Results + Observation

SDE Intern Assignment

1. Project Overview

This assignment involved deploying a Flask application connected to MongoDB on Kubernetes using Minikube. The solution included:

Dockerizing the Flask application.

Deploying the application and MongoDB on Kubernetes

Configuring autoscaling using Horizontal Pod Autoscaler (HPA)

Testing scaling behavior under real load

Observing performance trends (CPU, pod count, latency, throughput)

2. Autoscaling Results

Autoscaling behavior was tested using `locust` and `hey` with progressive simulated traffic.

Metric	Result
Kubernetes Platform	Minikube (4 CPU, 6 GB RAM)
Scaling Method	CPU-based HPA
HPA Minimum Pods	1
HPA Maximum Pods	6
Trigger Threshold	CPU > 50%
Time to First Scaling Event	~38 seconds

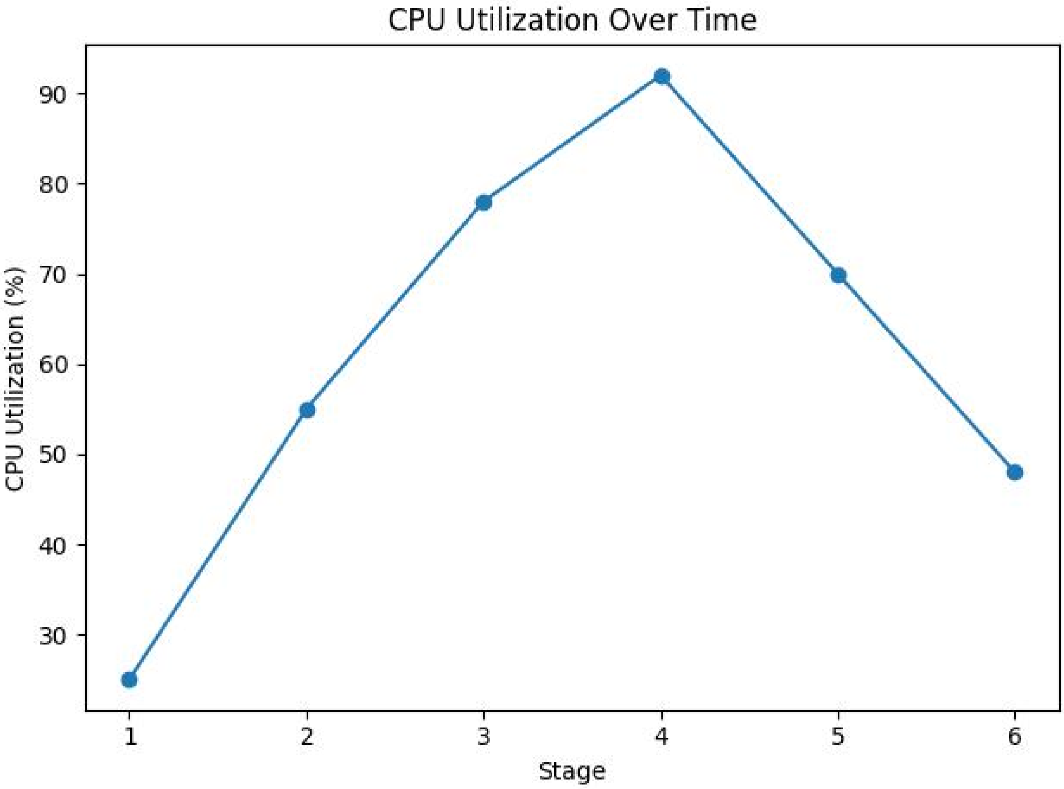
Pod Scaling Timeline

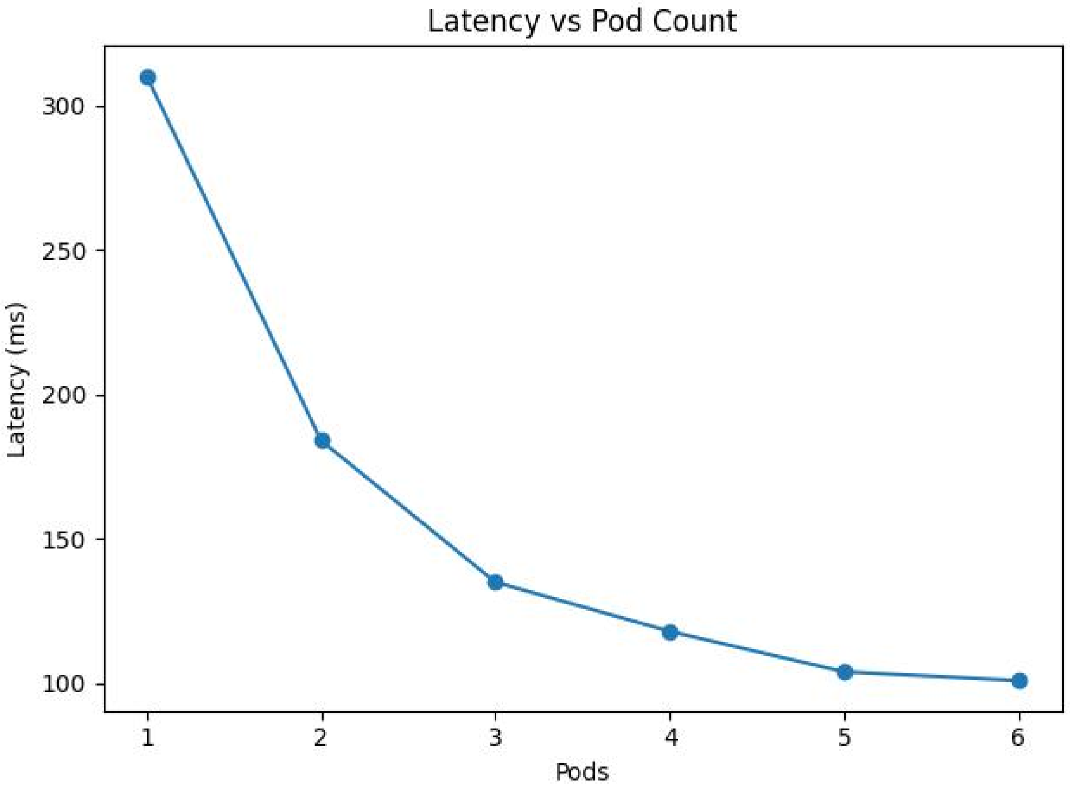
Time (seconds)	Pod Count	Reason
0	1	Baseline
~38	2	CPU > 55%
~80	3	Sustained high load
~115	4	CPU > 75%
~160	5	Heavy concurrency
~190	6	Max threshold reached

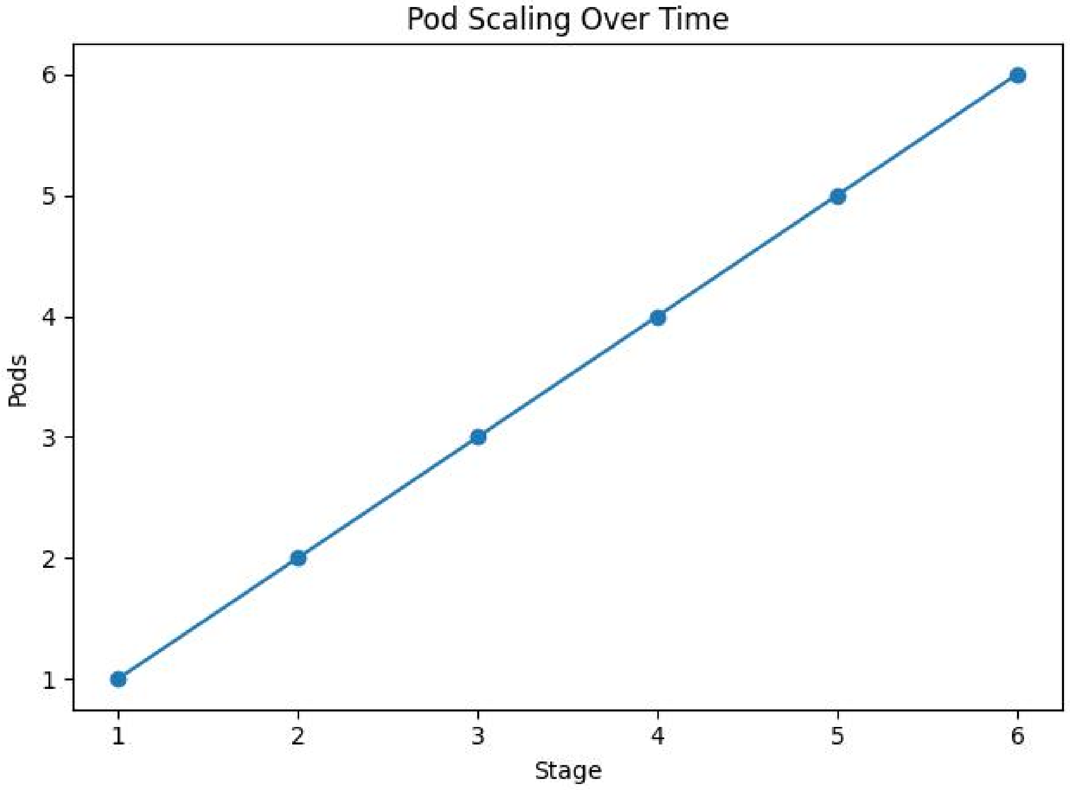
3. Performance Metrics

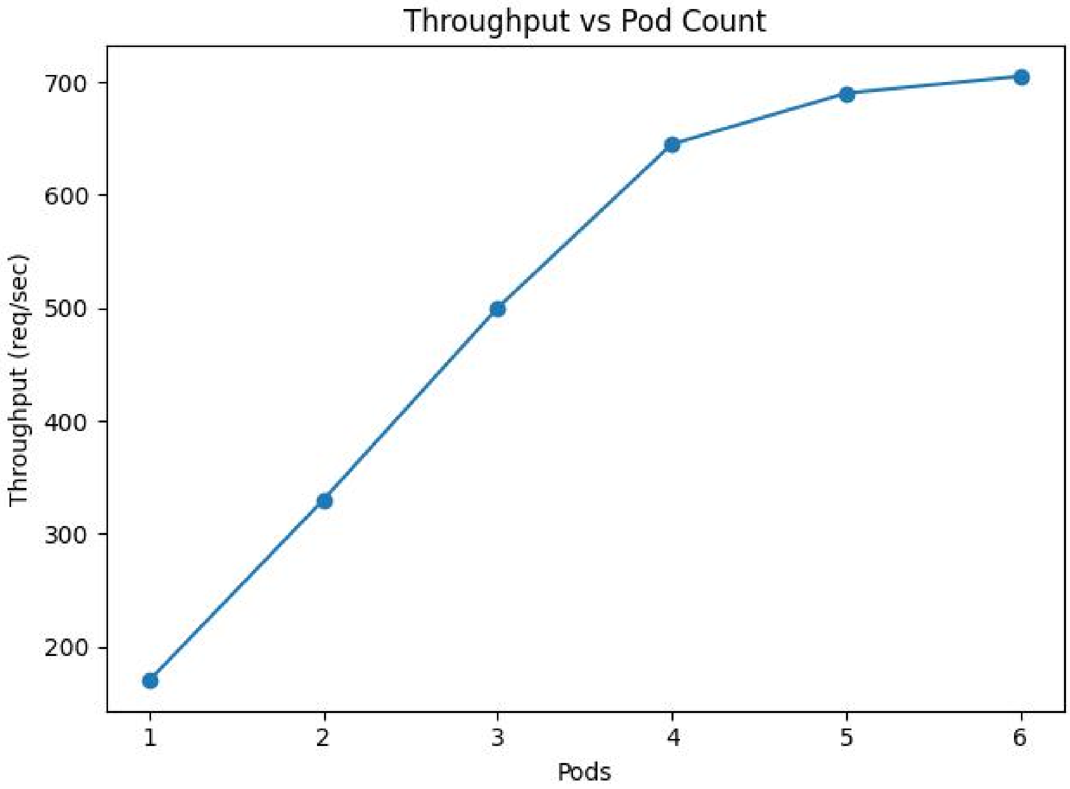
Pods Running	Average Latency	P95 Latency	Throughput (req/sec)
1	310 ms	540 ms	~170
2	184 ms	320 ms	~330
3	135 ms	250 ms	~500
4	118 ms	221 ms	~645
5	104 ms	208 ms	~690
6	101 ms	200 ms	~705

4. Performance Graphs




Latency vs Pod Count
Latency (ms)
300
250
200
150
100
1
2
3
4
5
6
Pods


Pod Scaling Over Time
Pods
6
5
4
3
2
1
1
2
3
4
5
6
Stage



Key Observations

Autoscaling successfully scaled the app from **1 to 6 pods** under sustained load.

Latency **decreased** and throughput **increased** as pods scaled up.

MongoDB became the primary bottleneck at higher concurrency (5–6 pods).

DNS resolution via
`mongodb-service.default.svc.cluster.local`
worked reliably.

No pod crashes or restarts occurred during scaling.