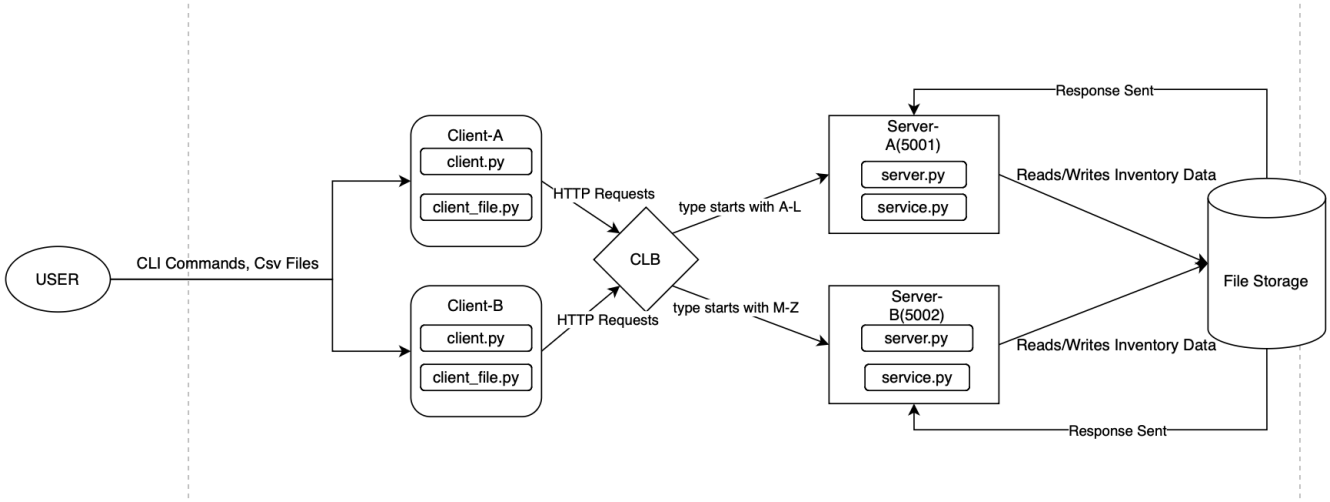
DESIGN DOCUMENT

Illustration Of the Design

This design introduces client-side load balancing (CLB) for the Inventory Service, which distributes API requests between two server instances based on the first letter of the ***type*** parameter. The routing is handled by the client using a function called locator, which routes requests for types starting with [A-L] or [a-l] to Server-a and [M-Z] or [m-z] to Server-b. The implementation preserves backward compatibility, meaning no changes are required on the server side. This design allows the system to scale by balancing the load without altering the API's behavior for end-users.

System Diagram



The load balancing is implemented entirely on the client side using the locator function, which determines the appropriate server based on the first character of the ***type*** parameter.
The InventoryClient class routes API calls to either Server-a or Server-b, depending on the type, and maintains backward compatibility with existing API calls. The design includes unit tests for the locator function and end-to-end tests to ensure that requests are routed correctly. The servers themselves remain unchanged, and environment variables are used to switch between local and Docker environments seamlessly.

Benefits of the Design

The design offers improved scalability by distributing API requests across two server instances, thus reducing bottlenecks. It maintains backward compatibility, meaning no changes are required to the server-side code or the existing API. The client-side load balancing is easy to implement, as it only involves changes on the client, and the use of environment variables allows the solution to work both in local testing and Docker-based deployments. This approach

also reduces the risk of overloading a single server, improving the system's overall performance and reliability.

Value of the Benefits

This design improves system scalability and performance by distributing the load across multiple servers. The backward compatibility ensures that existing users do not face any disruptions, while new users benefit from the enhanced performance. By offloading the balancing logic to the client, the implementation is simplified, avoiding the need for complex server-side configurations. Additionally, the flexibility to switch between local and Docker environments makes it easier for developers to test and deploy the system without extensive reconfiguration.

Drawbacks of the Design

The primary drawback of this design is the added complexity on the client side, as the load balancing logic is handled by the client rather than the server. This introduces potential points of failure in the client, which could impact the entire system. Additionally, the static nature of the routing logic based on the first letter of the type does not account for dynamic load variations. If one server receives disproportionately more traffic, the load distribution may become unbalanced, leading to performance degradation on that server.

Impact of the Drawbacks

The client-side complexity may require more maintenance in the future, especially if more servers or more sophisticated routing logic is introduced. If the client fails, the entire system could be affected, as the servers rely on the client to direct traffic correctly. Furthermore, the static load distribution based on the first letter of type could result in uneven traffic distribution, potentially causing one server to become overloaded while the other remains underutilized. This could impact the system's ability to handle traffic efficiently under certain conditions.

Value Analysis

The client-side load balancing provides a simple, effective way to handle increased load by distributing requests across multiple servers. This approach improves scalability without modifying the server-side code, preserving the existing API. While the system benefits from better performance and maintainability, the client-side logic adds some complexity. Nonetheless, the overall design strikes a good balance between scalability, simplicity, and maintainability, making it a valuable addition to the Inventory Service.