



SCALABLE IMAGE PROCESSING PLATFORM WITH KUBERNETES

Presented By : Yuxin Wang
Jingwen Huang
Lan Wang



TABLE OF CONTENT

- 01** INTRODUCTION
- 02** SYSTEM ARCHITECTURE
- 03** IMPLEMENTATION
- 04** RESULTS AND ANALYSIS
- 05** FUTURE DIRECTIONS
- 06** CONCLUSION



INTRODUCTION

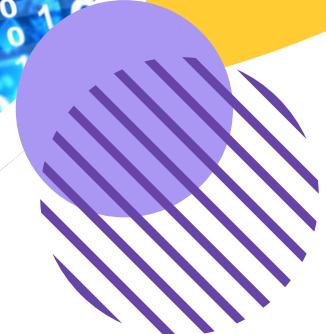
Traditional image processing systems suffer from:

- ✗ High latency due to sequential task execution
- ✗ Poor scalability from monolithic designs
- ✗ Single points of failure, causing service downtime
- ✗ Inability to handle traffic spikes without autoscaling

Goal: Build a scalable, resilient, and modular image processing platform

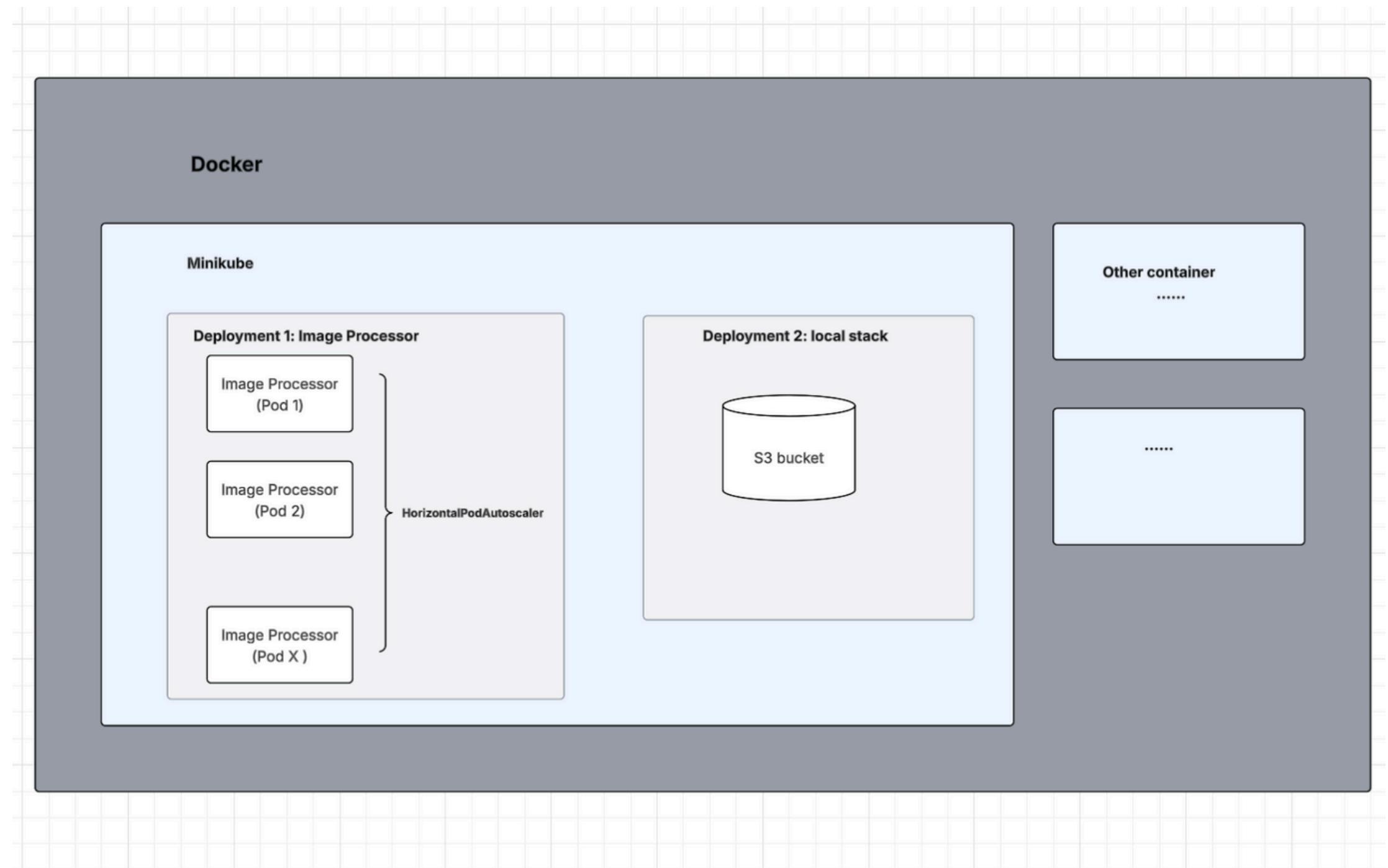
Key Features:

- REST API for image upload & transformations
- Java microservices (resize, watermark, filter)
- Kubernetes HPA for autoscaling
- S3-compatible storage via LocalStack
- Local deployment on Minikube



SYSTEM ARCHITECTURE

- **Containerized Microservices:** Each component runs in isolated Docker containers for portability and consistency.
- **Minikube Cluster:** Simulates cloud-native Kubernetes environment locally.
- **Image Processor Deployment:**
 - Stateless pods for resize, watermark, filter operations
 - Scales via Horizontal Pod Autoscaler (HPA) based on CPU load
- **LocalStack Deployment:**
 - Emulates AWS S3 for local image storage
 - Interacts with AWS SDK over port 4566
- **Extensible Design:**
 - Supports additional containers (e.g., monitoring, frontend, logging)



IMPLEMENTATION

- **Tech Stack:** Java (Spring Boot), Docker, Kubernetes, LocalStack (S3)
- **Core Endpoints:**
 - POST /resize, /watermark, /filter for image processing
 - GET /images/{imageKey} to retrieve results
- **Local Dev: Docker Compose + LocalStack for fast, offline testing**
- **K8s Deployment:**
 - Managed with YAML (Deployments, Services, HPA, Jobs)
 - Autoscaling via CPU thresholds
 - ConfigMaps for environment setup
- **Storage:** S3 API emulated with LocalStack; integrated
 - via AWS SDK



RESULTS & ANALYSIS

Load Testing Overview:

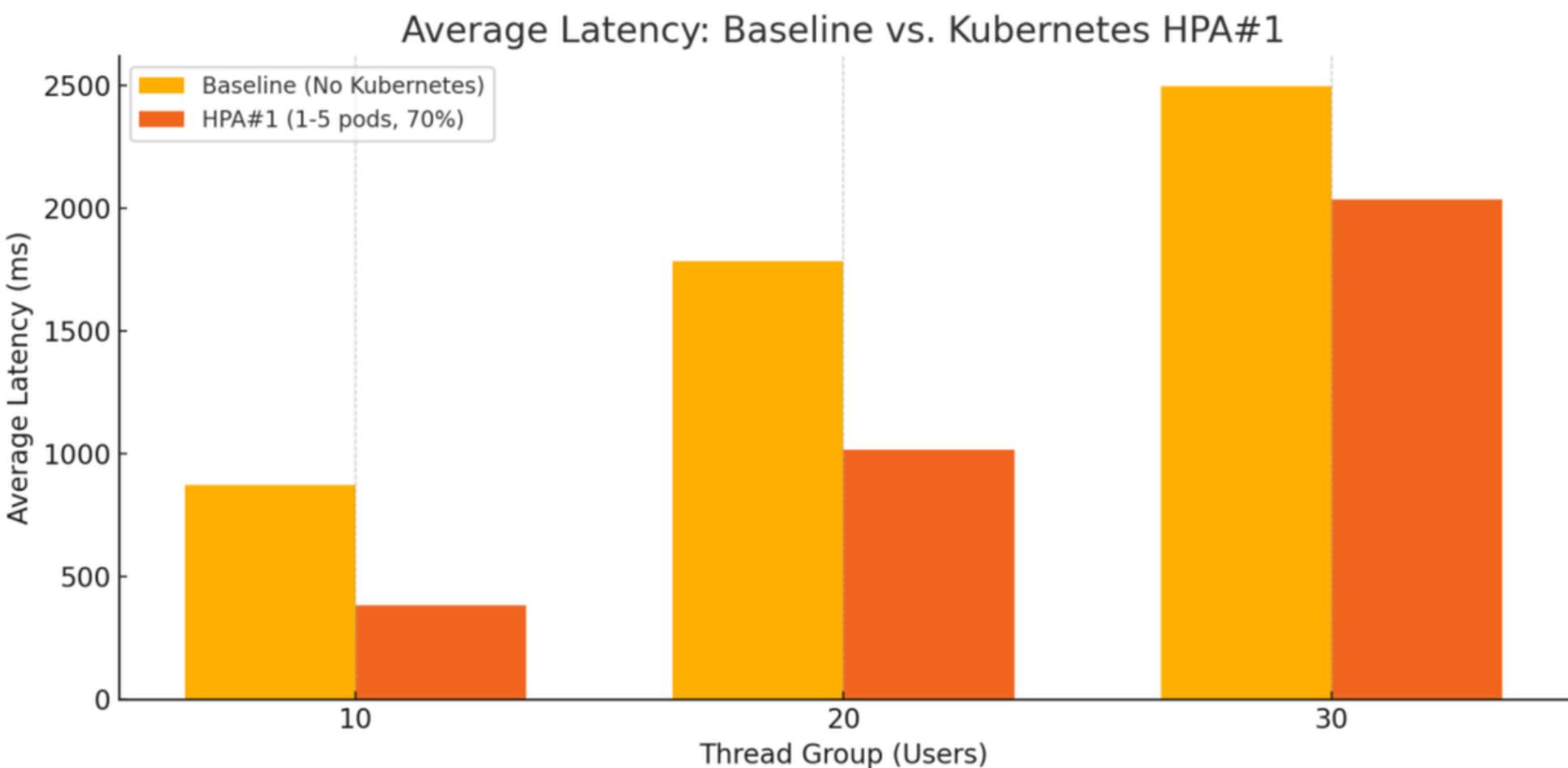
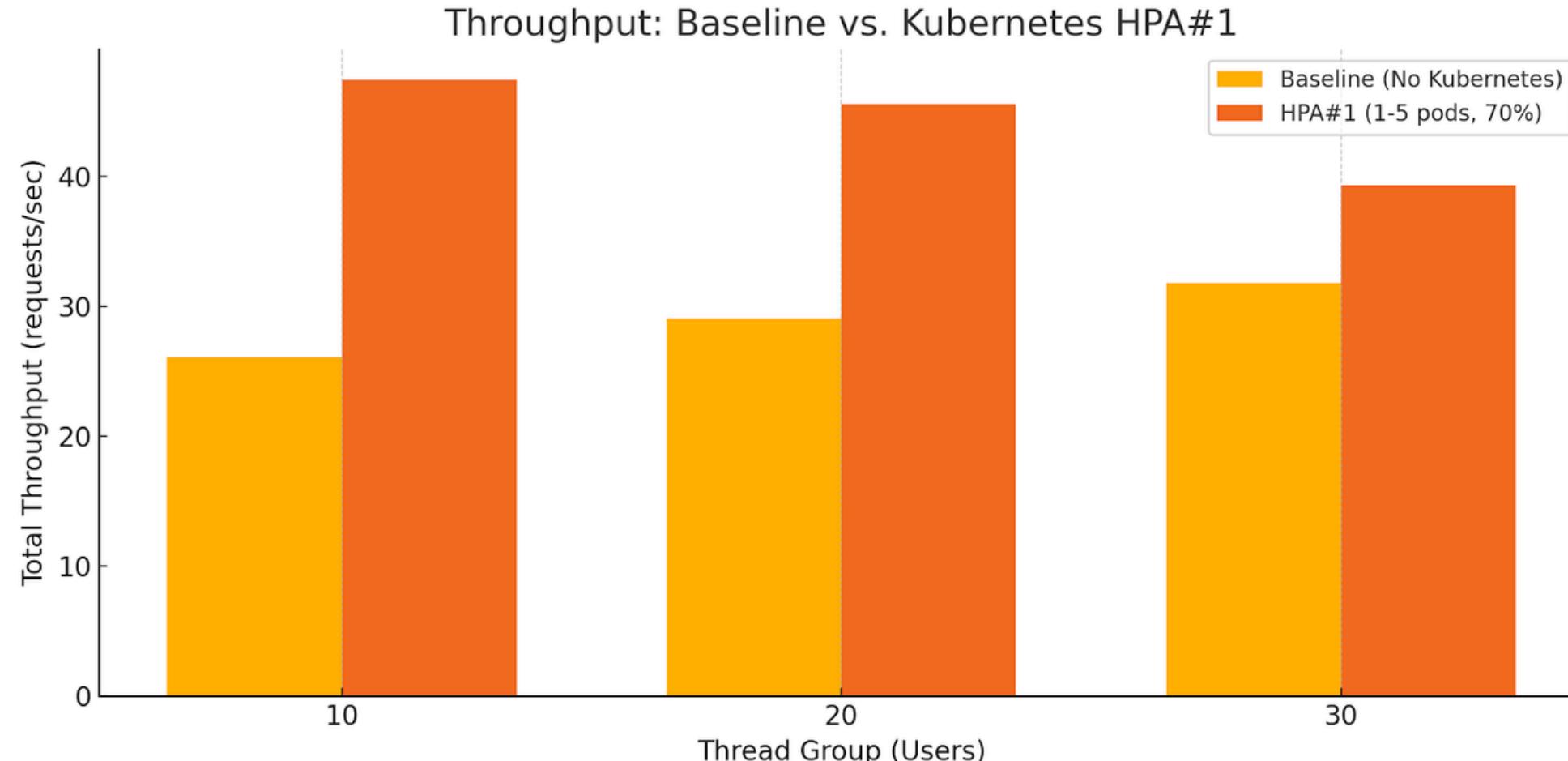
- Tested at 3 load levels: Base 10, 20, 30 (1K-3K requests)
- Focused on *POST* endpoints (resize, filter, watermark)
- Metrics:
 - Average latency
 - 90/99th percentile latency
 - Throughput (requests/sec)
 - Failed request percentage



RESULTS & ANALYSIS

Baseline v.s. Kubernetes (HPA#1):

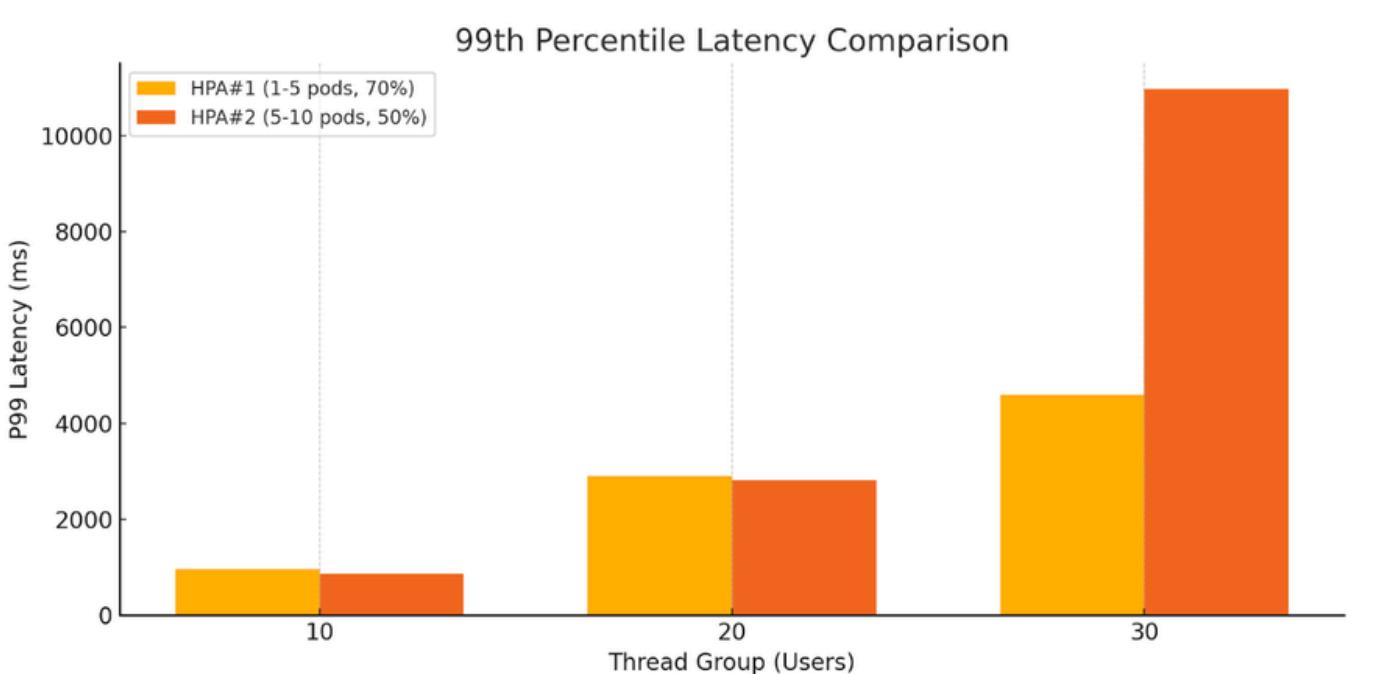
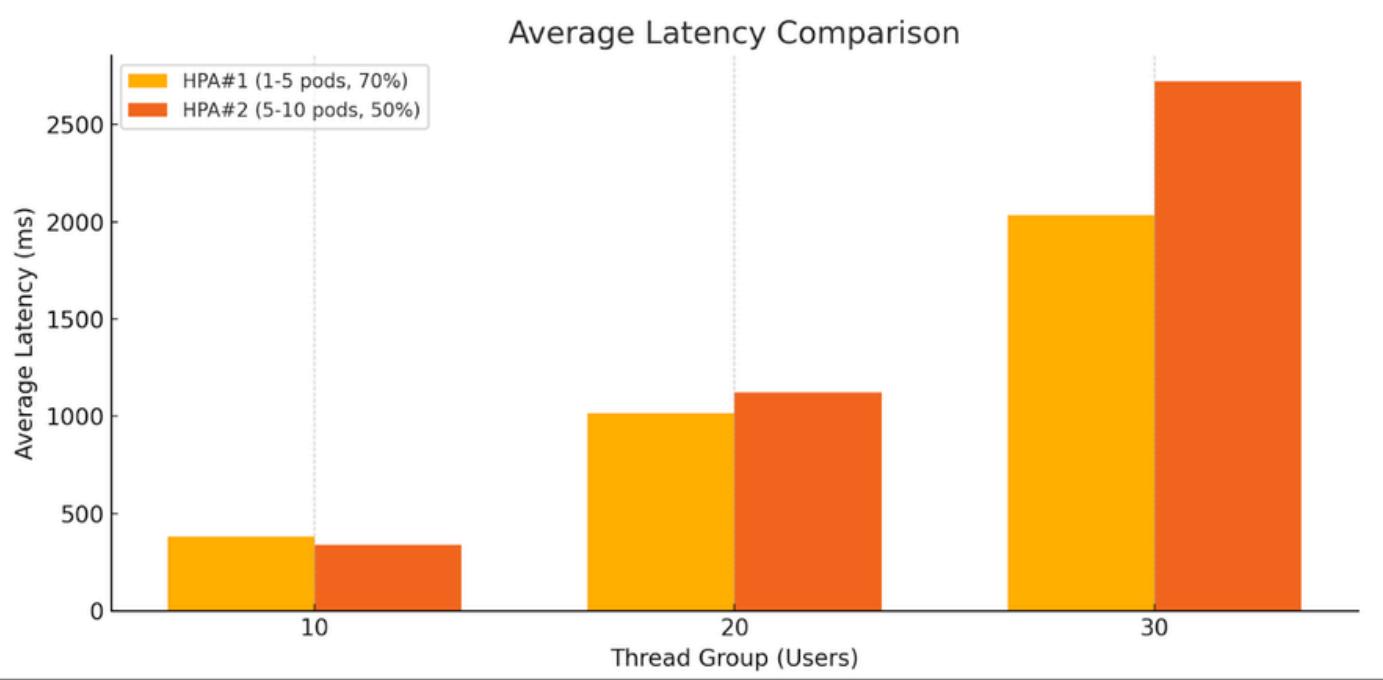
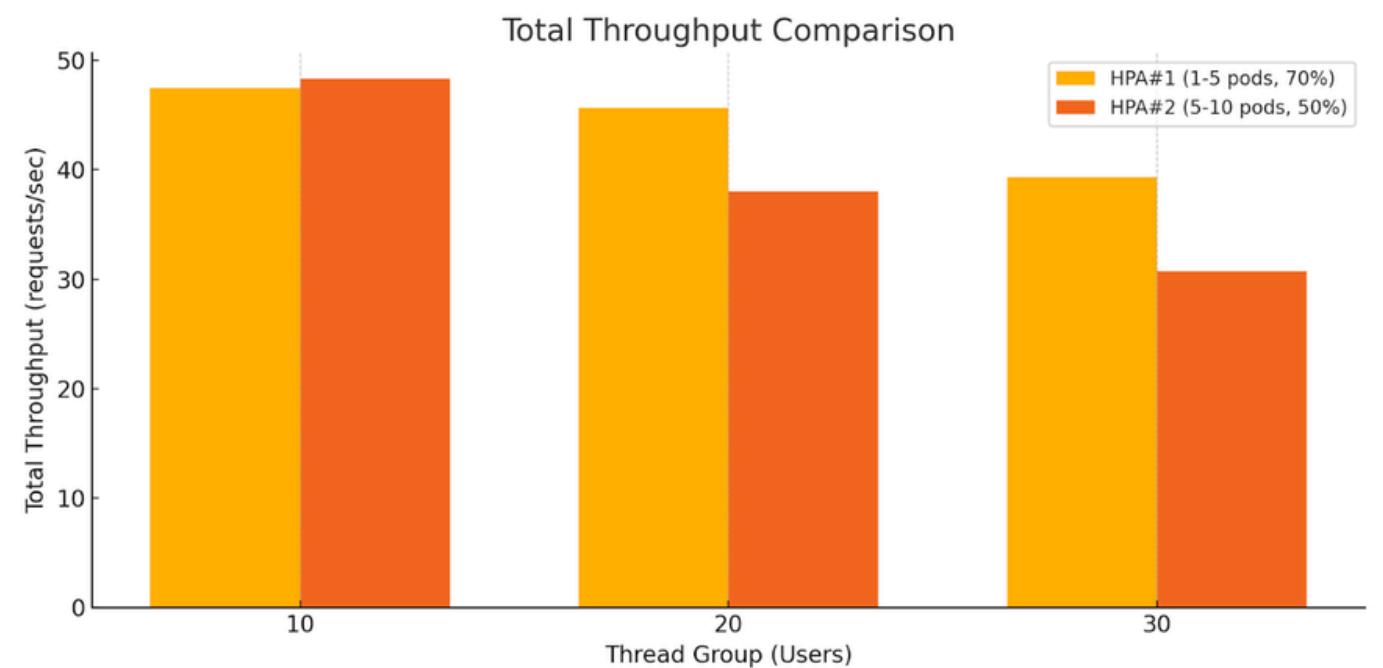
- Baseline server hit throughput limits early
- Kubernetes (HPA#1) scaled effectively with higher throughput
- Autoscaling reduced but didn't eliminate latency growth
- Kubernetes better handled concurrency under pressure



RESULTS & ANALYSIS

Comparing HPA#1 v.s. HPA#2:

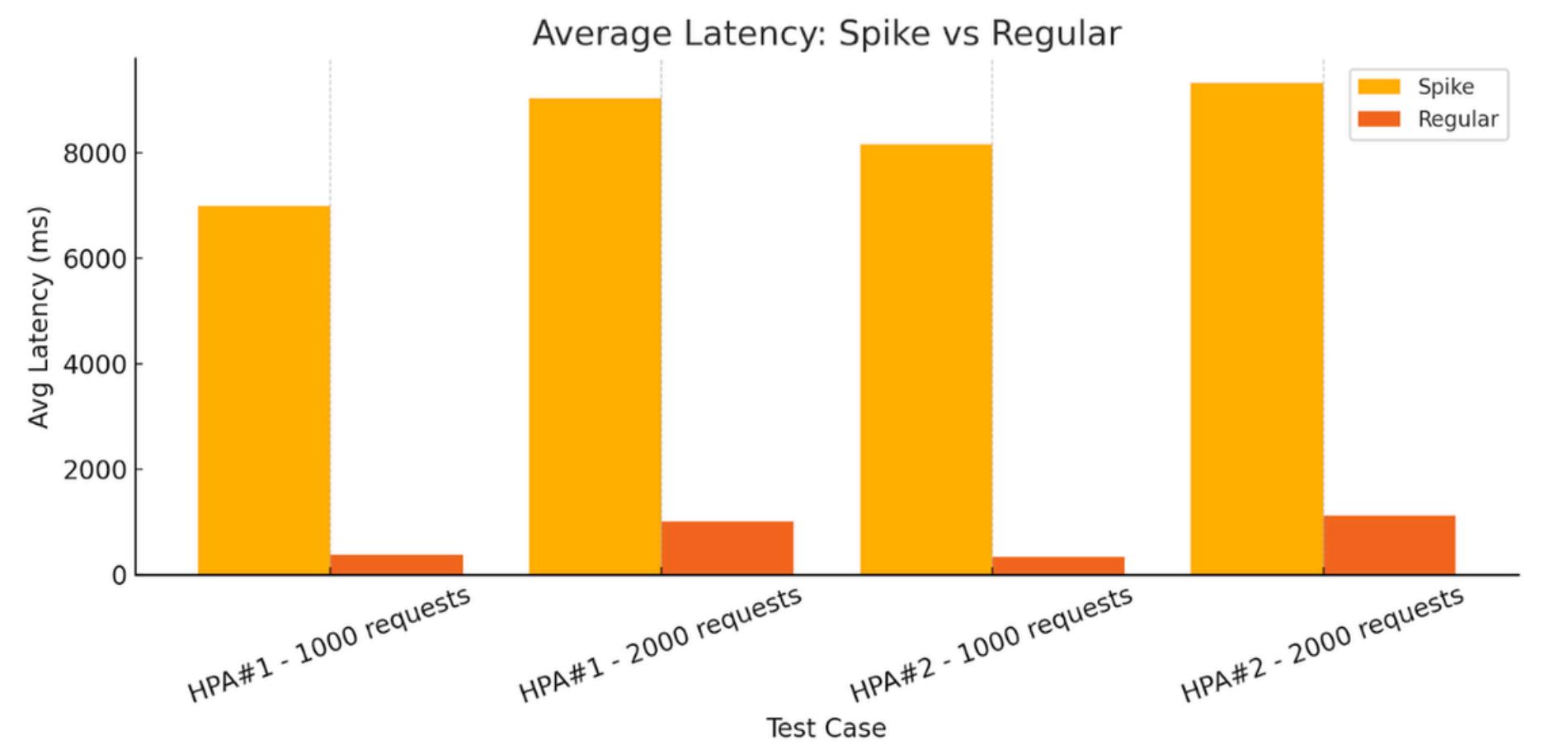
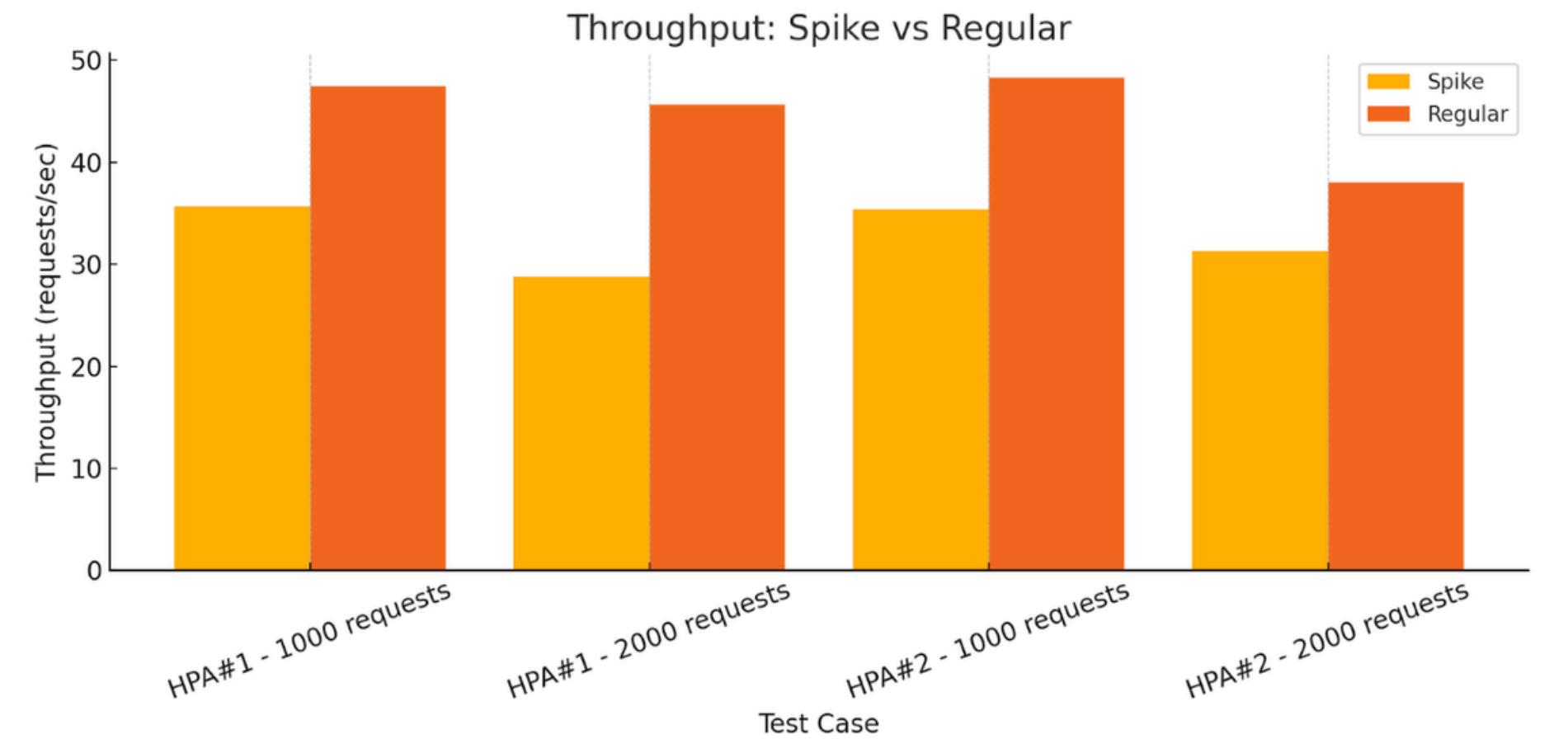
- HPA#1: 1-5 pods, 70% CPU threshold
- HPA#2: 5-10 pods, 50% threshold (more aggressive)
- HPA#1 had slightly better throughput at high load
- HPA#2's frequent scaling caused pod churn and delays
- Having more pods doesn't guarantee better performance if they're not fully initialized.



RESULTS & ANALYSIS

Spike Test Results:

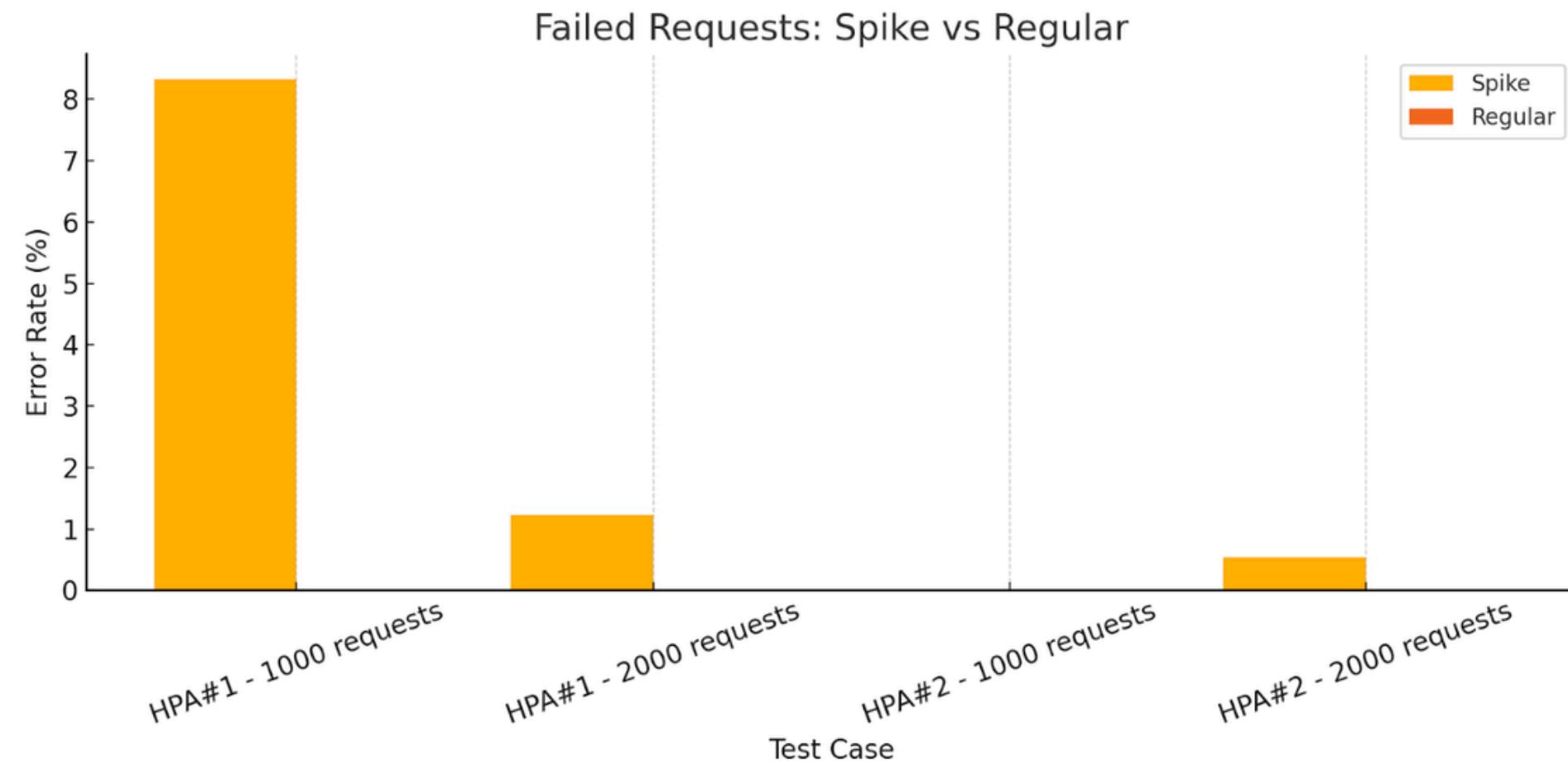
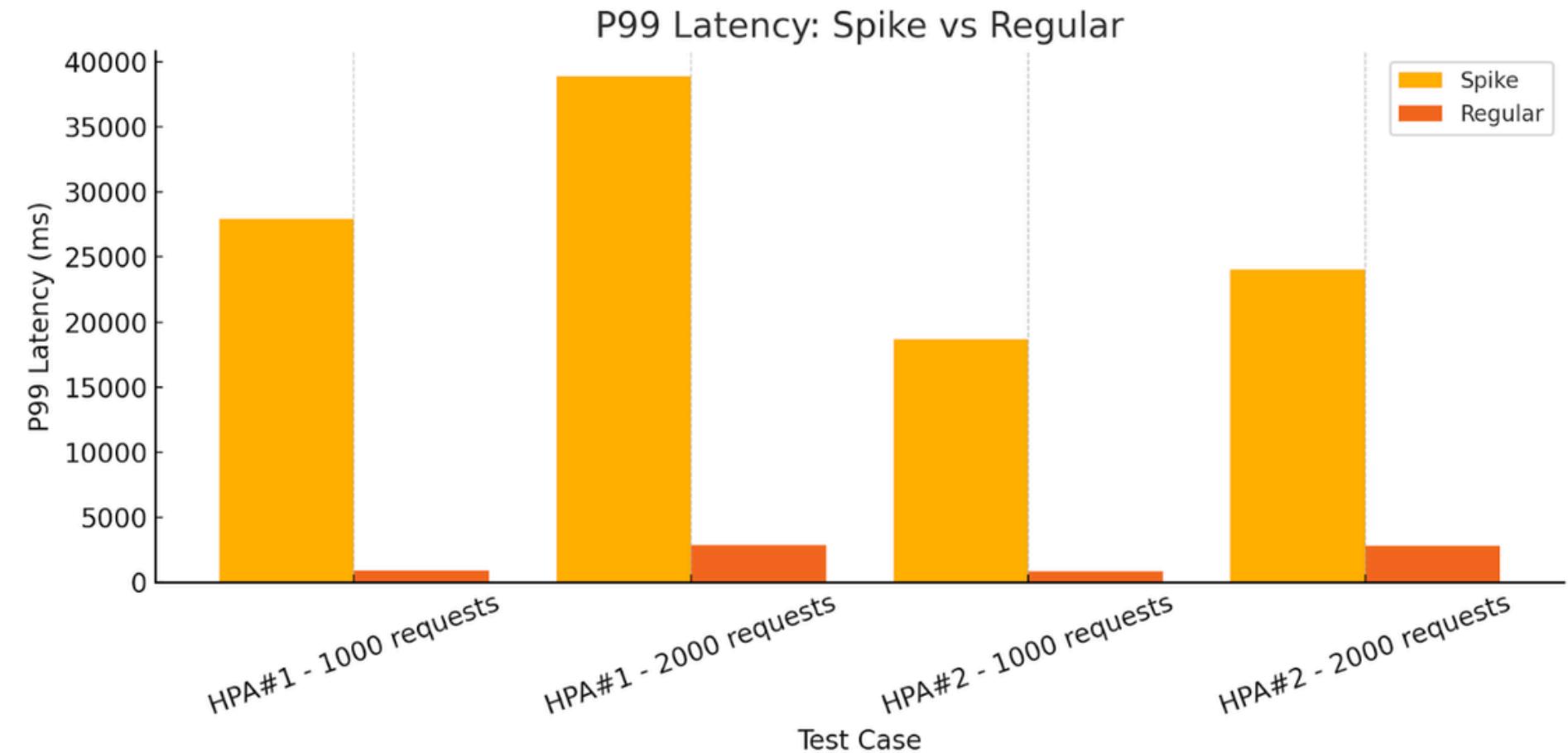
- Sudden traffic bursts: 3000-6000 requests
- HPA#2 handled spikes better with lower latency, 0% errors
- HPA#1 failed ~8% of requests due to pod startup lag
- Pre-warming pods or higher minReplicas recommended



RESULTS & ANALYSIS

Spike Test Results (continued):

- Sudden traffic bursts: 3000-6000 requests
- HPA#2 handled spikes better with lower latency, 0% errors
- HPA#1 failed ~8% of requests due to pod startup lag
- Pre-warming pods or higher minReplicas recommended



RESULTS & ANALYSIS

Key Bottlenecks & Recommendations:

- Cold starts slow response—keep warm pods
- POST ops are CPU-bound—optimize processing logic
- Autoscaler tuning essentials for stability
- Add Prometheus & Grafana for smart autoscaling





FUTURE DIRECTIONS

- **Support batch jobs and chained processing:**
 - Extend functionality by allowing batch image uploads and multi-step transformations in a single workflow (e.g., resize → filter → watermark).
- **Deploy on AWS EKS or GKE**
 - Moving from Minikube to a managed Kubernetes platform like **AWS EKS** or **Google Kubernetes Engine (GKE)** would enable real-world scalability testing. It also offers access to production-grade services (IAM, CloudWatch, real S3) and better reflects enterprise deployment environments.
- **Implement CI/CD with GitHub Actions or Jenkins**





CONCLUSION



- Built a modular, autoscalable, fault-tolerant image processing system
- Validated its performance with simulated workloads
- Learned valuable lessons in container orchestration, autoscaling, and benchmarking
- Ready to evolve into a production-grade pipeline



THANK YOU