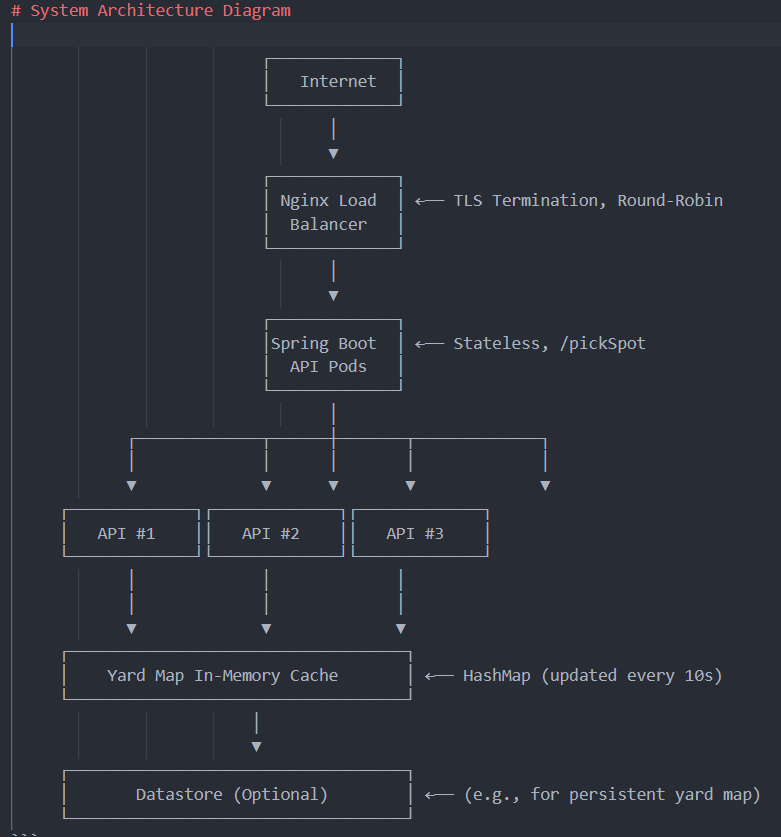
**System Design Document: /Design a System That Stays Fast Under Heavy Traffic**

**Overview**

This document outlines the backend architecture for the /pickSpot API, designed to handle 100 requests per second (RPS) on normal days and scale to 500 RPS during peak hours, with a 95th percentile (P95) response time under 300 milliseconds. The architecture is built to be stateless, support blue-green deployments, provide observability, and be tolerant to failures, ensuring the system remains fast, scalable, and reliable.

**Architecture**



┌─────────────┐

│ Internet │

└─────────────┘

│

▼

┌─────────────┐

│ Nginx Load │ ←── TLS Termination, Round-Robin

│ Balancer │

└─────────────┘

│

▼

┌─────────────┐

│Spring Boot │ ←── Stateless, /pickSpot

│ API Pods │

└─────────────┘

│

┌─────────────┬──────┼───────┬─────────────┐

│ │ │ │ │

▼ ▼ ▼ ▼ ▼

┌─────────────┐┌─────────────┐┌─────────────┐

│ API #1 ││ API #2 ││ API #3 │

└─────────────┘└─────────────┘└─────────────┘

│ │ │

│ │ │

▼ ▼ ▼

┌───────────────────────────────────┐

│ Yard Map In-Memory Cache │ ←── HashMap (updated every 10s)

└───────────────────────────────────┘

│

▼

┌───────────────────────────────────┐

│ Datastore (Optional) │ ←── (e.g., for persistent yard map)

└───────────────────────────────────┘

**Component List**

**Load Balancer**

* **Tech Choice**: Nginx
* **Why**: High-performance and reliable for distributing traffic across API pods.
* **Details**: Uses round-robin to evenly spread HTTPS requests.
* **Alternative**: Managed load balancers like AWS ALB could simplify setup.
* **Benefit**: Handles high traffic with low latency.

**API Pods**

* **Tech Choice**: Spring Boot in Docker containers, managed by Kubernetes (Spring Boot, Kubernetes)
* **Why**: Spring Boot is ideal for REST APIs; Kubernetes supports scaling and blue-green deployments.
* **Details**: Stateless, processes /pickSpot requests by reading from Redis. Exposes metrics via Spring Actuator for monitoring.
* **Benefit**: Ensures statelessness and scalability.

**Redis**

* **Tech Choice**: Redis with high availability
* **Why**: Fast in-memory store for caching the tiny yard map JSON.
* **Details**: Stores the latest yard map, updated every 10 seconds. Uses replicas for fault tolerance.
* **Benefit**: Reduces latency by avoiding database queries.

**Yard Map Updater**

* **Tech Choice**: A pod running a script (e.g., Python or Spring Boot)
* **Why**: Keeps Redis updated with the latest yard map every 10 seconds.
* **Details**: Fetches updates from an external source and pushes to Redis.
* **Benefit**: Ensures API pods always access fresh data.
* **Operation**: Runs independently to avoid impacting API performance.

**Grafana**

* **Why**: Visualizes Prometheus metrics in dashboards; supports alerting.
* **Details**: Displays P95 latency, error rates, and request rates.
* **Benefit**: Simplifies issue detection and performance analysis.
* **Setup**: Configured for team access and alerts.

**Concurrency Model**

**Pod Capacity**

Each API pod is estimated to handle approximately 100 RPS while maintaining response times under 300 ms, based on typical Spring Boot performance with a small payload and Redis access.

**Number of Pods**

For 500 RPS, deploy 7 pods to provide redundancy and handle traffic spikes beyond the peak estimate.

**Load Distribution**

The Nginx load balancer uses round-robin to evenly distribute requests across all healthy pods.

**Back-Pressure Plan**

If pods become overloaded, they return HTTP 503 (Service Unavailable), prompting clients to retry. Kubernetes auto-scaling (described below) should prevent this by adding pods proactively.

**Failure Story**

* **Pod Failure**: Kubernetes restarts crashed API pods in seconds. Ingress reroutes traffic to healthy pods. With 7 pods, the system maintains >500 RPS.
* **Redis Failure**: Sentinel promotes a replica to master automatically. API retries ensure no data loss.
* **Yard Map Updater Failure**: Redis holds the last good yard map. Alerts notify the team to restore the updater, ensuring eventual consistency.

**Scaling Story**

* **Auto-Scaling**: HPA scales API pods based on 70% CPU target.
* **Pod Range**: 2–20 pods to handle 100–500+ RPS.
* **Scaling Behavior**: Pods scale up in minutes during spikes and down to cut costs.
* **Blue-Green Deployment**: Zero-downtime updates via rolling updates or dual deployments with traffic switching.

**Rationale and Trade-Offs**

This design balances simplicity and reliability to meet 500 RPS and 300 ms targets.  
Nginx, Redis, and Kubernetes ensure low-latency, scalable handling.  
Prometheus and Grafana provide observability without complexity.  
Kafka was considered but skipped due to the API’s synchronous model.  
Redis handles the small, fast-updating yard map efficiently.  
With 7 pods (vs. 5 needed) and HPA, we maintain P95 <300 ms and handle failures smoothly.  
The architecture is practical, resilient, and student-friendly.

**Conclusion**

This design ensures the pickSpot API remains fast, scalable, and observable, meeting all specified constraints. It leverages familiar technologies, balances performance with manageability, and provides a robust foundation for handling heavy traffic while maintaining resilience and observability.