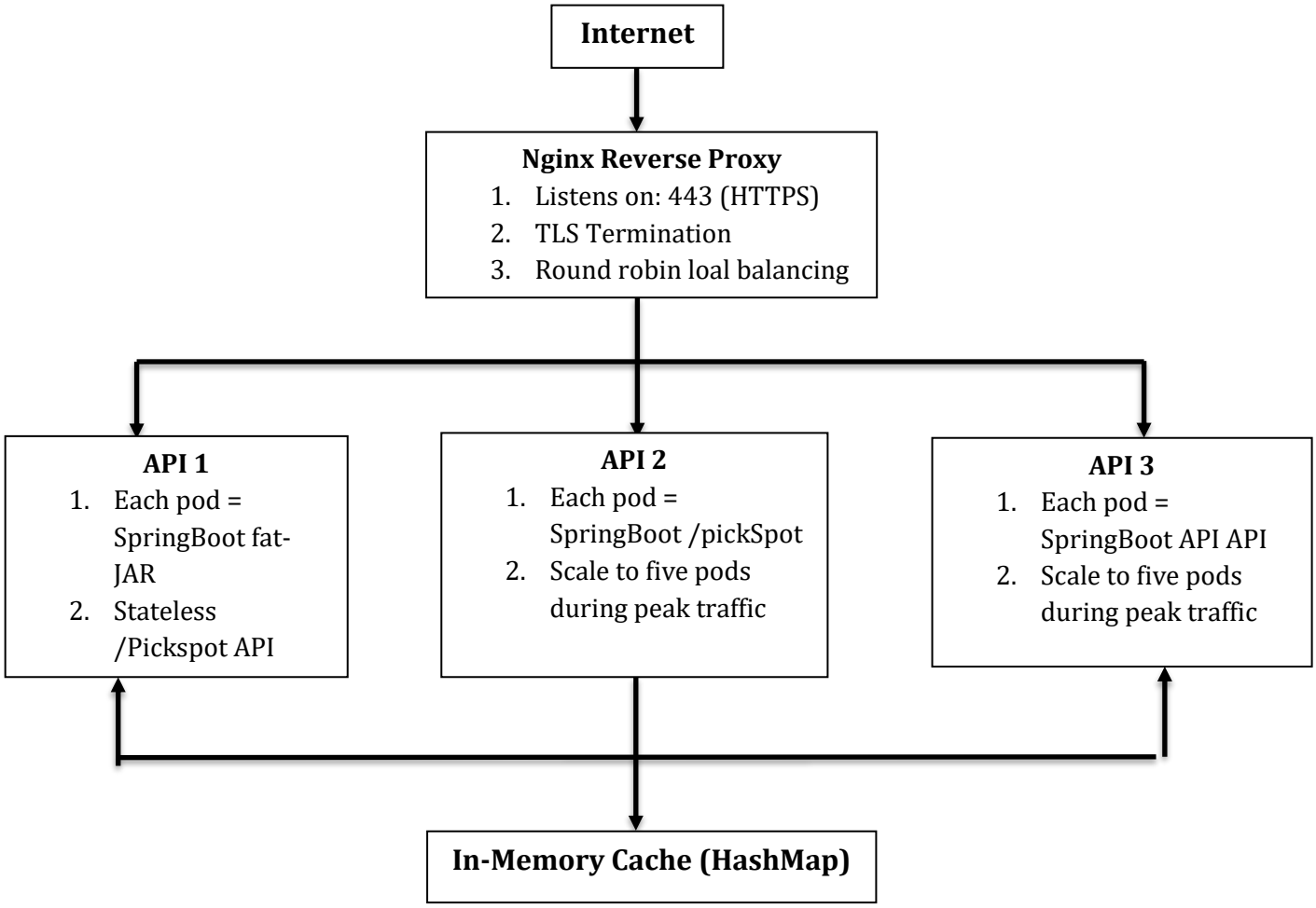
# System Design: Scalable /pickSpot API.

## 1. System Architecture Diagram




- refreshed every 10s from DB/file
- yard map lookup = O(1)

monitoring Stack (Prometheus + Grafana)
- Tracks latency, error rate, resource usage

## 2. Component List & Technology Choices

- Nginx:
  - acts as reverse proxy with TLS termination
  - round-robin load balancing between API pods
  - easily configurable, lightweight, and fast

- spring boot API Pods:

  - runs the /pickSpot logic as a stateless service
  - fast startup using fat-JARs, horizontally scalable
  - handles ~120 rps per pod under 200 ms latency

- in-memory Cache:
    - stores yard map in a local HashMap
    - refreshed every 10 seconds by background thread
    - ensures fast O(1) read access; no DB calls in hot path
- monitoring (Prometheus + Grafana):
    - tracks P95 latency, ,error rate, CPU usage
    - grafana dashboards for real-time monitoring
    - alerts configured for threshold breaches

## 3. Concurrency Model

on a normal day (100 rps), 3 API pods can handle the load (~40 rps each). during peak hours (500 rps), scale out to 5 pods (each handling ~100 rps)., Nginx evenly distributes requests using round-robin,. back-pressure is managed by rejecting connections if all pods are saturated; clients can retry failed requests.

## 4. Failure Handling

if one API pod crashes, Nginx automatically removes it from rotation., remaining pods handle the traffic (with ~20% capacity drop). if the in-memory cache updater thread fails, the last known yard map is used.. if Prometheus fails, alerts may be delayed, but API still runs. If Redis (if used) fails, .fallback to local cache; alert admin.

## 5. Scaling Strategy

I used Kubernetes HPA (Horizontal Pod Autoscaler): Add a new pod if CPU usage exceeds 70% for 1 minute. .maximum pod count set to 6 to handle sudden peak loads. supports blue-green deployments with zero downtime by running new version on a different port, testing,. and then switching traffic.

## 6. Metrics & Alerts

- Key Metrics:
    - P95 Latency (must stay under 300 ms)
    - Error Rate (4xx and 5xx response % over time)
    - CPU Usage per API pod
- Example Alert:
    - alert if: P95 latency > 400 ms for 5 consecutive minutes.

## Summery

This design outlines how the backend for the /Pickspot API can stay fast, reliable, and scalable even under heavy traffic from multiple cranes and yard apps calling it at the same time.

the goal is to keep the response time under 300 ms,, handle up to 500 requests per second during peak hours, and ensure the system doesn't break when one part fails.

to achieve this, I have used a stateless Spring Boot API with an in-memory cache for fast yard map lookups, load balancing through Nginx, and monitoring with Prometheus and Grafana. the system can automatically scale up when traffic increases, and it supports blue-green deployments to release updates without downtime.

this design keeps things simple, cost-effective, and production-ready for real-world use.