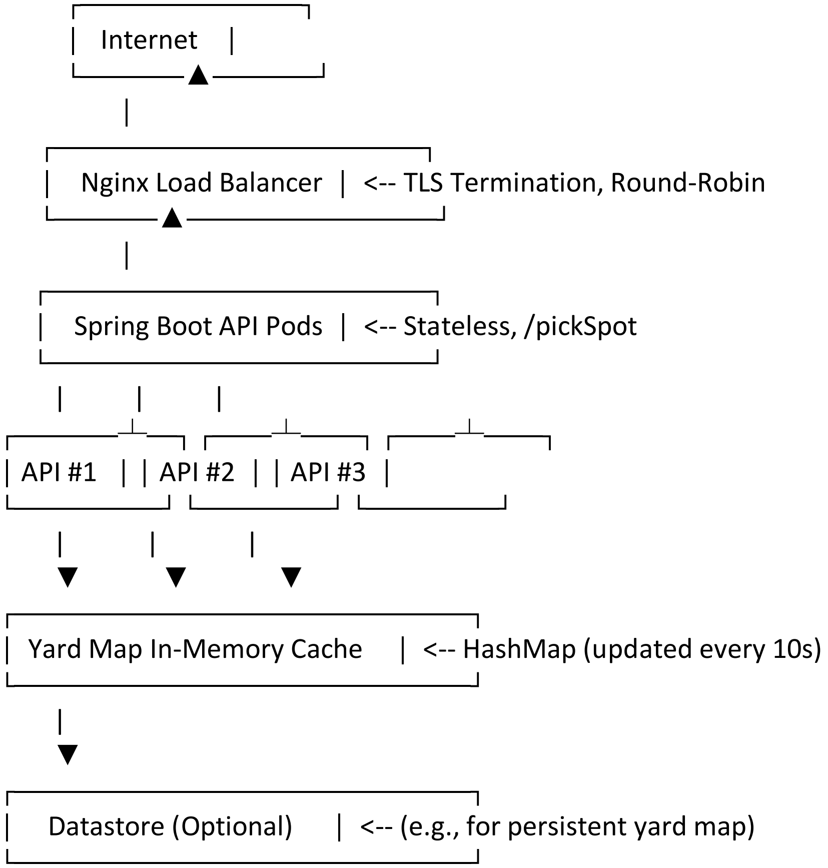
System Design

To create a system which remains fast under heavy traffic for your /pickSpot API, we must guarantee scalability, fault tolerance, and low latency even in high traffic times. Here is a detailing of how the system may be designed to ensure the requirements of 500 RPS handling and response time of ≤ 300 ms at peak hours.

1. System Architecture Diagram

Let's define the architecture first. We must ensure that the system is horizontally scalable, fault tolerant, and observability-friendly.



1. Architecture Diagram

Components:

Nginx (TLS Termination & Load Balancer) → routes to →

Spring Boot API Pods → read from →

In-Memory Cache (HashMap) ← refreshed every 10s

Optional DB (for logs or persistence)

Monitoring: Prometheus + Grafana

2. Component Summary

Nginx: Round-robin traffic balancing & TLS termination.

Spring Boot: Stateless API, horizontally scale easily.
In-Memory Cache: O (1) lookup from HashMap, refreshed every 10s.
Database (optional): For log or backup purpose, not read in real time.
Prometheus + Grafana: Monitor P95 latency, errors, uptime.

3. Concurrency Model
Each Spring Boot pod processes ~120 RPS.
For 500 RPS → use 5 pods.
Nginx evenly splits load.
Auto-scale if CPU > 70%.
Backpressure: Throttle via Nginx + pod autoscaling.

4. Scaling, Deployment & Monitoring
Horizontal Scaling: Automatically add API pods if CPU is more than 70%.
Blue-Green Deployment: Add new pod (e.g., port 9000), test, and route traffic through Nginx with zero downtime.

Monitoring:
Metrics:
Monitor P95 latency (<300ms),
4xx/5xx error rate.
Alerts:
P95 > 400ms for 5 min
Error rate >5% for 10 min