**System Designing**

System design involves planning and structuring complex software or hardware systems to meet specific requirements. It’s a crucial part of software engineering that involves high-level thinking about how to organize components, define data flows, ensure scalability, reliability, and manage resources efficiently.

Here are some key concepts and steps

1. **Understand Requirements**
2. **Identify Major Components**
3. **Choose Appropriate Technology**
4. **Define Interfaces**
5. **Design Data Model**
6. **Consider Scalability and Performance**
7. **Test and Validate**
8. **Deploy and Maintain**

**Architectural Patterns**:

**Client-Server:** The system is divided into clients that request services and servers that provide them.

**Microservices:** The system is broken down into small, independent services that communicate over a network.

**Monolithic Architecture**: The entire system is built as a single, unified application.

**Event-Driven Architecture:** The system responds to events, which trigger actions across different components.

**Scalability**: Designing the system to handle growth, whether in terms of data, traffic, or users. This can involve horizontal scaling (adding more servers) or vertical scaling (adding more resources to existing servers).

**Load Balancing**: Distributing incoming traffic across multiple servers to ensure no single server is overwhelmed. This helps in achieving better performance and reliability.

**Load Balancers**

It helps to distribute load across multiple resources.

It also keeps track of the status of all the resources while distributing requests. If a server is not available, it stops sending traffic.

Load Balancers: Where it can be added

• User — Web Server

• Web server — Internal Server

• Internal Server – Database

**Load Balancers: Types**

**Hardware LB:**

• They are hardware that works as LB but are very expensive.

• Even big companies use them only as the first point of contact & use another mechanism for load-balancing.

**Software Load balancers:**

• It's a hybrid approach. HAProxy is a popular open-source software LB.

• Every client request on this port (Where HAProxy is running) will be received by proxy & then passed to the backend service efficiently.

• HAProxy manages health checks & will remove or add machines to those pools.

• We should start with these.

**Load Balancers: Algorithms**

• Round Robin • Round Robin with Weighted Server

• Least connections • Least Response Time

• Source IP hash • URL hash

**Cache:**

• Caching works on locality of reference principle: recently requested data is likely to be requested again.

• It's like short-term memory which has limited space but is faster & contains most recently accessed items.