Verteilte Systeme - Gruppe X - RustyBalancer

Maximilian Müller und Vivian Berger

RustyBalancer:

Introduction: RustyBalancer is a prototype for a load balancer and deployment agent developed in Rust. It is supposed to manage and optimize the distribution of incoming network traffic across multiple backend-servers. This documentation provides a theoretical overview of RustyBalancer's architecture and its features.

System Architecture: RustyBalancer's architecture is designed to handle both load balancing and deployment processes efficiently. The project is implemented in Rust, a systems programming language known for its performance, safety, and concurrency features. These characteristics make sure that the distributed system can handle high loads and provides a reliable and secure environment. The system consists of the following main components:

Load Balancer

The load balancer is responsible for distributing incoming requests across multiple backend servers to ensure availability and the equal distribution of resources. (nimmt anfragen an und leitet die weiter an deployment agent) Key characteristics include:

- **Request Distribution:** The load balancer uses algorithms to determine how incoming requests are allocated to backend servers.
- **Fault Tolerance:** It can detect server failures and redirect traffic to healthy servers.
- **Scalability:** The load balancer can scale horizontally by adding more backend servers as needed.

Deployment Agent

The deployment agent manages the deployment and scaling of applications. (rechnet roundrobin aus und teilt die ergebnisse loadbalancer mit)

Load Balancing Algorithms

RustyBalancer incorporates a load balancing algorithm to distribute incoming traffic effectively. The algorithm used is Round Robin with Priorities.

Round Robin with Priorities

The Round Robin algorithm is a widely-used load balancing technique that distributes requests in a cyclic order. This approach ensures that each server receives an equal number of requests over time. The enhanced version, Round Robin with Priorities, introduces an additional layer to prioritize servers based on their importance or capacity:

- **Basic Round Robin:** Requests are distributed sequentially across a list of servers. After reaching the end of the list, the algorithm starts again from the beginning.
- **Priorities:** Requests are first distributed based on assigned priorities, ensuring that higher-priority requests receive more resources compared to lower-priority ones. This adjustment helps to balance the load more effectively according to the criticality of the services.

The combination of Round Robin and Priorities allows RustyBalancer to handle workloads more efficiently and ensure that critical services are prioritized while keeping the traffic balanced.

Proactive Handling

whats this do?

Docker

The application uses Docker and Docker Compose. Therefore it can be containerized and run in isolated environments, making the deployment process more manageable and scalable. By using the isolation provided by containers, dependencies and configurations can be minimized.

Key functions include:

- **Environment Configuration:** It supports multiple environments, such as production, development, and slim, allowing for different configurations and optimizations.
- **Automated Deployment:** The engine automates the deployment process, reducing manual intervention and minimizing errors.

Customization and Extensibility

RustyBalancer is designed with customization in mind.
Users can adapt the system to their specific needs by:

- **Customizing Deployment Configurations:** Users can create or modify deployment configurations to better fit their environment and application needs.
- **Extending Functionality:** The system is built to allow the integration of additional features and services, enhancing its capabilities as needed.

Implementation Details:

Communication across components

The load balancer and deployment agent communicate with each other with websocket connection. This allows for real-time updates and efficient data exchange between the components. The deployment agent sends the load balancer information about the current state of the backend servers, as well as compute their priorities and the round robin distribution. It is sent in a queue? The load balancers receives http-requests and forwards them to the deployment agent based on the queue received from the websocket connection.

Encountered Problems and Solutions

Handling the communication between the load balancer and deployment agent was a challenging task as it required real-time updates and efficient data exchange. Requiring an asynchronous communication model. After deciding to use websockets to establish a connection between the components, there were still struggles of keeping the channels organised and uni-directional as to avoid blockages.

Conclusion: RustyBalancer represents a robust solution for managing load distribution and application deployment. By combining sophisticated load balancing algorithms, proactive handling techniques, and flexible configuration options, RustyBalancer aims to provide a high-performance, reliable, and adaptable system for application environments.

Conclusion: J. Smith and A. N. Other (*The IET, Stevenage, UK*)

E-mail: jsmith@theiet.org

References

1 Anderson, P.: 'A poor man's derivation of scaling laws for the Kondo problem', *J. Phys. C.*, 1960, **3**, p. 2436
2 Coleman, P.: '1/N expansion for the Kondo lattice', *Phys. Rev. B*, 1983, **28**, pp. 5255-5262
3 Ludwig, I. and Ludwig A. W. W.: 'Kondo effect induced by a magnetic field', *Phys. Rev. B*, 2001, **64**, p. 045328