Performance

- There are two key factors that define the performance of a system:

1) **Low Latency**
    - The ability to respond quickly (fast response time).
2) **High Availability**
    - The ability to remain accessible (system is always available).

Low Latency

- **Latency**
    - The time taken by a system to process a request and return a response.
- **Low latency**
    - The system responds quickly (with minimal delay).

Why is Low Latency Important?

1) **Users expect fast interactions**
    - Websites and mobile applications should work quickly.
2) **Real-time systems require low latency**
    - Delays in real-time systems can cause serious problems:
        - **Financial transactions** (e.g., stock trading)
        - **Gaming** (e.g., multiplayer online games)
        - **Streaming services** (e.g., live video, audio)
3) **Improves user experience**
4) **Increases efficiency**

How to Achieve Low Latency?

1) **Caching**
    - Store frequently accessed data in memory instead of retrieving it from a slow database.
    - Examples:
        - **Redis**
        - **Memcached**
2) **Optimized Database Queries**
    - Use indexes for faster lookups.
    - Implement connection pooling.
    - Avoid expensive joins and unnecessary queries.
3) **Load Balancer**
    - Distribute traffic across multiple servers to prevent overload on a single server.
4) **CDN (Content Delivery Network)**
    - Serve static assets (images, videos, etc.) from the closest location to the user.

High Availability

- **Availability**
    - The ability to remain accessible.
- **High Availability (HA)**
    - A system is reliable and accessible with **minimal downtime**.

Why is High Availability Important?

1) Users expect **24/7 uptime**,
    - even on high-traffic days (e.g., Black Friday).
2) Ensures **business continuity**
    - prevents revenue loss
3) Reduces **downtime costs**
    - losses due to downtime

4) Builds **customer trust**.

How to Achieve High Availability?

1) **Failover Mechanisms**
   - Run multiple instances of a service.
   - If one instance fails, another should take over.
2) **Load Balancer**
   - Distributes traffic to multiple servers to prevent overloading a single server.
3) **Database Replication**
   - Have multiple copies of the database for failover protection.
4) **Auto-Scaling**
   - Dynamically increase or decrease the number of instances based on traffic.
5) **Graceful Degradation**
   - If part of the system fails, reduce functionality instead of going completely offline.