System Design

Archana Goli

Let's consider a live streaming system

Likes and comments from users are essential elements in the design of a live streaming system. Because each encounter is given a unique ID and timestamp, effective tracking and management are made possible. The requirements document records these exchanges, which are then converted into data definitions and assigned to database objects.

The system uses endpoint URLs to retrieve this data. Users can interact with the system in a seamless manner because these APIs encapsulate the data. To efficiently access video data, for example, click the getVideo endpoint. To make these exchanges possible, the APIs make use of network protocols like gRPC or HTTP.

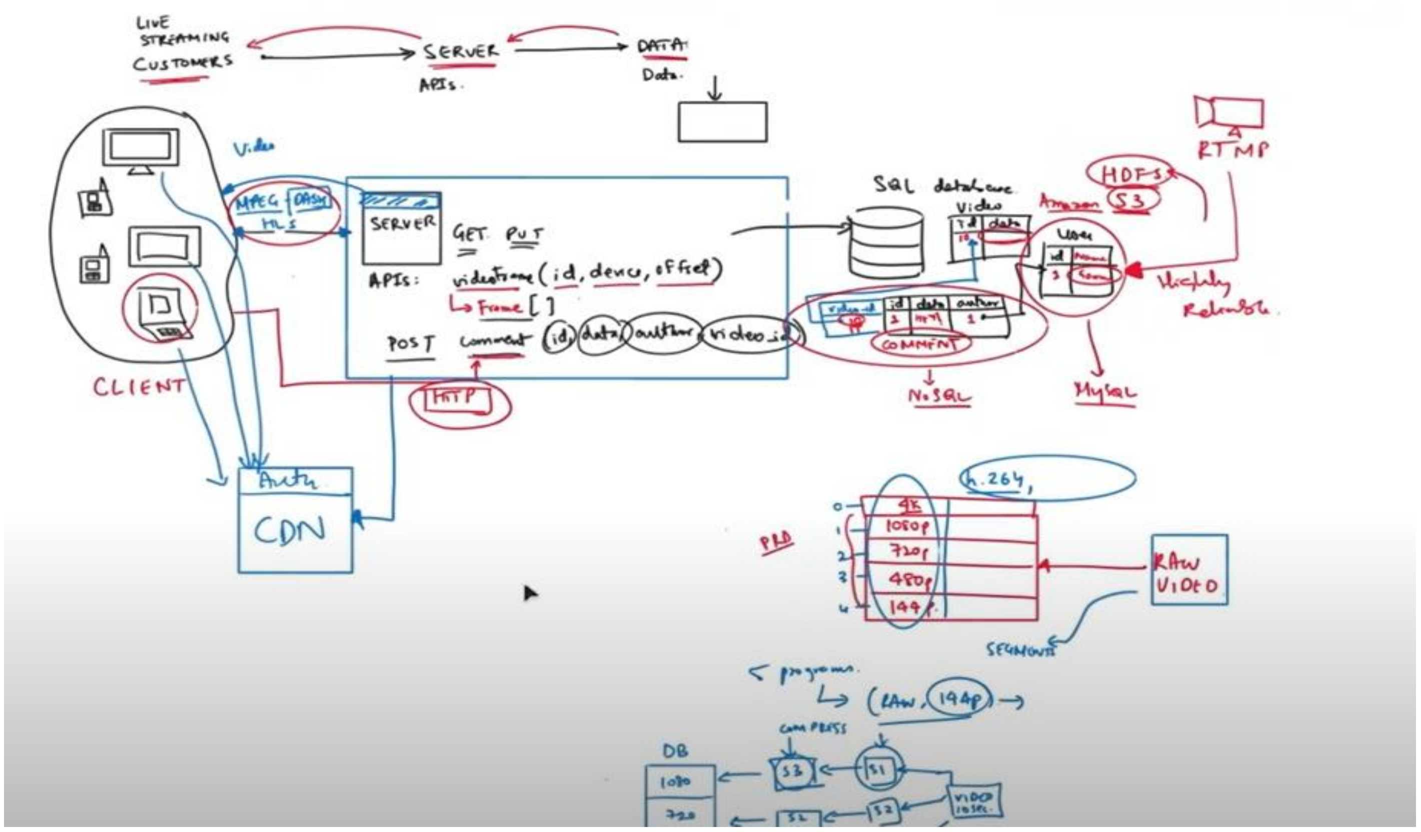
The architecture must account for potential failures, including database crashes or firewall problems, in order to guarantee stability and scalability. To improve system resilience, design concepts like fault tolerance and modularity should be used. The program may adjust to changing requirements by implementing edge case testing, assuring long-term sustainability and user happiness.

Additionally, raw video data is saved on the server and accessible to users with a single click, as opposed to being sent directly to consumers. By dynamically altering the quality dependent on network conditions, adaptive bitrate protocols like HTTP-DASH can optimize video streaming and enhance user experience.

What if the databases we are querying crashes? What if your firewall start blocking all your requests?

What if one piece of code starts misbehaving due to some bug in it?

We must use some design principles.



Efficient storage solutions are crucial for managing video files and related information in the server-side architecture of a live streaming system. For storing massive amounts of data, cloud storage solutions like Amazon S3 and HDFS (Hadoop Distributed File System) are ideal since they provide scalability and effective querying. Relational databases like MySQL or PostgreSQL are preferred for transactional data that requires intricate joins.

A NoSQL database like MongoDB, however, is perfect for handling massive volumes of unstructured data. It makes it possible for several data kinds to persist, including several writers and comments connected to a video. A distinct ID can be assigned to each movie, and relevant information can be easily retrieved by storing it as key-value pairs.

Streaming Protocol

Peer-to-peer streaming, made possible by technologies such as WebRTC, improves real-time communication and video delivery by enabling users to connect directly for live interactions. Both MPEG-DASH (Dynamic Adaptive Streaming over HTTP) and HLS (HTTP Live Streaming) are essential for adaptive streaming; HLS is especially designed for iOS and macOS devices, and both optimize video quality dependent on user bandwidth.

The HTTP Protocol

Lastly, to process user requests, the system makes use of the stateless HTTP protocol. This indicates that the server does not save session data between requests and that each request is independent. Because of this architecture, scalability is guaranteed because multiple users can be served by the server effectively without requiring the maintenance of separate user contexts.

We can use CDN to store and persist the static data. We can store web pages and some videos, so that client can directly access them. We need to write code to authenticate the user accessing these files and deploy the code on CDN.

Important points to be remembered:

- Define the requirements as the abstract concepts (objects)
- Objects can be manipulated and queried using APIs on server
- The data representation needs to be stored in the databases

Once the layout of system design is done, we think about

- what kind of network protocols can be used
- What kind of databases can be used
- What are the design patterns (message queues, load balancers) can be used

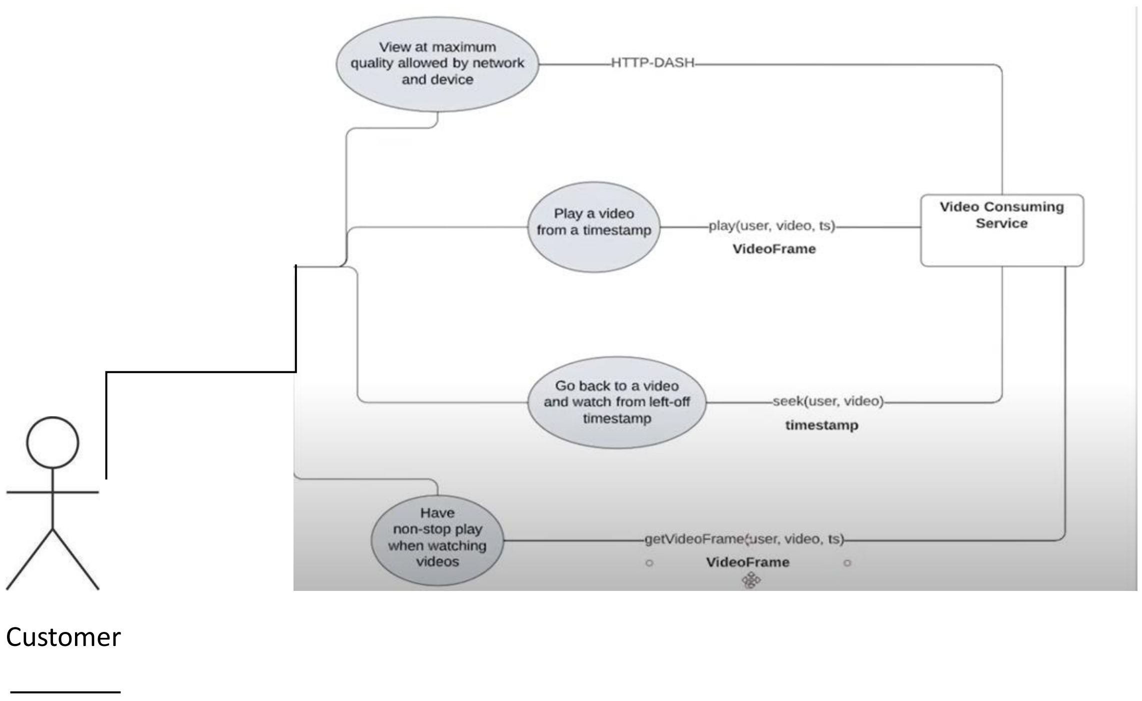
Low Level Design:

In case of high-level design,

- the client sends the requests to the API gateway
- API gateway forwards requests to multiple services
- The services send the requests to the data bases

But coding all these things and documenting them together is not an easy task. So, divide the entire system into the small chunks and work on each chunk which is called as a low-level design.

Let's consider a use case which describes the services to be provided by the system to the


View at maximum quality allowed by network and device
HTTP-DASH
Play a video from a timestamp
play(user, video, ts)
VideoFrame
Video Consuming Service
Go back to a video and watch from left-off timestamp
seek(user, video)
timestamp
Have non-stop play when watching videos
getVideoFrame(user, video, ts)
VideoFrame
Customer

Buffering and Video Frames
An uninterrupted stream of video is created by stacking collected frames in a live streaming system. Buffering the frames is a step in this procedure that guarantees seamless playing. A player can load enough content in advance to avoid pauses when streaming by using buffering, which temporarily stores video data. Because data is constantly sent into the video player, viewers are guaranteed a flawless experience.

The protocol for adaptive bitrates
Video streams are dynamically resized by an adaptive bitrate protocol according to the user's device capabilities and internet speed. Because of the system's ability to dynamically adjust bitrates and resolutions to match available bandwidth, buffering is reduced and user experience is enhanced, resulting in a fluid watching experience.

HTTP-DASH
The Operation of HTTP-DASH Inside: Video files are divided into manageable parts by the HTTP Dynamic Adaptive Streaming (DASH) protocol, which enables clients to request just the portions they require. There are several bitrates and resolutions for each part. The client keeps an eye on the state of its network and chooses the suitable section according to available bandwidth. DASH allows adaptive streaming without the need for user participation by using manifest files (MPD files) to offer information about the available segments.

Class diagram

The structure of the system is visually represented by a class diagram, which displays classes along with their properties, methods, and connections. Key classes for an HTTP-DASH live streaming system could be:
Video: Contains properties such as ID, title, and duration to represent video material.
Segment: A representation of a single video segment that includes bitrate and resolution information.

The player controls bitrate selection, buffering, and playback.
NetworkManager: Communicates with the Player and keeps an eye on the network's circumstances.

Sequence diagram

Object interactions inside a system are depicted by a sequence diagram. Inside the HTTP-DASH framework:
Someone makes a video request.
Requests for the MPD file from the Server are made by the Player first.
In response, the server sends the segment information-containing MPD file.
In light of the existing network conditions, the Player requests the first few video segments.
NetworkManager continually monitors bandwidth during video playback and modifies segment requests as necessary.

Considerations for Implementation
This system's implementation requires code to handle:
Logic for video frame buffering.
Using adaptive bitrate logic, several section resolutions can be selected.
API endpoints for segment delivery and video requests.
error management for playback pauses and network problems.
Integration for the production and delivery of segments using a video encoding library.

System design parts:

Load balancer: It is used to control the web traffic. It will perform the health check of available nodes and routes the traffic to the healthier nodes following round robin algorithm.

Where does load balancer can be placed?

- Can be placed between User and web server (UI)

- Can be placed between web server and internal server(Back end)
- Can be placed between between the internal server and databases

Types of Load balancers:

Hardware Loadbalacer:

Software loadbalancer: one of the software load balancer is HAProxy loadbalancer

Load Balancers: Algorithms

1. Round Robin - requests are assigned to the servers sequentially one by one
2. Round Robin with Waited server - once waited server is selected and once the normal server is selected
3. Least connections - Based on least number of server connections, server is selected
4. Least Response time - based the response time of the server, server is selected
5. Souce IP hash - Based on the request IP address, the hashing is applied on the IP address and server that handle is selected
6. URL hash - Based on the request URL, the hashing is applied on the URL and then server is selected

Cache:

Its like a short- term memory which has limited space but it is faster& contains most recently accessed items.

Cache can be used almost in every layer hardware, OS, web browser, web application but are often found nearest to the front end.

Cache Types:

- Application server chache - single node cache
- Distribute cache - multiple nodes
- Global cache - one single node central cache
- CDN - cache for static type of content

Cache invalidation:

When data is modified in DB, it should be invalidated in the cache.

For data consistency between a database and its cache to be maintained, cache invalidation is essential. The appropriate cache record needs to be invalidated whenever data is changed in the database to stop users from receiving outdated information. Users will always receive the

most recent info thanks to this approach.
Kinds of Validation for Caches
Write-Through Cache: With this approach, data is concurrently written to the database and the cache. The most recent data is always guaranteed to be in the cache thanks to this, while write operations may become more slowly.
Write-Around Cache: This approach only modifies the cache upon read; it writes data directly to the database. The first time data is accessed, it may cause cache misses even though it lessens cache pollution from rarely visited data.

Write-Back Cache: In this case, data is written to the cache and classified as dirty at first. Although efficiency is increased, there is a chance of data loss if the cache expires before the write-back takes place. This is because the cache writes the data back to the database at a later date.
Policies for Cache Evictions
The cache needs to remove certain data when it fills up in order to make place for more entries. Which data to erase depends on several eviction policies:

First in, first out, or FIFO, refers to the removal of the oldest entries first, regardless of usage.
The most recent additions are eliminated first, according to LIFO (Last In, First Out).
Less Recently Used (LRU): The entries that haven't been accessed in the longest period of time are removed.

Most Recently Used: The items that have been viewed the most recently are eliminated first.
Least Frequently Used: The least frequently accessed entries are removed.
Random Replacement: In certain situations, evicting a random entry offers a straightforward yet effective tactic.

Sharding methods
Data is divided into rows and dispersed over several tables or databases using horizontal partitioning.
Vertical Partitioning: For maximum performance, divide a table into smaller tables according to columns and distribute them throughout databases.
Directory-Based Partitioning: This method maps data to distinct shards by using a directory service.

Sharding Criteria
Hash-Based Partitioning: Provides an equitable distribution of data by using a hash function to identify the shard for each data entry.
List partitioning: Groups particular values into specified shards and divides data according to these lists.

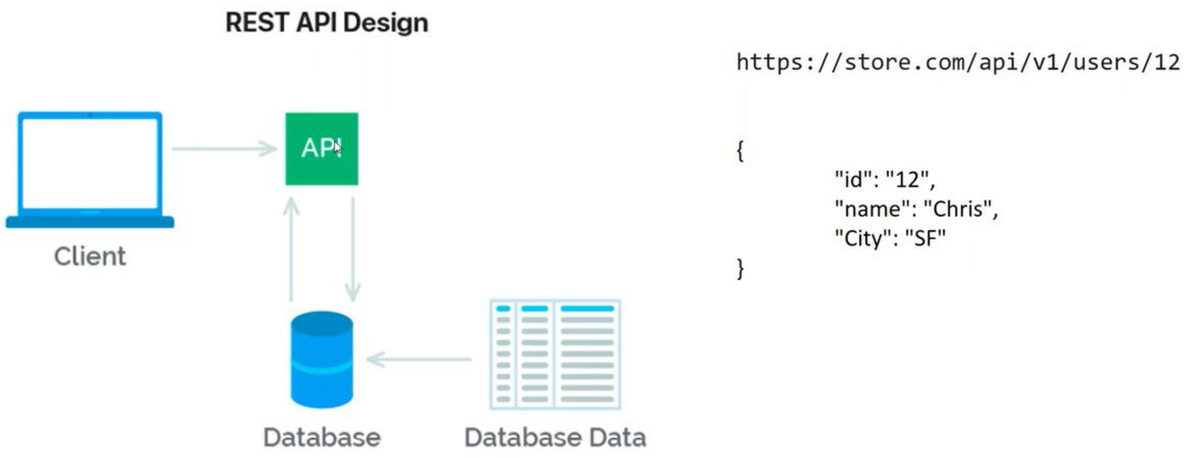
Round-Robin Partitioning: Ensures uniform load distribution by distributing data among shards in a sequential manner.
Composite Partitioning: Uses both main and secondary methods to combine numerous criteria for data distribution.

Sharding Challenges
In a sharded system, ensuring that all database transactions are Atomic, Consistent, Isolated, and Durable—also known as ACID compliance—may become increasingly difficult.
Inefficient Joins: Because data may reside on separate servers and require additional network calls, performing joins across shards may cause performance concerns.

Indexes
Data structures called indexes let databases retrieve information more quickly. They lessen the requirement for full table scans by offering rapid access channels to data, enabling speedier searches. Especially for big datasets, well-designed indexes can greatly improve query performance.

Scaling
Increasing the number of servers to spread the database load and strengthen the system's capacity to manage greater traffic is known as horizontal scaling.
Vertical scaling: Increasing the capacity of the current server (for example, by adding additional RAM or CPU) might enhance performance but has physical constraints.

RESTful API
A web service that follows the REST (Representational State Transfer) principles allows users to interact with web resources using the normal HTTP methods (GET, POST, PUT, DELETE). Since REST APIs are stateless, every client request must include all required data in order for the server to process and respond to it. They are frequently employed in the development of adaptable and scalable web applications.



```
https://store.com/api/v1/users/12

{
        "id": "12",
        "name": "Chris",
        "City": "SF"
}
```

A service is a RESTful service, if it follows following:

- Uniform interface
- Stateless
- Cachable
- Client server
- REST permits different data formats such as plain text, HTML, XML, JSON

Proxy Server: A proxy server facilitates communication between client devices and servers while providing security and anonymity. A user's request to access a web service travels via the proxy before being forwarded to the relevant server. The proxy forwards the response back to the user after obtaining it. Different kinds of proxies exist:

Forward proxies are used to send client requests to servers outside of the company. Frequently used for anonymous browsing and content filtering.
Reverse proxies: Placed in front of servers to control SSL encryption, cache content, and balance load. By hiding the user's IP address, blocking harmful content, and enhancing speed through caching, proxies increase security.

Message Queue: One way to facilitate asynchronous message transfer between various components in distributed systems is through the usage of message queues. A producer queues up messages, which a recipient then reads and processes. Services can now function independently of one another because to this decoupling. The two primary patterns are:

Single transmitter, single recipient in a point-to-point (queue) system.
On a topic, publish-subscribe Several receivers receive broadcast messages. RabbitMQ, Kafka, and ActiveMQ are a few popular message queues. Their ability to store and process messages temporarily, even in the event that some system components are unavailable, ensures system reliability.

Selecting a Database: SQL vs. NoSQL
The type of data you have and your scalability requirements should be taken into account while deciding between SQL and NoSQL databases. Relational and structured SQL databases (such as MySQL and PostgreSQL) use tables with rows and columns. They are appropriate for transactional systems where data integrity is vital because they uphold the ACID qualities (Atomicity, Consistency, Isolation, Durability). Unstructured or semi-structured data storage is possible with NoSQL databases, such as MongoDB and Cassandra, because they are flexible and non-relational. NoSQL databases are incredibly scalable, especially when it comes to distributed, high-speed data; yet, depending on the use case, they might not be as well-suited for complicated queries or joins.

Microservices vs. Monolithic Architecture A monolithic architecture entails creating a single, cohesive codebase for the whole program. Tight coupling between all functionalities, including UI, data management, and business logic, makes development easier but presents maintenance and scaling issues. It gets harder to implement new features and make modifications as the application gets larger. On the other hand, the application is divided into smaller, loosely linked services using microservices design. Every service manages a particular business task and uses APIs to connect with other services. Although this modularity makes it possible for services to be developed, scaled, and deployed independently, it also makes maintaining data consistency and inter-service communication more difficult.

Hashing
Hashing is a technique that is typically used for security or indexing reasons to transform input data (such a password or document) into a fixed-length string of characters. An input is fed into a hash function, which generates a distinct, deterministic output (hash). The hash produced by the same input is always the same. But since hashing is one-way, it is impossible to deduce the original input from the hash in reverse. Typical usage cases include of:

Data integrity is the assurance that no files have been altered.
Keeping hashed passwords on file for safe authentication is known as password protection. Two well-liked algorithms are MD5 and SHA-256.

Consistent Hashing
In distributed systems, consistent hashing is a method for distributing data uniformly among several nodes with the least amount of disturbance possible upon the addition or removal of nodes. Consistent hashing uses a ring structure to overcome the issue of large-scale data rehashing that arises when changes are made to the system, such as adding a new server. When a node changes, just a small amount of the data is remapped. For load balancing and fault tolerance, distributed databases and caching systems like Amazon Dynamo and Apache Cassandra frequently employ this technique.

Kafka

The Kafka Empire Kafka is an open-source distributed streaming framework made for real-time, high-throughput data pipelines. According to Kafka's publish-subscribe paradigm, data producers send messages to topics, and consumers subscribe to these topics in order to view the data. Kafka is composed of three primary parts:

Producer: Disseminates messages on Kafka-related subjects.
Broker: Oversees the distribution and storage of messages among several servers.
Consumer: Examines messages pertaining to various subjects. Because of its great scalability and fault tolerance, Kafka is perfect for log aggregation, event-driven architectures, and real-time analytics.

LRU Cache Eviction Policy: By eliminating the items that haven't been accessed in a long time, the LRU cache eviction policy helps manage the limited cache space. Data that has been accessed lately is more likely to be accessed again shortly, according to the LRU algorithm. It keeps track of how long each item is accessed, and when the cache fills up, the item that hasn't been used in the longest gets removed. Systems that require cache hits and quick data retrieval, such web applications, database management systems, and operating systems' memory management, frequently employ LRU.

Hadoop Apache
An open-source framework called Apache Hadoop was created to manage distributed clusters of commodity hardware used for large-scale data processing and storing. The main parts of Hadoop are as follows:

Hadoop Distributed File solution, or HDFS, is a distributed storage solution that redundancies the way massive datasets are stored across several servers.
MapReduce is a programming paradigm that uses distributed, parallel algorithms to process and create large data collections. In settings like data warehousing and analytics, Hadoop is frequently used for batch processing of enormous volumes of unstructured or semi-structured data because of its great fault tolerance and scalability.

HDFS

Large datasets can be distributedly stored across clusters using Apache Hadoop's Hadoop Distributed File System (HDFS), a crucial component. With the purpose of providing fault tolerance, HDFS replicates blocks of data before distributing them among cluster nodes. With its design to manage huge files written once and read numerous times, HDFS is excellent for batch processing. Applications like distributed data processing and large data analytics are a perfect fit for its high throughput access support. Data is replicated across several nodes to enhance fault tolerance, allowing for data recovery from a failed node.

HBase
Built on top of Hadoop, HBase is a distributed, scalable NoSQL database with the ability to manage massive volumes of sparse data. HBase can accommodate millions of rows and columns over thousands of nodes since it stores data in a column-oriented structure, in contrast to standard relational databases. It is frequently used in applications needing quick lookups and access to unstructured data since it allows for random, real-time read/write access to large data. Use cases where high availability and horizontal scalability are crucial, like real-time analytics, messaging systems, and large-scale web services, are well suited for HBase.

ZooKeeper: The ZooKeeper Apache Coordination, configuration management, synchronization, and naming services are all provided by ZooKeeper, a centralized platform for managing distributed systems. ZooKeeper keeps track of distributed application configurations, facilitating effective node-to-node communication and synchronization. It ensures that only one node does a specific operation at a time by providing an abstraction of locks and barriers. ZooKeeper is widely used in systems such as Kafka, Hadoop, and HBase. It is essential for managing distributed locks, electing leaders in distributed contexts, and preserving data consistency.

Solr: Apache Solr Developed on top of Apache Lucene, Solr is an open-source search engine optimized for indexing and searching massive datasets. Due to its capability for distributed searching, faceted search, hit highlighting, and full-text search, Solr is a well-liked option for enterprise-level search applications. Strong features like fault tolerance, load balancing, and real-time indexing are provided, guaranteeing excellent availability and dependability. Due to its scalability and strong query capabilities, Solr is frequently used in applications that need search capabilities, such as content management systems, e-commerce platforms, and log analysis tools.

Apache Cassandra: Apache Cassandra is a distributed NoSQL database that is highly scalable and intended for managing massive amounts of data over numerous commodity machines. Its peer-to-peer, decentralized architecture removes single points of failure and offers partition tolerance and high availability. Cassandra can support high write throughput with low latency because it uses a column-family data model. Applications needing horizontal scaling, such real-time data analytics, social media platforms, and Internet of Things systems, frequently employ it because it allows eventual consistency, which means updates are transmitted to all nodes asynchronously.

How a System Is Designed: Determining the architecture, parts, and data flow of a system to satisfy functional and non-functional requirements is the process of system design. The crucial actions consist of:

Identifying the issue and the use cases for the system is known as requirement gathering.
Choosing a Design: Deciding between serverless, microservices, and monolithic designs.
Selecting relational (SQL) or non-relational (NoSQL) databases according to the data structure for data storage.
Scalability: The ability to accommodate growth by using load balancing, caching, and horizontal or vertical scaling.
Including replication, redundancy, encryption, and authentication to ensure security and fault tolerance. Performance, scalability, and maintainability are guaranteed by a well-designed system.

Construct a URL Shortener: Long URLs can be shortened with a URL shortener into short, distinct keys that lead users back to the original URL. Billions of URLs must be handled by the system with efficiency. Essential elements:
Database: Short URL to original URL mappings can be stored in a NoSQL database such as Redis or MongoDB. SQL can also be used to create unique keys when combined with an auto-incrementing ID.
Hashing: To create short URLs with a unique ID, use Base62 encoding or a hash function.
Scalability: To guarantee that the system can withstand heavy traffic, use caching and load balancing. Link expiration, statistics, and custom URLs are extra features.

Pastebin Design: Text snippets can be shared and stored on Pastebin. Important elements consist of:

Data Storage: To store pastes with metadata, such as expiration time, use a relational database like MySQL or NoSQL.

Pastebin Design Users can save and distribute text or code snippets with distinct URLs using Pastebin. Important components of the design include:

Database: Text, metadata (such as creation and expiration times), and access logs can all be stored using SQL or NoSQL. An ID is given to each paste so that it may be retrieved.

Generate a Unique URL: Create brief, one-of-a-kind IDs by encoding Base62 or using hash algorithms.

Scalability: To manage heavy traffic, use load balancing and caching (such as Redis).

Make sure that data encryption, paste expiration, and access control are in place. User registration and syntax highlighting for coding are extra features.

Design Dropbox: Dropbox is a synchronization and file storage tool that lets users upload, distribute, and view files on several devices. Important elements consist of:

Storage: Store files on decentralized storage platforms such as Amazon S3, and keep track of metadata in relational or NoSQL databases.

Synchronization: Changes are distributed among devices and file updates are monitored through event-based systems.

File versioning: Maintain a history of changes to enable rollbacks.

Security: Access to files is restricted by user authentication, and files are encrypted both at rest and during transmission (TLS/SSL). Load balancing, sharding, and replication provide scalability and reliability.

HTTP vs HTTPS: The protocols used to send data between clients and servers are HTTP (Hypertext send Protocol) and HTTPS (HTTP Secure). Security is where the biggest distinction is. Since HTTP transfers data in plain text, it is susceptible to interception. HTTPS, on the other hand, encrypts data during transmission using SSL/TLS to guarantee its secrecy and integrity. When conducting important operations, such online banking or e-commerce, where security is crucial, HTTPS is frequently employed. Additionally, HTTPS offers trust signals (such as the browser lock icon) and aids in search engine ranking.

The CAP Theorem: Consistency, Availability, and Partition Tolerance are the three properties that a distributed system can only accomplish concurrently, according to the CAP theorem.

All nodes view the same data simultaneously, which is known as consistency.

Availability: If a few nodes go down, the system still functions.
In the event of a network partition or a node's inability to communicate with one another, the system maintains its functionality. Traditional SQL databases concentrate on consistency and availability, whereas NoSQL databases, such as Cassandra, prioritize availability and partition tolerance. What is required for fault tolerance and real-time consistency in the system will determine which option to use.