Course Discussions Share

Uses of Load Balancing

ON THIS PAGE

Load balancing is a technique used to distribute workloads evenly across multiple computing resources, such as servers, network links, or other devices, in order to optimize resource utilization, minimize response time, and maximize throughput. Here are the five fundamental uses of a Load Balancer.

1. High Availability & Fault Tolerance (The "Survival" Rule)

Hardware fails. Networks flap. Memory leaks happen. If you have one server and it dies, your business is dead. If you have ten servers and one dies, but your client doesn't *know* which one is alive, your business is *still* dead.

A load balancer performs **Health Checks**. It acts as the heartbeat monitor for your cluster. It constantly pings your backend servers ("Are you alive? Can you take a request?"). If a server fa to answer or returns a 5xx error, the LB cuts it off instantly. It stops sending traffic to the corpse and reroutes it to the living.

- **Without an LB:** 2% of your users (the ones unlucky enough to be routed to Server #3's IP) open the app and see a spinning wheel. They switch to Lyft. You lose revenue.
- **With an LB:** The load balancer sees that Server #3 failed its health check (e.g., failed to return a `200 OK on /health` within 2 seconds). It immediately removes Server #3 from the active rotation. 100% of the traffic is instantly spread across the remaining 49 servers. The users never even noticed a glitch.

2. Horizontal Scalability (The "Black Friday" Defense)

Vertical scaling (buying a bigger machine) hits a ceiling. Eventually, you can't buy a bigger CPU. You need **Horizontal Scaling**, which is adding *more* machines. But how do you tell the entire internet that you suddenly have 50 new servers? You don't. You tell the Load Balancer.

The LB acts as the **Unified Entry Point** (Virtual IP). Clients only know the LB's address. When traffic spikes, you spin up more backend instances, register them with the LB, and boom, you have more capacity.

Example: You are the lead engineer for an **e-commerce site on Black Friday**. Normal traffic is 1,000 requests per second (RPS). You have 5 servers. Suddenly, a flash sale starts. Traffic spikes to 100,000 RPS.

- **The Strategy:** Your Auto-Scaling Group detects high CPU usage and boots up 100 new EC2 instances.
- **The LB Role:** As soon as those new instances boot, they register with the Load Balancer. The LB immediately starts throwing requests at them. The massive surge of customers is dilut across the new fleet. The client (the browser) didn't have to update DNS records or know anything changed. It just worked.

You need to update your application. In the junior world, you put up an "Under Maintenance" page. That is unacceptable here. You need to update code while users are still using the site.

Load balancers allow for **Connection Draining** and strategies like **Blue-Green Deployment**. You can signal the LB to stop sending *new* connections to a specific server while allowing *existing* connections to finish naturally, then take it offline for patching.

Example: You are deploying a new version of a **Banking API**.

- **The Process:** You have a "Blue" pool (current version) and a "Green" pool (new version).
- **The LB Role:** You tell the load balancer, "Send 1% of traffic to the Green pool." You monitor the logs. No errors? Good. "Send 10%." Still good? "Send 50%." "Switch to 100%."
- **The Save:** If you spot a critical bug at the 1% mark, you instantly tell the LB "Revert to Blue." The rollback is instant. No user downtime, no failed transactions.

4. Security & Attack Mitigation (The "Shield")

Never expose your application servers directly to the internet. If an attacker knows your backend server's IP, they can bypass your firewalls and hit you directly.

A Load Balancer acts as a **Reverse Proxy**. It terminates the connection. The client talks to the LB; the LB talks to the server. The internet never touches your backend. Furthermore, the LB can absorb **DDoS attacks** (Distributed Denial of Service) and filter malicious traffic before it even reaches your expensive application logic.

Example: You are running a **Social Media Platform**. A botnet targets your login page with a **SY** **Flood attack** (millions of fake connection requests) to crash your database.

- **The Outcome:** The Load Balancer detects the abnormal traffic pattern. It drops those connections at the edge. Your backend servers see only valid traffic volumes and continue processing legitimate user logins. The LB took the punch so your app didn't have to.

5. SSL Termination (The "Offloader")

Encryption is expensive. Handshaking SSL/TLS (decrypting HTTPS traffic) takes significant CPU power. If your web servers have to do this for every request, they are wasting cycles on math instead of running your business logic.

You can offload this to the Load Balancer. This is called **SSL Termination**. The client speaks HTTPS to the Load Balancer. The Load Balancer decrypts it and speaks HTTP (or lighter encryption) to your backend servers inside your secure private network.

Example: You are building a **High-Frequency Trading Dashboard**. Latency is everything.

- **The Bottleneck:** Your servers are hitting 90% CPU usage, but your code profiling shows that 30% of that is just OpenSSL handling encryption overhead.
- **The Fix:** You move your SSL certificates to the Load Balancer.
- **The Result:** Your backend servers no longer have to perform the heavy decryption math. Their CPU usage drops to 60%, effectively increasing your capacity by a third without buying a single new server.

← Previous Next →

Load Balancing Algorithms Load Balancer T

Mark as Completed