

Redis Rate Limiter

Production Documentation

Token Bucket Algorithm with Distributed State

Version 1.1.0 (Security Hardened)

Generated: 2025-12-16 23:04

Jules MCP Server - Antigravity Orchestration

Table of Contents

1.	Architecture Overview	3
2.	Tier Configuration (Updated)	4
3.	Integration Guide	5
4.	Security Hardening (New)	6
5.	API Reference	7
6.	Metrics & Monitoring	8

1. Architecture Overview

System Architecture

The rate limiter implements a distributed token bucket algorithm using Redis for state management. It supports per-API-key rate limiting with tiered configurations and graceful failover to local memory when Redis is unavailable.

Key Components:

- **Express Middleware** - Intercepts requests and applies rate limiting
- **Redis Backend** - Distributed state management with atomic Lua scripts
- **Failover Cache** - Local memory backup when Redis is unavailable
- **Tier System** - Configurable limits per subscription tier

2. Tier Configuration (Updated)

The rate limiter supports three tiers with configurable limits. Each tier uses the token bucket algorithm with different refill rates and burst capacities.

Tier	Requests/Min	Burst Capacity	Refill Rate	Window
Free	100	150	1.67/sec	60s
Pro	1,000	1,500	16.67/sec	60s
Enterprise	100,000	150,000	1,666.67/sec	60s

Security Change (v1.1.0): Enterprise tier bypass removed

The enterprise tier no longer bypasses rate limiting entirely. Instead, it has very high limits (100,000 requests/minute) to maintain protection against abuse while providing practically unlimited access for legitimate use.

3. Integration Guide

Quick Start

```
// 1. Import the rate limiter
import { createRateLimiter } from './middleware/rateLimiter.js';

// 2. Create and initialize
const rateLimiter = createRateLimiter();
await rateLimiter.initialize();

// 3. Apply middleware
app.use('/api/', rateLimiter.middleware());

// 4. Add metrics endpoint
app.get('/api/rate-limit/metrics', (req, res) => {
  res.json(rateLimiter.getMetrics());
});
```

Environment Variables

Variable	Description	Default
REDIS_URL	Redis connection (use rediss:// for TLS)	redis://localhost:6379
NODE_ENV	Environment (enables TLS warning)	development
RATE_LIMIT_FAILOVER	Failover strategy	fail-closed

4. Security Hardening (v1.1.0)

Critical security fixes were applied to address vulnerabilities identified in the security audit.

Issue	Severity	Fix	Status
Stack trace in logs	CRITICAL	Removed err.stack from all error logs	FIXED
Redis TLS not enforced	CRITICAL	Added validateRedisUrl() warning	FIXED
X-Forwarded-For spoofing	HIGH	Use req.socket.remoteAddress	FIXED
API key in query string	HIGH	Deprecated and ignored	FIXED
Enterprise bypass	HIGH	Replaced with 100k/min limit	FIXED

Security Best Practices

- **Use TLS for Redis:** Set REDIS_URL to rediss://... in production
- **API keys in headers only:** Never pass API keys in query strings
- **Monitor error logs:** Stack traces no longer expose credentials
- **Trust socket address:** X-Forwarded-For can be spoofed by attackers

5. API Reference

RedisRateLimiter Class

Method	Parameters	Returns	Description
initialize()	None	Promise<boolean>	Connect to Redis
middleware()	None	Express middleware	Create middleware
getTier(apiKey)	string	Promise<string>	Get tier for key
setTier(apiKey, tier)	string, string	Promise<object>	Set tier for key
getMetrics()	None	object	Get metrics
close()	None	Promise<void>	Close connection

Response Headers

Header	Description	Example
RateLimit-Limit	Maximum requests per window	100
RateLimit-Remaining	Requests remaining	42
RateLimit-Reset	Unix timestamp for reset	1702814460
Retry-After	Seconds until next request	45

6. Metrics & Monitoring

The rate limiter exposes Prometheus-ready metrics via /api/rate-limit/metrics endpoint.

Metric	Type	Description
totalRequests	Counter	Total requests processed
allowedRequests	Counter	Requests allowed through
deniedRequests	Counter	Requests denied (429)
redisErrors	Counter	Redis connection errors
redisConnected	Gauge	Redis connection status
requestsPerSecond	Gauge	Current request throughput

Dashboard Component

A React component (RateLimiterMetrics.jsx) is available for visualizing rate limiter metrics in the dashboard. It polls the metrics endpoint every 5 seconds and displays requests/sec, allowed/blocked counts, and Redis status.