



Published in Dev Genius

You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)

Open in app ↗

Sign up

Sign In



Search Medium



Save



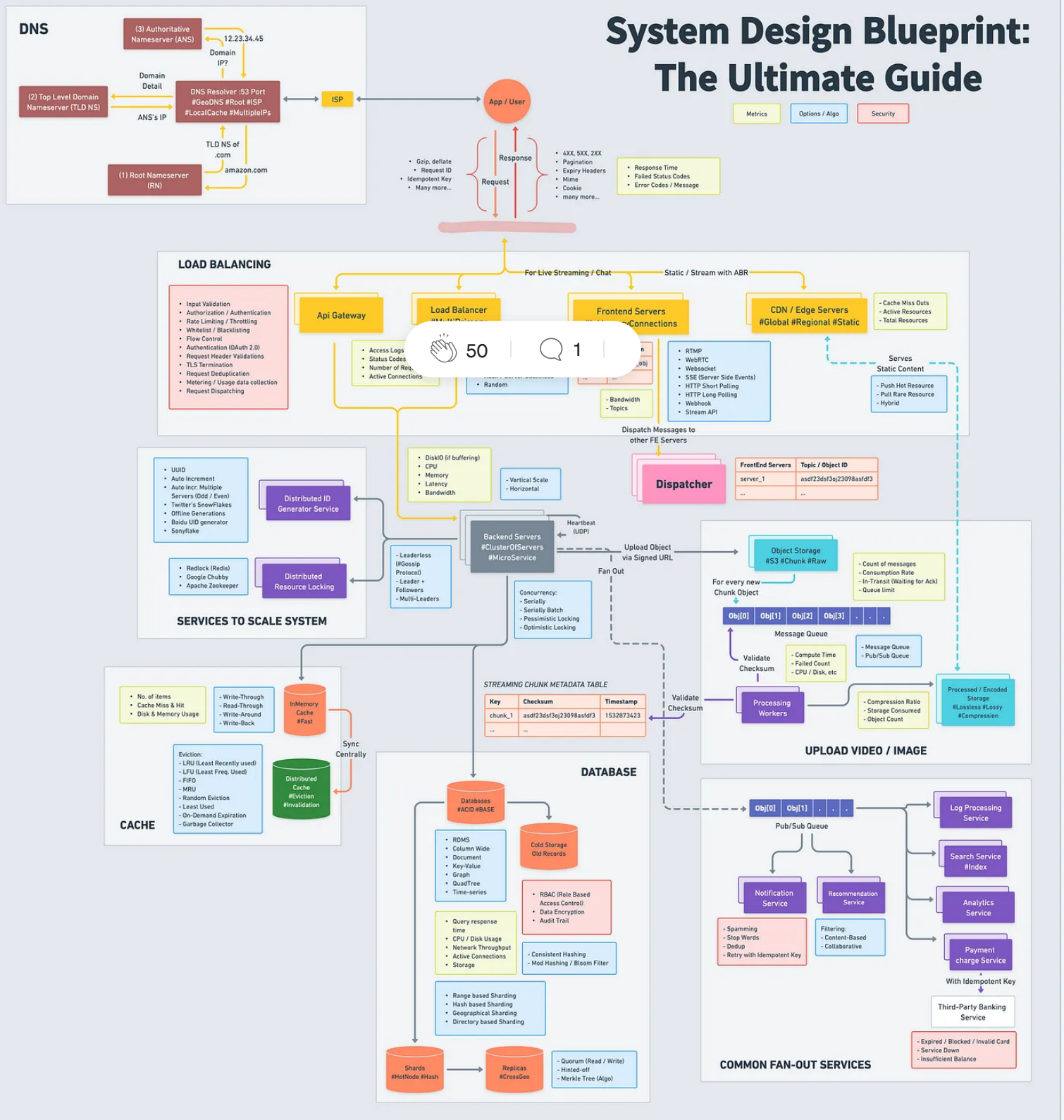
System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the process more manageable. In this blog post, we'll explore essential system design components such as DNS, load balancing, API Gateway, and more, along with a concise cheat sheet that can help developers design systems of varying complexity.

System Design Blueprint / Cheatsheet

A comprehensive visual guide that provides developers with a quick and easy reference to key concepts and best practices in system design. This handy cheatsheet or blueprint covers essential topics such as DNS, load balancing, API Gateway, video and image handling, caching, databases, unique ID generation, standard components like payment and recommendation services, and chat and streaming protocols. With this valuable resource at your fingertips, you'll be well-equipped to tackle the challenges of designing and implementing scalable, efficient, and reliable systems.

System Design Blueprint: The Ultimate Guide



System Design Blueprint: The Ultimate Guide — Full Resolution: <https://bytebyte-go.s3.amazonaws.com/System+Design+Blueprint.pdf>

Section 1: Principles of System Design

1.1: Modularization

Dividing the system into smaller, manageable modules help reduce complexity, improve maintainability, and increase reusability.

1.2: Abstraction

Hiding the implementation details and showing only the essential features helps simplify complex systems and promote modularity.

1.3: Layering

Organizing the system into layers, each layer providing a specific set of functionalities promotes the separation of concerns and enhances maintainability.

1.4: Scalability

Design systems to handle the increased load by adding more resources (horizontal scaling) or optimizing the system's capacity (vertical scaling).

1.5: Performance

Optimizing the system's response time, throughput, and resource utilization is crucial for a successful design.

1.6: Security

Ensure the system's confidentiality, integrity, and availability by implementing proper security measures and practices.

1.7: Fault Tolerance and Resilience

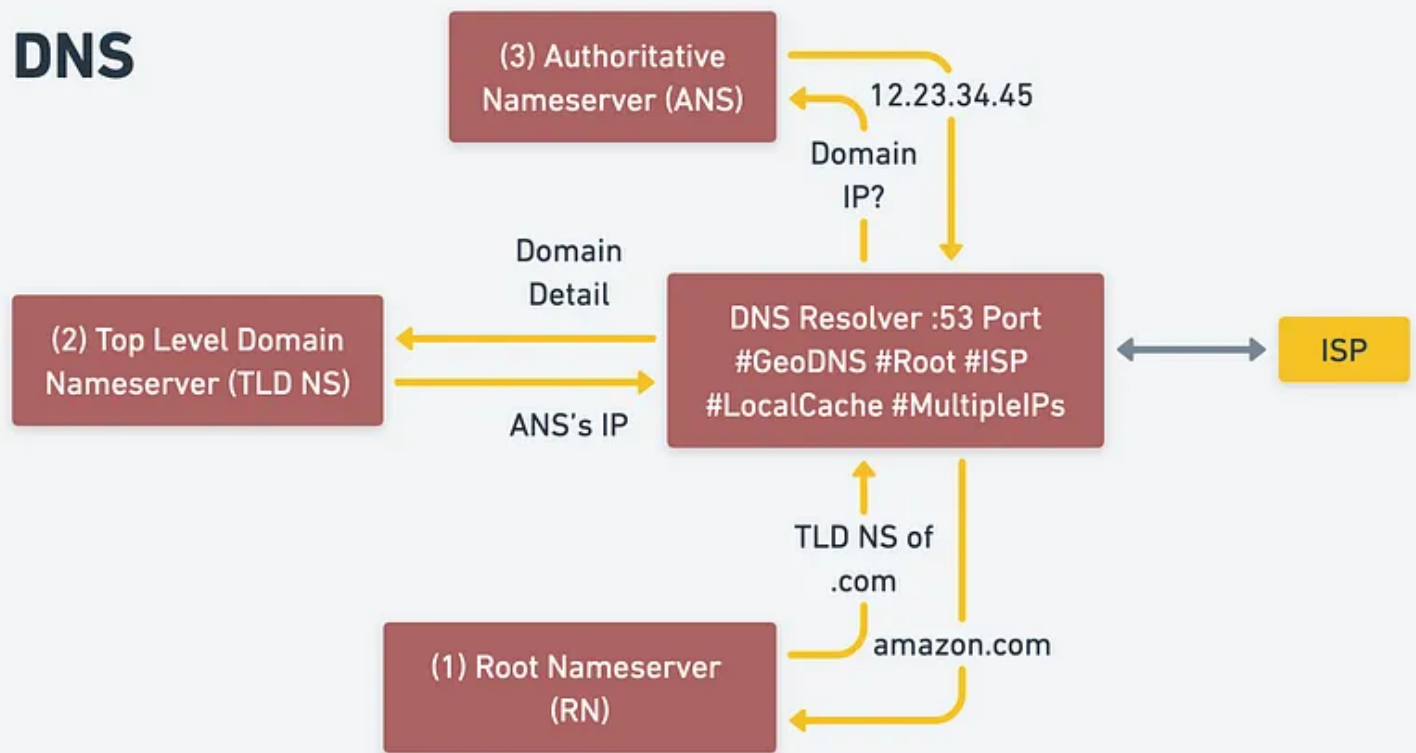
Design systems to withstand failures and recover gracefully from errors, ensuring reliability and availability.

Section 2: Key Components of System Design

2.1: DNS (Domain Name System)

DNS is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It translates human-readable domain names (e.g., www.example.com) into IP addresses, allowing users to access websites and services more efficiently. [Read More](#)

DNS



2.2: Load Balancing

Load balancing refers to distributing network traffic across multiple servers to ensure no single server is overwhelmed. This approach improves the system's availability, reliability, and performance. Standard load balancing algorithms include Round Robin, Least Connections, and IP Hash. [Read More with Cheatsheet](#)

2.3: API Gateway

An API Gateway is a server that acts as an intermediary between clients and microservices in a distributed system. It manages and routes requests, enforces security policies, and may provide additional features such as caching, logging, and monitoring.

2.4: Content Delivery Network (CDN)

A CDN is a network of servers distributed across various locations, designed to serve content to users with lower latency and higher bandwidth. CDNs cache content on edge servers close to end-users, improving the system's performance and reducing the load on origin servers. [Read More with Cheatsheet](#)

2.5: Message Queue

Message queues facilitate communication between distributed system components by temporarily storing messages in a queue. They enable asynchronous processing and help decouple components, improving the system's scalability and fault tolerance. [Read more](#)

2.6: Communication Protocols

Different communication protocols are used in system design, such as HTTP/HTTPS, WebSocket, and gRPC. These protocols have advantages and trade-offs, and the choice depends on factors like latency, security, and data transmission requirements.

2.7: Cache

Caching is a temporary technique used to store copies of data, allowing for faster retrieval in future requests. It helps reduce latency, server load, and bandwidth consumption. Popular caching mechanisms include in-memory caching, distributed caching, and browser caching. [Read more with Cheatsheet.](#)

2.8: Database

Choosing the appropriate database for a system depends on data structure, scalability, consistency, and latency. Common database types include relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Cassandra), and NewSQL databases (e.g., Cockroach DB, Google Spanner).

2.9: Replication Techniques

Replication is maintaining multiple copies of data across different nodes for increased reliability, availability, and fault tolerance. Standard replication techniques include synchronous replication, asynchronous replication, and semi-synchronous replication.

2.10: Distributed Unique ID Generation

Creating unique identifiers in a distributed system can be challenging but is essential for maintaining data consistency and integrity. [Read more with Comparison Table](#)

Section 3: Uploading Videos and Images in Chunks Using Signed URLs

In this section, we'll explore how to upload large video and image files in chunks using signed URLs. This method can significantly improve the efficiency and reliability of file uploads, especially in scenarios where network conditions are less than ideal.

3.1: What are Signed URLs?

Signed URLs are specially-crafted URLs that grant temporary, secure access to a specific resource, such as an object in cloud storage. These URLs contain an authentication signature that allows the user to perform a particular action, such as uploading or downloading a file, for a limited period. Popular cloud storage providers like Amazon S3 and Google Cloud Storage support generating signed URLs. Here's an example of how a signed URL might look:

```
https://example-bucket.s3.amazonaws.com/my-file.txt?\nX-Amz-Algorithm=AWS4-HMAC-SHA256&\nX-Amz-Credential=AKIAIOSFODNN7EXAMPLE%2F20220407%2Fus-east-1%2Fs3%2Faws4_request&\nX-Amz-Date=20220407T123456Z&\nX-Amz-Expires=3600&\nX-Amz-SignedHeaders=host&\nX-Amz-Signature=a9c8a7d1644c7b351ef3034f4a1b4c9047e891c7203eb3a9f29d8c7a74676d88
```

3.2: Uploading in Chunks

Uploading large files in a single request can lead to timeouts, high memory consumption, and an increased risk of failure due to network instability. Instead, breaking large files into smaller chunks and uploading them sequentially or in parallel can improve upload efficiency and reliability. This approach is known as “chunked” or “multipart” uploads.

3.3: Combining Signed URLs and Chunked Uploads

To upload video and image files in chunks using signed URLs, follow these general steps:

- 1. Divide the file into smaller chunks:** Split the large file into smaller chunks on the client side, typically using JavaScript. The chunk size can vary, but it's essential to balance the number of requests and the size of each chunk to optimize upload performance.
- 2. Request signed URLs for each chunk:** Send a request to your server to generate a signed URL for each chunk. The server should create a signed URL with the appropriate permissions and expiry time and return it to the client.
- 3. Upload chunks using signed URLs:** Using the signed URLs, upload each chunk to the cloud storage service. Depending on the desired concurrency level and network conditions, these uploads can be done sequentially or in parallel.
- 4. Confirm successful uploads and reassemble:** Once all chunks have been uploaded successfully, notify the server to confirm the completion of the upload process. The server can then reassemble the chunks into the original file and perform any necessary processing or validation.
- 5. Handle failed uploads:** If any chunk fails to upload, retry the upload using a new signed URL or implement an error-handling strategy to ensure a smooth user experience.

By using signed URLs and chunked uploads, developers can efficiently and securely handle large video and image uploads, improving the reliability and performance of their systems.

Section 4: Chat and Streaming Protocols

This section will discuss various chat and streaming protocols facilitating real-time communication and data streaming between clients and servers. Understanding these protocols can help developers build responsive and interactive applications.

4.1: RTMP (Real-Time Messaging Protocol)

RTMP is a proprietary protocol developed by Adobe Systems for streaming audio, video, and data over the Internet. It is commonly used in video streaming applications and provides low-latency communication between clients and servers. However, due to its reliance on the Flash Player, its popularity has waned in recent years.

4.2: WebRTC (Web Real-Time Communication)

WebRTC is an open-source project that enables real-time audio, video, and data communication in web browsers and mobile applications. It supports peer-to-peer connections, reducing latency and server load. WebRTC is widely used in video conferencing, online gaming, and other applications requiring real-time communication.

4.3: WebSocket

WebSocket is a communication protocol that enables bidirectional, full-duplex communication between a client and a server over a single, long-lived connection. Due to its low latency and efficient communication capabilities, WebSocket is often used for real-time applications like chat, notifications, and live updates.

4.4: SSE (Server-Sent Events)

Server-Sent Events (SSE) is a technology that enables servers to push client updates over an HTTP connection. It is designed for one-way, real-time communication from the server to the client, making it suitable for applications like live updates, news feeds, and notifications.

4.5: HTTP Short Polling

Short polling involves clients repeatedly sending HTTP requests to the server to check for new updates. While simple to implement, short polling can result in high server load and increased latency due to constant polling, especially when updates are infrequent.

4.6: HTTP Long Polling

Long polling is an improvement over short polling, where the client sends a request to the server, and the server holds the request open until new data is available. This approach reduces the number of requests and server load, but it may still suffer from latency issues and requires careful management of server resources.

4.7: Webhook

Webhooks are user-defined HTTP callbacks triggered by specific events in a system. When an event occurs, the source site makes an HTTP request to the URL configured for the webhook. This approach allows for efficient, event-driven communication between different systems or services.

4.8: Stream API

Stream APIs enable clients to consume a continuous data stream from a server, often using HTTP or WebSocket connections. These APIs are designed for applications that require real-time updates, such as social media feeds, stock market data, or live analytics.

By understanding and utilizing these chat and streaming protocols, developers can build responsive, real-time applications that cater to various use cases and provide engaging user experiences.

Section 5: Common Components in System Design

This section will explore some standard components often found in modern system designs. Understanding these components can help developers seamlessly integrate them into their systems and enhance overall functionality.

5.1: Payment Service

Payment services handle transactions between customers and businesses. Integrating a reliable payment service is crucial for e-commerce and subscription-based platforms. Popular payment service providers include Stripe, PayPal, and Square. These services usually provide APIs to facilitate secure transactions and manage recurring payments, refunds, etc.

5.2: Analytic Service

Analytic services enable data collection, processing, and visualization to help businesses make informed decisions. These services can track user behaviour, monitor system performance, and analyze trends. Standard analytic service providers include Google Analytics, Mixpanel, and Amplitude. Integrating analytic services into a system can help businesses optimize their offerings and improve the user experience.

5.3: Notification

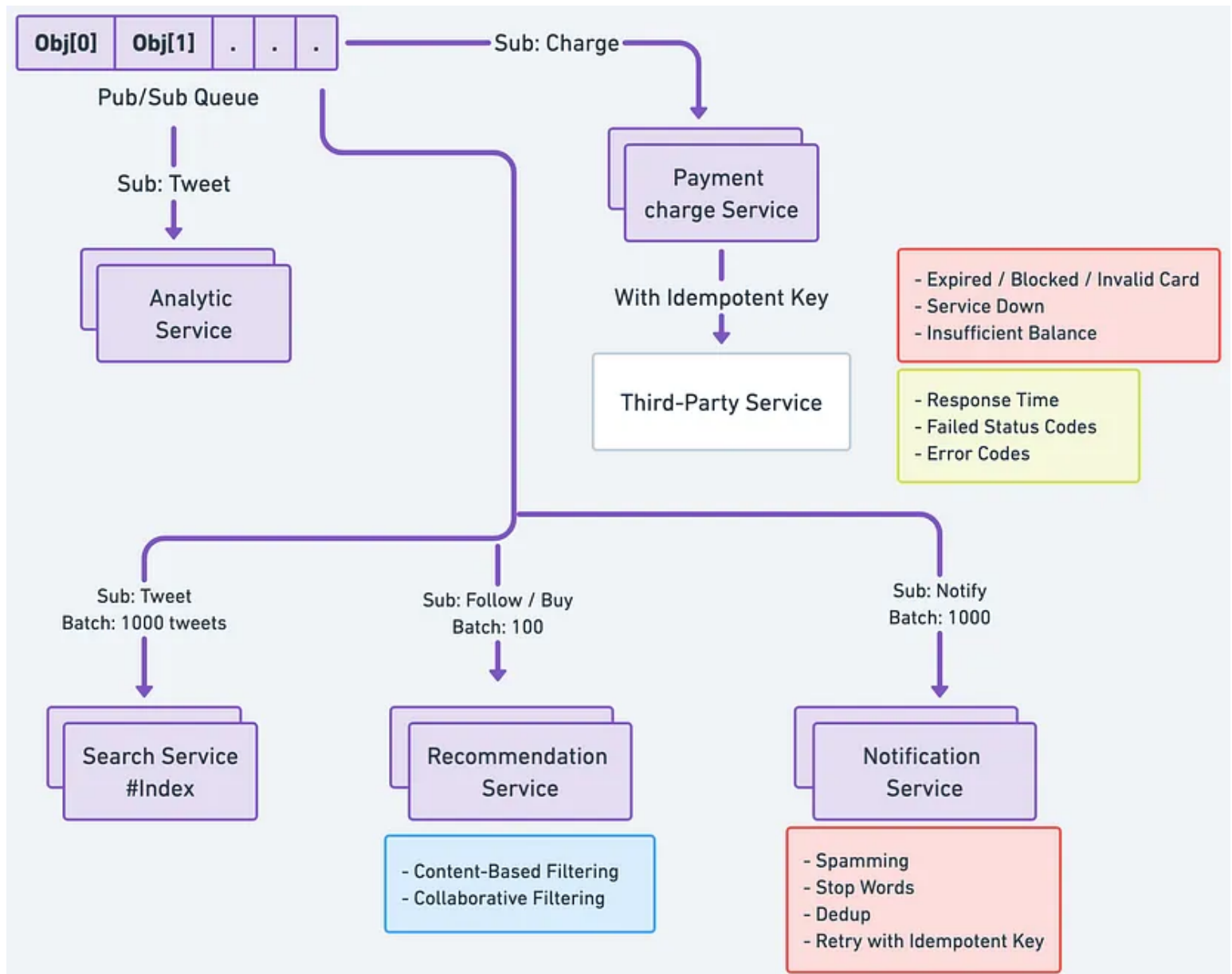
Notification services keep users informed about updates, alerts, and important information. These services can deliver notifications through various channels, such as email, SMS, and push notifications. Examples of notification service providers include Firebase Cloud Messaging (FCM), Amazon Simple Notification Service (SNS), and Twilio.

5.4: Search

Integrating a powerful search component is essential for systems with large amounts of data or content. A search service should provide fast, relevant, and scalable search capabilities. Elasticsearch, Apache Solr, and Amazon CloudSearch are popular choices for implementing search functionality. These services typically support full-text search, faceted search, and filtering, enabling users to find the information they're looking for quickly and efficiently.

5.5: Recommendation Service

Recommendation services use algorithms to provide personalized suggestions to users based on their preferences, behaviour, and other factors. These services can significantly improve user engagement and satisfaction. Techniques for generating recommendations include collaborative filtering, content-based filtering, and hybrid approaches. Machine learning algorithms, such as matrix factorization and deep learning, can also be used to generate more sophisticated recommendations.



By incorporating these standard components into their system designs, developers can enhance the functionality of their applications and provide a more seamless and engaging experience for users.

Section 6: Best Practices for System Design

6.1: Requirement Gathering

Thoroughly understand and document the system requirements before starting the design process.

6.2: Design Patterns

Leverage proven design patterns to address recurring design problems and improve the overall architecture.

6.3: Documentation

Document your design decisions, assumptions, and rationale to ensure better communication and maintainability.

6.4: Iterative Design

Refine your Design through multiple iterations and feedback, allowing it to evolve and improve.

6.5: Testing and Validation

Validate your Design against the requirements and conduct testing to identify and address potential issues.

Conclusion

In conclusion, system design is a multifaceted and complex process that requires a deep understanding of various components, protocols, and techniques. This blog post has provided an overview of essential topics such as DNS, load balancing, API Gateway, video and image handling, caching, databases, unique ID generation, standard components like payment and recommendation services, and chat and streaming protocols.

By leveraging this knowledge, developers can create scalable, efficient, and reliable systems that cater to diverse requirements and provide a seamless user experience. It's crucial to remember that system design is an iterative process and that continuous improvement is vital to building and maintaining successful applications. With a solid foundation in these core concepts and a focus on adaptability, developers can confidently tackle the challenges of designing and implementing robust systems.

I hope you've gained some valuable insights from this learning experience. Please share this information with your colleagues and friends if you found it helpful. And if you're interested in accessing even more content like this, consider becoming a [Medium member](#). Thank you for your support, and happy learning!

- Software Engineering
- Distributed Systems
- System Design Concepts
- Software Architecture
- System Design Interview

Sign up for DevGenius Updates

By Dev Genius

Get the latest news and update from DevGenius publication. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

 Get this newsletter

Get the Medium app

