**1. What is Load Balancing?**

**Definition:**

Load balancing is the process of distributing incoming network traffic across multiple servers or resources. It ensures that no single server becomes overwhelmed, which improves responsiveness, availability, and fault tolerance of applications, websites, and other services.

**Purpose:**

* **Optimize resource use** – prevents overuse of one server.
* **Maximize throughput** – more consistent and faster service.
* **Minimize response time** – distributes requests to reduce delay.
* **Ensure reliability** – if one server fails, others can handle the load.

**Where It’s Used:**

* Web servers (handling HTTP requests)
* Application servers (APIs, databases)
* Data centres and cloud platforms (AWS, Azure)
* DNS systems and content delivery networks (CDNs)

**Components Involved:**

* **Load balancer** – the tool or device that manages the distribution.
* **Server pool (backend)** – a group of servers where traffic is sent.
* **Health checks** – monitoring servers to route traffic only to healthy ones.

**2. Load Balancing Strategies Overview**

Different algorithms determine how requests are distributed. These strategies impact performance, resource utilization, and user experience.

**Types of Load Balancing:**

1. **Static Load Balancing**  
   Predefined rules are used, doesn't consider server state.
2. **Dynamic Load Balancing**  
   Considers current server load, response time, or number of connections.

**Common Strategies:**

* Round Robin
* Least Connections
* Random
* IP Hashing
* Weighted Round Robin
* Least Response Time

**Strategy 1 – Round Robin**

**How It Works:**

* Requests are distributed in a circular order.
* Each server gets a request in turn, regardless of current load.

**Example:**  
If there are 3 servers (A, B, C) and 6 incoming requests, the distribution would be:

* Request 1 → A
* Request 2 → B
* Request 3 → C
* Request 4 → A
* Request 5 → B
* Request 6 → C

**Advantages:**

* Simple to implement.
* Effective when all servers have similar capacity.

**Disadvantages:**

* Doesn’t account for current load or server performance.
* If one server is slower or has issues, performance suffers.

**Strategy 2 – Least Connections**

**How It Works:**

* The request is sent to the server with the fewest active connections.

**Example:**  
If Server A has 2 active connections, Server B has 1, and Server C has 4:

* The next request goes to Server B.

**Advantages:**

* Efficient for handling variable-length sessions or requests.
* Good for systems with unpredictable traffic patterns.

**Disadvantages:**

* Requires real-time tracking of connections.
* Slightly more complex to implement.

**Strategy 3 – Random**

**How It Works:**

* Requests are assigned to servers randomly.
* All servers have an equal chance of being selected.

**Example:**

* Request 1 → C
* Request 2 → A
* Request 3 → B
* Request 4 → C  
  (Even distribution isn't guaranteed)

**Advantages:**

* Very simple and fast to implement.
* Can work well in systems with similar servers.

**Disadvantages:**

* No intelligence or optimization involved.
* May overload some servers by chance.

**Conclusion:**

Choosing the right load balancing strategy depends on:

* Traffic pattern
* Server performance
* Required reliability and responsiveness

Each method has its pros and cons. In real-world systems, advanced load balancers often combine multiple strategies or use AI to dynamically adjust traffic routing.