Annotation-Based Rate Limiting Library

Technical Specification v2.0

1. Executive Summary

This document defines the architecture for a high-performance, annotation-driven rate-limiting library for the Java ecosystem. The primary goal is to provide a declarative way to protect APIs while remaining framework-agnostic (supporting Spring Boot, Quarkus, and Jakarta EE) and cloud-ready (Kubernetes-native).

2. System Architecture

The system follows a **Hexagonal Architecture** pattern to separate core rate-limiting logic from framework-specific "glue" code.

2.1 Core Module (rl-core)

- **Responsibility:** Logic for bucket management, window calculations, and coordination of SPIs.
- **Dependency Policy:** Zero external framework dependencies (no Spring, no Quarkus, no Jakarta).
- **Key Components:**
    - **Limiter Engine:** Processes incoming requests against defined thresholds.
    - **Registry:** Maintains the lifecycle and state of active rate-limiters.

2.2 Framework Adapters

- **rl-spring-boot-adapter:** Provides Spring AOP support, @ConfigurationProperties mapping, and SpEL evaluation for dynamic key resolution.
- **rl-quarkus-adapter:** Provides CDI Interceptors and SmallRye Config integration.
- **rl-jakarta-adapter:** Provides standard jakarta.interceptor.Interceptor support for legacy or pure EE environments.

3. Rate Limiting Algorithms

The library must support multiple algorithms to handle different traffic shapes:

3.1 Token Bucket (for Smooth Bursts)

Provides a "bucket" of tokens that refills at a fixed rate.

- Mathematical Model:
  $$T_{available} = \min(B, T_{last} + (t_{current} - t_{last}) \times R)$$

  Where: B = Bucket Capacity, R = Refill Rate, T_{last} = Token count at the last

request.

3.2 Sliding Window Counter (for Accuracy)

Uses a weighted moving average between the current and previous time windows to provide high accuracy without the $O(N)$ memory overhead of a sliding window log.

4. Service Provider Interfaces (SPI)

To ensure the library can scale and integrate with any infrastructure, the following extension points are defined:

- **StorageProvider**: Abstraction for the persistence layer (Redis, Hazelcast, In-Memory). Must support atomic operations (e.g., Lua scripts for Redis).
- **ConfigProvider**: Abstraction for where limits are stored (Properties files, Kubernetes ConfigMaps, Consul).
- **KeyResolver**: Abstraction for identifying the unique caller (IP, User ID, API Key).
- **MetricsExporter**: Abstraction for telemetry (Micrometer, Prometheus, StatsD).

5. Resilience & Fallback Strategy

To prevent the rate-limiter from becoming a Single Point of Failure (SPOF), the following tiered logic is required:

5.1 L1/L2 Tiered Defense

1. **L1 (Primary):** A distributed storage provider (e.g., Redis) to ensure consistent limits across a cluster.
2. **L2 (Fallback):** An in-memory cache (e.g., Caffeine) that activates if L1 is unreachable, providing per-node protection during outages.

5.2 Circuit Breaker & Fail-Open

- **Monitoring:** The library must track the health (timeouts/failures) of the StorageProvider.
- **Behavior:** If the failure rate exceeds a configurable threshold, the system must trip the circuit and bypass L1, defaulting to **Fail-Open** (allow traffic) or **Fail-Closed** (deny traffic) based on the specific security requirements of the endpoint.

6. Kubernetes & Cloud Integration

The library is designed for dynamic, containerized environments.

- **Hot-Reloading:** Support for a KubernetesConfigProvider that monitors volume-mounted **ConfigMaps**. Threshold updates must be reflected in-memory without a JVM restart via file-watch listeners.
- **Environment Awareness:** Native resolution of Kubernetes environment variables for configuration.
- **Sidecar Compatibility:** Ability to communicate with local Redis/Hazelcast sidecars via low-latency Unix Domain Sockets or localhost.

7. Configuration Schema

The library must be fully configurable via the host framework's native configuration system (e.g., application.yml).

Category	Property	Description
Global	rl.enabled	Global kill-switch for the library.
Storage	rl.storage.type	REDIS, HAZELCAST, CAFFEINE, LOCAL.
Thresholds	rl.config.source	ANNOTATION, CONFIG_MAP, REMOTE_API.
Resilience	rl.fail.strategy	OPEN (Allow) or CLOSED (Deny) on storage failure.
K8s	rl.k8s.watch-enabled	Toggle for hot-reloading from mounted volumes.

8. Observability & Telemetry

The library must provide transparency through:

- **HTTP Headers:** Automatic injection of X-RateLimit-Limit, X-RateLimit-Remaining, and X-RateLimit-Retry-After.
- **Event Logging:** Asynchronous logging of "Limit Exceeded" events to prevent blocking request threads.
- **Metrics:** Integration with framework-native registries (Micrometer/SmallRye) to export counters for hits, misses, and system failures.