CS436 Term Project Infinity Search Engine

Group members:

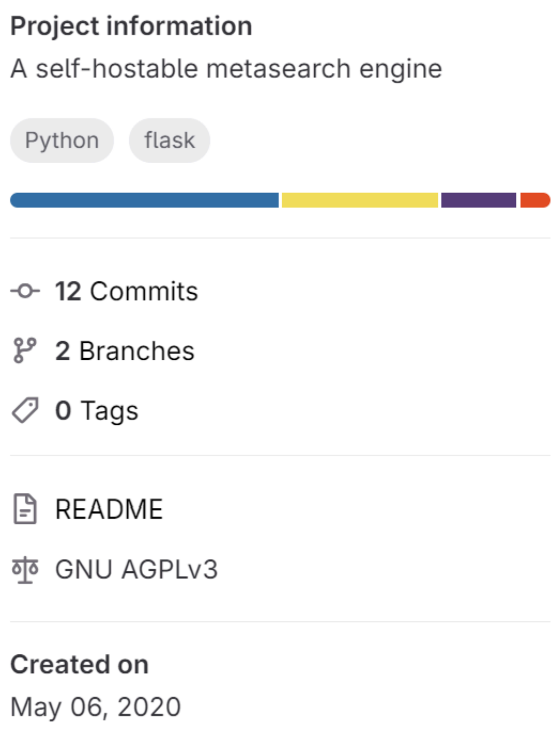
- **Yağız Gürdamar 22534**
- **Muammer Tunahan Yıldız 27968**
- **Ömer Faruk Tarakçı 28226**

26/05/2024

Sabancı Üniversitesi

The Application

Infinity Search Solo is a lightweight, privacy-focused search engine designed to provide users with fast and relevant search results while prioritizing data privacy and security.


Project information
A self-hostable metasearch engine

Python flask

12 Commits
2 Branches
0 Tags

README
GNU AGPLv3

Created on
May 06, 2020

1. Key Features

- **Advanced Search Algorithms:** Utilizes cutting-edge algorithms to deliver accurate and efficient search results.
- **Privacy Protection:** Implements robust privacy measures to ensure user data is protected and not shared with third parties.
- **Customization Options:** Offers customizable search settings to tailor the user experience according to individual preferences.
- **Open Source:** Infinity Search Solo is an open-source project, promoting transparency and community collaboration.

2. Architecture

- **Cloud-Based Infrastructure:** Infinity Search Solo is built to leverage cloud computing resources for scalability, reliability, and performance.
- **Microservices Architecture**: Utilizes a modular architecture with microservices to facilitate easier development, deployment, and maintenance.

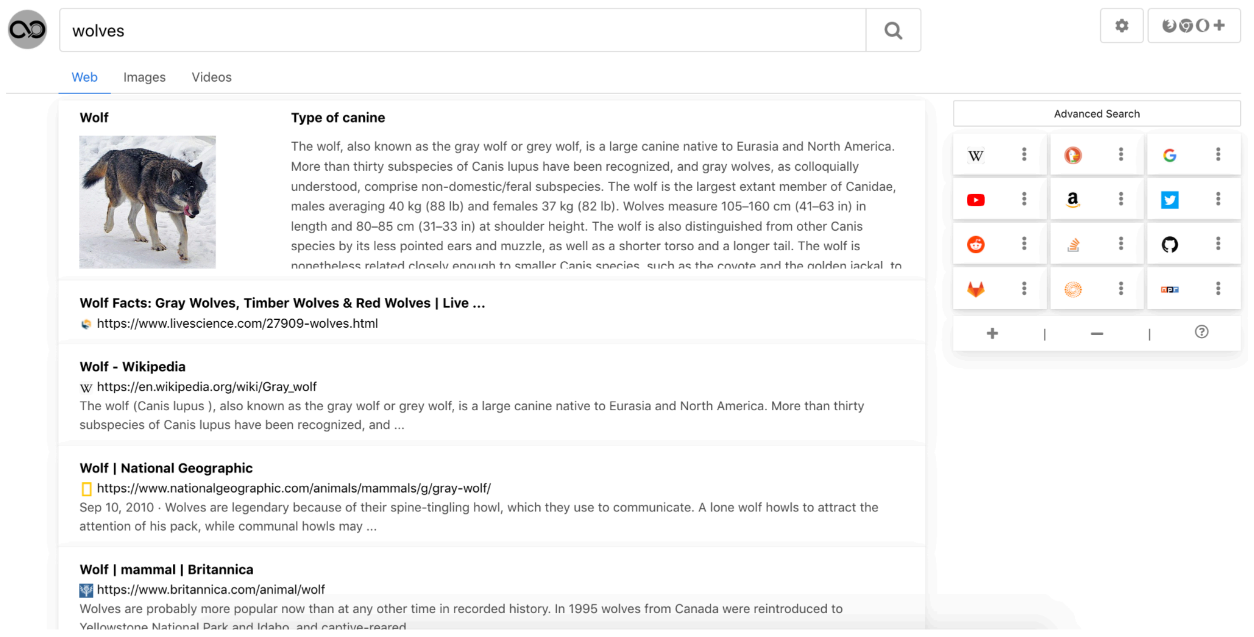
- **Containerization:** Implements containerization using technologies like Docker to package the application and its dependencies for seamless deployment across different environments.
- **Orchestration:** Utilizes orchestration tools like Kubernetes to manage and scale containers effectively.

3. Deployment Model

- **Cloud Deployment:** Infinity Search Solo is deployed on cloud platforms such as AWS, Google Cloud Platform, or Azure to leverage their infrastructure and services.
- **Scalability**: The application is designed to scale horizontally to handle varying levels of traffic and user demand.
- **High Availability:** Implements redundancy and failover mechanisms to ensure high availability and minimal downtime.

4. Performance Metrics

- Latency: Infinity Search Solo aims to minimize latency and provide users with fast search results by optimizing query processing and response times.
- Throughput: Measures the application's ability to handle a large number of search queries concurrently while maintaining performance.
- Uptime: Ensures that the application remains accessible and operational for users with minimal downtime.



Wolf

Type of canine

The wolf, also known as the gray wolf or grey wolf, is a large canine native to Eurasia and North America. More than thirty subspecies of Canis lupus have been recognized, and gray wolves, as colloquially understood, comprise non-domestic/feral subspecies. The wolf is the largest extant member of Canidae, males averaging 40 kg (88 lb) and females 37 kg (82 lb). Wolves measure 105–160 cm (41–63 in) in length and 80–85 cm (31–33 in) at shoulder height. The wolf is also distinguished from other Canis species by its less pointed ears and muzzle, as well as a shorter torso and a longer tail. The wolf is nonetheless related closely enough to smaller Canis species, such as the coyote and the golden jackal, to

Wolf Facts: Gray Wolves, Timber Wolves & Red Wolves | Live ...
https://www.livescience.com/27909-wolves.html

Wolf - Wikipedia
https://en.wikipedia.org/wiki/Gray_wolf
The wolf (Canis lupus), also known as the gray wolf or grey wolf, is a large canine native to Eurasia and North America. More than thirty subspecies of Canis lupus have been recognized, and ...

Wolf | National Geographic
https://www.nationalgeographic.com/animals/mammals/g/gray-wolf/
Sep 10, 2010 · Wolves are legendary because of their spine-tingling howl, which they use to communicate. A lone wolf howls to attract the attention of his pack, while communal howls may ...

Wolf | mammal | Britannica
https://www.britannica.com/animal/wolf
Wolves are probably more popular now than at any other time in recorded history. In 1995 wolves from Canada were reintroduced to Yellowstone National Park and Idaho, and captive-reared

Architecture and System Design

The Infinity Search Solo application was set up and deployed using Docker, Kubernetes on Google Kubernetes Engine (GKE), Terraform, and a number of auxiliary technologies. This report describes the process. Using a cloud-native architecture, the objective was to provide an automated, scalable, and reliable deployment environment.

Technologies Used

- **Docker**: Containerization platform for packaging the application.
- **Kubernetes (GKE)**: Orchestrates containerized applications across a cluster of nodes.
- **Google Kubernetes Engine (GKE)**: Managed Kubernetes service on Google Cloud Platform.
- **Terraform**: Infrastructure as Code (IaC) tool for provisioning cloud resources.
- **Google Cloud Storage and Container Registry**: Storage for Docker images and other assets.

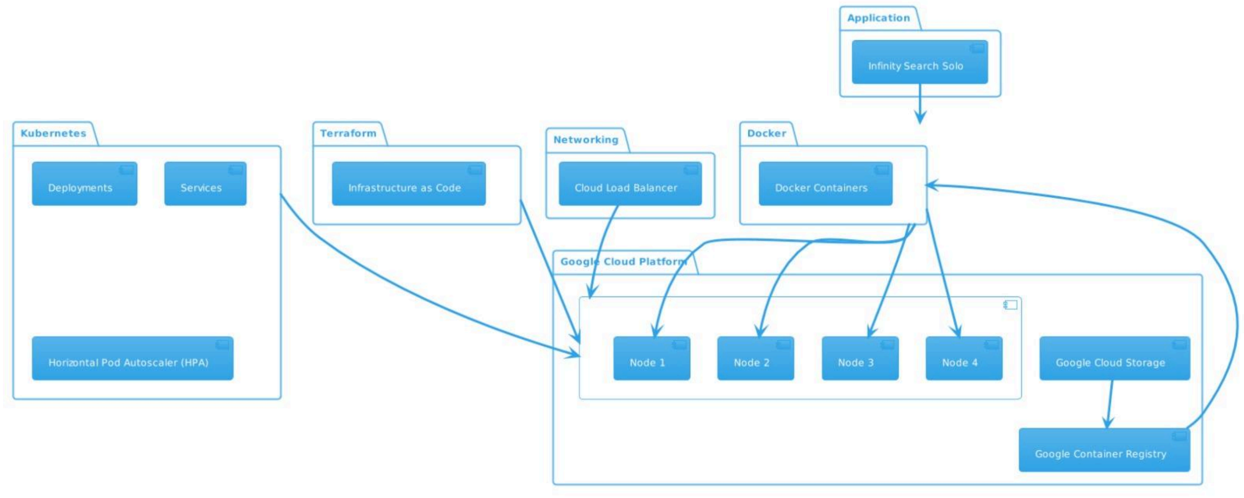
Infrastructure Setup

1. **Google Cloud Platform**:
    - **GKE Cluster**: Set up with 4 nodes using e2-medium machine type for balanced capacity.
    - **Load Balancing**: Managed by GKE services.
2. **Terraform Configuration**:
    - Defined infrastructure as code to provision GKE clusters and associated resources.
    - Configuration includes setting up the cluster, specifying node configurations, and defining firewall rules.
    - The cluster was configured to be zonal, ensuring it operates within a single zone for simplicity and reduced latency.

Application Deployment

1. **Docker**:
    - A Dockerfile was created for the application, specifying the Python environment, dependencies, and application start commands.
    - Docker images were built and stored in the Google Container Registry.
2. **Startup Script**:
    - The setup procedure was made automated with the help of a script which is written by us. It contained instructions for configuring the Google Cloud SDK, installing and upgrading the necessary packages, starting Terraform, and launching the Docker container on GKE.

3. **Kubernetes Configuration**:
    - **Deployment**: Defines how to deploy the application pods. The deployment ensures that a specified number of replicas of the application are running at all times.
    - **Service**: Exposes the application to the internet through a LoadBalancer service, making it accessible externally.
    - **Horizontal Pod Autoscaler (HPA)**: Automatically scales the number of pods based on CPU and memory usage to handle varying loads efficiently.



Monitoring

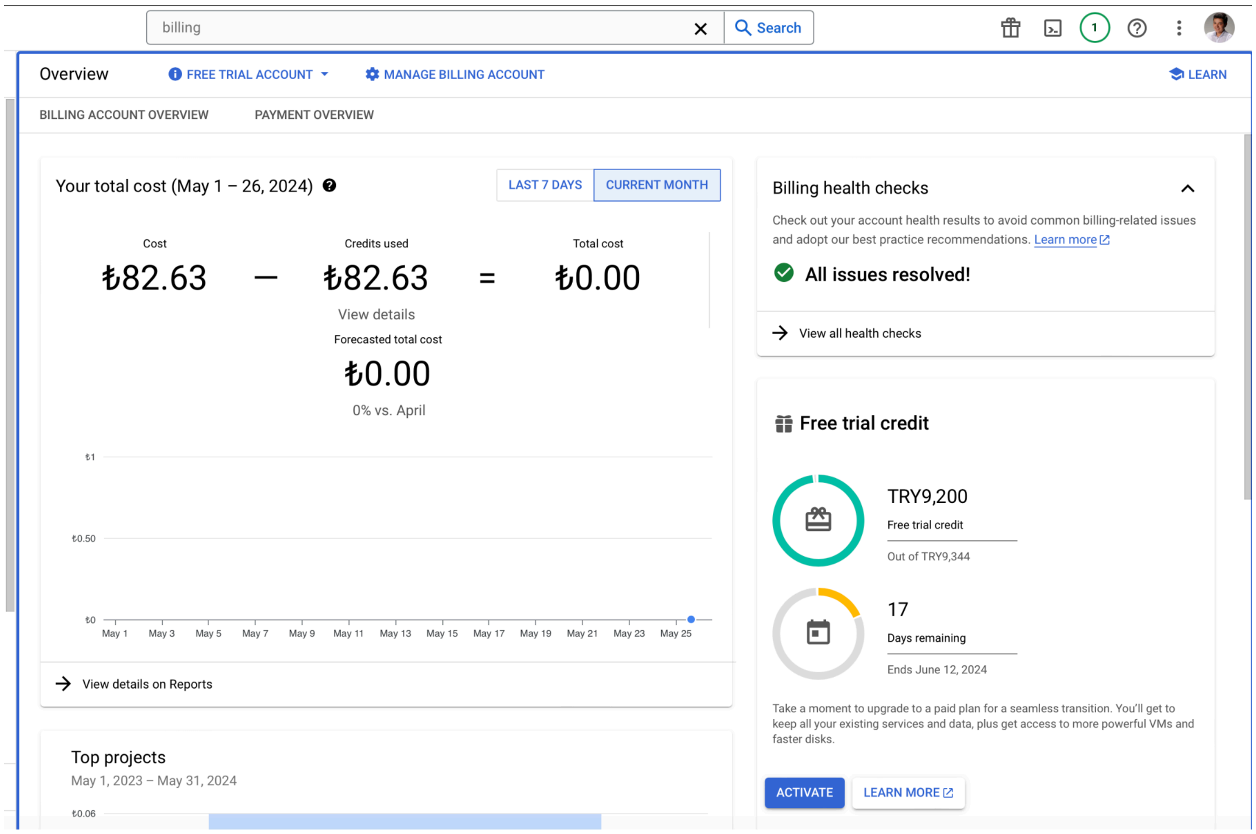
- **Monitoring**: The application's seamless operation was ensured by monitoring resource utilization (CPU, memory, and storage) using GKE and Google Cloud Monitoring tools. While the monitoring revealed that CPU and memory utilization were within anticipated bounds, sporadic spikes suggested that additional scaling considerations were necessary.

Cloud-Native Architecture

- **Scalability**: The use of Kubernetes and HPA ensures that the application can scale dynamically based on load, providing high availability and reliability.
- **Automation**: Terraform automates the provisioning of cloud resources, ensuring consistent and repeatable deployments.
- **Resilience**:In order to maintain the application's availability and resilience, the cloud-native design makes use of GKE's managed services, which include automated upgrades, backups, and failover procedures.

Cost Management

- The project was completed using the Google Cloud Platform's free trial credits. There were no out-of-pocket expenses because the trial credits paid for the entire amount incurred.
- The billing summary showed that the free trial credits paid for the entire cost of the time, indicating effective resource management and cost control.



Your total cost (May 1 – 26, 2024)

Cost		Credits used		Total cost
₺82.63	−	₺82.63	=	₺0.00

View details

Forecasted total cost

₺0.00

0% vs. April

Billing health checks

Check out your account health results to avoid common billing-related issues and adopt our best practice recommendations. Learn more

All issues resolved!

Free trial credit

TRY9,200
Free trial credit
Out of TRY9,344

17
Days remaining
Ends June 12, 2024

Take a moment to upgrade to a paid plan for a seamless transition. You'll get to keep all your existing services and data, plus get access to more powerful VMs and faster disks.

Top projects
May 1, 2023 – May 31, 2024

This project shows how to use contemporary cloud and containerization technologies to set up a reliable, scalable, and automated deployment environment for the Infinity Search Solo application. Terraform is used to manage the infrastructure, guaranteeing consistent and repeatable deployments, while Kubernetes on GKE handles the essential orchestration and scaling. Docker makes sure that the program operates reliably in many settings. By using HPA, the application may scale dynamically in response to load, guaranteeing peak performance.

A startup script streamlines the deployment process and makes it simple to set up the entire environment from the beginning. This method increases the infrastructure's dependability and maintainability while simultaneously increasing deployment efficiency.

All things considered, this project effectively illustrates the strength and adaptability of cloud-native architectures in the deployment and administration of scaled applications.

Experiments

The purpose of the experiment is to evaluate the performance and scalability of our cloud deployment of Infinity Search Solo application under different load conditions using Apache JMeter. The focus is on assessing how the system responds to varying numbers of concurrent users and different node configurations of the system.

The test scenarios included a low load test simulating 100 concurrent users and a high load test simulating 500 concurrent users, both with a ramp-up period of 53 seconds, and repeated 5 times. To simulate realistic user behavior, the tests involved sending a query request along with a mini-image search request. Users were given a random think time between 1.5 to 4 seconds to browse the page before similar query requests for the second search results page was sent. After an additional 1 to 1.5 seconds of think time, a query for image results was made, followed by a final think time of 2 to 3 seconds. Three listeners-Results Tree, Aggregate Graph, and Summary Report- were included to facilitate monitoring of the tests in the GUI mode, although the final stress tests were executed in the console.

The hardware used in these tests consisted of e2-medium machines on Google Kubernetes Engine (GKE), with configurations set up for both single node and 4 nodes. The software stack included the Infinity Search Solo application, Docker for containerization, Kubernetes (GKE) for orchestration, and Terraform for infrastructure provisioning. Key metrics monitored during the tests included response time, throughput, and CPU and memory usage. Apdex score is used to summarize and compare the performance results.

Effective stress testing was a crucial component of the experiment, designed to push the system to its limits by simulating high numbers of concurrent users. The high load test with 500 threads aimed to simulate peak load conditions, monitoring key metrics such as response time, throughput, and resource utilization to observe system behavior under extreme conditions. The low load test with 100 threads served as a baseline to understand system performance under normal operating conditions and to compare against high load results, thus assessing scalability and performance improvements with additional nodes.

Overall, the Infinity Search Solo application was tested using a structured and methodical approach to understand its performance and scalability. Apache JMeter provided a robust framework for simulating various load conditions, while the deployment on Google Kubernetes Engine (GKE) facilitated scalable and reliable application orchestration.

Experiment Results

As mentioned above, our tests consisted of the same test plan with 2 different load values, and GKE setted up with 2 different configurations. At total, we performed a low-load test for single noded configuration, and performed low-load and high-load tests for 4 noded configuration.

1- Low Load Test (100 Concurrent Threads)
The low load test demonstrates significant performance improvement when the application is scaled from a single node to four nodes. The average response time decreased by approximately 84%, and throughput more than doubled, indicating that the application handles concurrent requests more efficiently with additional nodes.

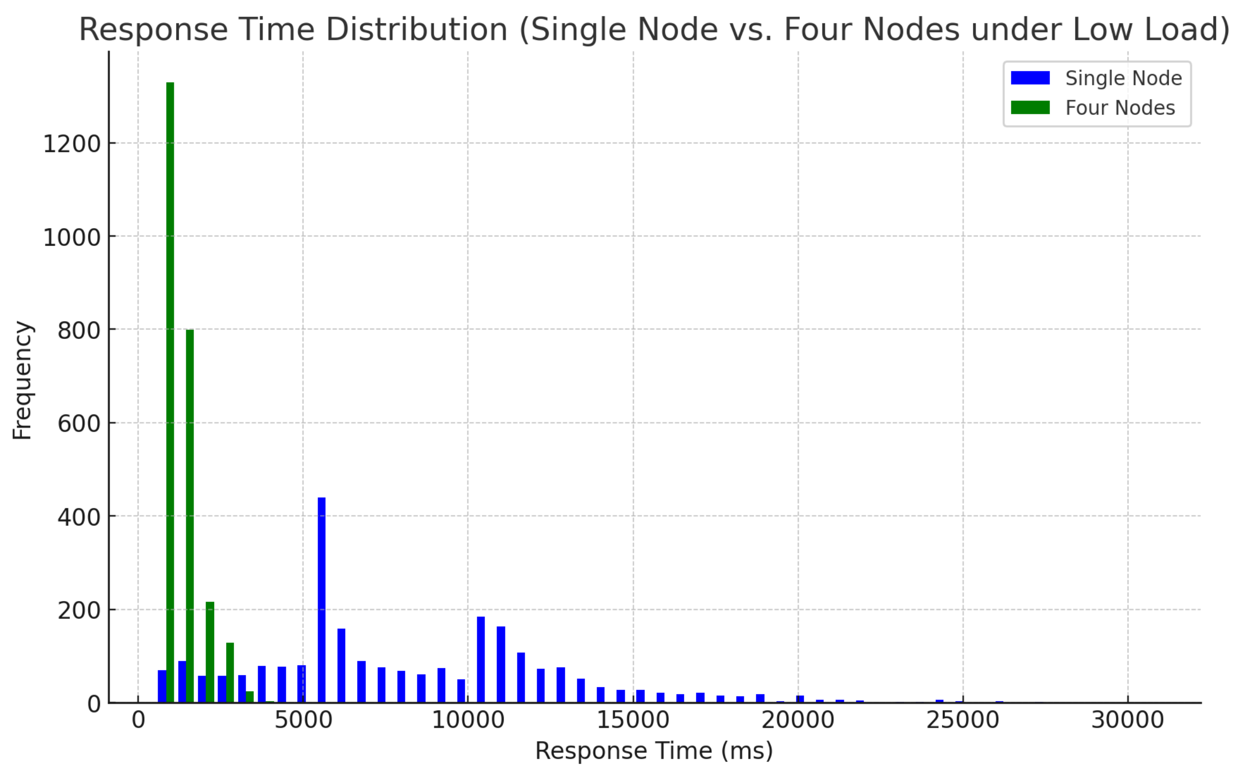
Single Node:

- **Average Response Time**: 8311.72 ms
- **Median Response Time**: 7102.50 ms
- **90th Percentile Response Time**: 13986.90 ms
- **Max Response Time**: 30763 ms
- **Throughput**: 7.92 requests per second
- **Success Rate**: 100%
- **Overall Apdex Score:** 0.025

Four Nodes:

- **Average Response Time**: 1286.71 ms
- **Median Response Time**: 1122.00 ms
- **90th Percentile Response Time**: 2099.20 ms
- **Max Response Time**: 3736 ms
- **Throughput**: 16.61 requests per second
- **Success Rate**: 100%
- **Overall Apdex Score:** 0.394

These results indicate that while the server can handle requests from 100 concurrent threads, the single-node configuration exhibits high response latency and a low Apdex score, reflecting a suboptimal user experience. In contrast, the four-node configuration significantly improves response times and the Apdex score, demonstrating better performance and user experience under similar load conditions.



Graph 1: The distribution shows a higher frequency of lower response times with four nodes, indicating more consistent performance compared to a single node.

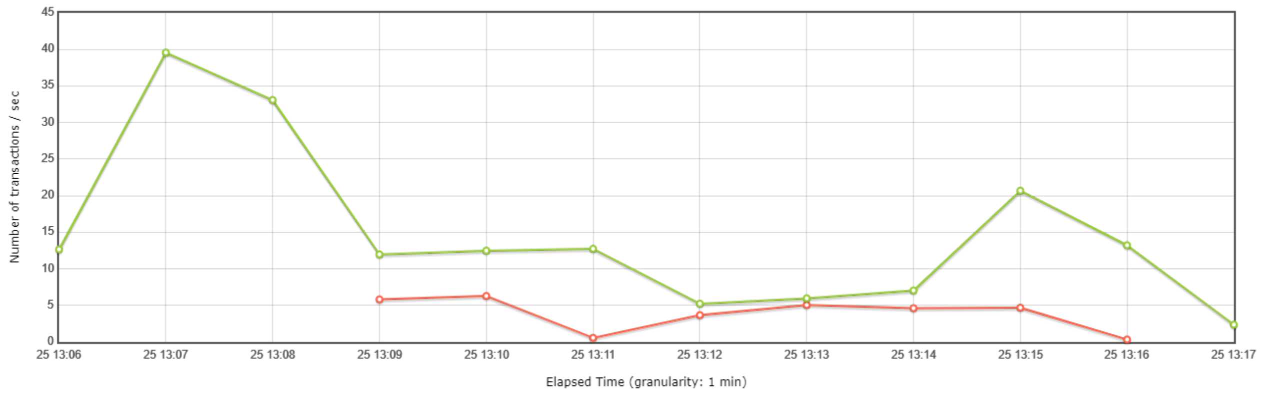
2- High Load Test (500 Concurrent Threads)

Since the single-node deployment already struggled with 100 concurrent threads, we tested only the four-node deployment with the high load test. This test aimed to determine the server's limits by attempting to receive failed responses.

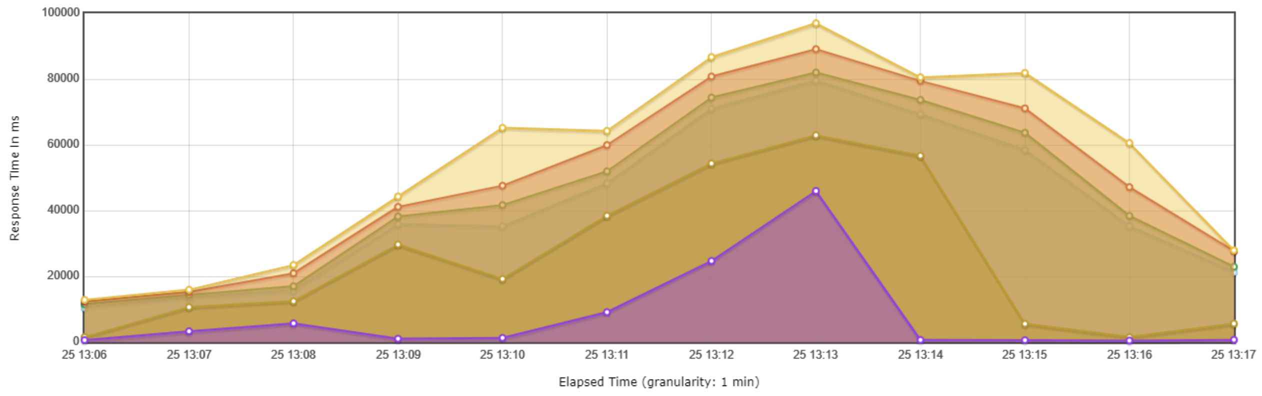
- **Average Response Time**: 20650.28 ms
    - **Median Response Time**: 13979.50 ms
    - **90th Percentile Response Time**: 49219.40 ms
    - **Max Response Time**: 96999 ms
    - **Throughput**: 19.09 requests per second
    - **Success Rate**: 85.05%
    - **Overall Apdex Score:** 0.049

Under high load conditions, the four-node configuration maintained a reasonable throughput of 19.09 requests per second. However, the average response time increased significantly, and the success rate dropped to 85.05%. This indicates that while the system can handle a higher number of concurrent users, performance degradation occurs, and some requests fail under peak load conditions.

Throughput Over Time



Response Time Percentiles Over Time (Successful Responses)



These two shows the throughput (requests per second) over time:

- **Initial Spike**: Throughput peaks over 40 requests per second initially, indicating efficient handling of the initial load.
- **Rising Response Times**: Response times increase significantly as the test progresses, with the 99th percentile approaching 100,000 ms.
- **Peak and Decline**: Response times peak midway and then decline but remain high.
- **Stabilization**: Throughput stabilizes between 5 and 15 requests per second, showing the system's struggle to maintain high throughput under sustained load.
- **Final Phase**: Slight increase followed by a decrease in throughput towards the end of the test.

This indicates that while the system can handle bursts of high load, it struggles to maintain consistent throughput over time.

In conclusion, the high load test shows that while the system can initially handle the load, it struggles to maintain consistent performance, with high variability in response times. Even though the server handled the high load well initially, its performance degraded over time, indicating a need for further optimization to improve sustained performance under peak load conditions.