System Design Framework

Prepared By: Aatiz Ghimire, for Herald Center for AI.

Summer, 2025

1 Learning Objectives.

- Use real-world architecture to reproduce system design frameworks
- Understand how large-scale systems scale and fail
- Gain experience applying architecture patterns from industry leaders

---

A 4-Step Practical Framework for System Design

System design challenges are open-ended by nature. There's rarely a "right" answer—only solutions that make well-reasoned trade-offs based on the problem constraints. This guide offers a practical and iterative 4-step framework to follow when tackling any system design problem.

Step 1 – Understand the Problem and Establish Design Scope

Before proposing a solution, fully understand the requirements. Hasty assumptions or overconfidence can lead to poor design choices.

Key Actions:

- Clarify requirements and constraints.
- Understand functional and non-functional requirements.
- Write down your assumptions.

Example: Design a News Feed System

- Is it a web or mobile app or both?
- What are the most important features? (e.g., creating posts, seeing friends' feed)
- Is the feed ordered chronologically or algorithmically?
- Max number of friends per user?
- What is the expected scale? (e.g., 10M DAU)
- Will content support media (images/videos)?

Step 2 – Propose a High-Level Design and Get Buy-In

Develop a bird's-eye view of the system. Sketch major components and show how they interact. Collaborate and validate your blueprint.

Key Components to Sketch:

- Client (mobile/web)
- Load Balancers
- Web Servers (Stateless)
- Cache (e.g., Redis)
- Database (e.g., MySQL, NoSQL)
- CDN for static content
- Message Queue (for asynchronous operations)

Use Case: News Feed System

- **Feed publishing:** User posts content, it is written to cache/db and propagated to friends' feeds.

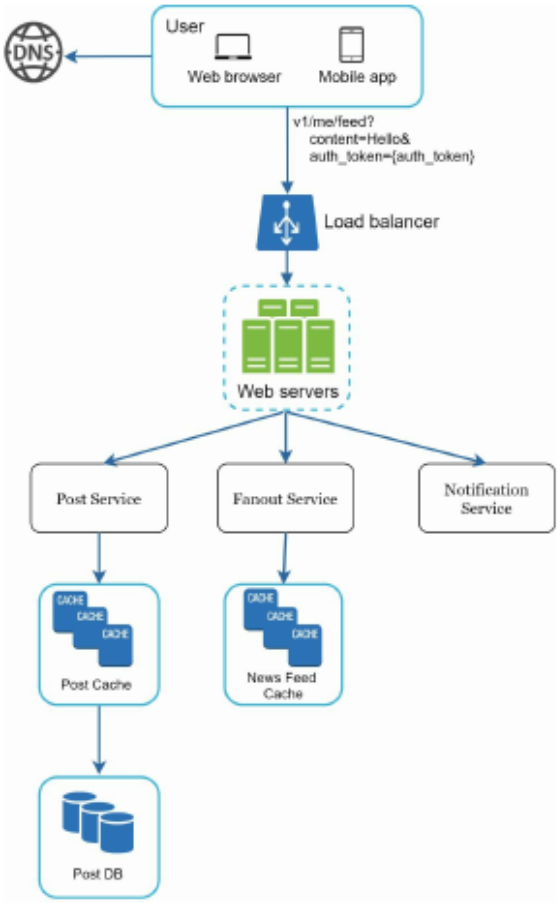

Figure 1: Feed publishing: when a user publishes a post, corresponding data is written into cache/-database, and the post will be populated into friends' news feed.

- **Feed retrieval:** Aggregates posts from friends in reverse chronological order.

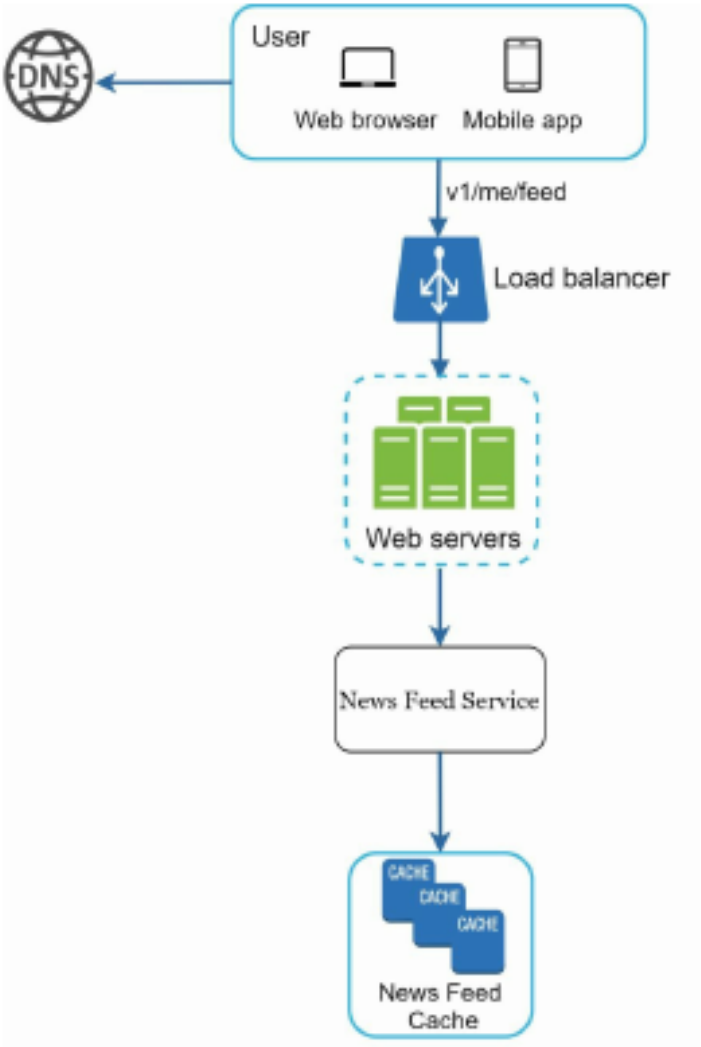

Figure 2: Newsfeed building: the news feed is built by aggregating friends' posts in a reverse chronological order.

Step 3 – Dive into the Details

With the high-level agreed upon, focus on depth. Choose one or two components or use-cases and explore them thoroughly.

Prioritization Criteria:

- Which part is most performance-sensitive?
- Where are the main bottlenecks?
- What could fail under heavy load?

Detailed Use Case: News Feed Publishing

- User posts are pushed into a message queue.
- Backend workers persist it into DB and update friend caches.

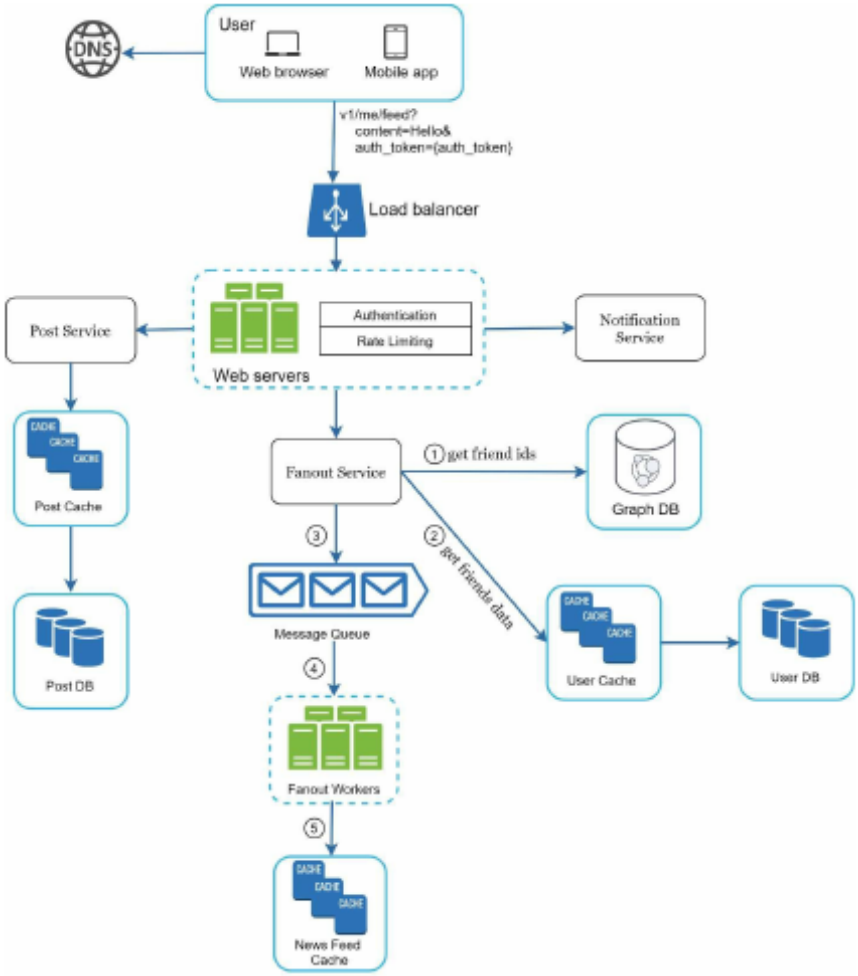

Figure 3: News Feed Publishing

Detailed Use Case: Feed Retrieval

- Web tier queries cache first. If a cache miss occurs, DB is queried.
- Media assets are served via CDN.

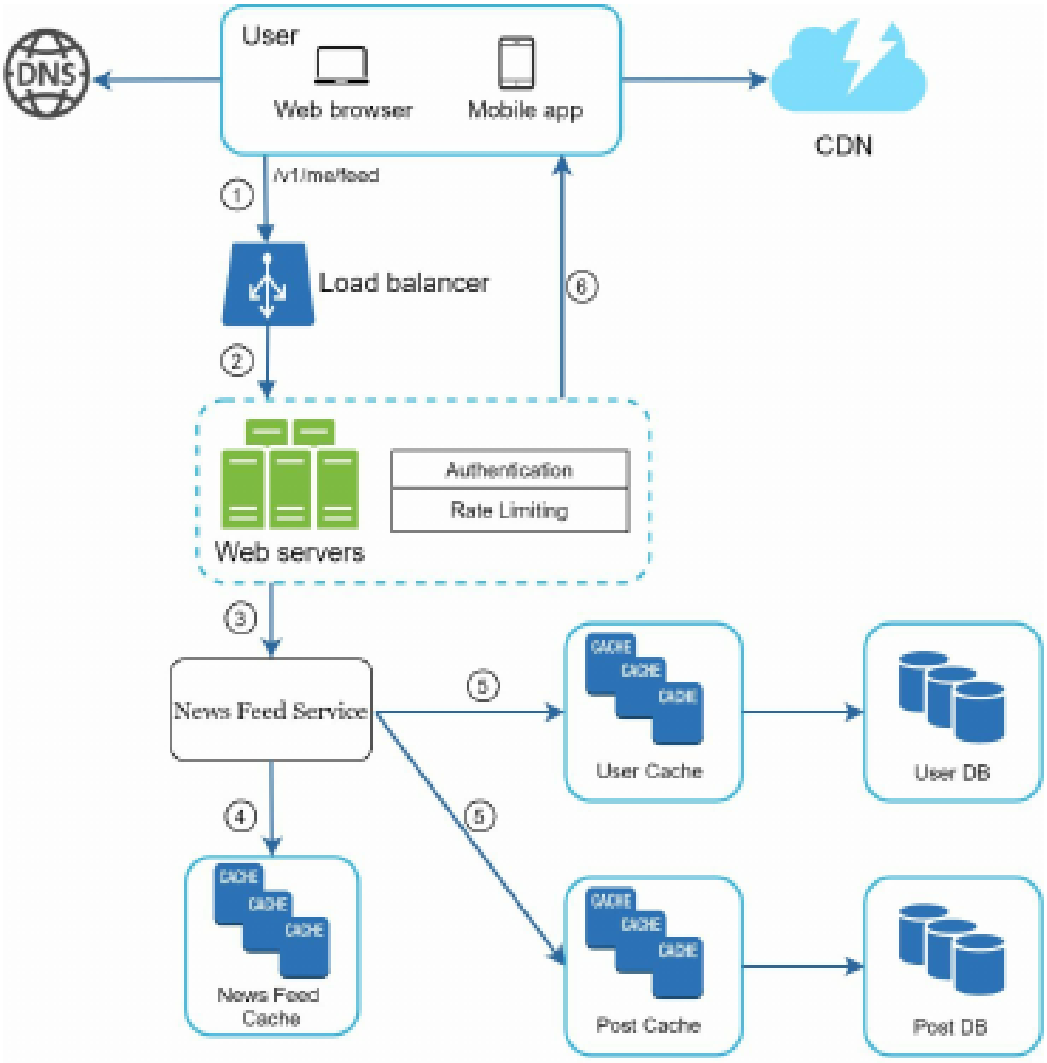

Figure 4: News Feed Publishing

Step 4 – Wrap Up with Bottlenecks, Improvements, and Scaling Paths

Reflect, evaluate, and iterate:

Checklist to Conclude:

- Identify bottlenecks (e.g., DB write throughput, cache eviction). ✓
- Propose monitoring/logging strategies. ✓
- Mention fallback mechanisms and failure recovery. ✓
- Show how to scale from 1M to 10M users. ✓
- Offer optional refinements if time permits. ✓

Back-of-the-Envelope Estimation Example: Twitter QPS and Storage

1. Assumptions:

- 300 million monthly users
- 50
- 2 tweets/day
- 10
- Store data for 5 years

2. QPS Estimate:

- DAU = 300M × 0.5 = 150M
- Tweet QPS = $\frac{150M \times 2}{24 \times 3600} \approx 3{,}500$
- Peak QPS = $2 \times 3{,}500 \approx 7{,}000$

3. Storage for Media:

- Average media size = 1MB
- Daily media volume = 150M × 2 × 10
- 5-year media = 30 TB × 365 × 5 ≈ 55 PB

Dos & Don'ts of System Design

Dos:

- Ask clarifying questions.
- Write down and validate assumptions.
- Communicate your thinking process.
- Break system into components.
- Prioritize trade-offs clearly.

Don'ts:

- Don't rush into architecture.
- Don't focus on just one layer early on.
- Don't ignore fault tolerance.
- Don't neglect capacity planning.
- Don't stay silent—talk through your logic.

This framework offers structure for both interviews and real-world system planning. Iterate, reason critically, and communicate effectively at each stage.

Workshop: Apply System Design Framework to Netflix & Facebook

1. Define Framework Steps

- Clarify Requirements & Scope
- Define Workloads & Constraints (QPS, data size, latency, availability)
- Design High-Level Architecture
- Identify Bottlenecks
- Plan Data Flow, Availability & Recovery
- Scale & Optimize (Caching, Sharding, Data Centers)
- Evaluate Trade-offs & Iteratively Improve

2. Define Framework Steps

Netflix

- Microservices-Based architecture on AWS. Ref: "System Design Netflix — A Complete Architecture" "Netflix Blogs and System Design" "Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems" "How did Facebook design their Real-Time Processing ecosystem"
- Event sourcing and Scalable video delivery using Open Connect. Ref: "Netflix System Design-Backend Architecture"
- Practices like Chaos Monkey for resilience. Ref: "Chaos engineering"

Facebook News Feed

- Fan-out model (pull-based timeline generation). Ref: "An Introduction to Facebook's System Architecture: Social Graph and TAO" Baseline System Design — Facebook Newsfeed And "Fanout" "Facebook System Design"
- Backend system TAO + MySQL + Memcached + Redis. Ref: The Tao of Facebook: 'Social Graph' Takes New Path
- Presto engine for petabyte-scale querying. Ref: Presto (SQL query engine)

3. Tasks

Step	Activity
A. Select Case (Netflix or Facebook)	Choose one
B. Requirements	Document QPS, batch vs stream, global latency
C. Sketch Architecture	Include entry points, microservices, caches, data tier
D. Identify Bottlenecks & Mitigation	E.g., CDN, Microservices decoupling, fan-out/pull
E. Replication & Scaling Plan	Sharding, data centers, consistency methods
F. Resilience Methods	Chaos testing, circuit breakers, fallback designs
G. Trade-off Vitals	E.g., performance vs eventual consistency, cost vs latency

4. Deliverables

- Simplified diagram + 2-page design doc per team
- Highlight trade-offs, scaling choices, and resilience strategies
- Compare your design to public architecture after publication

——————— The - End ———————-