Lecture: Scaling Laravel from 1K to 5M Users (Practical Comparison)

0) A key idea before anything else

The same code can run in both cases… but what changes dramatically is:

- Architecture
- Database design (DB Design + Indexing + Sharding/Partition)
- Caching strategy
- Background processing (Queues + Jobs)
- Deployment + Infrastructure
- Observability
- Cost control

1) Requirements and goals

At 1,000 users

- Goal: “works well” + “fast iteration”
- You can rely on:
  - One or two servers
  - A single DB
  - Simple cache or none
  - Simple queue or Cron

At 5,000,000 users

- Goal: “always on” + “handles pressure” + “no downtime” + “steady response time”
- You must think about:
  - High Availability
  - Horizontal scaling
  - Read/Write separation
  - Caching layers
  - Queue at scale
  - Zero-downtime deploy
  - Monitoring + Alerting
  - Rate limiting + Abuse protection

2) Performance vs capacity

1K

- Relatively few requests
- A few extra queries won’t show much
- A page can run 5–20 queries and pass

5M

- Any extra query = a problem
- N+1 destroys throughput
- You must:
  - do proper eager loading
  - paginate every listing
  - avoid loading unnecessary relations
  - compress payloads (API responses) and drop heavy fields

3) Database: the biggest difference

1K users

- One MySQL instance
- Simple indexes on (email, user_id, created_at)
- Basic backups
- Minimal lock pressure

5M users Treat data like a product of its own:

A) Indexing and query discipline

- Every production query should be:
  - selective (specific columns)
  - backed by the right index
  - avoiding full table scans
  - using composite indexes (e.g., user_id, status, created_at)

B) Read replicas

- Separate reads from writes:
  - writes on primary
  - reads on replicas
  - Laravel supports this in config

C) Partitioning / archiving

- Logs, events, and transaction tables explode in size
- Apply:
  - partition by date
  - or archive historical data

D) Lock management

- At 5M, a bad update across many rows can stall the system
- You need:
  - short transactions
  - avoid “update all” during peak time
  - queues for heavy operations

4) Caching: from “optional” to “essential”

1K

- Simple config cache
- File cache or small Redis

5M

- Redis (or Memcached) is a must, with layered caching:

Key cache types:

- Response/HTTP cache (when applicable)
- Query result cache (expensive queries)
- Object cache (user profile, settings, permissions)
- Rate-limit cache (protection)

Golden rules:

- Clear cache invalidation strategy
- Proper TTLs
- Organized cache keys

5) Background jobs & queues: the heart of 5M

1K

- Emails/notifications can be sync or a simple Cron

5M

- Any heavy task must go to a queue:
  - emails / SMS / push
  - report generation
  - imports/exports
  - image processing
  - webhook retries
  - analytics events

Laravel tooling that matters:

- Redis queue
- Laravel Horizon to monitor workers
- Priority queues:
  - high, default, low

6) Sessions and auth

1K

- File-based sessions are okay

5M

- With multiple servers:
  - sessions must be shared
  - use Redis session driver
  - tokens/JWT can be better for some systems
- Also consider:
  - password reset throttling
  - 2FA (for sensitive products)
  - device/session management

7) Deployment and infrastructure

1K

- One VPS + Nginx + PHP-FPM
- Manual deploy or Git pull

5M

- Load balancer + multiple app servers
- Auto-scaling (CPU/RPS driven)
- Zero-downtime deploy (Blue/Green or Rolling)
- Separate services:
  - App
  - DB
  - Redis
  - Queue workers
  - Scheduler
- CDN for assets (assets/images)

8) Observability: from “we’ll know when it breaks” to “know before it breaks”

1K

- Basic logs
- Some server metrics

5M

- You need:
  - Centralized logging
  - APM (traces)
  - Metrics + Alerting
  - SLO/SLA targets

Metrics to watch:

- p95 latency

- error rate
- DB slow queries
- queue lag
- cache hit ratio
- CPU/memory per node

9) Security and abuse protection

1K

- validation + auth + basic rate limit

5M

- Strong rate limiting on:
  - login
  - OTP
  - search endpoints
- WAF / bot protection (depending on product)
- anti-scraping measures
- permissions caching
- audit logging for sensitive actions

10) Laravel-specific items that matter

- `php artisan config:cache` and `route:cache` in production
- Use Redis for:
  - cache
  - sessions
  - queues
- Horizon for workers
- Octane (if it fits) for throughput
- Database pooling/connection tuning
- Disable debug tools in production

Quick comparison “as a table” (conceptually)

1K users

- Single server possible
- One DB
- light caching
- few jobs
- simple deploy
- basic monitoring

5M users

- multi-server + load balancer
- DB primary + replicas + partitioning

- layered caching
- heavy queues + priorities
- CI/CD + rolling deploy
- APM + alerts + SLOs

Practical roadmap: if you expect growth from 1K → 5M

1. Fix DB queries + indexing + pagination
2. Redis caching
3. Queues + Horizon
4. Read replicas
5. CDN + optimize assets
6. Observability
7. Auto-scaling + zero-downtime deploy
8. Partition/Archive for large tables