15 System Design Building Blocks You Should Know

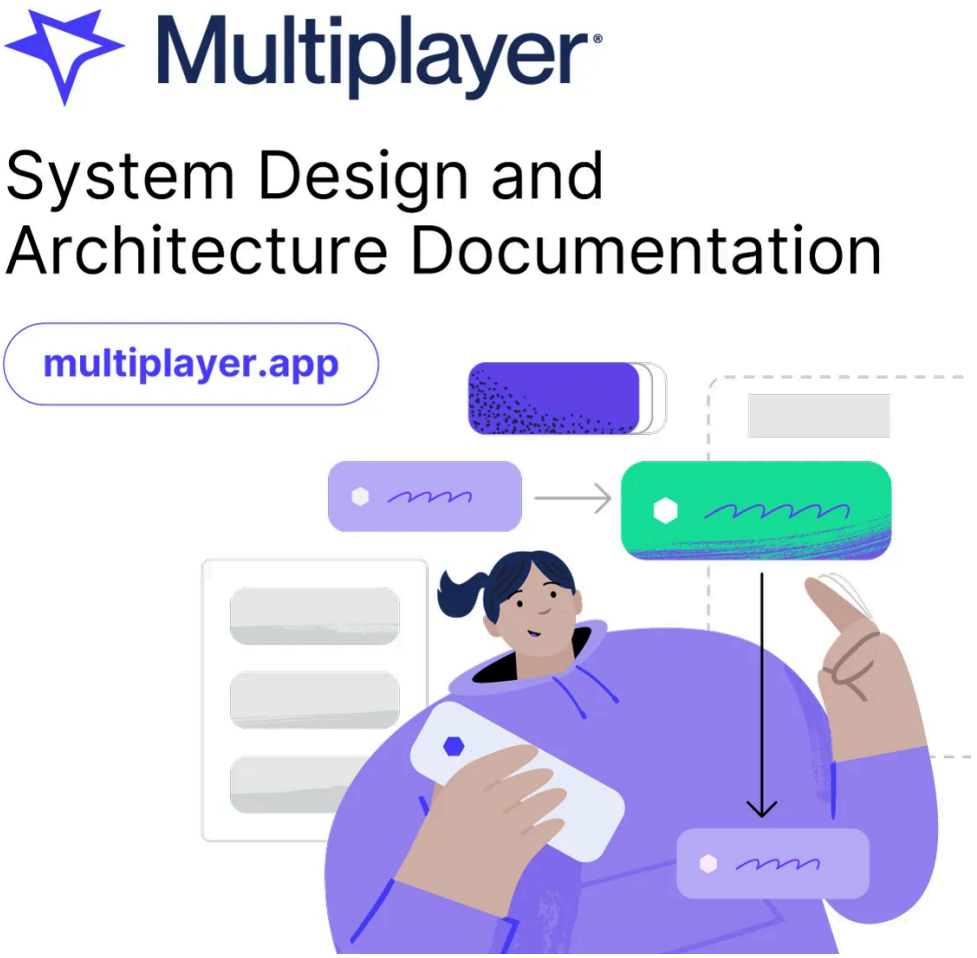
ASHISH PRATAP SINGH
OCT 17, 2024

System design can feel complex, but once you understand its **fundamental building blocks** and how to **stitch them together**, everything falls into place.

In this post, we're going to break down the **top 15 building blocks** of system design, every developer should know.

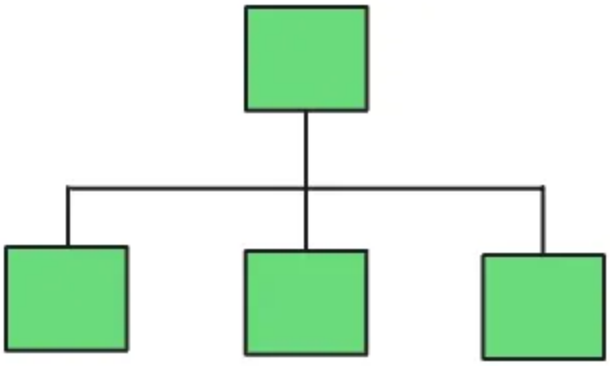
Knowing these will help you make sense of a large system and help you in answering your next system design interview problem.

📣 Design, develop and manage distributed software better (Sponsored)



[Multiplayer](#) auto-documents your system, from the high-level logical architecture down to the individual components, APIs, dependencies, and environments. Perfect for teams who want to speed up their workflows and consolidate their technical assets.

1. Load Balancers



Visualized using Multiplayer

Load balancers distribute incoming requests across **multiple servers** to ensure no single server bears too much load. It helps maintain **availability** and **reliability** by automatically rerouting traffic if a server fails.

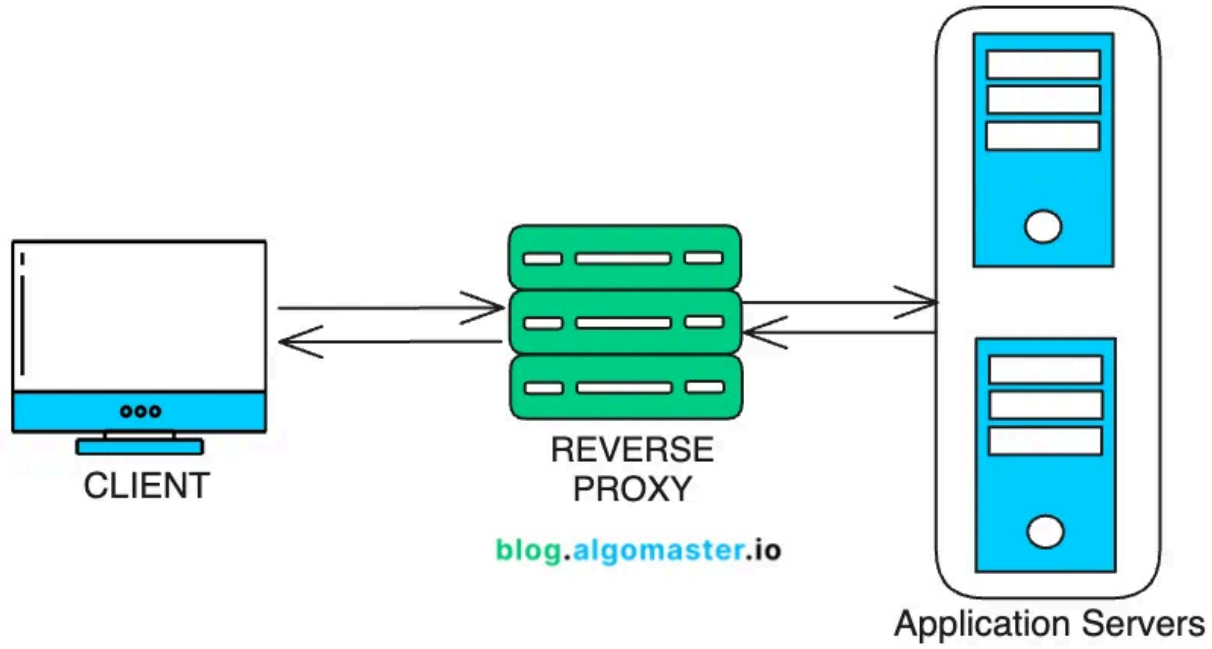
Use it when your application grows beyond the capacity of a single server and needs **horizontal scaling** to maintain performance and availability.

Types:

- **Layer 4 Load Balancers:** Operate at the transport layer (e.g., TCP, UDP).
- **Layer 7 Load Balancers:** Operate at the application layer (e.g., HTTP, HTTPS).

> **Examples:** Nginx, HAProxy, **AWS ELB**

2. Reverse Proxy



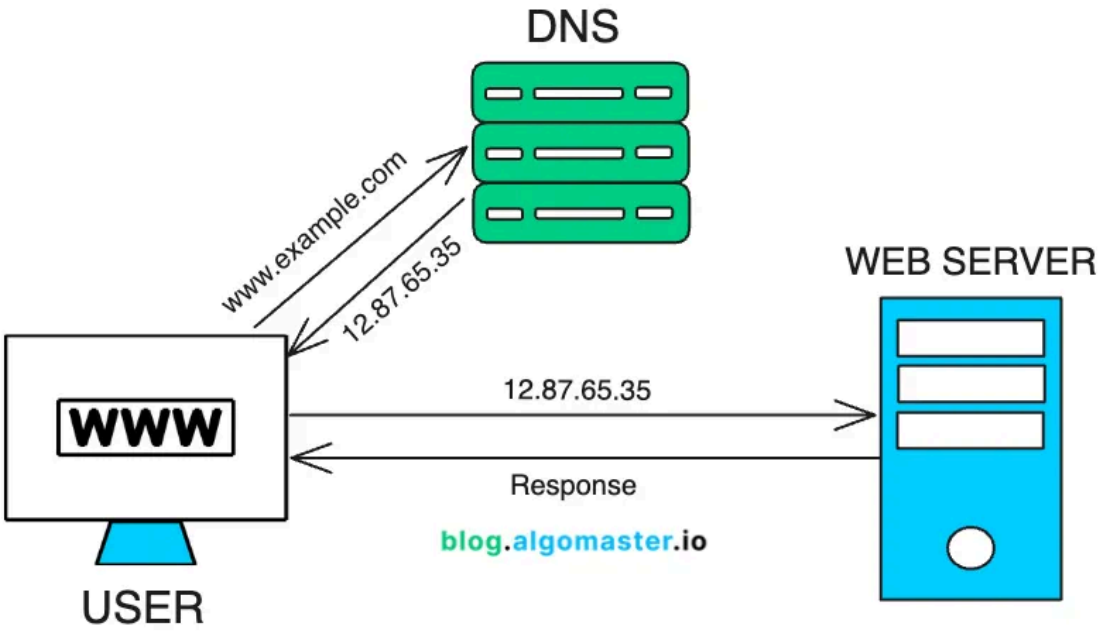
Visualized using Multiplayer

A **reverse proxy** acts as an intermediary between clients and servers. It forwards client requests to appropriate backend servers and then returns the server's response back to the client.

It **enhances security** by hiding the backend servers and **optimizes performance** through caching.

> **Example:** Cloudflare acts as a reverse proxy for websites, securing and accelerating content delivery by caching resources and blocking malicious traffic.

3. Domain Name System (DNS)



Visualized using Multiplayer

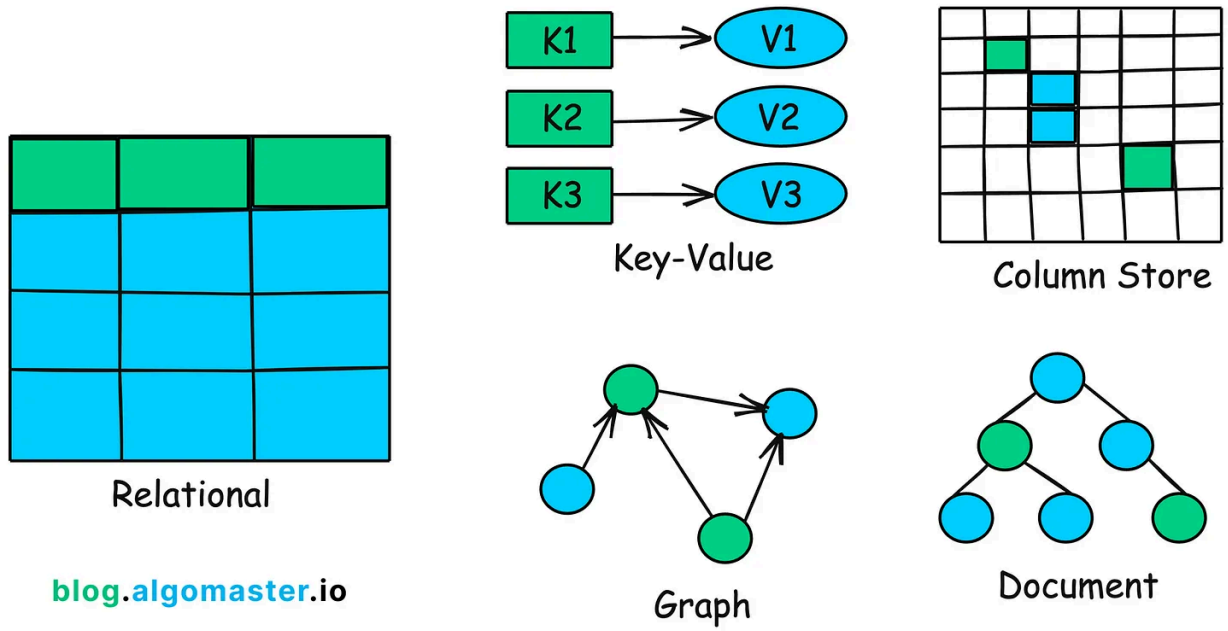
The **DNS** is like a phonebook for the internet.

It translates human-readable domain names (e.g., `example.com`) into IP addresses that computers use to identify each other on the network.

DNS allows users to access websites and services without needing the users to memorize numeric IP addresses.

> **Examples:** Route 53, Google Cloud DNS

4. Databases



Visualized using Multiplayer

Databases are the core storage units. They store **structured** or **unstructured** data that applications rely on for performing operations like queries, updates, and deletes.

Relational databases (SQL) like MySQL or PostgreSQL offer structured data storage with ACID properties, while NoSQL databases like MongoDB handle unstructured, high-volume data more efficiently.

> **Example:** Netflix uses Apache Cassandra, a NoSQL database, to store and retrieve massive amounts of user data and streaming logs.

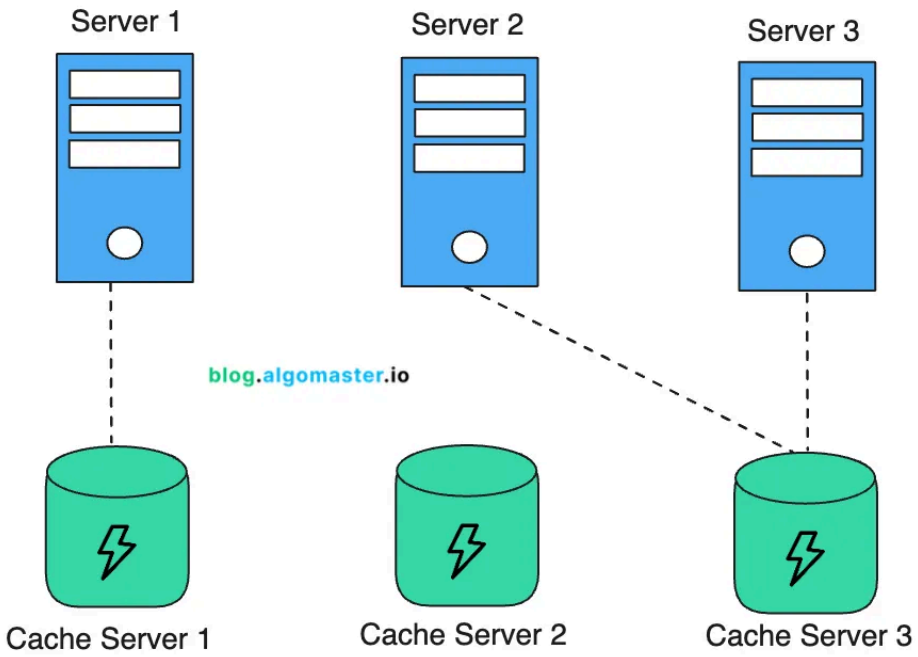
5. Blob Store

A **Blob** (**Binary Large Object**) store is a storage service designed for storing large volumes of unstructured data like images, videos, and backups that are typically difficult to manage in regular databases

Blob stores are perfect for applications that handle a large volume of media files, such as video streaming services (e.g., YouTube), image hosting platforms (e.g., Instagram), or content delivery networks (CDNs) that serve multimedia content.

> **Examples:** Amazon S3, Google Cloud Storage

6. Distributed Cache



Visualized using Multiplayer

A **distributed cache** stores frequently accessed data across multiple nodes, making it faster to retrieve and reducing the load on the primary database.

It's quite useful to scale read-heavy applications, or when certain pieces of data are frequently accessed by users, such as user sessions or product catalogs.

> **Examples:** Redis, Memcached

7. Content Delivery Network (CDN)

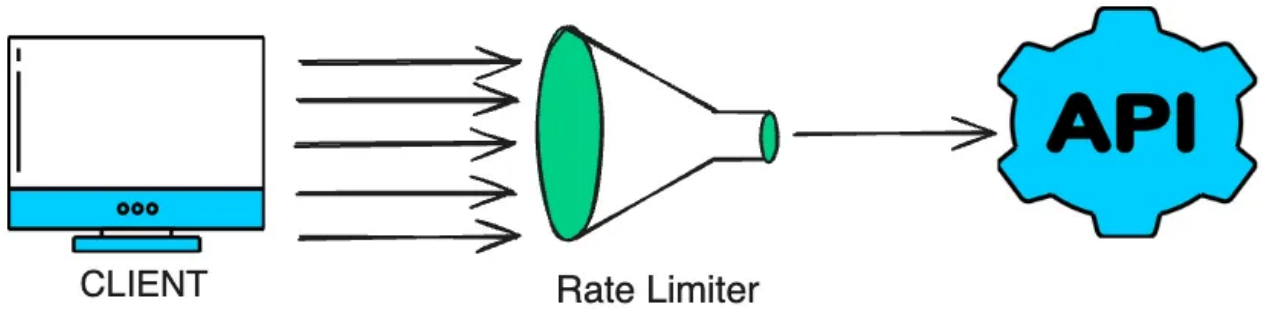
A CDN is a geographically distributed network of servers that deliver static and dynamic content (e.g., HTML pages, JavaScript files, images) to users based on their location.

By caching static content at edge servers closer to users, a CDN reduces latency and speeds up content delivery.

Netflix uses a CDN to serve video content from edge servers located near the user, reducing buffering and improving stream quality.

> **Examples:** Cloudflare, Akamai

8. Rate limiter



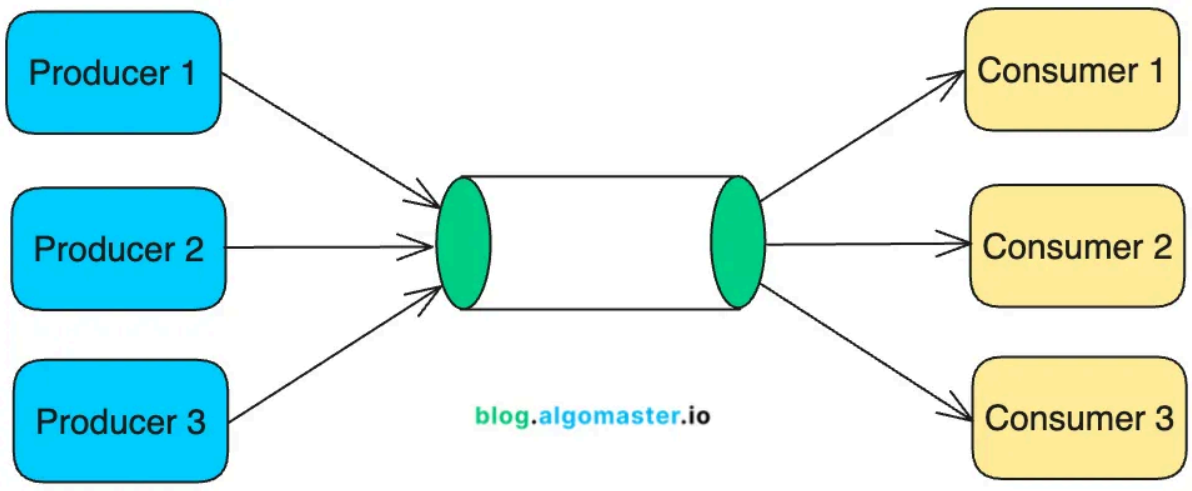
Visualized using Multiplayer

Rate limiters control the number of requests a user or system can make to a service over a specific period.

It's used to protect the system from overload due to malicious attacks (e.g., DDoS) or unintentional traffic spikes.

Popular services apply rate limits on it's APIs to control the number of requests a user/developer can make within a certain time frame.

9. Distributed Messaging Queues



Visualized using Multiplayer

A **distributed messaging queue** allows asynchronous communication between different parts of a system by sending messages between producers and consumers.

They help decouple services, ensuring that even if the consumer is unavailable, the producer can continue sending messages.

> **Example:** Uber uses Apache Kafka to manage millions of ride events in real-time, ensuring they are processed reliably and at scale.

10. Microservices

Microservices is an architectural style where an application is broken down into small, loosely coupled services that communicate over a network, often via HTTP or messaging queues.

It enables independent development, deployment, and scaling of different parts of an application.

> **Example:** Amazon uses a microservices architecture to manage different parts of their platform, such as payments, product search, and inventory management.

11. Distributed Unique ID Generator

A **distributed unique ID generator** is a service that creates globally unique identifiers in a distributed system without requiring centralized coordination.

These unique IDs are crucial for systems that operate across multiple servers or regions to avoid conflicts and ensure that every entity (like a user, transaction, or order) has a distinct identifier.

In traditional systems, a centralized database might generate auto-incrementing IDs, but in distributed systems, this approach doesn't scale well and can introduce bottlenecks.

Distributed ID generators solve this problem by enabling multiple systems to generate IDs independently, without collisions.

> **Example:** Twitter's Snowflake generates unique IDs for each tweet in a globally distributed system.

12. Distributed Task Scheduler

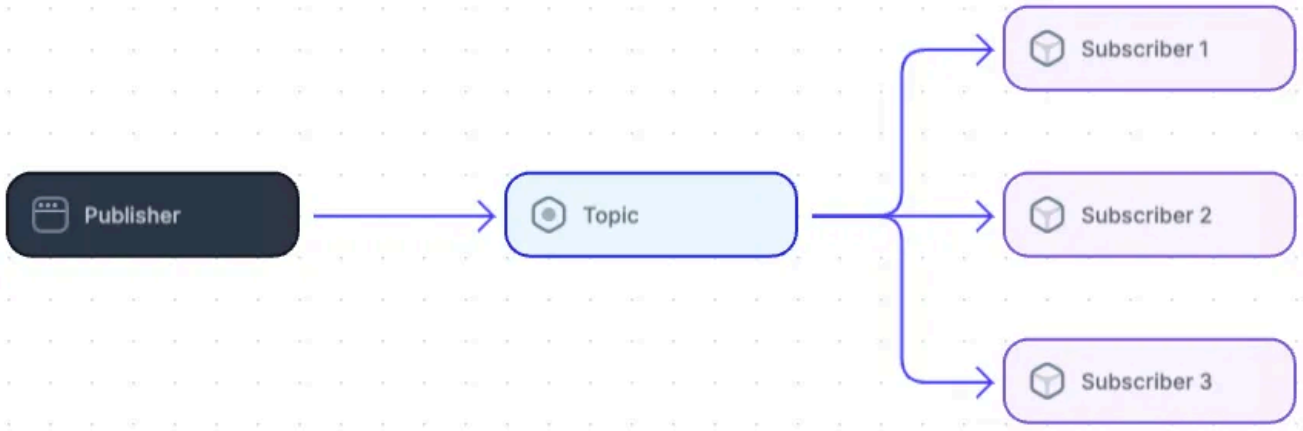
A **distributed task scheduler** coordinates and manages task execution across multiple nodes. It ensures that scheduled tasks are executed on time, even in a distributed environment.

It helps break down a system's workload into individual tasks and distributes them across different nodes. The scheduler ensures that each task is completed by keeping track of their execution, monitoring for failures, and retrying tasks if necessary.

It's commonly used to schedule and execute periodic batch processing tasks like data backup, ETL, report generation etc..

> **Examples:** Apache Airflow, Celery, AWS Step Function

13. Pub-Sub System



Created using Multiplayer

A **publish-subscribe** (**pub-sub**) system allows **one-to-many** communication, where publishers send messages to multiple subscribers without knowing who they are.

It's widely used for decoupling event producers and consumers.

Whenever a new message is published, all subscribers receive it simultaneously, enabling real-time communication.

> **Examples:** Amazon SNS, Google Cloud Pub/Sub

14. Distributed logging system

A **distributed logging system** aggregates logs from multiple services and servers into a central location, making it easier to monitor, analyze, and troubleshoot distributed systems.

Uber operates a complex microservices architecture that generates an enormous volume of logs. They use **ELK Stack** (Elasticsearch, Logstash, and Kibana) to collect, store, and analyze logs from their services globally.

> **Examples:** Elastic Stack (ELK), Splunk

15. Monitoring systems

Monitoring systems track the health, performance, and uptime of applications.

They collect metrics like CPU usage, memory consumption, disk I/O and response times to ensure the system operates efficiently.

You should always use monitoring systems in production environments to ensure your system runs efficiently and issues are addressed as soon as they arise.

> **Examples:** Prometheus, Grafana

Hope you enjoyed reading this article.

If you found it valuable, hit a like ❤️ and consider subscribing for more such content every week.

If you have any questions or suggestions, leave a comment.

This post is public so feel free to share it.

Subscribe for free to receive new articles every week.

Checkout my **Youtube channel** for more in-depth content.

Follow me on **LinkedIn**, **X** and **Medium** to stay updated.

Checkout my **GitHub repositories** for free interview preparation resources.

I hope you have a lovely day!

See you soon,

Ashish