Rate Limiter

Why?

- To stop DDOS attack
- to prevent system resources
- reduce company cost to save some server
- Rate limiters limit the number of events a person, device, or IP address can do in a given time range.
- Rate limiters limit a sender's request volume. A rate limiter blocks requests after the cap is hit.
- They can help limit traffic to a website to prevent DDOS attacks.

We also inform the user we are blocking the request to send HTTP error code
we can send 429 to many request

HOW
to identify the specific IP and make rate limit
we have multiple algorithem

we have multiple algorithem

we have token bucket algo

we have a token a

we have fixed windows system algo

Requirement
1: Server side
block based IP user id
send 439 HTTP request on blocing
Loggin matching

Functional Requirements:

1. Identify and track requests by IP address or user ID.
2. Implement rate limiting algorithms (token bucket, fixed window, sliding window).
3. Block requests that exceed the defined rate limit.
4. Send HTTP 429 (Too Many Requests) error code for blocked requests.
5. Allow configurable rate limits per API endpoint or globally.
6. Provide real-time updating of rate limit rules.
7. Log all rate limiting events for analysis.

Non-Functional Requirements:

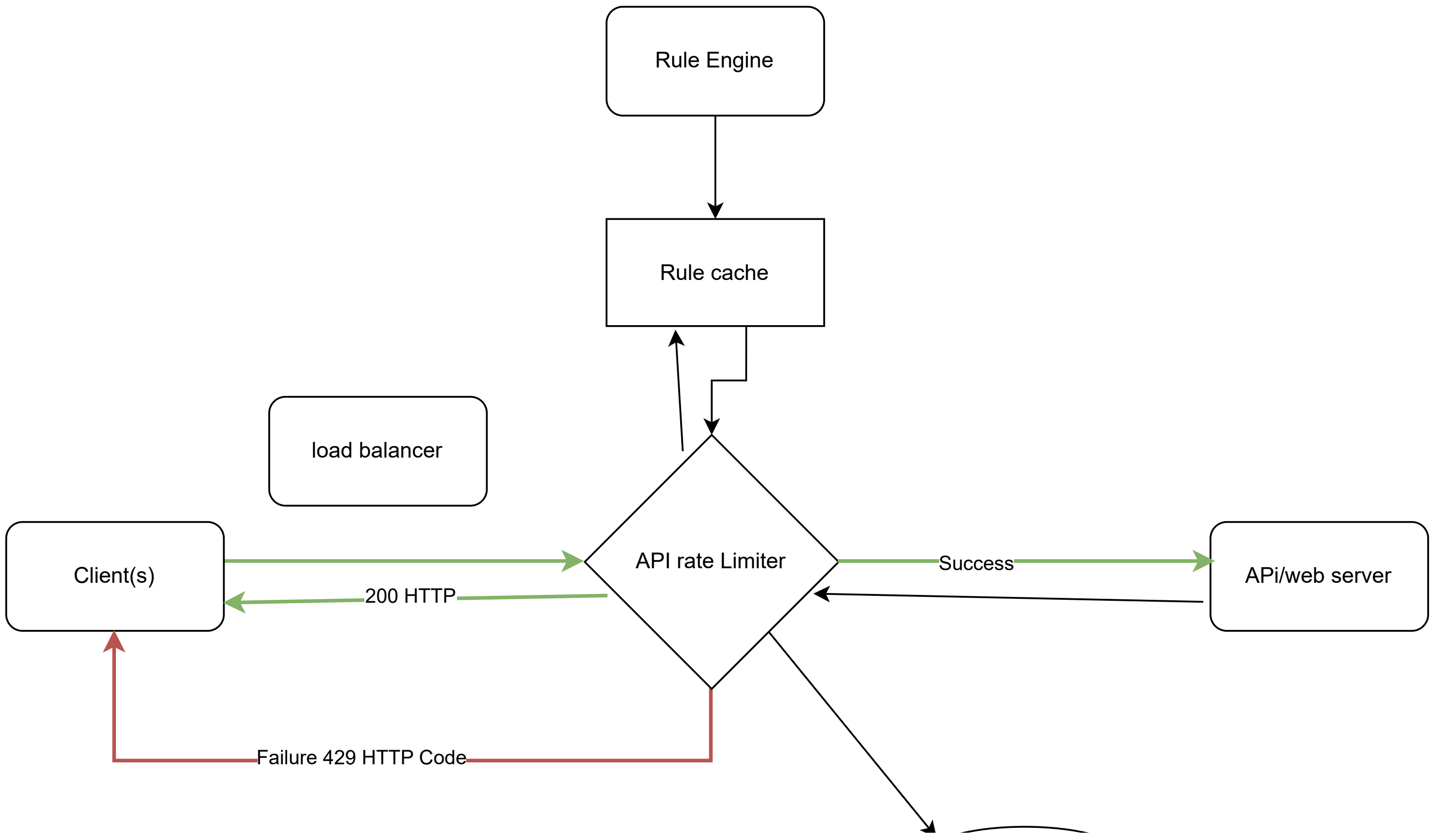
1. High Performance: Minimal latency impact on API requests.
2. Scalability: Able to handle increasing loads and distribute across multiple servers.
3. Reliability: Ensure consistent rate limiting across all instances.
4. Flexibility: Easy to adjust rate limits and algorithms.
5. Observability: Provide metrics and alerts for rate limiting events.

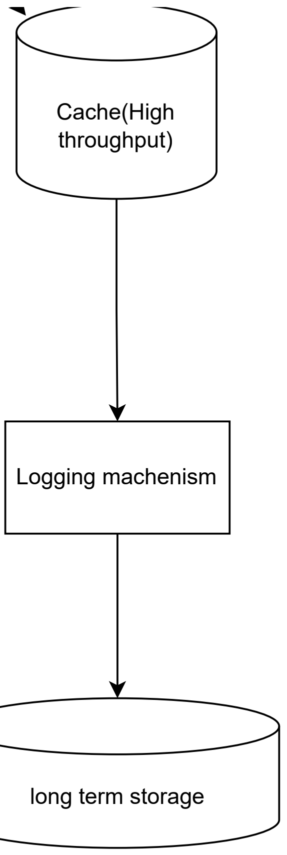
- **Distributed Rate Limiter:**
  - Use a distributed cache (e.g., Redis) for storing rate limit counters.
  - Implement a consistent hashing mechanism for distributing load across multiple rate limiter instances.

- **Rule Engine:**
  - Provide a user interface for defining and managing rate limit rules.
  - Store rules in a database for persistence.
  - Implement a rule cache for fast access.

- **Rate Limiting Service:**
  - Implement multiple rate limiting algorithms (token bucket, sliding window, etc.).
  - Use the distributed cache for maintaining rate limit state.
  - Provide a fast path for frequent offenders using bloom filters.

- **API Gateway:**
  - Integrate rate limiting as a middleware in the API gateway.
  - Handle authentication and extract user/IP information.
  - Make rate limit decisions based on rules and current limits.
- **Logging and Analytics:**
  - Implement asynchronous logging to minimize impact on request processing.
  - Use a time-series database for storing rate limiting events and metrics.
  - Provide a dashboard for visualizing rate limiting trends and patterns.
- **Configuration Service:**
  - Allow dynamic updates to rate limit rules without service restart.
  - Implement a publish-subscribe mechanism for propagating rule changes.
- **Load Balancer:**
  - Distribute incoming requests across multiple API gateway instances.
  - Implement health checks and automatic failover.




Cache(High throughput)
Logging machenism
long term storage