API Rate Limit Monitor - Complete Documentation

Table of Contents

What is Rate Limiting?

Rate limiting is controlling how many requests a client can make to your API in a given time period.

Why Do We Need It?

Without Rate Limiting:

```
User A sends 10,000 requests per second → Server crashes
Malicious bot sends 1 million requests → Your bill explodes
Single user consumes all resources → Other users can't access the API
```

With Rate Limiting:

```
User A: Maximum 100 requests per minute → Controlled usage
Bot: Blocked after exceeding limit → System protected
Resources: Fairly distributed → All users get access
```

Real-World Examples

- **Twitter API**: 300 requests per 15 minutes
- **GitHub API**: 5,000 requests per hour
- **Google Maps API**: Based on your billing plan
- **Stripe API**: Different limits per endpoint

How This System Works

Think of it like a **ticket counter at a concert**:

1. **Registration**: Each person gets a ticket with their name (API Consumer with API Key)
2. **Entry Rules**: Each ticket allows 10 entries per hour (Rate Limit)
3. **Counter**: Someone tracks how many times you've entered (Usage Record)
4. **Time Windows**: The count resets every hour (Time Window)

System Components

```
┌─────────────────┐
│ API Consumer    │ → Who is making requests
│ (name, apiKey)  │
└─────────────────┘
        │
        │ has limit of
        ▼
┌─────────────────┐
│ Rate Limit      │ → How many requests allowed
│ (100/minute)    │
└─────────────────┘
        │
        │ tracked in
        ▼
┌─────────────────┐
│ Usage Record    │ → Current count for this minute
│ (87 requests)   │
└─────────────────┘
```

Core Concepts

1. API Consumer

What is it? A user, application, or service that calls your API.

Example:

json

```
{
  "id": 1,
  "name": "Mobile App v2.0",
  "apiKey": "a1b2c3d4e5f6",
  "limitPerMinute": 100,
  "status": "ACTIVE"
}
```

Think of it as: A Netflix account. Each account has an ID, name, and subscription plan (rate limit).

---

2. API Key

What is it? A unique identifier that authenticates the consumer.

Example: `a1b2c3d4e5f67890123456789012 34`

How it works:

```
Request without API Key → ❌ Rejected
Request with valid API Key → ✅ Proceed to rate limit check
Request with invalid API Key → ❌ Rejected (404 Not Found)
```

Think of it as: Your credit card number. Unique to you, used to identify you.

---

3. Time Windows

What is it? A fixed time period for counting requests.

Types:

- **MINUTE**: Resets every 60 seconds (10:30:00 → 10:31:00)
- **HOUR**: Resets every 60 minutes (10:00:00 → 11:00:00)

Example Timeline:

```
10:30:00 ────────────────────────────────▶ 10:31:00
│                          │
│  Window Start: 10:30:00          │  Window Start: 10:31:00
│  Requests in this window: 47       │  Requests in this window: 0 (RESET)
└──────────────────────────────────────────────┘
```

Think of it as: A gym membership that allows 10 visits per week. Every Monday, your counter resets to 0.

---

4. Usage Record

What is it? A record of how many requests a consumer made in a specific time window.

Database Row Example:

```
id | consumer_id | window_type | window_start        | request_count
1  | 5           | MINUTE      | 2026-01-18 10:30:00 | 87
2  | 5           | MINUTE      | 2026-01-18 10:31:00 | 12
3  | 7           | MINUTE      | 2026-01-18 10:30:00 | 5
```

Reading this table:

- Row 1: Consumer #5 made 87 requests between 10:30:00 and 10:31:00
- Row 2: Consumer #5 made 12 requests between 10:31:00 and 10:32:00
- Row 3: Consumer #7 made 5 requests between 10:30:00 and 10:31:00

Think of it as: A call log on your phone. It shows who called, when, and how many times.

Data Flow Examples

Example 1: Creating a New API Consumer

Step-by-Step:

```
1. User sends request:
   POST /api/consumers
   {
     "name": "Weather App",
     "limitPerMinute": 50
   }

2. System generates unique API Key:
   apiKey = "a1b2c3d4e5f6" (random UUID)

3. System saves to database:
   INSERT INTO api_consumers (name, api_key, limit_per_minute, status)
   VALUES ('Weather App', 'a1b2c3d4e5f6', 50, 'ACTIVE')

4. System responds:
   {
     "id": 1,
     "name": "Weather App",
     "apiKey": "a1b2c3d4e5f6",
     "limitPerMinute": 50,
     "status": "ACTIVE"
   }
```

Now the consumer can use `a1b2c3d4e5f6` to make API calls!

Example 2: First Request (Cold Start)

Scenario: Consumer makes their very first request at 10:30:15

```
1. Request arrives:
   POST /api/rate-limit/record?apiKey=a1b2c3d4e5f6

2. System finds consumer:
   SELECT * FROM api_consumers WHERE api_key = 'a1b2c3d4e5f6'
   Result: {id: 1, limitPerMinute: 50, status: ACTIVE}

3. System checks if consumer is active:
   ✅ Status is ACTIVE → Proceed

4. System calculates current time window:
   Current time: 10:30:15
   Window start: 10:30:00 (truncate to minute)

5. System looks for usage record:
   SELECT * FROM usage_records
   WHERE consumer_id = 1
     AND window_type = 'MINUTE'
     AND window_start = '2026-01-18 10:30:00'
   Result: ❌ No record found (first request ever)

6. System creates new usage record:
   INSERT INTO usage_records (consumer_id, window_type, window_start, request_count)
   VALUES (1, 'MINUTE', '2026-01-18 10:30:00', 0)

7. System increments count:
   UPDATE usage_records
   SET request_count = 1
   WHERE id = <new_id>

8. System checks limit:
   1 < 50 → ✅ ALLOWED

9. System responds:
   {
     "success": true,
     "currentUsage": 1
   }
```

Database state after this request:

```
api_consumers:
id | name        | api_key      | limit_per_minute | status
1  | Weather App | a1b2c3d4e5f6 | 50               | ACTIVE

usage_records:
id | consumer_id | window_type | window_start        | request_count
1  | 1           | MINUTE      | 2026-01-18 10:30:00 | 1
```

Example 3: Multiple Requests in Same Minute

Scenario: Consumer makes 5 more requests at 10:30:20, 10:30:25, 10:30:40, 10:30:50, 10:30:55

```
Request #2 (10:30:20):
  Current window: 10:30:00
  Existing count: 1
  New count: 2
  Check: 2 < 50 → ✅ ALLOWED

Request #3 (10:30:25):
  Current window: 10:30:00 (SAME WINDOW)
  Existing count: 2
  New count: 3
  Check: 3 < 50 → ✅ ALLOWED

Request #4 (10:30:40):
  Current window: 10:30:00 (SAME WINDOW)
  Existing count: 3
  New count: 4
  Check: 4 < 50 → ✅ ALLOWED

... and so on
```

Database state:

```
usage_records:
id | consumer_id | window_type | window_start        | request_count
1  | 1           | MINUTE      | 2026-01-18 10:30:00 | 6
```

Key insight: The SAME row is updated because it's the same (consumer, window_type, window_start).

Example 4: New Minute Window

Scenario: Consumer makes a request at 10:31:05 (new minute!)

```
1. Request arrives at 10:31:05

2. System calculates window:
   Current time: 10:31:05
   Window start: 10:31:00 ← NEW WINDOW!

3. System looks for usage record:
   SELECT * FROM usage_records
   WHERE consumer_id = 1
     AND window_type = 'MINUTE'
     AND window_start = '2026-01-18 10:31:00'
   Result: ❌ No record (different window!)

4. System creates NEW record:
   INSERT INTO usage_records (consumer_id, window_type, window_start, request_count)
   VALUES (1, 'MINUTE', '2026-01-18 10:31:00', 1)

5. Response: ✅ ALLOWED (count reset to 1)
```

Database state:

```
usage_records:
id | consumer_id | window_type | window_start        | request_count
1  | 1           | MINUTE      | 2026-01-18 10:30:00 | 6    ← OLD WINDOW
2  | 1           | MINUTE      | 2026-01-18 10:31:00 | 1    ← NEW WINDOW
```

This is why the unique constraint is critical! It ensures we have exactly ONE row per time window.

Example 5: Rate Limit Exceeded

Scenario: Consumer has made 49 requests, now makes the 50th and 51st

```
Request #50:
  Current count: 49
  New count: 50
  Check: 50 < 50 → ❌ FALSE, but we allow it (49 + 1 = 50 is at limit)
  Actually, let's check the code...

  Code says: if (requestCount >= limitPerMinute) throw exception

  So before recording:
    49 < 50 → ✅ Record it
    Update count to 50

Request #51:
  Current count: 50
  Check: 50 >= 50 → ❌ EXCEEDED!
  Response: 429 TOO MANY REQUESTS
  {
    "error": "Rate limit exceeded"
  }
```

The user must wait until 10:31:00 for their counter to reset.

Database Design

Why Two Tables?

api_consumers = WHO can access the API **usage_records** = WHAT they've done recently

Separation of concerns:

- Consumer data changes rarely (name, limit)
- Usage data changes constantly (every request)

The Critical Constraint

```sql
UNIQUE (consumer_id, window_type, window_start)
```

What it prevents:

```
❌ WITHOUT CONSTRAINT:
id | consumer_id | window_type | window_start        | request_count
1  | 1           | MINUTE      | 2026-01-18 10:30:00 | 25
2  | 1           | MINUTE      | 2026-01-18 10:30:00 | 30  ← DUPLICATE!
3  | 1           | MINUTE      | 2026-01-18 10:30:00 | 15  ← DUPLICATE!

Total requests: 70, but we can't know which is correct!

✅ WITH CONSTRAINT:
id | consumer_id | window_type | window_start        | request_count
1  | 1           | MINUTE      | 2026-01-18 10:30:00 | 70

Single source of truth! Accurate counting!
```

API Endpoints Guide

1. Create Consumer

Purpose: Register a new API consumer

Request:

```http
POST /api/consumers
Content-Type: application/json

{
  "name": "My Mobile App",
  "limitPerMinute": 100
}
```

Response:

```json
```

```
{
  "id": 1,
  "name": "My Mobile App",
  "apiKey": "a1b2c3d4e5f6g7h8i9j0",
  "limitPerMinute": 100,
  "status": "ACTIVE"
}
```

Use Case: When onboarding a new developer/application.

---

2. Check Rate Limit (Without Recording)

Purpose: Check if a consumer can make a request without actually counting it

Request:

```http
POST /api/rate-limit/check?apiKey=a1b2c3d4e5f6g7h8i9j0
```

Response (Under Limit):

```json
{
  "allowed": true,
  "currentUsage": 47
}
```

Response (At Limit):

```json
{
  "allowed": false,
  "currentUsage": 100
}
```

Use Case: Gateway wants to check before forwarding request.

---

3. Record Request

Purpose: Record that a request was made (increments counter)

Request:

```http
POST /api/rate-limit/record?apiKey=a1b2c3d4e5f6g7h8i9j0
```

Response (Success):

```json
{
  "success": true,
  "currentUsage": 48
}
```

Response (Exceeded):

```json
HTTP 429 Too Many Requests
{
  "error": "Rate limit exceeded"
}
```

Use Case: After successfully processing a request, record it.

---

4. Get Current Usage

Purpose: Check how many requests consumer has made in current window

Request:

```http
GET /api/rate-limit/usage?apiKey=a1b2c3d4e5f6g7h8i9j0&windowType=MINUTE
```

Response:

```json
```

```
{
  "apiKey": "a1b2c3d4e5f6g7h8i9j0",
  "windowType": "MINUTE",
  "currentUsage": 47
}
```

Use Case: Dashboard showing real-time usage statistics.

---

5. Suspend Consumer

Purpose: Block a consumer from making requests (abuse, non-payment, etc.)

Request:

```http
PATCH /api/consumers/1/suspend
```

Response:

```
204 No Content
```

Effect: All future requests with this consumer's API key will be rejected with `403 Forbidden`.

---

6. Activate Consumer

Purpose: Re-enable a suspended consumer

Request:

```http
PATCH /api/consumers/1/activate
```

Response:

```
204 No Content
```

Real-World Scenario

Scenario: Weather API Service

You're running a weather API. You have 3 customers:

Customers:

1. Free Tier App: 10 requests/minute
2. Pro App: 100 requests/minute
3. Enterprise App: 1000 requests/minute

Timeline of Events

10:30:00 - Free Tier App starts hammering the API

```
10:30:05 → Request #1  → ✅ Allowed (1/10)
10:30:10 → Request #2  → ✅ Allowed (2/10)
10:30:15 → Request #3  → ✅ Allowed (3/10)
...
10:30:50 → Request #10 → ✅ Allowed (10/10)
10:30:55 → Request #11 → ❌ BLOCKED (429 Too Many Requests)
```

10:31:00 - New minute, counter resets

```
10:31:05 → Request #1 (of new window) → ✅ Allowed (1/10)
```

10:31:30 - Pro App makes a batch request

```
Sends 50 requests simultaneously
All arrive at same time (10:31:30)

System handles:
  Request 1 → Creates record with count=1
  Request 2 → Updates record to count=2
  Request 3 → Updates record to count=3
  ...
  Request 50 → Updates record to count=50

All 50 requests: ✅ ALLOWED (under 100 limit)
```

10:32:00 - Admin detects Free Tier abuse

```
Admin reviews logs:
  Free Tier App tried to make 50 requests in one minute
  Only 10 were allowed, 40 were blocked

Admin decision: SUSPEND

Action:
  PATCH /api/consumers/1/suspend

Result:
  All future requests from Free Tier App → 403 Forbidden
  "Consumer is suspended"
```

Summary: How Everything Connects

```
┌─────────────────────────────────────────────────────────────┐
│              REQUEST ARRIVES                                │
│         POST /api/rate-limit/record?apiKey=XXX              │
└──────────────────────────┬──────────────────────────────────┘
                           │
                           ▼
          ┌─────────────────────────┐
          │ Find API Consumer       │
          │ by apiKey               │
          └──────────┬──────────────┘
                     │
                     ▼
          ┌─────────────────────────┐
          │ Check Status            │
          │ ACTIVE or SUSPENDED?    │
          └──────────┬──────────────┘
                     │
          ┌──────────┴──────────┐
          │                     │
      SUSPENDED            ACTIVE
          │                     │
          ▼                     ▼
    ┌──────────┐     ┌──────────────────────┐
    │ REJECT   │     │ Calculate Window     │
    │ 403      │     │ window_start=10:30:00│
```

```
└──────────────┘  └──────────────┬──────────────┘
                │
                ▼
        ┌──────────────────────────────────┐
        │ Find or Create     │
        │ Usage Record       │
        │ (consumer, MINUTE, 10:30)│
        └──────────────┬───────────────────┘
               │
               ▼
        ┌──────────────────────────────────┐
        │ Check Current Count   │
        │ count < limit?        │
        └──────────────┬───────────────────┘
               │
        ┌──────────────┴───────────┐
        │         │
       YES        NO
        │         │
        ▼         ▼
   ┌────────────────────┐  ┌────────────────┐
   │ Increment Count │   │ REJECT     │
   │ count = count+1 │   │ 429        │
   │ Save Record    │   │ "Rate limit │
   └──────────┬─────────┘  │ exceeded"  │
        │        └────────────────┘
        ▼
   ┌────────────────────┐
   │ ALLOW REQUEST   │
   │ Return success  │
   └────────────────────┘
```

Key Takeaways

1. **API Consumer** = Identity (who)
2. **API Key** = Authentication (proof of identity)
3. **Rate Limit** = Rule (how many allowed)
4. **Time Window** = Period (when counter resets)
5. **Usage Record** = Counter (current usage)
6. **Unique Constraint** = Accuracy (one truth per window)

The system protects your API from overuse while providing fair access to all consumers.