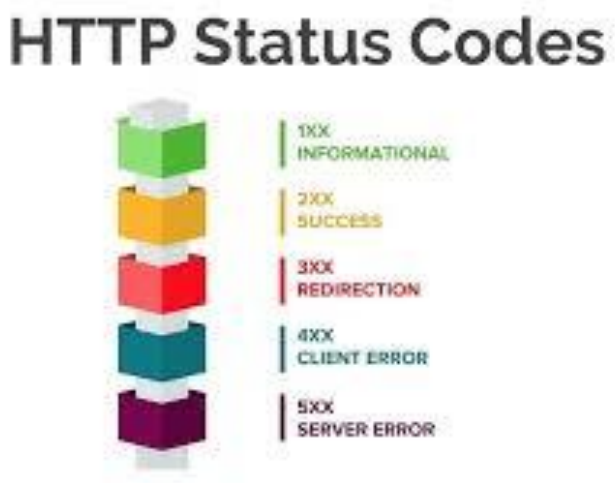
RESPONSE PARAMETERS OF API

Body | Cookies | Headers (19) | Test Results | 200 OK | 1174 ms | 9.89 KB | Save Response

Pretty | Raw | Preview | Visualize | JSON

STATUS CODE


```
{
    "count": 28,
    "recipes": [
        {
            "publisher": "101 Cookbooks",
            "title": "Best Pizza Dough Ever",
            "source_url": "http://www.101cookbooks.com/archives/001199.html",
            "recipe_id": "47746",
            "image_url": "http://forkify-api.herokuapp.com/images/best_pizza_dough_recipe1b20.jpg",
            "social_rank": 100,
            "publisher_url": "http://www.101cookbooks.com"
        },
        {
            "publisher": "The Pioneer Woman",
            "title": "Deep Dish Fruit Pizza",
            "source_url": "http://thepioneerwoman.com/cooking/2012/01/fruit-pizza/",
            "recipe_id": "46956",
            "image_url": "http://forkify-api.herokuapp.com/images/fruitpizza9a19.jpg",
            "social_rank": 100,
            "publisher_url": "http://thepioneerwoman.com"
        },
```

Source : Google images

REST API

- Based on seven principles
- Representation of Resources -> URL

  Example:

  http://example.com/api/books

- Basic operations on resources - CRUD

C → Create (SQL INSERT) : POST – Create a resource

R → Read (SQL SELECT) : GET - Retrieve a representation of a resource

U → Update (SQL UPDATE) : PUT - Update a resource using a full representation

D → Delete (SQL DELETE) : DELETE - Delete a resource.

EXAMPLE

Sample Data source

- JSON data
- Can also be from DB

FLASK IMPLEMENTATION

```python
#!flask/bin/python
import json

from flask import Flask, jsonify, abort, request

app = Flask(__name__)

with open('books.json', 'r') as books_file:
    books = json.loads(books_file.read())
```

books.json > {} 3


```json
[
    {
        "title": "A Light in the ...",
        "stars": "5 out of 5",
        "price": "$51.77",
        "link": "http://books.toscrape.com/catalogue/a-ligh
        "picture": "http://books.toscrape.com/media/cache/2
    },
    {
        "title": "Tipping the Velvet",
        "stars": "5 out of 5",
        "price": "$53.74",
        "link": "http://books.toscrape.com/catalogue/tippin
        "picture": "http://books.toscrape.com/media/cache/2
    },
    {
        "title": "Soumission",
        "stars": "5 out of 5",
        "price": "$50.10",
        "link": "http://books.toscrape.com/catalogue/soumis
        "picture": "http://books.toscrape.com/media/cache/3
    },
    {
        "title": "Sharp Objects",
```

API TO PERFORM BASIC CRUD OPERATIONS

C → Create (SQL INSERT) : POST – Create a resource

```
@app.route('/api/v1.0/books', methods=['POST'])
def create_task():
    if not request.json or not 'title' in request.json:
        abort(400)
    book = {
        'id': books[-1]['id'] + 1,
        'title': request.json['title'],
        'description': request.json.get('description', ""),
        'done': False
    }
    books.append(book)
    return jsonify({'book': book}), 201
```

NEW DATA

API TO PERFORM BASIC CRUD OPERATIONS

R → Read (SQL SELECT) : GET - Retrieve a representation of a resource

```
@app.route('/api/v1.0/books', methods=['GET'])
def get_books():
    return jsonify({'books': books})
```

Get all resources

Get a single resource

```
@app.route('/api/v1.0/books<int:book_id>', methods=['GET'])
def get_book(book_id):
    book = [book for book in books if book['id'] == book_id]
    if len(book) == 0:
        abort(404)
    return jsonify({'task': book[0]})
```

API TO PERFORM BASIC CRUD OPERATIONS

U → Update (SQL UPDATE) : PUT - Update a resource using a full representation

```
@app.route('/api/v1.0/books/<int:book_id>', methods=['PUT'])
def update_book(book_id):
    book = [book for book in books if book['id'] == book_id]
    if len(book) == 0:
        abort(404)
    if not request.json:
        abort(400)
    book[0]['stars'] = request.json.get('stars', book[0]['stars'])
    book[0]['price'] = request.json.get('price', book[0]['price'])
    return jsonify({'book': book[0]})
```

API TO PERFORM BASIC CRUD OPERATIONS

D → Delete (SQL DELETE) : DELETE - Delete a resource.

```
@app.route('/api/v1.0/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    book = [book for book in books if book['id'] == book_id]
    if len(book) == 0:
        abort(404)
    books.remove(book[0])
    return jsonify({'result': True})
```

FLASK-RESTFUL TEMPLATE

```
from flask_restful import Api, Resource
'''
class BookListAPI(Resource):
    def get(self):
        pass

    def post(self):
        pass

class BookAPI(Resource):
    def get(self, id):
        pass

    def put(self, id):
        pass

    def delete(self, id):
        pass

api.add_resource(BookListAPI, '/api/v1.0/books', endpoint = 'books')
api.add_resource(BookAPI, '/api/v1.0/books/<int:id>', endpoint = 'book')
'''
```

LONG RUNNING TASKS

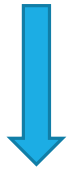
Example – Sending an email

```
from threading import Thread
from basic_api import app

def send_async_email(app, msg):
    with app.app_context():
        mail.send(msg)

def send_email(subject, sender, recipients, text_body, html_body):
    msg = Message(subject, sender=sender, recipients=recipients)
    msg.body = text_body
    msg.html = html_body
    thr = Thread(target=send_async_email, args=[app, msg])
    thr.start()
```

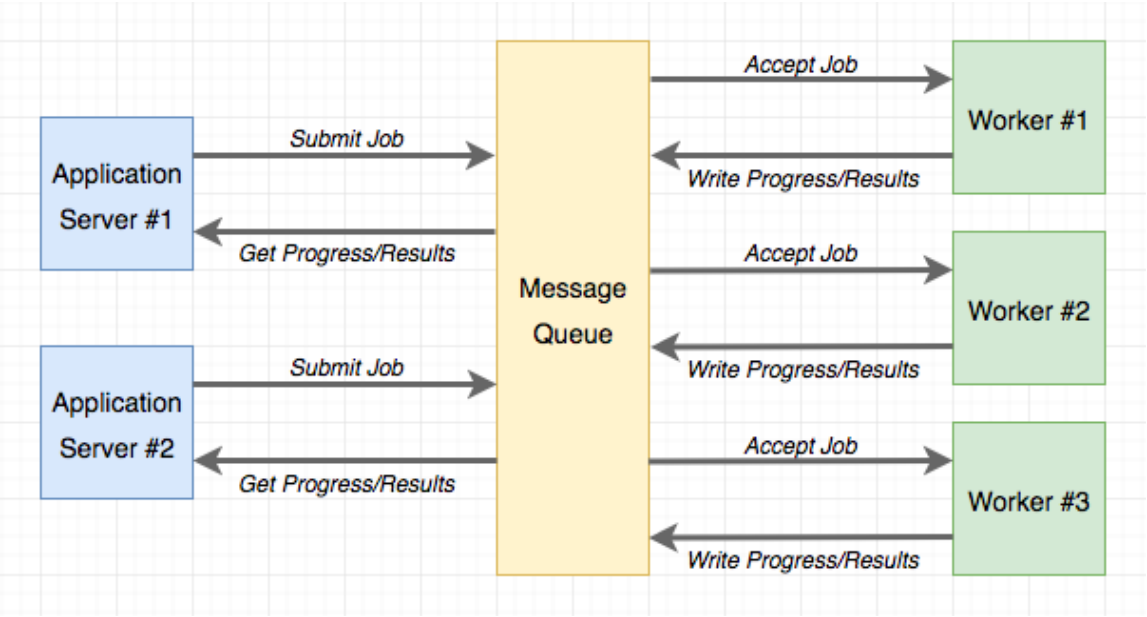
It can block the API.



```
@app.route('/api/v1.0/email_books', methods=['GET'])
def email_books():
    send_email('List of books', 'aa@abc.com', ['bb@abc.com'], books, None)
    return jsonify({'books': books})
```

LONG RUNNING TASKS – BACKGROUND TASKS

- Need non-blocking request/responses
- Task Queue – Celery



Source : Google images

THREADING VS. TASK QUEUE

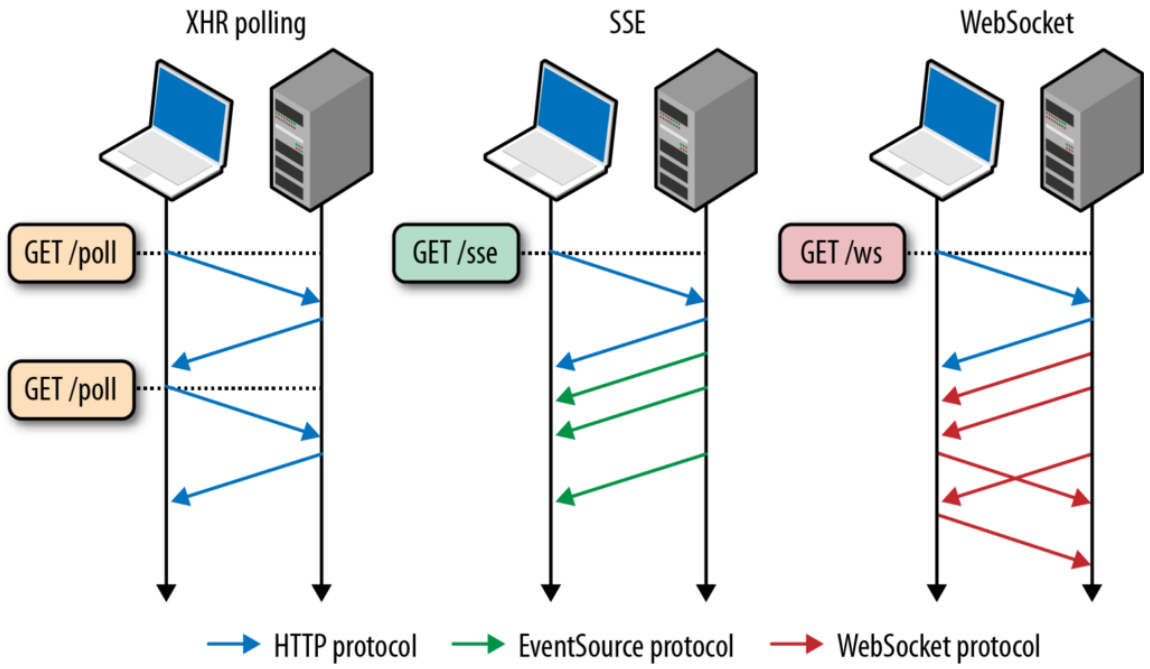
1. Task queues have distributed architecture
2. Scales application extremely well
3. Add more workers according to load
4. Does not block the API

STATUS OF BACKGROUND TASK

Client will know that background task started.

Update status in non blocking way

- Polling
- Web socket
- Server sent events

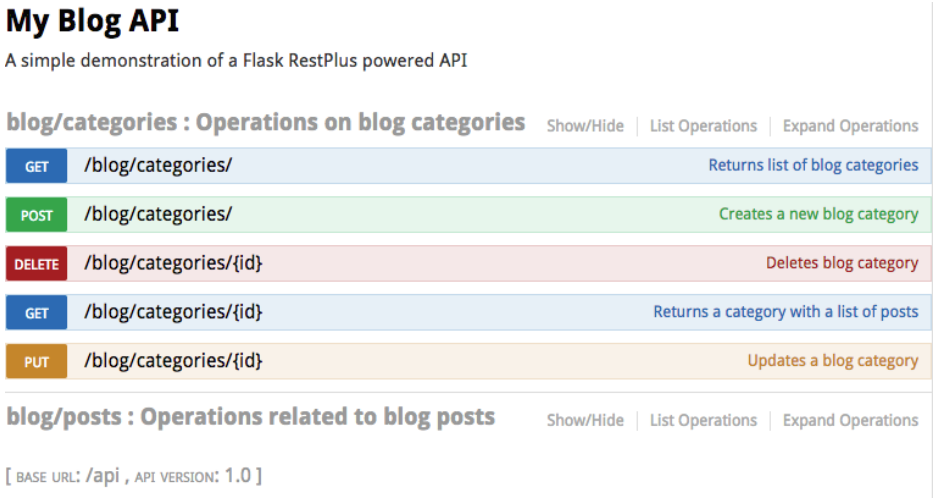


Source : Google images

API DOCUMENTATION WITH SWAGGER TOOL

- Good documentation is essential for APIs
- Take care of versioning and backward compatibility

<code snippet of swagger plugin in flask>



Source : Google images

Output of a well- organized API doc

SERVERLESS DEPLOYMENT WITH ZAPPA

While developing, APIs run on local server

For your APIs to be accessed by clients/3rd party users, they must be in a server.

Instead of dedicated server, we can go serverless

For personal projects or quick testing we can go with the option of serverless deployment.

Easy to setup

- Demo/screenshot of serverless deployment with Zappa