# Load Test Performance Report

# Contents

# Introduction

This report is based on the monitoring findings during the load tests that I conducted for one of my microservices in the Movimingle software. Here I will give information about the test configuration, and configuration of the auto scaling. The purpose of this report and findings is to analyze them and determine the optimal resource allocation for the microservice and to understand the impact of scaling on the overall system performance. The load tests are performed both with and without horizontal scalability.

# Test Configuration

## Load Test Script

I have used a JavaScript file in combination with k6 to perform the test. The script consists of simulation of HTTP GET requests sent to an endpoint of the API Gateway, which routs them to the voting-service. The scripts I used were two different ones, since the findings were not what I had expected and initially thought it was because of the setup of the load test. The initial script was simulating the requests and was gradually increasing the number of virtual users to 5000, while the second one was up to 4000 Vus.

## Microservice Deployment

The setup of the cluster is such that I have implemented an API Gateway to handle the requests for the services in the application. The configuration of the Kubernetes cluster is set using YAML files.

## Monitoring Tools

Performance metrics were monitored using K6 and Grafana dashboards.

## Scope of Scaling

Only the voting-service microservice is scaled and tested.

# Initial Results

## K6 Dashboard – No Scaling


### Metrics

- Iteration Rate: 404.9/s
- HTTP Request Rate: 338.2/s
- HTTP Request Duration: 5ms
- HTTP Request Failed: 0/s
- Received Rate: 115 kB/s
- Sent Rate: 343 kB/s

## Grafana Dashboard – No Scaling


### Cluster Optimization

- Cluster CPU Usage: Peaked at around 5 cores (physical capacity: 12 cores)
- Cluster Memory Usage: Peaked at around 5.58 GiB (physical capacity: 7.65 GiB)

## K6 Dashboard – With Scaling


## Metrics:

- iteration Rate: 411/s
- HTTP Request Rate: 344.3/s
- HTTP Request Duration: 7ms
- HTTP Request Failed: 0/s
- Received Rate: 116 kB/s
- Sent Rate: 349 kB/s

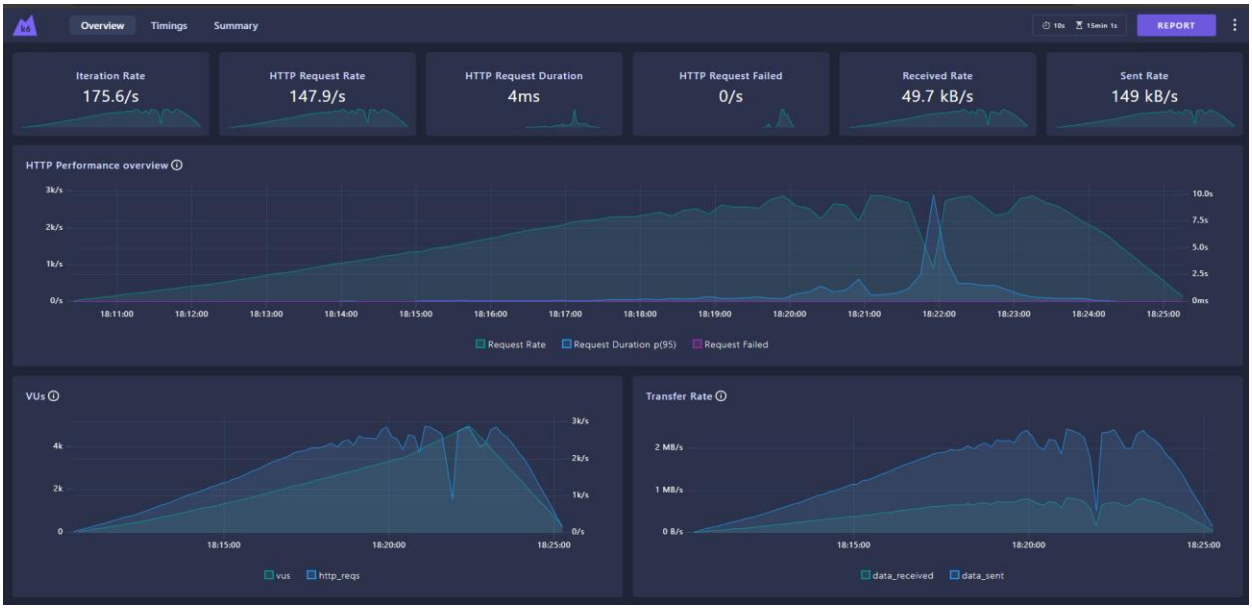
## Grafana Dashboard – With Scaling


## Node Optimization

- Node CPU Usage: Peaked at around 8 cores (physical capacity: 12 cores)
- Node Memory Usage: Peaked at around 6.13 GiB (physical capacity: 7.65 GiB)

# New Results

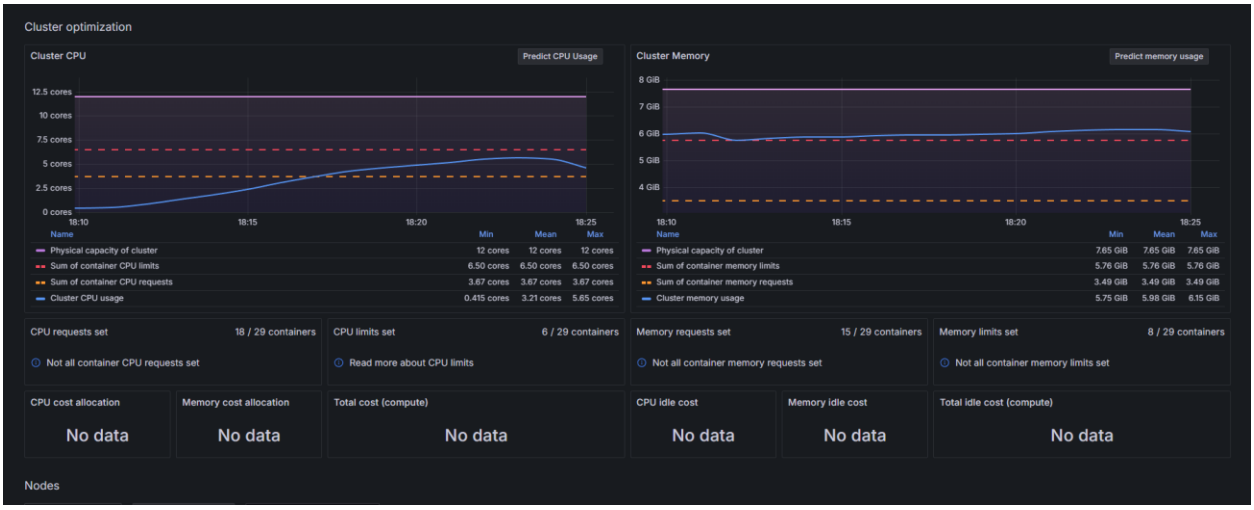
## K6 Dashboard – No Scaling




## Metrics

- Iteration Rate: 175.6/s
- HTTP Request Rate: 147.9/s
- HTTP Request Duration: 4ms
- HTTP Request Failed: 0/s
- Received Rate: 49.7 kB/s
- Sent Rate: 149 kB/s

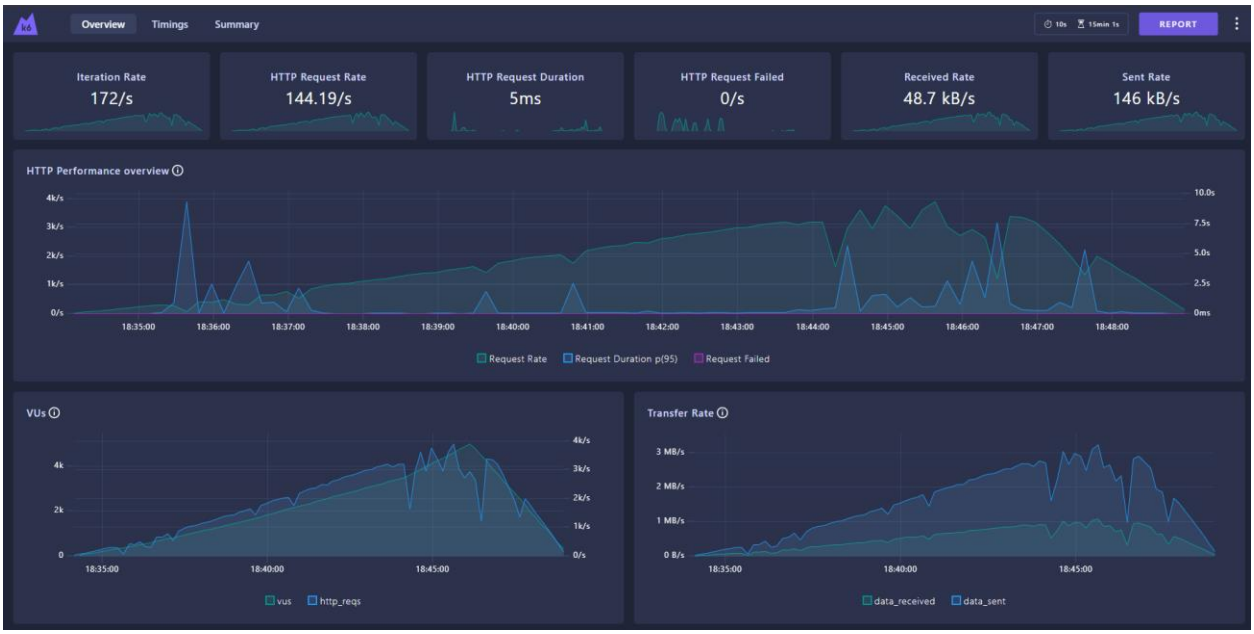
## Grafana Dashboard – No Scaling




## Cluster Optimization

- Cluster CPU Usage: Peaked at around 5.65 cores (physical capacity: 12 cores)
- Cluster Memory Usage: Peaked at around 6.15 GiB (physical capacity: 8 GiB)

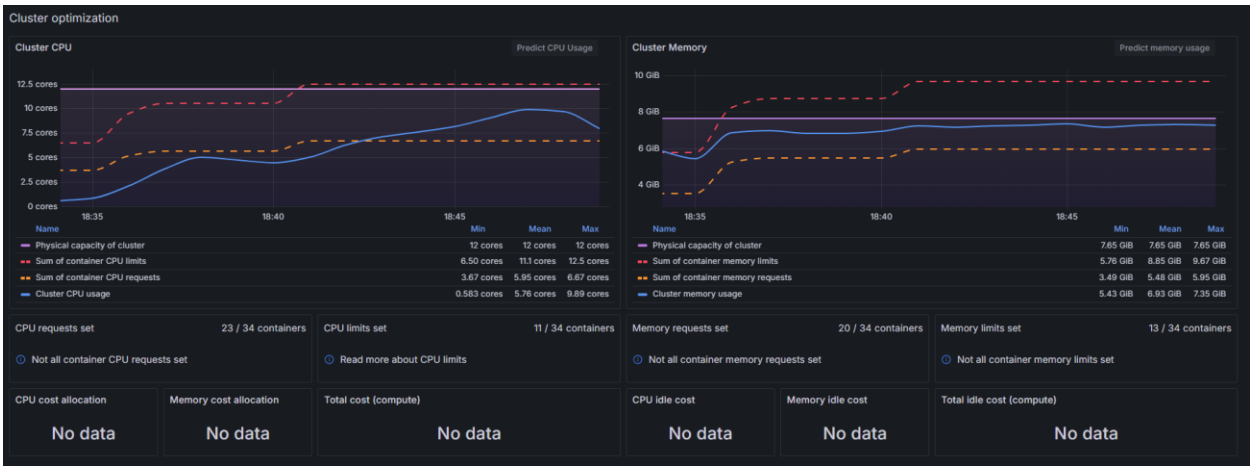
## K6 Dashboard – With Scaling




## Metrics

- Iteration Rate: 172/s
- HTTP Request Rate: 144.19/s
- HTTP Request Duration: 5ms
- HTTP Request Failed: 0/s
- Received Rate: 48.7 kB/s
- Sent Rate: 146 kB/s

## Grafana Dashboard – With Scaling




## Cluster Optimization

- Cluster CPU Usage: Peaked at around 9.89 cores (physical capacity: 12 cores)
- Cluster Memory Usage: Peaked at around 7.35 GiB (physical capacity: 8 GiB)

# Analysis

## Iteration and Request rates

### Observation

The iteration rate and HTTP requests rate are lower in the new scaled setup in comparison to the new non-scaled setup.

### Explanation

The results show that a single instance of the microservice handles the increased load on the application better than the one with auto-scaling. This could be due to improper setup of the HPA (Horizontal Pod Autoscaler), improper memory and CPU allocation, and of course by introducing new pods to the system the complexity increases and my be too many things that have to be handled for the current setup of the software.

## HTTP Request Duration

### Observation

The request duration is slightly higher in the new scaled setup (5ms) compared to the new non-scaled setup (4ms).

### Explanation

Again the exact reason of this difference in the favor of the non-scaled setup is not known to me yet, however my assumption is that the added complexity and overhead of managing multiple replicas might be impacting the performance. Thus the non-scaled setup shows lower request duration, because the management of the resources is simpler.

### Impact

The increased request duration may lead to higher latency, which will affect the user experience and because of that further optimization is needed in the scaled setup.

## Resource Utilization

### CPU & Memory Usage

The new scaled setup utilizes more CPU and memory resources, peaking at 9.89 cores (CPU) and 7.35 GiB (Memory), compared to the non-scaled setup – 5.65 cores (CPU) and 6.15 GiB (Memory).

The higher overall resource consumption occurs because the scaled setup uses more replicas. This implies that it can handle higher loads, but this is not the case. The reason for this is because the current configuration is not leveraging the additional resources effectively. To do that a proper set up of the requests and limits will have to be done to ensure optimal distribution among all services.

## Steps for Improvement

In order to properly configure the horizontal scaling, analysis of the metrics that monitoring tools like Grafana, Metric-Server, and k6 Web-Dashboard will have to be done. This will help with determining what the optimal resource allocation is. Together with that, testing different strategies for the HPA will have to be tested and with enough gathered data and tests I think I will be able to make the setup optimal.

## Conclusion

The load testing analysis shows that both the scaled and non-scaled setups can handle the load somewhat efficiently, with the scaled setup showing higher resource utilization and worse performance metrics due to the sub-optimal setup and increased complexity. This can be fixed with more testing. Additionally, the fact that my knowledge of Kubernetes and scaling is somewhat limited, since I have not dived deep into that realm yet should be considered. Gaining more experience for sure will help in fine-tuning the configuration.