Consistency Levels:

- **Strong Consistency:** Every read receives the most recent write, ensuring that all nodes in the system return the same data at any given time, making it ideal for systems where data accuracy and integrity are critical.
- **Eventual Consistency:** Over time, all replicas of data will converge to the same value, but immediate consistency is not guaranteed, making it suitable for systems prioritizing availability and partition tolerance over strict consistency.
- **Read-Your-Write Consistency:** A guarantee that after a write operation, any subsequent read from the same client will return the updated value, providing a useful level of consistency for scenarios where immediate data visibility is essential, like user sessions or shopping carts in e-commerce systems.

Availability:

- Denotes the system or service's capability to be operational and accessible when needed by users.
- Commonly expressed as a percentage of total system downtime during a specific period.
- High availability is often a primary goal in distributed systems to ensure uninterrupted access.

Service Level Agreement (SLA):

- An agreement or contract between a service provider and a customer.
- Defines the services the provider commits to offering, typically in quantifiable terms.
- Specifies performance metrics, response times, uptime assurances, and consequences for non-compliance.
- Vital for establishing expectations and ensuring accountability in service partnerships.

DownTime per year corresponding to Availability:

- 90% (one nine) - More than 36 days
- 95% - About 18 days
- 98% - About 7 days
- 99% (two nines) - About 3.65 days
- 99.9% (three nines) - About 8.76 hours
- 99.99% (four nines) - About 52.6 minutes
- 99.999% (five nines) - About 5.26 minutes
- 99.9999% (six nines) - About 31.5 seconds
- 99.99999% (seven nines) - About 3.15 seconds

How to Increase the availability of your system ?

- Replication: Create duplicate instances of data or services.
- Redundancy: Maintain backup components to take over if the primary fails.
- Scaling: Add more resources to handle increased loads.
- Geographical Distribution (CDN): Distribute resources across different locations.
- Load-Balancing: Distribute workload across multiple systems.
- Failover Mechanisms: Automatically switch to redundant systems on failure.
- Monitoring: Keep track of system performance and operation.
- Cloud Services: Use scalable cloud resources.
- Scheduled Maintenances: Perform routine maintenance during off-peak times.
- Testing & Simulation: Regularly test system performance and failover procedures.

CAP Theorem

It states that, in a distributed data store, you can achieve at most two out of three desirable properties: Consistency, Availability, and Partition Tolerance.

Components of CAP:

Consistency: All nodes in the distributed system will have the same data at the same time. If you write data to one node and then read from another, you'll always get the most recent write.

Availability: Every request to the system will receive a response (not necessarily with the most recent data). The system is always responsive, even if it provides data that is slightly outdated.

Partition Tolerance: The system continues to function correctly even in the presence of network partitions or node failures. Even if some network connections fail, the system still operates without data loss or inconsistencies.

In a Nutshell:

- Trade-offs: CAP theorem implies that you must make trade-offs between these properties.
- Choose Two: You can prioritize any two properties, but you may have to sacrifice the third based on your system's requirements.

Examples:

- YouTube Comments: Availability. In a video streaming platform like YouTube, ensuring that users can access and watch videos without interruptions is crucial. Comments might experience slight delays, but video availability is paramount.
- Instagram Post/Feed: Availability. Keeping the platform available so that users can post photos and view their feeds. Temporary inconsistencies in likes and comments are more acceptable.
- Amazon Cart: Consistency. Maintaining consistency in cart items is crucial for e-commerce. Users rely on accurate cart contents during their shopping experience.
- Uber Payment: Consistency. Confirming payments before booking a ride is essential to avoid disputes and ensure a smooth user experience.
- Uber Search Cab: Availability. In ride-sharing, finding a nearby cab quickly is more critical than perfect consistency in cab availability, which can change rapidly.
- ATM: Consistency. ATMs must ensure that account balances are accurate to prevent errors and disputes.
- File Sharing (Google Docs): Consistency. Google Docs emphasizes real-time consistency to support collaboration and document editing.

Note: Examples can vary depending on our requirements and there is no one correct answer

Functional Requirements vs Non-Functional Requirements:

Functional Requirements:

- Basic system tasks and operations.
- Focus on "what" the system must do.
- Examples: User registration, payment processing.
- Measurable and validated through testing.
- Essential for core functionality.

Non-Functional Requirements:

- Qualities and attributes for user satisfaction.
- Emphasize "how" the system should perform.
- Examples: Performance, availability.
- Qualitative measures for system quality.
- Enhance user experience and reliability.

Properties of a System:

- **Latency**: The time it takes for data to move, affecting server response time.
- **Throughput**: The actual amount of data transferred through the system in a given time, impacting how many transactions can be processed per second.
- **Bandwidth**: The maximum data transfer capacity, like a 100 Mbps internet connection, affecting how much data can flow through the system.
- **Fault**: Refers to a defect or malfunction in a system component, like a server's hard drive crash, which can disrupt operations.
- **Failure**: Occurs when a system or component is unable to perform its intended function, such as when a server stops working due to a hard drive crash.
- **Resiliency**: The system's ability to recover from failures and continue functioning, like a server switching to a backup hard drive after a primary one crashes.
- **Redundancy**: The duplication of critical system components to ensure a backup is available in case of a failure, such as a backup hard drive in a server.

Stateful Systems vs Stateless systems:

Stateful Systems:

- Definition: Systems that maintain or remember the state of interactions.
- Example: E-commerce websites that remember items in your shopping cart.
- Advantages: Can provide a more personalized experience based on past interactions.
- Disadvantages: More difficult to scale due to dependency on previous interactions. May need more resources (e.g., memory) for maintaining state.

Stateless Systems:

- Definition: Systems that don't maintain any state information from previous interactions.
- Example: The HTTP protocol treating each request independently.
- Advantages: Easier to scale and manage since no state information is stored.
- -Disadvantages: Can't provide personalized experiences based on past interactions. Requires all needed data to be sent with each request.