<u>Notes on Load Balancer</u>

A load balancer is a critical component in modern distributed systems and web architecture. Its primary function is to distribute incoming network traffic across multiple servers, ensuring reliability, scalability, and optimal resource utilization.

1. What is a Load Balancer?
A load balancer acts as an intermediary between clients and backend servers. It receives client requests, determines the most suitable server to handle each request, and forwards it accordingly. This prevents any single server from becoming a bottleneck or failing under heavy traffic.

2. Importance of Load Balancers
- High Availability: Load balancers detect server failures and reroute traffic to healthy servers, ensuring minimal downtime.
- Scalability: They enable horizontal scaling by distributing workloads across additional servers as demand grows.
- Performance Optimization: Load balancers distribute requests evenly, preventing server overload and reducing response times.

- Security: By acting as a gateway, they can block malicious traffic, enforce SSL/TLS encryption, and protect against DDoS attacks.

3. Types of Load Balancers
1. Hardware Load Balancers:
   - Dedicated physical devices.
   - High performance but expensive and less flexible.
2. Software Load Balancers:
   - Software applications deployed on standard hardware or virtualized environments.
   - Flexible, cost-effective, and widely used.
3. Cloud Load Balancers:
   - Provided as a service by cloud providers like AWS, Azure, or Google Cloud.
   - Fully managed, highly scalable, and integrates with cloud ecosystems.

4. Load Balancing Algorithms
- Round Robin: Distributes requests sequentially among servers.
- Least Connections: Directs traffic to the server with the fewest active connections.

- Weighted Round Robin: Allocates traffic based on server capacity or weight.
- IP Hashing: Routes requests based on client IP address for session persistence.

5. Features of Load Balancers
- Health Monitoring: Periodically checks server health and routes traffic away from unresponsive servers.
- SSL Termination: Offloads SSL/TLS decryption, reducing server overhead.
- Session Persistence: Ensures requests from a client are directed to the same server.
- Auto-Scaling Integration: Automatically adjusts server pools based on demand.

6. Use Cases
- Web Applications: Handles high traffic websites and ensures uptime.
- APIs: Manages API traffic for microservices architecture.
- Media Streaming: Distributes high bandwidth requests for video and audio.

7. Challenges and Best Practices

- Proper Configuration: Misconfigurations can lead to uneven traffic distribution.
- Monitoring: Regularly monitor load balancer performance to identify bottlenecks.
- Scaling: Ensure adequate backend server capacity for unexpected traffic surges.