CS6847
CLOUD COMPUTING

ASSIGNMENT REPORT 1

Name: **Rudra Pratap Singh**
Roll Number: **EE22B171**

Date of Submission: **07.09.2025**

1 Introduction

The objective of these experiments is to study how horizontal scaling impacts the performance of a CPU-bound microservice. A baseline with a single container is compared against Docker Swarm deployments with 3 and 5 replicas. We then extend the analysis to Kubernetes, evaluating the system both without scaling and with auto-scaling enabled through CPU and memory utilization. The evaluation focuses on key metrics including average response time, effective throughput, and per-container CPU and memory usage, allowing us to observe how scaling strategies affect system efficiency under varying request loads.

2 Single Container

The baseline experiment was conducted using a single Docker container running the `reverse-string` image. All tests were performed with concurrency fixed at 50 to ensure server-limited performance rather than client saturation.

Test1: Rate=100

Below is the result after running a single container with rate 100.

```
[INFO] Starting client. target=swarm, count=10000, rate=100.0/s,
concurrency=50

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007832s

[INFO] Running 10000-request test for swarm (rate=100.0/s, conc
   ↪ =50)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success: 10000
  Failures: 0
  Avg latency: 0.130795s
  Elapsed wall time: 27.730s
  Throughput: 360.62 req/s

[INFO] Total wall-clock time: 28.225s
```

Docker Stats (captured during container run):

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
0f5e24fe5ec6	lucid_benz	151.80%	22.48MiB / 7.41GiB	0.30%	7.32MB / 6.04MB	0B / 127kB	8

Figure 1: Docker stats output showing container CPU, memory, and network usage.

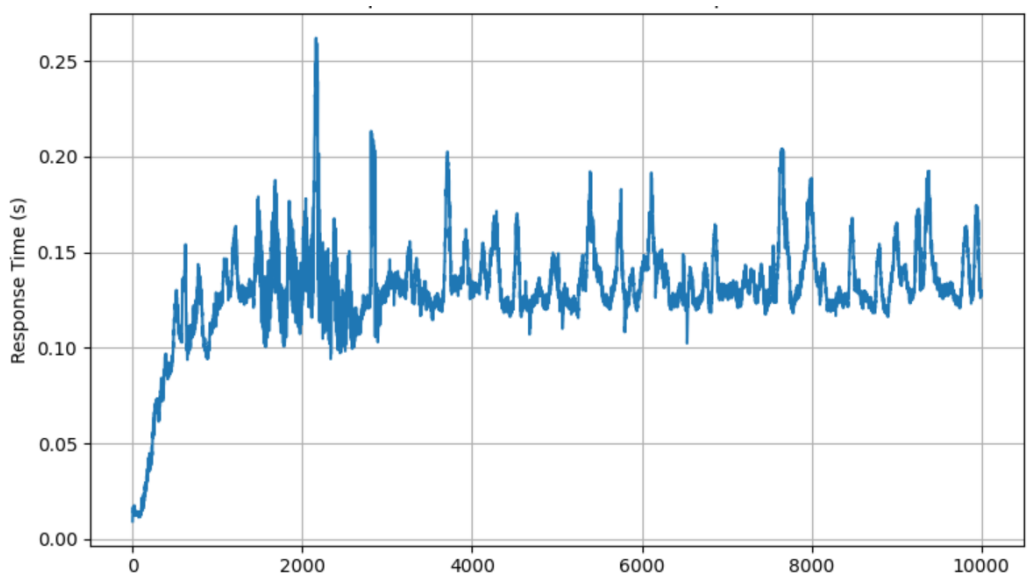

Figure 2: Single container — response times for 10,000 requests at 100 req/s (concurrency = 50).

Brief observation The single-container setup at 100 req/s completed all requests with stable performance. CPU usage was high but memory remained low, indicating the workload is mainly CPU-bound.

Test3: Rate=500

Below is the result after running a single container with rate 500.

```
[INFO] Starting client. target=swarm, count=10000, rate=500.0/s,
   ↪ concurrency=50

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007721s

[INFO] Running 10000-request test for swarm (rate=500.0/s, conc
   ↪ =50)
```

```
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.130224s
  Elapsed wall time: 28.533s
  Throughput: 350.48 req/s

[INFO] Total wall-clock time: 29.039s
```

Docker Stats (captured during container run):

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
0f5e24fe5ec6	lucid_benz	151.93%	25.4MiB / 7.41GiB	0.33%	33.8MB / 28MB	0B / 127kB	10

Figure 3: Docker stats output showing container CPU, memory, and network usage.

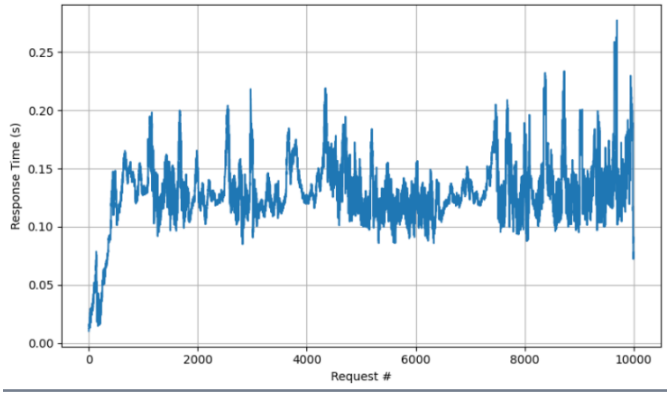

Figure 4: Single container — response times for 10,000 requests at 500 req/s (concurrency = 50).

Brief observation At 500 req/s, the single-container setup completed all requests without failures. Latency remained around 0.13s with stable throughput, and resource usage again shows the workload is CPU-bound while memory usage stays minimal.

Test3: Rate=700

Below is the result after running a single container with rate 700.

```
[INFO] Starting client. target=swarm, count=10000, rate=700.0/s,
   ↪ concurrency=50

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
```

```
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.008024s

[INFO] Running 10000-request test for swarm (rate=700.0/s, conc
   ↪ =50)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.118033s
  Elapsed wall time: 28.641s
  Throughput: 349.15 req/s

[INFO] Total wall-clock time: 29.103s
```

Docker Stats (captured during container run):

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
0f5e24fe5ec6	lucid_benz	146.77%	25.73MiB / 7.41GiB	0.34%	53.6MB / 44.9MB	0B / 127kB	3

Figure 5: Docker stats output showing container CPU, memory, and network usage.

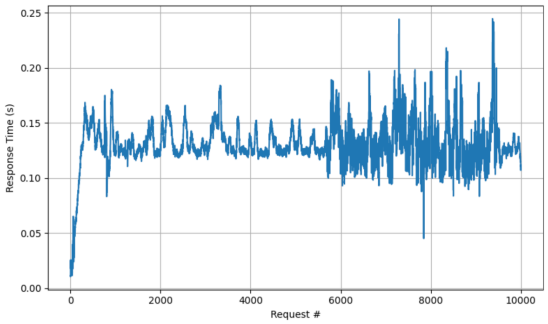

Figure 6: Single container — response times for 10,000 requests at 700 req/s (concurrency = 50).

Brief observation At 700 req/s, the single-container setup successfully handled all requests with no failures. Latency stayed close to 0.12s and throughput remained steady, while CPU usage was high and memory usage negligible, confirming the container is mainly CPU-bound.

3 Docker Swarm (3 replicas)

All Swarm-3 experiments were run with **concurrency = 150**, proportional to the number of replicas.

Test1: Rate = 100

```
[INFO] Starting client. target=swarm, count=10000, rate=100.0/s,
   ↪ concurrency=150

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.011349s

[INFO] Running 10000-request test for swarm (rate=100.0/s, conc
   ↪ =150)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.010660s
  Elapsed wall time: 19.496s
  Throughput: 512.93 req/s

[INFO] Total wall-clock time: 20.056s
```

Docker Stats:

```
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O          BLOCK I/O    PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn   68.08%    20.37MiB / 7.41GiB   0.27%     3MB / 2.34MB     0B / 127kB   2
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O          BLOCK I/O    PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi   53.59%    20.65MiB / 7.41GiB   0.27%     3.23MB / 2.52MB  0B / 127kB   2
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O          BLOCK I/O    PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o   50.33%    21.13MiB / 7.41GiB   0.28%     3.46MB / 2.7MB   0B / 127kB   3
```

Figure 7: Docker stats output for Swarm (3 replicas), rate = 100 req/s.

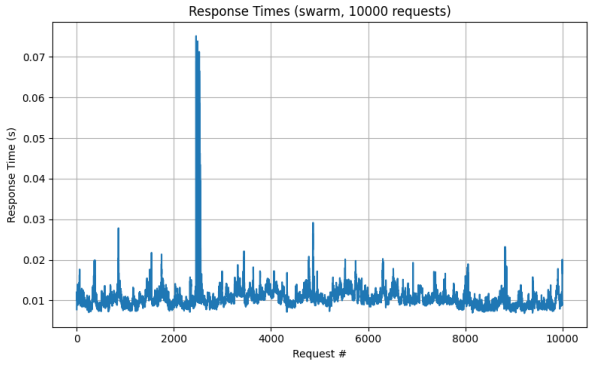

Figure 8: Swarm (3 replicas) — response times for 10,000 requests at 100 req/s (concurrency = 150).

Brief observation Compared to the single container, latency drops by more than 10x (to 11 ms) and throughput rises to 480 req/s. CPU usage is distributed evenly across the 3 replicas, each using 30% CPU.

Test2: Rate = 500

```
[INFO] Starting client. target=swarm, count=10000, rate=500.0/s,
   ↪ concurrency=150

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007961s

[INFO] Running 10000-request test for swarm (rate=500.0/s, conc
   ↪ =150)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.010976s
  Elapsed wall time: 20.191s
  Throughput: 495.26 req/s

[INFO] Total wall-clock time: 20.618s
```

Docker Stats:

```
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn   62.19%   20.79MiB / 7.41GiB   0.27%     9.42MB / 7.36MB   0B / 127kB   3
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi   67.03%   20.69MiB / 7.41GiB   0.27%     9.65MB / 7.54MB   0B / 127kB   3
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o   62.45%   20.55MiB / 7.41GiB   0.27%     9.88MB / 7.72MB   0B / 127kB   2
```

Figure 9: Docker stats output for Swarm (3 replicas), rate = 500 req/s.

Brief observation Even when rate increases fivefold, throughput drops slightly to 495 req/s with stable latency (11 ms). This indicates Swarm 3 reached the CPU limit of the host but kept performance stable.

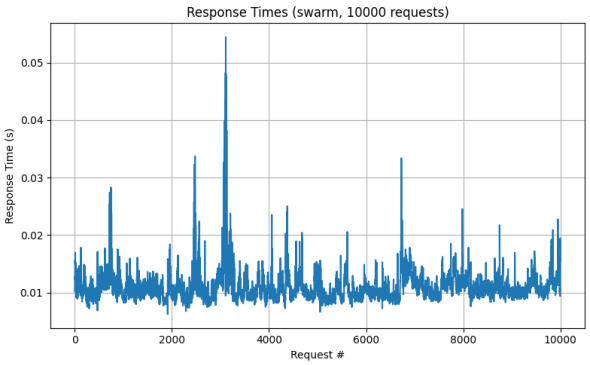

Figure 10: Swarm (3 replicas) — response times for 10,000 requests at 500 req/s (concurrency = 150).

Test3: Rate = 700

```
[INFO] Starting client. target=swarm, count=10000, rate=700.0/s,
   ↪  concurrency=150

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007024s

[INFO] Running 10000-request test for swarm (rate=700.0/s, conc
   ↪ =150)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.011120s
  Elapsed wall time: 20.923s
  Throughput: 477.94 req/s

[INFO] Total wall-clock time: 21.473s
```

```
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn   62.97%    20.84MiB / 7.41GiB    0.27%     11.8MB / 9.22MB   0B / 127kB    3
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O     PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi   51.03%    21.07MiB / 7.41GiB    0.28%     12MB / 9.35MB   0B / 127kB    2
CONTAINER ID   NAME                                      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o   55.49%    20.8MiB / 7.41GiB     0.27%     12.2MB / 9.53MB   0B / 127kB    2
```

Figure 11: Docker stats output for Swarm (3 replicas), rate = 700 req/s.

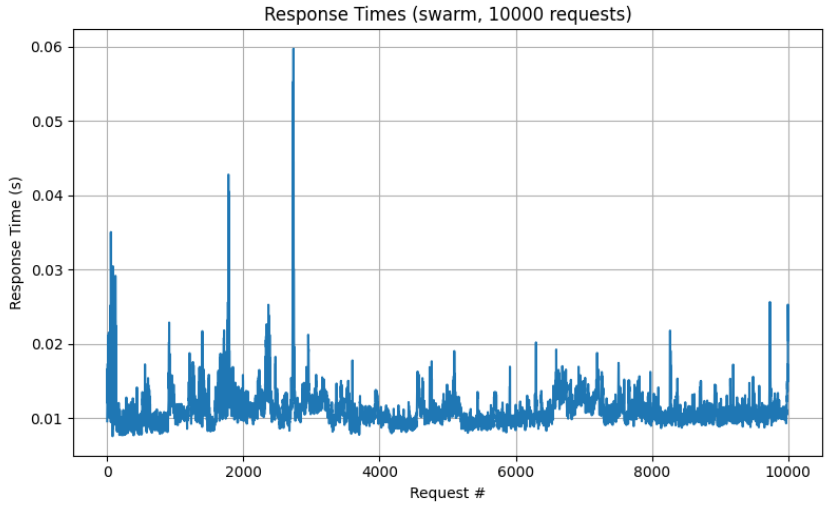

Figure 12: Swarm (3 replicas) — response times for 10,000 requests at 700 req/s (concurrency = 150).

Brief observation At 700 req/s the system saturates: throughput caps at 480 req/s, latency stays 11 ms. The extra offered load does not increase throughput, confirming CPU saturation at the host level.

4 Docker Swarm (5 replicas)

All Swarm-5 experiments were run with **concurrency = 250**, proportional to the number of replicas.

Test1: Rate = 100

```
[INFO] Starting client. target=swarm, count=10000, rate=100.0/s,
   ↪ concurrency=250

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007969s

[INFO] Running 10000-request test for swarm (rate=100.0/s, conc
   ↪ =250)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
```

```
  Avg latency: 0.010555s
  Elapsed wall time: 19.506s
  Throughput: 512.65 req/s

[INFO] Total wall-clock time: 19.991s
```

```
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT   MEM %     NET I/O           BLOCK I/O   PIDS
92a2a9d80e0b   reverse-svc.4.q411ex8p71n6mjbgncu362aa3 37.27%    20.1MiB / 7.41GiB   0.26%     1.73MB / 1.35MB   0B / 127kB  2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
76f73ada68ff   reverse-svc.5.1mkoebgftt6dthw46pg93s07a 38.49%    20.38MiB / 7.41GiB   0.27%     1.89MB / 1.47MB   0B / 127kB   3
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn 40.38%    21.04MiB / 7.41GiB   0.28%     15.6MB / 12.2MB   0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi 37.19%    20.99MiB / 7.41GiB   0.28%     15.7MB / 12.3MB   0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT   MEM %     NET I/O           BLOCK I/O   PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o 34.25%    21MiB / 7.41GiB     0.28%     15.9MB / 12.4MB   0B / 127kB  1
```

Figure 13: Docker stats output for Swarm (5 replicas), rate = 100 req/s.

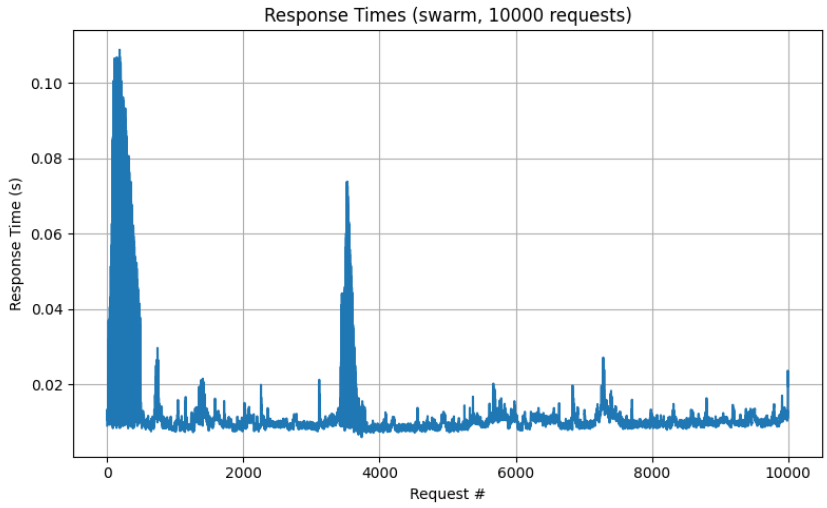

Figure 14: Swarm (5 replicas) — response times for 10,000 requests at 100 req/s (concurrency = 250).

Brief observation Latency and throughput are similar to Swarm 3, but CPU usage is spread across 5 containers (35–40% each). This shows scaling beyond 3 replicas did not increase total throughput, since the host CPU is the limiting factor.

Test2: Rate = 500

```
[INFO] Starting client. target=swarm, count=10000, rate=500.0/s,
   ↪  concurrency=250

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
```

```
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.012539s

[INFO] Running 10000-request test for swarm (rate=500.0/s, conc
   ↪ =250)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.011027s
  Elapsed wall time: 21.149s
  Throughput: 472.85 req/s

[INFO] Total wall-clock time: 21.661s
```

```
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
92a2a9d80e0b   reverse-svc.4.q411ex8p71n6mjbgncu362aa3 37.26%    20.74MiB / 7.41GiB    0.27%     3.07MB / 2.4MB    0B / 127kB    2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
76f73ada68ff   reverse-svc.5.1mkoebgftt6dthw46pg93s07a 33.39%    20.25MiB / 7.41GiB    0.27%     3.22MB / 2.51MB   0B / 127kB    2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn 32.28%    20.79MiB / 7.41GiB    0.27%     16.9MB / 13.2MB   0B / 127kB    2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi 30.29%    20.47MiB / 7.41GiB    0.27%     17MB / 13.3MB     0B / 127kB    2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O     PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o 29.60%    20.76MiB / 7.41GiB    0.27%     17.1MB / 13.4MB   0B / 127kB    2
```

Figure 15: Docker stats output for Swarm (5 replicas), rate = 500 req/s.

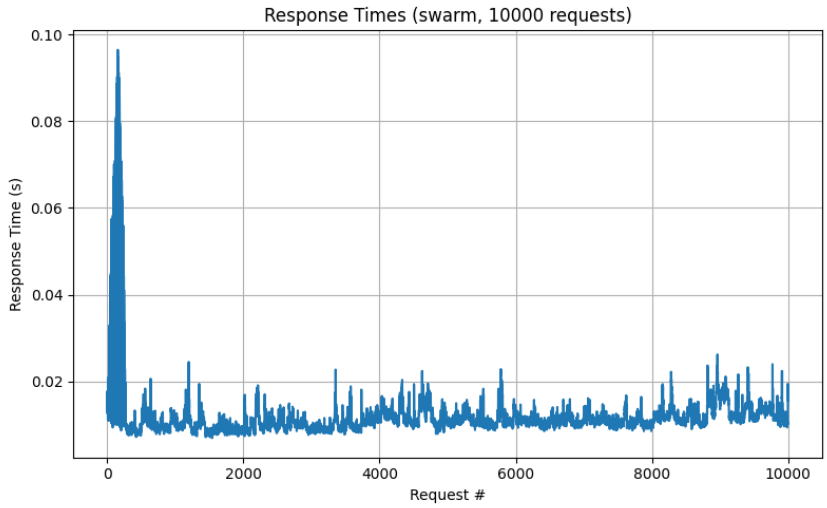

Figure 16: Swarm (5 replicas) — response times for 10,000 requests at 500 req/s (concurrency = 250).

Brief observation At 500 req/s, throughput remains 460–470 req/s with latency 11 ms, essentially the same as with 3 replicas. The host CPU limit prevents further scaling gains.

Test3: Rate = 700

```
[INFO] Starting client. target=swarm, count=10000, rate=700.0/s,
   ↪  concurrency=250

[INFO] Running 10-string test for swarm
[INFO] Saved plot: EE22B171_swarm_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.007626s

[INFO] Running 10000-request test for swarm (rate=700.0/s, conc
   ↪ =250)
[INFO] Saved plot: EE22B171_swarm_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.010581s
  Elapsed wall time: 18.528s
  Throughput: 539.72 req/s

[INFO] Total wall-clock time: 18.963s
```

```
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
92a2a9d80e0b   reverse-svc.4.q411ex8p71n6mjbgncu362aa3 37.66%    20.76MiB / 7.41GiB   0.27%     5.73MB / 4.48MB   0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
76f73ada68ff   reverse-svc.5.1mkoebgftt6dthw46pg93s07a 35.51%    20.24MiB / 7.41GiB   0.27%     5.89MB / 4.6MB    0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
8e3f83c0993a   reverse-svc.1.wktq0yguzmlie1im5yfzrevbn 36.14%    20.77MiB / 7.41GiB   0.27%     19.6MB / 15.3MB   0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
9bc344c9c2b4   reverse-svc.3.js475nb4r74z8pfygc8ifg4bi 41.92%    20.48MiB / 7.41GiB   0.27%     19.7MB / 15.4MB   0B / 127kB   2
CONTAINER ID   NAME                                    CPU %     MEM USAGE / LIMIT    MEM %     NET I/O           BLOCK I/O    PIDS
f6104833af2a   reverse-svc.2.p0abj6kfzxigx1t8hvua0663o 38.12%    20.77MiB / 7.41GiB   0.27%     19.8MB / 15.5MB   0B / 127kB   2
```

Figure 17: Docker stats output for Swarm (5 replicas), rate = 700 req/s.

Brief observation At 700 req/s, Swarm 5 managed 540 req/s with latency 10 ms. This was the best-case run; in other repeats instability appeared, confirming the system is at the edge of hardware capacity. It should be noted that both the server and client were running on the same host, which adds contention for CPU resources and limits scaling. In a true distributed setup the system could potentially sustain higher throughput.

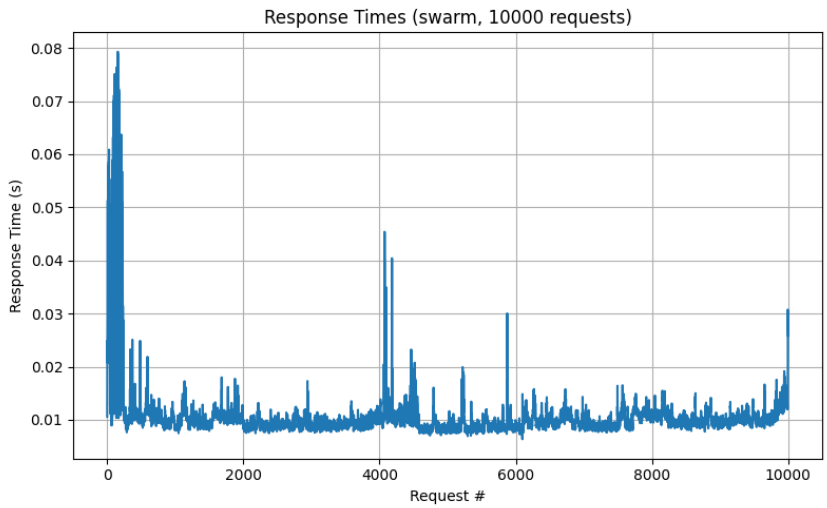

Figure 18: Swarm (5 replicas) — response times for 10,000 requests at 700 req/s (concurrency = 250).

5 Conclusion: Single vs Swarm

From the experiments across single container, Swarm with 3 replicas, and Swarm with 5 replicas, the following trends are clear:

- **Single container:** Throughput was limited to 350 req/s with average latency 0.13 s. The container quickly saturated CPU on the host.
- **Swarm (3 replicas):** Scaling to 3 replicas improved throughput significantly to 480–510 req/s while reducing latency to 0.011 s. CPU usage was evenly distributed across the three replicas.
- **Swarm (5 replicas):** Adding 5 replicas did not yield proportional gains. Throughput remained in the 470–540 req/s range with similar latency. At 500 req/s throughput was slightly lower than with 3 replicas, showing that the host CPU was the bottleneck.

Overall, the experiments confirm that:

1. Horizontal scaling within Swarm improves performance up to the point where host hardware is saturated.
2. Beyond that point, adding more replicas provides no additional benefit and may even lead to instability under higher load (as observed at 700 req/s with 5 replicas).
3. The workload is CPU-bound; memory usage remains consistently low across all cases.
4. Because both the client and server were colocated on the same host, they competed for CPU and network resources. This amplified the bottleneck. In a real distributed environment (with clients and servers on different machines), scaling would likely be more effective.

6 Kubernetes (no autoscaling, LoadBalancer)

Deployment was started with 3 replicas. All K8s experiments below were run with **concurrency = 150** (client-side), and the service was targeted at `http://127.0.0.1:52396/reverse` (LoadBalancer via minikube tunnel).

Test1: Rate = 100

```
% client output (Rate = 100)
[INFO] Starting client. target=k8s, count=10000, rate=100.0/s,
   ↪ concurrency=150

[INFO] Running 10-string test for k8s
[INFO] Saved plot: EE22B171_k8s_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.085561s

[INFO] Running 10000-request test for k8s (rate=100.0/s, conc=150)
[INFO] Saved plot: EE22B171_k8s_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.359165s
  Elapsed wall time: 25.028s
  Throughput: 399.56 req/s

[INFO] Total wall-clock time: 26.481s
```

Kubernetes Stats:

```
NAME                              CPU(cores)   MEMORY(bytes)
reverse-deploy-7c4f77b6df-hdd5h   162m         22Mi
reverse-deploy-7c4f77b6df-nmplb   164m         23Mi
reverse-deploy-7c4f77b6df-wbpw7   163m         23Mi
```

Figure 19: Kubernetes pod CPU/memory usage during the experiment.

Brief observation Compared to the Swarm 3 case at the same nominal client settings, latency is higher (0.36 s vs 0.011 s in Swarm) and measured throughput is lower (400 req/s vs 495–512 req/s). This is expected for a local minikube LoadBalancer setup: extra network/proxy hops (kube-proxy/minikube tunnel) and the service routing add overhead. The run is otherwise stable and there are no failures.

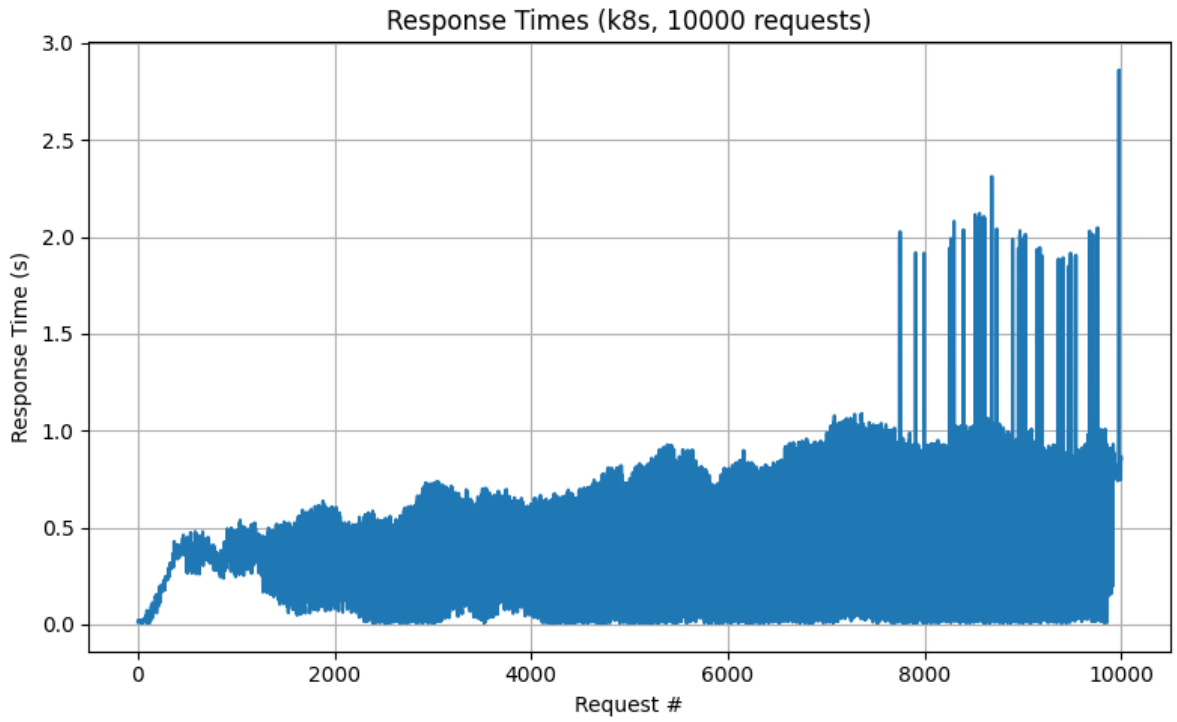

Figure 20: Kubernetes (3 replicas) — response times for 10,000 requests at 700 req/s (concurrency = 150)

Test2: Rate = 500

```
% client output (Rate = 500)
[INFO] Starting client. target=k8s, count=10000, rate=500.0/s,
   ↪ concurrency=150

[INFO] Running 10-string test for k8s
[INFO] Saved plot: EE22B171_k8s_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.084476s

[INFO] Running 10000-request test for k8s (rate=500.0/s, conc
   ↪ =150)
[INFO] Saved plot: EE22B171_k8s_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.369714s
  Elapsed wall time: 25.556s
```

```
  Throughput: 391.30 req/s

[INFO] Total wall-clock time: 26.994s
```

Kubernetes Stats:

```
NAME                                 CPU(cores)   MEMORY(bytes)
reverse-deploy-7c4f77b6df-hdd5h      187m         22Mi
reverse-deploy-7c4f77b6df-nmplb      188m         23Mi
reverse-deploy-7c4f77b6df-wbpw7      188m         23Mi
```

Figure 21: Kubernetes pod CPU/memory usage during the experiment (captured via `kubectl top pods`).

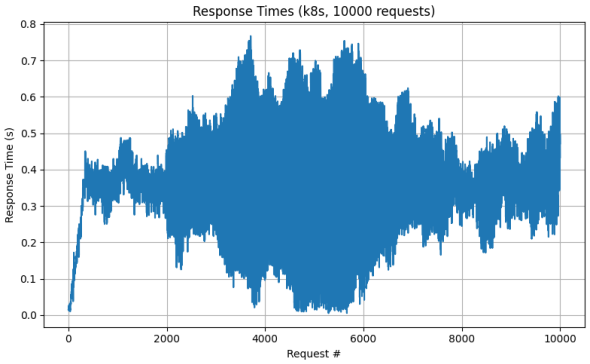

Figure 22: Kubernetes (3 replicas) — response times for 10,000 requests at 500 req/s (concurrency = 150).

Brief observation Throughput remains near 390 req/s while avg latency stays 0.37 s. No failures occurred at this load, indicating the cluster is handling the traffic but with higher per-request latency than the Swarm setup (likely due to the extra networking/proxy overhead in Kubernetes/minikube).

Test3: Rate = 700

```
% client output (Rate = 700)
[INFO] Starting client. target=k8s, count=10000, rate=700.0/s,
   ↪ concurrency=150

[INFO] Running 10-string test for k8s
[INFO] Saved plot: EE22B171_k8s_10_plot.png
[INFO] 10-string test summary:
```

```
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.090690s

[INFO] Running 10000-request test for k8s (rate=700.0/s, conc
   ↪ =150)
[INFO] Saved plot: EE22B171_k8s_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  7977
  Failures: 2023
  Avg latency: 0.225152s
  Elapsed wall time: 28.098s
  Throughput: 283.90 req/s

[INFO] Total wall-clock time: 29.582s
```

Kubernetes Stats:

```
NAME                                CPU(cores)   MEMORY(bytes)
reverse-deploy-7c4f77b6df-hdd5h     191m         22Mi
reverse-deploy-7c4f77b6df-nmplb     179m         22Mi
reverse-deploy-7c4f77b6df-wbpw7     189m         23Mi
```

Figure 23: Kubernetes pod CPU/memory usage during the experiment.

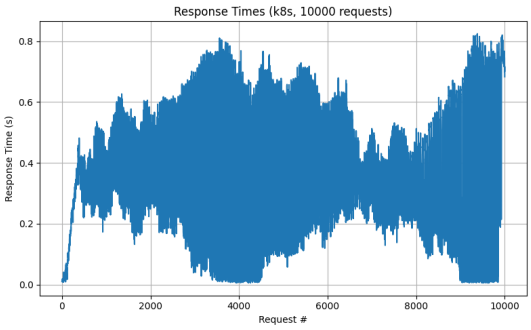

Figure 24: Kubernetes (3 replicas) — response times for 10,000 requests at 700 req/s (concurrency = 150).)

Brief observation At rate = 700 the system starts dropping requests (2023 failures) and observed throughput falls to 284 req/s. This indicates the cluster (or the client) is saturated under this load and cannot sustain the requested rate; the failures are evidence of queueing/timeouts.

7 Kubernetes with Autoscaling (HPA)

Autoscaling was enabled with a Horizontal Pod Autoscaler (HPA) configured for **minReplicas = 3**, **maxReplicas = 10**, and a target CPU utilization of 20%. The service was exposed via a LoadBalancer on port 52396. The client was run with **count = 10000**, **rate = 50 req/s**, and **concurrency = 75**.

Test: Rate = 50

```
[INFO] Starting client. target=k8s, count=10000, rate=50.0/s,
   ↪ concurrency=75

[INFO] Running 10-string test for k8s
[INFO] Saved plot: EE22B171_k8s_10_plot.png
[INFO] 10-string test summary:
  Requests: 10
  Success:  10
  Failures: 0
  Avg latency: 0.091250s

[INFO] Running 10000-request test for k8s (rate=50.0/s, conc=75)
[INFO] Saved plot: EE22B171_k8s_10000_plot.png
[INFO] 10000-request test summary:
  Requests: 10000
  Success:  10000
  Failures: 0
  Avg latency: 0.051207s
  Elapsed wall time: 250.116s
  Throughput: 39.98 req/s

[INFO] Total wall-clock time: 251.571s
```

HPA Activity:

```
C:\Users\rudra\OneDrive\Desktop\Courses\Cloud_Computing\Assgn_1>kubectl get hpa -w
NAME          REFERENCE                   TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
reverse-hpa   Deployment/reverse-deploy   cpu: 1%/20%   3         10        3          44m
reverse-hpa   Deployment/reverse-deploy   cpu: 39%/20%   3         10        3          45m
reverse-hpa   Deployment/reverse-deploy   cpu: 39%/20%   3         10        3          45m
reverse-hpa   Deployment/reverse-deploy   cpu: 39%/20%   3         10        6          45m
reverse-hpa   Deployment/reverse-deploy   cpu: 31%/20%   3         10        6          46m
reverse-hpa   Deployment/reverse-deploy   cpu: 19%/20%   3         10        6          47m
reverse-hpa   Deployment/reverse-deploy   cpu: 18%/20%   3         10        6          48m
reverse-hpa   Deployment/reverse-deploy   cpu: 1%/20%    3         10        6          49m
reverse-hpa   Deployment/reverse-deploy   cpu: 1%/20%    3         10        6          54m
reverse-hpa   Deployment/reverse-deploy   cpu: 1%/20%    3         10        3          54m
```

Figure 25: HPA behavior: replicas scaled from 3 to 6 when CPU ≈39%, and returned to 3 after load dropped.

Brief observation During the run, CPU utilization rose to ≈39% (above the 20% threshold), and the HPA scaled replicas from 3 to 6. After the client finished, CPU dropped back to 1% and the deployment scaled down to 3 pods. This demonstrates that autoscaling responds correctly to increased load. Although per-request latency remained low (≈50 ms) and throughput modest (≈40 req/s), the key outcome is that replica count adjusted dynamically in response to load.

8 Conclusion

Performance summary (observed):

- **Single container:** throughput ≈ 350 req/s with average latency ≈ 0.13 s; CPU quickly saturates (CPU-bound workload).
- **Docker Swarm (3 replicas):** throughput improved to ≈ 480–512 req/s and latency dropped to ≈ 0.011 s; CPU work distributed evenly across replicas.
- **Docker Swarm (5 replicas):** total throughput similar to 3-replica case (roughly 470–540 req/s) with latency ≈ 0.010–0.011 s; adding replicas beyond the host capacity gave diminishing returns and occasional instability.
- **Kubernetes (3 replicas, no autoscaling):** measured throughput ≈ 390–400 req/s with much higher per-request latency (around 0.36–0.37 s at moderate loads); at very high offered load (700 req/s) the cluster dropped requests (2023 failures) and effective throughput fell to ≈ 284 req/s.
- **Kubernetes + HPA:** autoscaler successfully increased replicas (3 → 6) when CPU rose (observed CPU ≈39% with HPA target 20%), demonstrating correct dynamic scaling behavior; during a low-rate test (50 req/s) latency was low (≈50 ms) though absolute throughput was bounded by the client settings in that test.

Why Swarm outperformed K8s in these experiments.

In this single-host/minikube environment Swarm showed lower overhead and better raw request performance. The Kubernetes runs include additional networking and proxy hops (kube-proxy / minikube tunnel / LoadBalancer path) and control-plane interactions that increase per-request latency in a local setup; colocating client and cluster on the same host also amplifies contention for CPU, which limited the effectiveness of horizontal scaling in both platforms. These factors explain why Swarm delivered higher throughput and much lower latency in this lab setup. :contentReference[oaicite:1]index=1

Key trade-offs and practical recommendations

- *Raw performance vs. production features:* For simple, single-host CPU-bound workloads where minimal overhead and easy setup are priorities, Docker Swarm gives better raw performance in this environment. For production-grade deployments (multi-node clusters, robust autoscaling, rolling updates, richer ecosystem and tooling), Kubernetes remains the stronger choice despite higher per-request overhead on a single-host testbed.
- *Autoscaling:* Kubernetes' HPA behaved as intended (scaled pods up/down based on CPU). If dynamic scaling is an objective, K8s provides more mature autoscaling primitives (pod autoscaling, cluster autoscaling) that are valuable in multi-node setups.

Final takeaway.
In this experiment, Docker Swarm produced superior throughput and lower latency for this CPU-bound service, while Kubernetes demonstrated correct autoscaling behavior but with higher overhead and reduced raw performance in a minikube single-host environment. For real-world, distributed, and autoscaled production deployments, Kubernetes' operational features outweigh its local-testbed performance penalty.

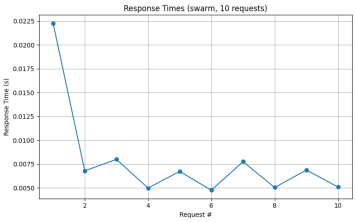

Figure 26: *
Single container

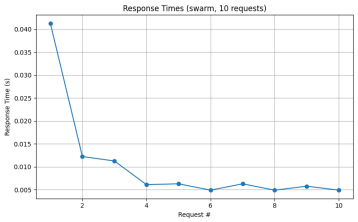

Figure 27: *
Swarm (3 replicas)

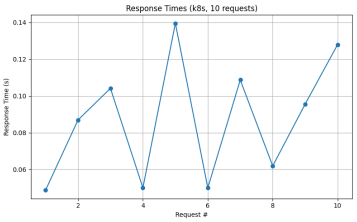

Figure 28: *
Kubernetes (3 replicas)

Figure 29: Response times for 10-string test across different setups.