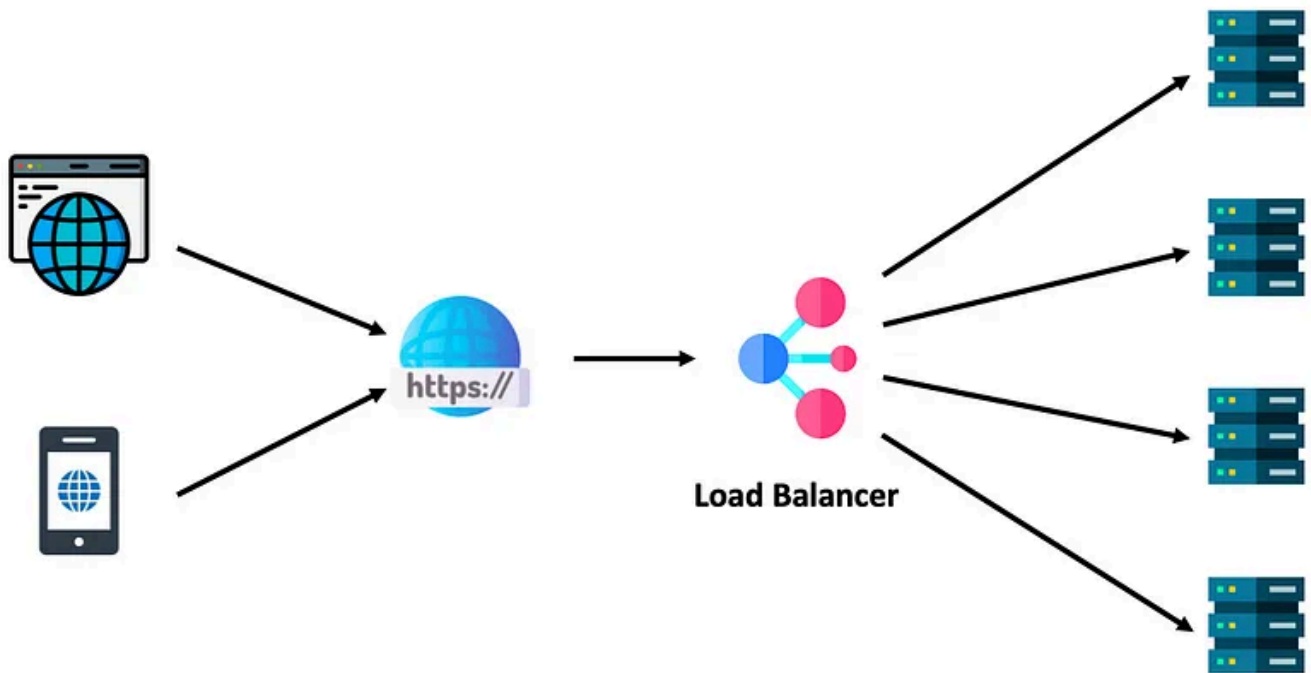
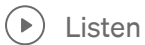


Load Balancing in Distributed Systems: Exploring Concepts and Practical Demonstration



Truong Bui · Follow

7 min read · Aug 17, 2023



Load Balancing in Distributed Systems

Part of the Domain Knowledge Series: If you haven't read my other articles yet, please refer to the following links:

1. [Scaling SQL Read Operations: Effective Strategies and Detailed Examples](#)
2. [Exploring Caching in Distributed Systems: Concepts and Practical Demonstration](#)

Recently, I've been thinking a lot about improving my Domain Knowledge. In the software engineering world, having good Domain Knowledge is essential for becoming a better software engineer. Today, I will explore the load balancer in

distributed systems and do a practical demonstration as part of my self-directed learning.

Table of Contents

1. Concepts
2. Types
3. Algorithms
4. Practical Demonstration of Load Balancing
5. Time to test what we did

Concepts

Load balancing is the process of distributing incoming network traffic across a set of resources ensuring the performance and reliability of the system. This provides the flexibility to add or subtract resources as demand dictates.

A load balancer can sit in front of the server and direct client requests across all servers capable of servicing them, optimizing speed and capacity use. This prevents one server from getting too busy and slowing down. If a server goes down, the load balancer redirects traffic to the remaining online servers. When we add a new server, the load balancer automatically starts sending requests to it.

Types

DNS load-balancers

A technique used to distribute incoming network traffic across multiple servers or resources by manipulating the Domain Name System (DNS) to provide different IP addresses in response to DNS queries.

Unfortunately, DNS load balancing might not provide instant failover. Clients might continue to use the IP address of a failed server until the DNS record expires and is refreshed.

Layer 4 load balancer

Utilize data from the networking transport layer (Layer 4) to determine the distribution of client requests among a server cluster. For additional insights, visit: [layer-4-load-balancing](#)

Layer 7 load balancer

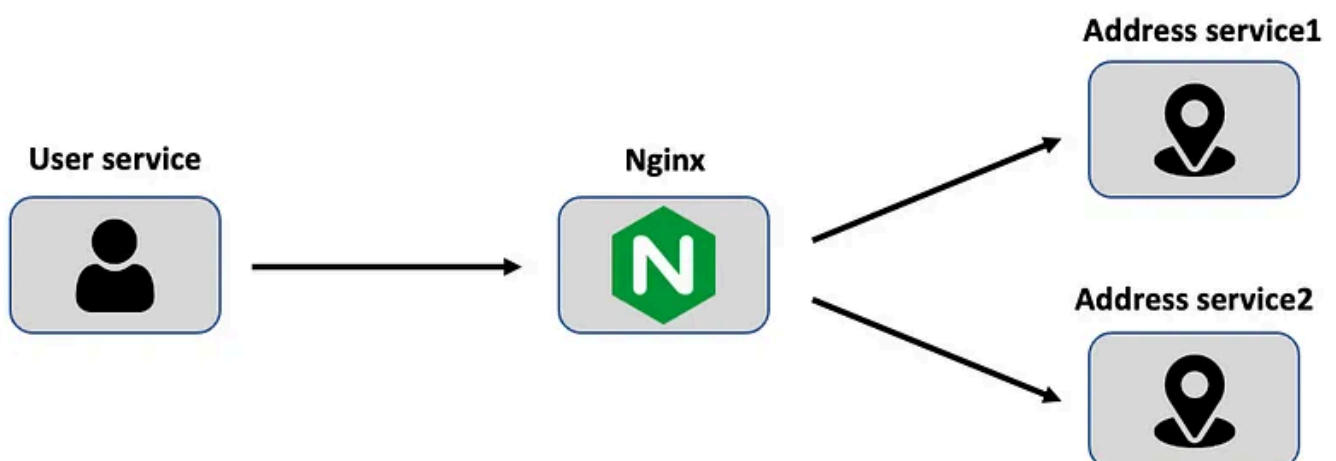
Base the routing decisions on various characteristics of the HTTP header and on the actual contents of the message, such as the URL, the type of data (text, video, graphics), or information in a cookie. For additional insights, visit: [layer-7-load-balancing](#)

Algorithms

- **Round-robin:** The most basic approach. The load balancer holds a list of active servers and directs the initial request to the first server, the second request to the second server, and so forth.
- **Weighted Round-robin:** Builds on the round-robin technique to account for server characteristics, so that the servers with more resources get proportionally more of the requests.
- **Least Connections:** Directs new requests to the server with the fewest active connections at any given moment. The directing decision can be made based on the number of active connections for each server in the load balancer's pool.
- **Least Response Time:** Distributing incoming network traffic to the server that has the lowest response time. The directing decision can be measured with various metrics, including latency, round-trip time for each server.
- **Hashing:** Utilizing a hash function to transform the client's identifier (e.g., IP address or session token) into a hash value, allows for consistent routing of requests from a specific client to the same server.

Practical Demonstration of Load Balancing

Scenario



We employ **Nginx as a load balancer**, distributing requests from User-service to Address services.

Prerequisite

- Spring Boot 3+
- JDK 17
- Docker

Implementations

Address services

For the purpose of this demonstration, we keep address services very simple. Each service exposes just one API, returning a distinct string, This distinction can help us identify load balancer-directed services.

The source code is available at these GitHub links: [address-service1](#), [address-service2](#)

Remember, we're using Docker to deploy our system. Each service requires a Dockerfile to build Docker images.

```
## Address Service1 Dockerfile
# Use the official OpenJDK base image for Java 17
FROM openjdk:17-jdk-slim-buster
# Set maintainer name
MAINTAINER truongbn
# Set the working directory inside the container
WORKDIR /app
# Copy the compiled Java application JAR file into the container
COPY target/address-service1-0.0.1-SNAPSHOT.jar /app/address-service1.jar
# Expose the port that the application will run on
EXPOSE 8080
# Command to run the Java application
CMD ["java", "-jar", "address-service1.jar"]
```

```
## Address Service2 Dockerfile
# Use the official OpenJDK base image for Java 17
FROM openjdk:17-jdk-slim-buster
# Set maintainer name
MAINTAINER truongbn
# Set the working directory inside the container
WORKDIR /app
```

```
# Copy the compiled Java application JAR file into the container
COPY target/address-service2-0.0.1-SNAPSHOT.jar /app/address-service2.jar
# Expose the port that the application will run on
EXPOSE 8080
# Command to run the Java application
CMD ["java", "-jar", "address-service2.jar"]
```

User services

User service is also kept simple. You can find its source code at this [GitHub link](#): **user-service**

```
## User Service Dockerfile
# Use the official OpenJDK base image for Java 17
FROM openjdk:17-jdk-slim-buster
# Set maintainer name
MAINTAINER truongbn
# Set the working directory inside the container
WORKDIR /app
# Copy the compiled Java application JAR file into the container
COPY target/user-service-0.0.1-SNAPSHOT.jar /app/user-service.jar
# Expose the port that the application will run on
EXPOSE 9090
# Command to run the Java application
CMD ["java", "-jar", "user-service.jar"]
```

There is one aspect requires explanation: Calling external address services. We achieve this using **RestTemplate**, a central spring class used to consume the web services for all HTTP methods.

```
@RestController
@RequestMapping("/api/v1/users")
@RequiredArgsConstructor
public class UserController {
    private final RestTemplate restTemplate;
    private static final String ADDRESS_SERVICE_URL = "http://nginx-service:80/";
    @GetMapping("/addresses")
    public String getUserAddress() {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<String> entity = new HttpEntity<>(null, headers);
        ResponseEntity<String> response = restTemplate.exchange(ADDRESS_SERVICE
            entity, String.class);
```

```
        return response.getBody();  
    }  
}
```

Observe that `ADDRESS_SERVICE_URL` represents the URL of the Nginx load balancer. The user service is meant to send requests to this URL, while the responsibility of request distribution among the address services is handled by Nginx.

The `nginx-service` hostname I used is just a placeholder. In practice, the hostname of your Nginx container will depend on how you named the container when you started it. When you run a Docker container, you can provide a custom name using the `--name` option. If you don't provide a name, Docker will generate a random name for the container.

To find out the actual hostname of your Nginx container, follow these steps:

1. List the active containers with this command: `docker ps`
2. Identify the container corresponding to your Nginx instance. The container's name is displayed in the first column, labeled NAMES.
3. The Nginx container's name visible in the NAMES column is the hostname to utilize when making requests to the container.

We'll explore this further in the upcoming section.

Build & Run

```
# Create a Docker network to allow communication between the services and the l  
# This will enable the load balancer to forward requests to the appropriate cor  
docker network create load-balancer-network
```

```
# Navigate to the directory of the first address service  
# build the first address service project  
mvn clean install  
  
# Build Docker Image  
docker build -t address-service1 .
```

```
# Run service container
```

```
docker run -d --name address-service1 -p 8080:8080 --network load-balancer-netw
```

```
# Navigate to the directory of the second address service
```

```
# build the second address service project
```

```
mvn clean install
```

```
# Build Docker Image
```

```
docker build -t address-service2 .
```

```
# Run service container
```

```
docker run -d --name address-service2 -p 8081:8081 --network load-balancer-netw
```

```
# Navigate to the directory of the user service
```

```
# build the user service project
```

```
mvn clean install
```

```
# Build Docker Image
```

```
docker build -t user-service .
```

```
# Run service container
```

```
docker run -d --name user-service -p 9090:9090 --network load-balancer-network
```

Create an nginx directory including 2 files: nginx.conf and Dockerfile

```
# nginx.conf
```

```
# Define an upstream block that lists backend servers
```

```
upstream addressservices {
```

```
    # Hostname discovery follows a similar approach as mentioned for Nginx above
```

```
    # "weight" indicates the relative server weights.
```

```
    # A higher weight assigns more requests to a server compared to a lower-weight
```

```
    server address-service1:8080 weight=2;
```

```
    server address-service2:8081 weight=8;
```

```
}
```

```
server {
```

```
    # Listen on port 80
```

```
    listen 80;
```

```
    # Handle requests for this domain
```

```
    server_name nginx-service.com;
```

```
# Define a location block for requests to "/api/v1/addresses"
location /api/v1/addresses {
    # Pass the request to the "addressservices" upstream
    proxy_pass http://addressservices;
}
}
```

```
## Nginx Dockerfile
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Now let's build and run Nginx docker image.

```
# Navigate to the directory of the Nginx Dockerfile
docker build -t nginx-image .
docker run -d --name nginx-service -p 80:80 --network load-balancer-network ngi
```

Time to test what we did

Now, everything is ready! 😎

Access the URL below 10 times: <http://localhost:9090/api/v1/users/addresses>

The result is:

```
Address Service 2
Address Service 2
Address Service 1
Address Service 2
Address Service 2
Address Service 2
Address Service 2
Address Service 2
Address Service 1
Address Service 2
Address Service 2
```


Observe that the `address-service2` received approximately 4 times more requests than `address-service1`.

We have just explored the concept of load balancing in distributed systems and conducted a brief demonstration to observe its behavior.

Hope you can find something useful!

The completed source code can be found in this GitHub repository:
<https://github.com/buingoctruong/Load-balancer-distributed-system>

Happy learning!

Bye!

System Design Interview

Load Balancing

Load Balancer

Nginx

Scalability



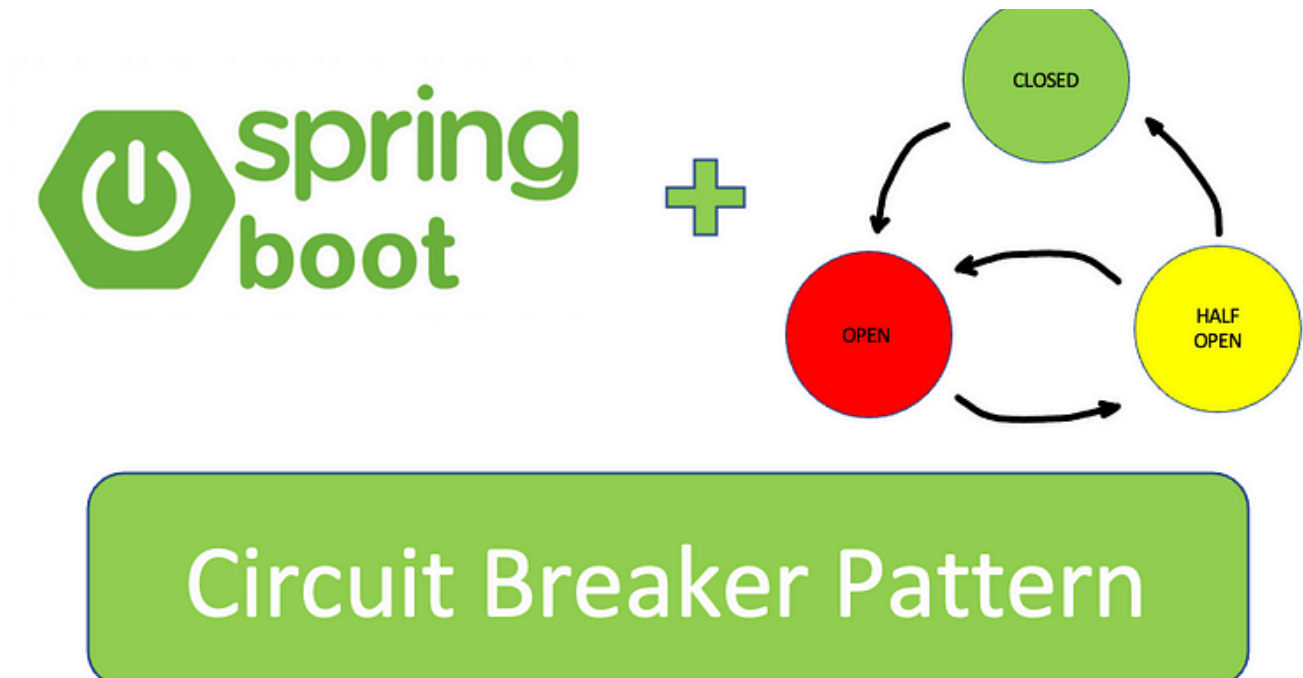
Follow

Written by Truong Bui

537 Followers

Software Engineer - Dreamer - Writer - Reader

More from Truong Bui



 Truong Bui

Circuit Breaker Pattern in Spring Boot

Part of the Resilience4J Article Series: If you haven't read my other articles yet, please refer to the following links: 1. MicroService...

Apr 18, 2023  170  3



Spring Boot 3
Spring Security 6

 Truong Bui

JWT Authentication and Authorization with Spring Boot 3 and Spring Security 6

During my journey of learning Spring Security, I had some thinking whether there are other developers who share a similar experience. Those...

May 11, 2023

👏 621

💬 21



Spring Boot + Resilience4J *RateLimiter*



Truong Bui

MicroService Patterns: Rate Limiting with Spring Boot

Part of the Resilience4J Article Series: If you haven't read my other articles yet, please refer to the following links: 1. Circuit Breaker...

Aug 13, 2023

👏 97

💬 3





JaCoCo Code Coverage with Spring Boot

I'm pretty sure that each one of us has and will continue to write Unit Testing in our projects. Unit Testing is playing a very important...


May 19, 2023 🖱 76



See all from Truong Bui

Recommended from Medium



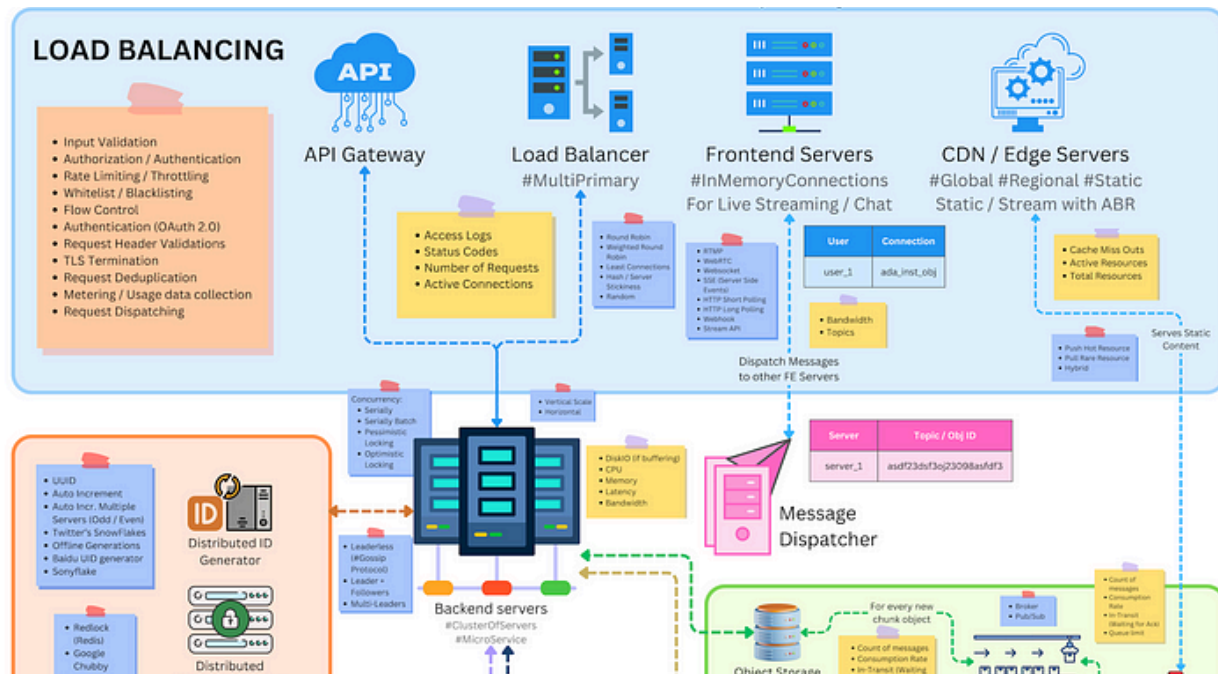
 Ayush Gupta

Concurrency Control: Pessimistic Vs Optimistic Locking

Explore how Pessimistic and Optimistic Locking strategies ensure data integrity in concurrent transactions. Which one fits your needs best?

★ Aug 18 🖱 54





Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

🌟 Sep 17, 2023 🖱 8.7K 💬 59

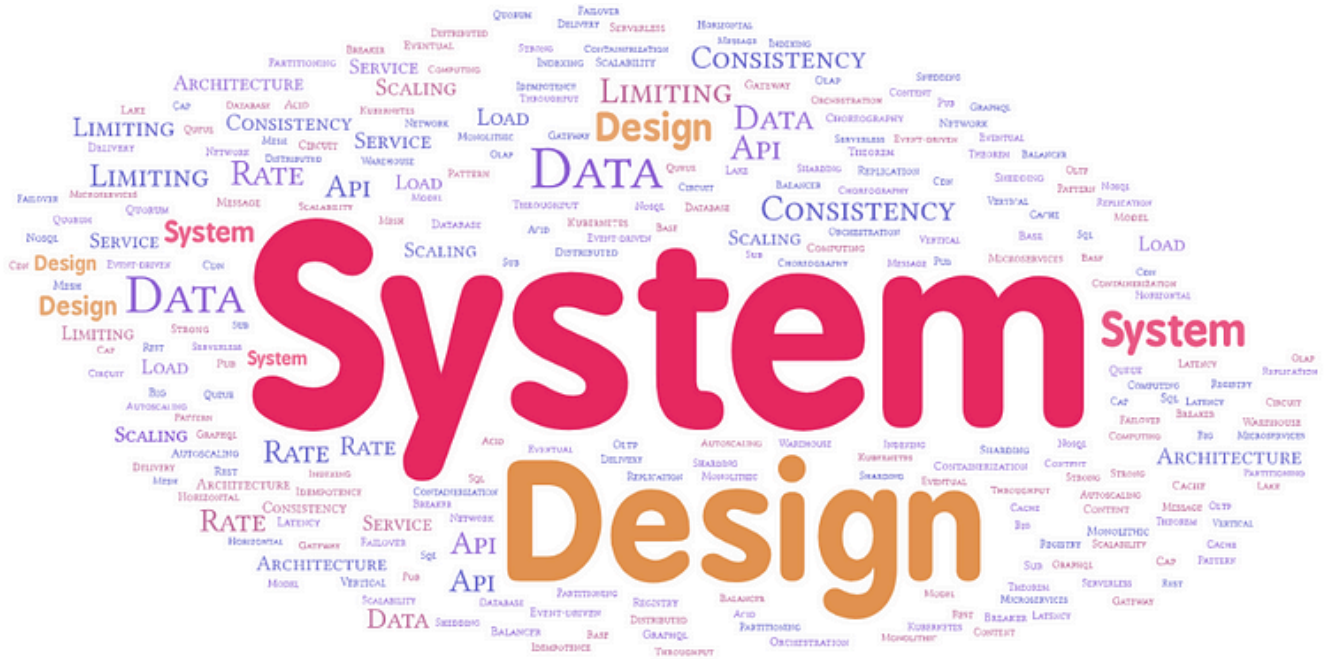
🔖 ...

Lists



Natural Language Processing

1733 stories · 1314 saves

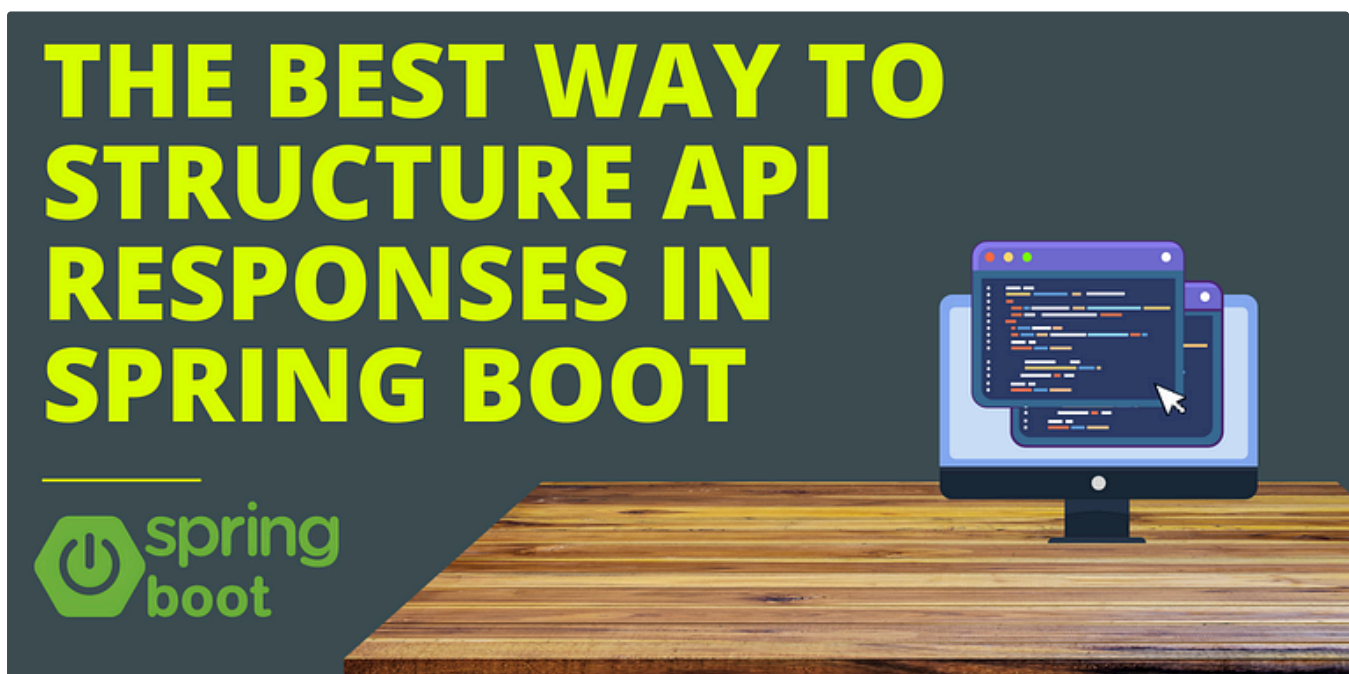


Tharun Kumar Reddy Polu

Top 50 System Design Terminologies You Must Know

Master the Essential Terms to Ace Your System Design Interviews with Explanations, Practical Examples, and Comprehensive Resources

Jul 4 🖱️ 1.5K 💬 17

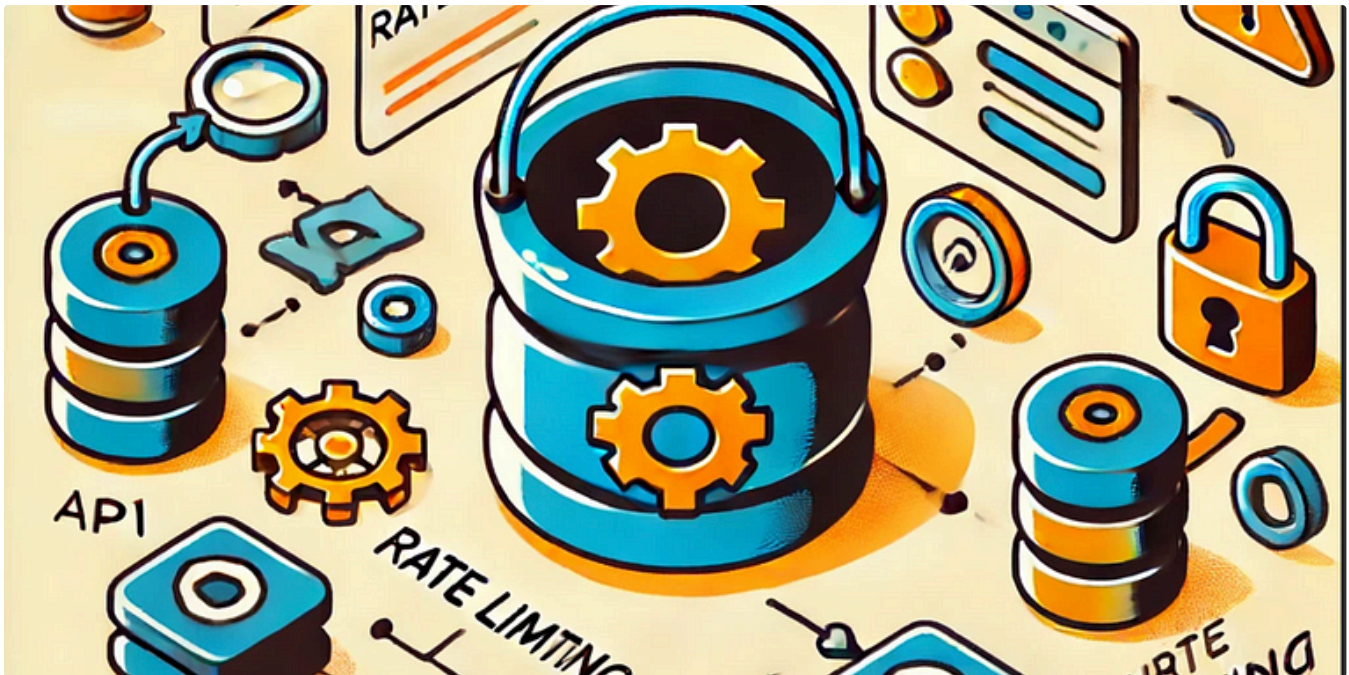


Rabinarayan Patra in Insights from ThoughtClan

The Best Way to Structure API Responses in Spring Boot

Discover the best practices for structuring API responses in Spring Boot. Enhance clarity, consistency, and maintainability in your APIs...

★ Aug 2 🖱 351 💬 13

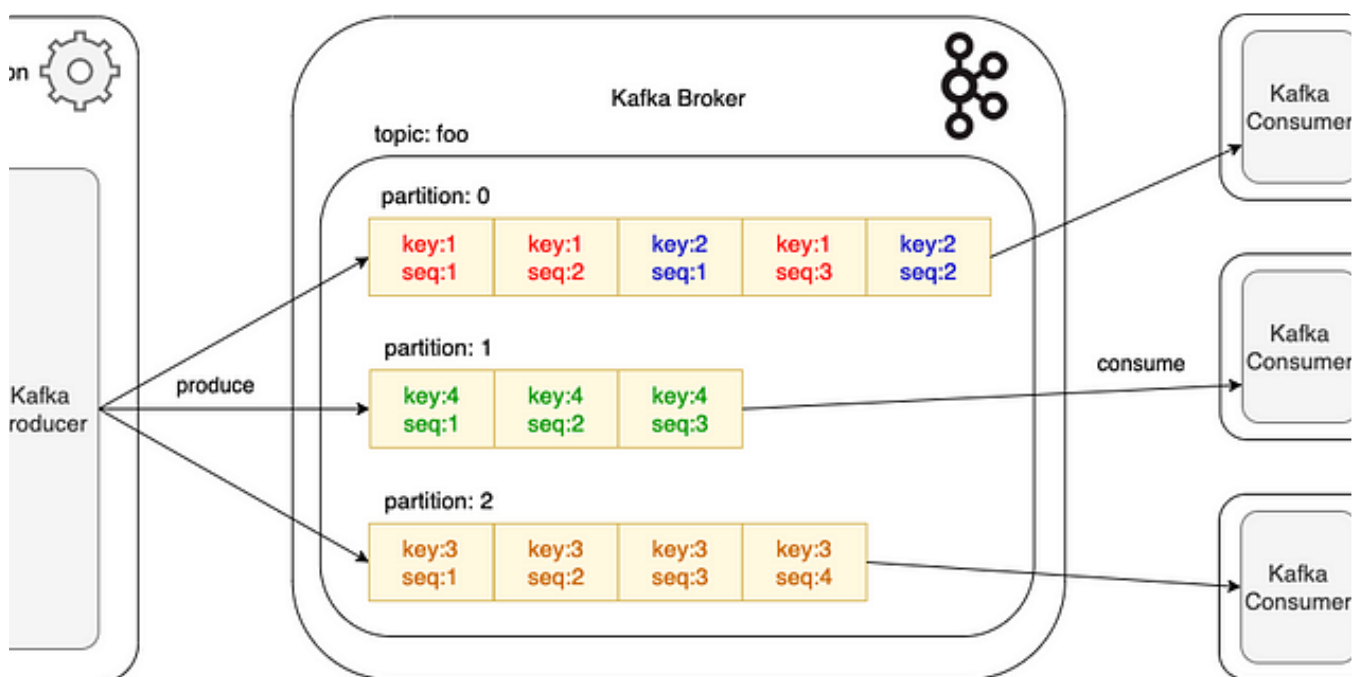


Master Spring Ter

Implementing Rate Limiting in Spring Boot with Bucket4j

As web applications grow and handle more traffic, ensuring their reliability and performance becomes crucial. One effective way to achieve...

Jun 16 🖱 10



Sutanu Dutta

Apache Kafka Explained: Real-World Use Cases and Practical Insights

We have learnt how Kafka works behind the scene earlier in this article. Now it's time to understand some nuances while working with Kafka...

★ Jul 13 🖱 13



See more recommendations