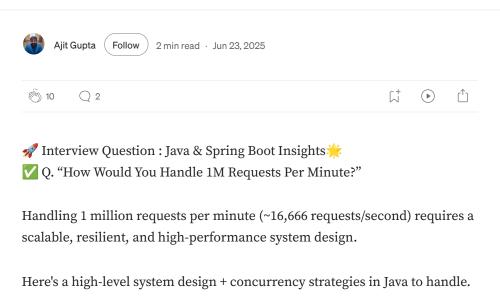






ın in



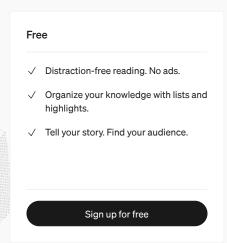


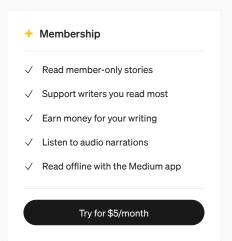
✓ High-Level System Design

1. Architecture Overview
Client -> API Gateway -> Load Balancer -> Web/Application Servers -> Service
Layer -> DB / Cache / MQ

# Medium

Sign up to discover human stories that deepen your understanding of the world.





X

Horizontally scalable (containers, Kubernetes) REST APIs using Spring Boot, Netty, or Vert.x

4. Asynchronous Processing (for high throughput) Use Message Queues like Kafka, RabbitMQ Queue writes, slow operations

#### 5. Database

Use read-replicas for scale NoSQL (e.g., Cassandra, MongoDB) for write-heavy SQL (e.g., PostgreSQL, MySQL) with caching

6. Caching LayerRedis / MemcachedCache hot data, responses, rate limits

#### 7. CDN

Offload static content (images, videos, JS) to a CDN (Cloudflare, Akamai)



#### 1. Use Thread Pooling Efficiently

Avoid creating threads manually. Use:

ExecutorService executor = Executors.newFixedThreadPool(100);

✓ For reactive workloads, consider ForkJoinPool or CompletableFuture.

## 2. Use Asynchronous & Non-Blocking I/O

Use Spring WebFlux, Project Reactor, or Vert.x for reactive programming. Prefer non-blocking I/O over traditional servlet thread-per-request.

## 3. Avoid Blocking Calls

Avoid using Thread.sleep, join, or any long CPU-bound operations in the request thread.

Use CompletableFuture.supplyAsync() for non-blocking service calls.

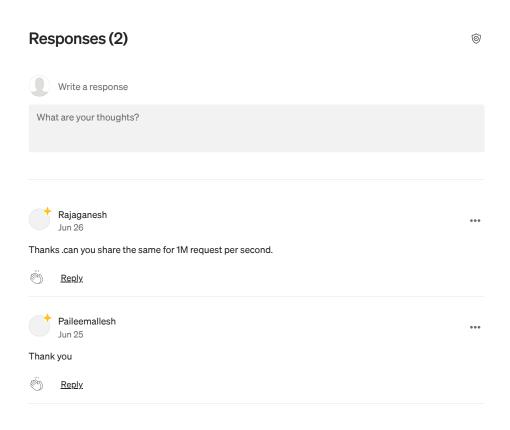


```
4. Leverage Caching
Use @Cacheable (Spring) or Redis for frequent, expensive queries:
@Cacheable("userProfile")
public UserProfile getUser(String userId) {
// DB call
5. Connection Pooling
Use HikariCP (default in Spring Boot) for efficient DB connections:
spring.datasource.hikari.maximum-pool-size=50
6. Use Backpressure / Throttling
Protect downstream systems with:
Bulkhead pattern (limit concurrent calls)
Circuit breaker (e.g., Resilience4j)
Rate limit using token bucket or leaky bucket algorithm
7. Monitoring & Auto-scaling
Use Prometheus + Grafana
Autoscale containers/pods based on CPU/memory/load
Sample Load Handling Strategy (for 1M RPM)
Tier: Scale Required
API Gateway: Auto-scaled, throttling
App Servers: 10-20 instances with 100-200 threads each
DB: Read replicas, connection pool, cache
Cache: Redis Cluster for horizontal scaling
MQ: Kafka for async buffering (e.g., emails, logs)
10 Q 2
                                                                L†
                                                                     Û
```

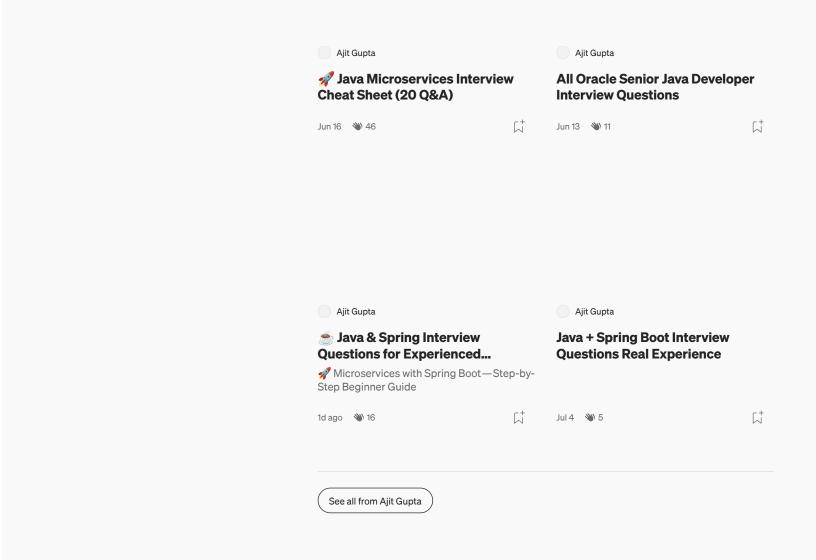




Follow



More from Ajit Gupta



## **Recommended from Medium**

