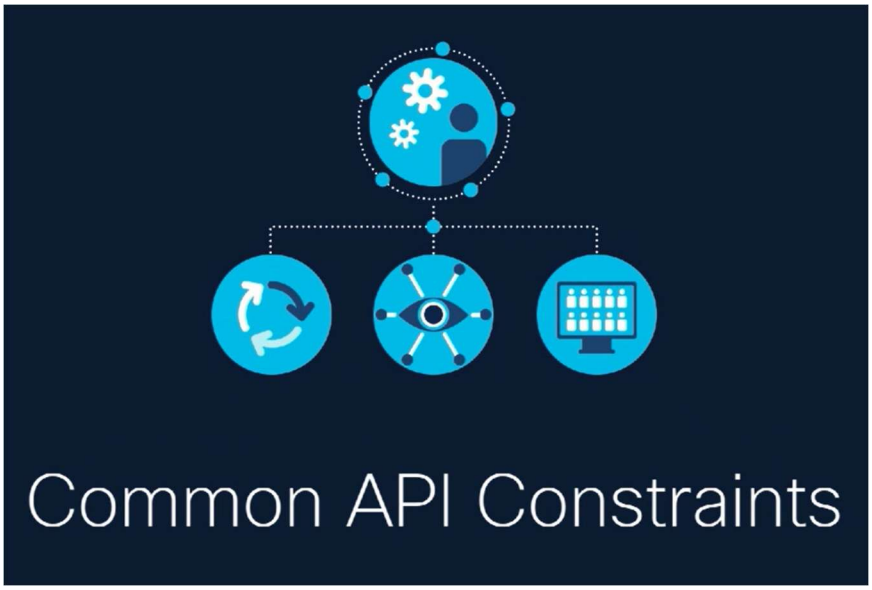
CÁC RÀNG BUỘC CỦA API

Khi anh em bắt đầu "đụng" vào REST API, có thể nghĩ đơn giản: chỉ cần curl một phát là nhận được dữ liệu, đúng không? Sai lầm thường gặp! Thực tế là: API luôn đi kèm với những rào cản (constraints) để kiểm soát truy cập, giới hạn tải và bảo vệ hệ thống backend. Và nếu không hiểu kỹ, bạn sẽ dễ bị chặn như chơi, gặp lỗi 429, 413, hoặc thậm chí làm quá tải chính hệ thống mình đang quản lý.



Vậy cần biết những gì khi làm việc với API? Hãy cùng điểm nhanh các "rào cản phổ biến" mà bạn phải đối mặt và vượt qua:

1. Pagination – Chia trang cho dữ liệu khổng lồ
   Không ai trả về nguyên bảng SQL với 10,000 bản ghi cả. Giải pháp phổ biến:
   - Offset Pagination:
     GET /users?limit=10&offset=30
     ➞ Lấy 10 bản ghi, bỏ qua 30 bản đầu tiên.
   - Link-based Pagination:
     API trả về các link kiểu: "next": "/users?page=4", "prev": "/users?page=2"
     Dựa trên RFC 5988 – được Cisco áp dụng chuẩn chỉnh!

2. Filtering & Sorting
Đừng kéo về tất cả dữ liệu rồi lọc bằng code! Hãy dùng query params:
GET /users?age>=50&sort=salary_desc
➞ Nhẹ nhàng, hiệu quả, và server xử lý nhanh chóng bằng SQL backend.

3. Rate Limiting – API có giới hạn!
- Ví dụ:
+ Gói miễn phí: 60 requests/phút
+ Gói Pro: 1000 requests/giờ
- Kỹ thuật giới hạn phổ biến:
+ Header X-RateLimit-Limit, X-RateLimit-Remaining
+ Thuật toán: leaky bucket, fixed window, sliding log
+ Reverse proxy như Nginx
➞ Nếu vượt giới hạn? Bạn sẽ thấy lỗi: 429 Too Many Requests + Retry-After

4. API Availability – Không phải lúc nào cũng online
- Thói quen tốt:

+ Gửi request tới /api/status để kiểm tra trước
+ Nếu thấy 500, 404 hay 408 → retry

- Chiến thuật retry:
+ Linear backoff: đợi 30s, thử lại 10 lần
+ Exponential backoff: 1s → 2s → 4s → 8s

5. Payload Limiting – Gửi nhiều quá là bị từ chối

- Nếu server giới hạn 256KB, mà bạn gửi 512KB? 👉 413 Request Entity Too Large
- Tip: resize ảnh, nén JSON, chia nhỏ batch nếu cần

➞ Payload nhỏ giúp:
+ Server xử lý nhanh hơn
+ Hạn chế DDoS hoặc memory exhaustion

☑ Tổng Kết – Đừng để API "đuổi bạn ra khỏi cửa"

- Biết cách phân trang và lọc dữ liệu
- Kiểm soát số lượng request
- Xử lý retry thông minh
- Giới hạn payload phù hợp

🧠 Ví dụ thực chiến:

```python
import requests, time
for i in range(5):
    resp = requests.get("https://api.example.com/status")
    if resp.status_code == 200:
        break
    else:
        time.sleep(2**i)
```

API Pagination, Filtering, Sorting và Rate Limiting

Bạn đã bao giờ gọi một REST API trả về hàng ngàn kết quả, chỉ để phải parse thủ công từng dòng JSON trong Python? Nếu rồi, bạn sẽ hiểu vì sao pagination, filtering, sorting, và rate limiting là những kỹ thuật "cứu cánh" cho cả client lẫn server trong thế giới lập trình hiện đại. Bài viết này là để giúp bạn — DevOps, NetDev, Automation engineer — không còn bỡ ngỡ khi gặp chúng trong thực chiến.

Pagination – Khi dữ liệu quá nhiều để nhét vào một lần:

Thay vì trả về 10.000 user, API hiện đại thường chia nhỏ kết quả thành từng trang, ví dụ GET /users?page=2&limit=100. Với Cisco Web API, họ chuẩn hóa việc này theo RFC 5988 (Web Linking), nghĩa là mỗi response thường đi kèm các header như:

Link: <https://api.example.com/users?page=3>; rel="next",
<https://api.example.com/users?page=1>; rel="prev"

Protip: Nếu rel="next" bị null hoặc thiếu → bạn đang ở trang cuối rồi đó!

Filtering – Chỉ lấy cái bạn cần:

Giống như WHERE trong SQL, filtering giúp bạn lấy subset dữ liệu dựa trên điều kiện, ví dụ:
GET /devices?model=C9300
GET /alerts?severity=critical&timestamp>2024-01-01
Kỹ thuật này giúp bạn giảm băng thông, RAM và thời gian xử lý client-side.

Sorting – Đưa dữ liệu về đúng thứ tự bạn cần:

Bạn muốn danh sách user theo thứ tự đăng nhập gần nhất? Hay top thiết bị bị cảnh báo nhiều nhất?
GET /users?sort=last_login:desc
GET /devices?sort=alerts_count:desc
Sắp xếp server-side không chỉ tiện mà còn giúp cache hiệu quả hơn.

Rate Limiting – Bức tường bảo vệ API:

Không chỉ giúp chống DoS, rate limit còn là biện pháp phân chia tài nguyên công bằng giữa các client:
- Theo người dùng: mỗi user được 1000 call/ngày.
- Theo thời gian: tối đa 10 request/giây.
- Theo vùng địa lý: chống các API call kỳ lạ từ vùng không liên quan.

Có thể implement rate limiting ở:

- Client-side: dùng timer, loading bar, retry mechanism…
- Server-side: áp dụng hiệu quả với Redis token bucket hoặc API Gateway như Kong, Apigee, AWS API Gateway...

```
def checkStatus(url):
    try:
        # Check API status
        status = requests.get(url + "/api/status")
        if status.status_code != 200:
            raise Exception("Unexpected status code: " +  status.status_code)

    except Exception as e:
        print("API endpoint encountered an error: " + e)
```

Một ví dụ thực tế (Python + Cisco DNA Center):
import requests
url = "https://sandboxdnac.cisco.com/dna/intent/api/v1/network-device"
headers = {"X-Auth-Token": "<your_token>"}
params = {"limit": 100, "offset": 200}
response = requests.get(url, headers=headers, params=params)
print(response.json())

Tổng kết – Làm chủ API là làm chủ automation:

Trong thời đại Infrastructure-as-Code, nếu bạn đang xây tool với Python, Ansible, hay Postman để tương tác API, thì việc hiểu rõ pagination, filter, sort, rate limiting là tối quan trọng để:
- Không bị timeout
- Không "ăn" hết quota
- Không làm sập API của chính mình

Rate Limiting – Khi API cần được "thở" đúng cách!

Rate Limiting là gì?

Rate limiting là kỹ thuật kiểm soát số lượng request mà một client có thể gửi đến API trong một khoảng thời gian nhất định. Mục tiêu là bảo vệ tài nguyên hệ thống khỏi lạm dụng, lỗi lập trình, hay tệ hơn – tấn công DoS/DDoS.

Client-side vs Server-side Rate Limiting

Client-side rate limiting:

- Được triển khai trực tiếp trong ứng dụng phía client.

- Hạn chế việc spam API qua đồng hồ đếm, thanh loading hoặc delay thủ công.

- Hiệu quả nếu client "ngoan", nhưng không chống lại client bị xâm nhập hay độc hại.

```
def getUsers(url):
    max_retries = 10
    timeout = 30

    for retry_count in range(max_retries):
        try:
            api_users = requests.get(url + "/api/users")
        except Exception as e:
            print("Error encountered while reading users. Retrying...")

        if api_users.status_code  ==200:
            return api_users
        else:
            time.sleep(timeout)

    raise Exception("Retry limit reached!")
```

Server-side rate limiting:

- Phía server chủ động giới hạn số lượng request.

- Hiệu quả cao hơn, ngăn ngừa:

  - Tấn công từ chối dịch vụ (DoS),

  - Hành vi bất thường như xoá hàng loạt tài nguyên (dấu hiệu bị compromise).

  - Quá tải hệ thống khi người dùng tăng đột biến.

Ví dụ: Mỗi user chỉ được phép gọi 1000 API/ngày. Gói premium có thể mở rộng giới hạn này.

Chiến thuật throttle API thông minh

- HTTP headers: Dùng X-RateLimit-Limit và X-RateLimit-Remaining để thông báo số lượng request còn lại.

- Message queue: Xếp hàng các request thay vì xử lý ngay để tránh nghẽn cổ chai.

- Thuật toán kiểm soát: Fixed Window, Leaky Bucket, Sliding Log.

- Reverse Proxy & Load Balancer: Hỗ trợ rate limiting như một tính năng mặc định.

Phản hồi khi vượt giới hạn

Khi user gọi API quá giới hạn:

- Server trả về HTTP 429 Too Many Requests

- Kèm theo Retry-After header → chỉ định thời gian chờ trước khi gọi lại.

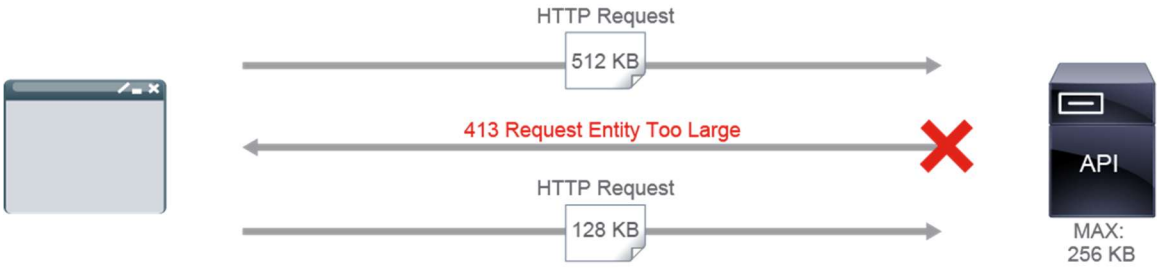
Xử lý sự cố & kiểm tra sức khỏe API

Đôi khi API sẽ bị "mệt":

- Đang cập nhật hệ thống,

- Quá tải,

- Mất kết nối mạng.



Best practice:

- Gửi request kiểm tra /api/status

- Theo dõi HTTP response codes:

  - 200 OK: ổn.

- 404 Not Found, 500 Internal Server Error: API down hoặc sai endpoint.

Chiến lược retry hiệu quả:

- Linear backoff: thử lại sau mỗi 60 giây.

- Exponential backoff: 1 → 2 → 4 → 8 → 16s...

Nếu gặp timeout:

- 408 Request Timeout

- 504 Gateway Timeout

Giới hạn kích thước payload – Payload Limiting

Payload quá lớn (ảnh HD, file XML nặng...) có thể:

- Làm server chậm hoặc tắt ngóm vì quá tải RAM.

- Bị gián đoạn khi truyền tải.

- Bị tấn công bằng dữ liệu độc hại.

API nên giới hạn kích thước payload để bảo mật, tối ưu hiệu năng, ngăn chặn tấn công.

Nếu vượt giới hạn, server trả về HTTP 413 Request Entity Too Large

Tóm lại:

- Client-side rate limiting: giới hạn từ phía client.

- Server-side rate limiting: bảo vệ server hiệu quả hơn.

- Payload limiting: giới hạn kích thước request để tăng bảo mật và hiệu suất.

Câu hỏi ôn tập cuối bài

Sự khác nhau giữa client-side rate limiting và payload limiting là gì?

✅ Client-side rate limiting giới hạn số lượng request gửi đi, còn payload limiting giới hạn kích thước nội dung mỗi request.

Gợi ý thực hành

Viết một đoạn script Python nhỏ:

- Gọi API /api/status mỗi 30 giây.

- Kiểm tra response code.

- Áp dụng exponential backoff nếu thất bại.