Short-Form Video Platform System Design (TikTok/Instagram Reels)

Table of Contents

Introduction

Design a short-form video sharing platform similar to TikTok or Instagram Reels where users can:

- Upload short videos (15-60 seconds)
- Browse an infinite feed of personalized videos
- Interact through likes, comments, shares
- Follow creators and discover trending content
- Add music, effects, and filters

Key Characteristics:

- High engagement (users watch 50-100+ videos/session)
- Personalized recommendations (ML-driven feed)
- Real-time interactions
- Mobile-first design
- Global scale (100M+ DAU)

Requirements

Functional Requirements

Core Features:

1. **Video Upload & Publishing**
    - Upload videos (15-60 seconds)
    - Add captions, hashtags, music
    - Apply filters and effects
    - Geo-tagging and visibility settings
2. **Feed Experience**
    - Infinite scroll feed (For You Page)
    - Personalized recommendations
    - Following feed (content from followed users)
    - Trending/Discover page
3. **Social Interactions**
    - Like/Unlike videos
    - Comment on videos
    - Share videos (internal & external)
    - Follow/Unfollow users
    - Save videos to collections
4. **User Profile**
    - Profile page with user's videos
    - Follower/Following count
    - Bio, profile picture
    - Video analytics for creators
5. **Discovery**
    - Search (users, videos, hashtags, sounds)
    - Trending hashtags
    - Viral sounds/music library
    - Category browsing
6. **Notifications**
    - New followers
    - Likes/Comments on videos
    - Mentions in comments
    - Trending video alerts

Non-Functional Requirements

1. **Scalability**
    - Support 500M DAU (Daily Active Users)
    - Handle 100M video uploads per day
    - Serve 50B video views per day

2. **Performance**
    - Video load time: < 2 seconds
    - Feed generation: < 1 second
    - Like/comment: < 500ms
    - Search results: < 1 second
3. **Availability**
    - 99.9% uptime (8.76 hours downtime/year)
    - Graceful degradation (serve cached content if recommendation engine down)
4. **Reliability**
    - No video loss during upload
    - Consistent view counts
    - Eventual consistency acceptable for likes/comments
5. **Storage**
    - Retain videos indefinitely (unless user deletes)
    - Multiple quality versions per video
    - CDN caching for popular videos
6. **Security**
    - Content moderation (inappropriate content)
    - User privacy controls
    - Copyright protection
    - DRM for premium content

Back-of-Envelope Calculations

Assumptions

```
Users:
- Total users: 1B
- Daily Active Users (DAU): 500M (50% active daily)
- Average session: 30 minutes
- Videos watched per session: 50 videos

Content:
- Video uploads: 100M per day (0.2 per user)
- Average video duration: 30 seconds
- Video sizes:
  - Original: 50 MB (raw 1080p)
  - Processed 1080p: 20 MB
  - 720p: 8 MB
  - 480p: 3 MB
```

```
  - 360p: 1.5 MB
- Total per video: 32.5 MB (all versions)

Interactions:
- Each user likes 10 videos per session
- Each user comments on 1 video per session
- Each user shares 0.5 videos per session
```

Traffic Calculations

Video Uploads:

```
Daily uploads: 100M videos/day
Upload QPS: 100M ÷ 86,400 = 1,157 QPS
Peak upload QPS: 1,157 × 3 = ~3.5K QPS

Processing jobs (per video):
- Transcoding: 5 quality versions
- Thumbnail generation: 3 thumbnails
- Audio extraction
- Content moderation check
- Total: ~10 jobs per video
Job QPS: 3.5K × 10 = 35K jobs/sec (peak)
```

Video Views:

```
Daily views: 500M users × 50 videos = 25 billion views/day
View QPS: 25B ÷ 86,400 = 289,351 QPS
Peak view QPS: 289K × 3 = 867K QPS

Read-to-Write Ratio: 25B / 100M = 250:1 (extremely read-heavy)
```

Social Interactions:

```
Likes:
- 500M users × 10 likes = 5B likes/day
- 5B ÷ 86,400 = 57,870 QPS (round to 58K QPS)
- Peak: 174K QPS

Comments:
- 500M users × 1 comment = 500M comments/day
- 500M ÷ 86,400 = 5,787 QPS (round to 6K QPS)
- Peak: 18K QPS

Shares:
- 500M users × 0.5 shares = 250M shares/day
```

```
- 250M ÷ 86,400 = 2,893 QPS (round to 3K QPS)
- Peak: 9K QPS

Total interaction writes: 58K + 6K + 3K = 67K QPS (peak: 201K QPS)
```

Feed Generation:

```
Feed requests:
- Each user opens app: 3 times/day
- 500M × 3 = 1.5B feed requests/day
- 1.5B ÷ 86,400 = 17,361 QPS (round to 17K QPS)
- Peak: 51K QPS

Each feed request needs:
- User preferences lookup: 17K QPS
- ML model inference: 17K predictions/sec
- Video metadata fetch: 17K × 50 videos = 850K QPS (cached)
```

Storage Calculations

Video Storage:

```
Daily video storage:
- 100M videos × 32.5 MB = 3,250 TB/day = 3.25 PB/day

Yearly: 3.25 PB × 365 = 1,186 PB/year = 1.2 EB/year
5 years: 1.2 EB × 5 = 6 exabytes

With compression improvements (20% savings):
Actual: 6 EB × 0.8 = 4.8 EB over 5 years
```

Metadata Storage:

```
Video metadata:
- 100M videos/day × 5 KB = 500 GB/day
- 5 years: 500 GB × 365 × 5 = 912 TB

User data:
- 1B users × 10 KB = 10 TB

Comments:
- 500M/day × 500 bytes = 250 GB/day
- 5 years: 250 GB × 365 × 5 = 456 TB

Likes:
- 5B/day × 16 bytes (user_id + video_id) = 80 GB/day
```

```
- 5 years: 80 GB × 365 × 5 = 146 TB

Total metadata: 912 TB + 10 TB + 456 TB + 146 TB = 1.5 PB
```

Total Storage: 4.8 EB (videos) + 1.5 PB (metadata) ≈ 4.8 EB

Bandwidth Calculations

Video Upload Bandwidth:

```
Incoming:
- 3.5K QPS × 50 MB (original) = 175 GB/s
- Peak bandwidth: 175 GB/s
```

Video Streaming Bandwidth:

```
Average video size served: 8 MB (720p most common)
Outgoing: 867K QPS × 8 MB = 6,936 GB/s = 6.9 TB/s

With CDN (95% cache hit ratio):
- Origin: 6.9 TB/s × 0.05 = 345 GB/s
- CDN: 6.9 TB/s × 0.95 = 6.5 TB/s

CDN absolutely critical!
```

Memory Calculations

Hot Video Cache:

```
Popular videos (last 24 hours):
- 100M videos/day × 20% hot = 20M videos
- 20M × 8 MB (720p) = 160 TB

Cache at CDN edges:
- 200 edges × 800 GB each = 160 TB ✓
```

Metadata Cache:

```
Video metadata (hot):
- 20M videos × 5 KB = 100 GB

User profiles (active):
- 500M DAU × 10 KB = 5 TB
- Cache 20%: 1 TB
```

```
Feed recommendations (pre-computed):
- 500M users × 20 videos × 5 KB = 50 TB
- Cache recent sessions: 10 TB

Total metadata cache: 100 GB + 1 TB + 10 TB = 11 TB
Distributed: 200 Redis nodes × 55 GB each
```

Server Calculations

Application Servers:

```
Peak QPS: 867K (video views) + 51K (feed) + 201K (interactions) = 1.1M
QPS
Servers needed: 1.1M ÷ 5K per server = 220 servers
With redundancy: 400 servers globally
```

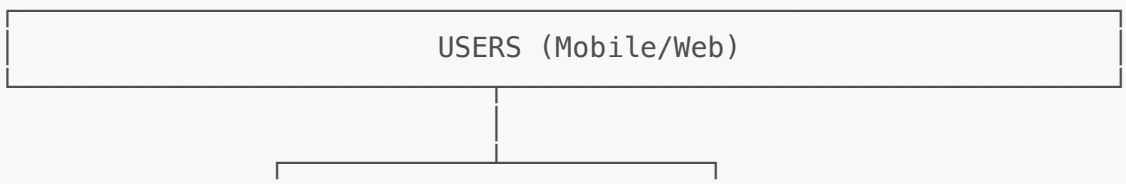
Video Processing Workers:

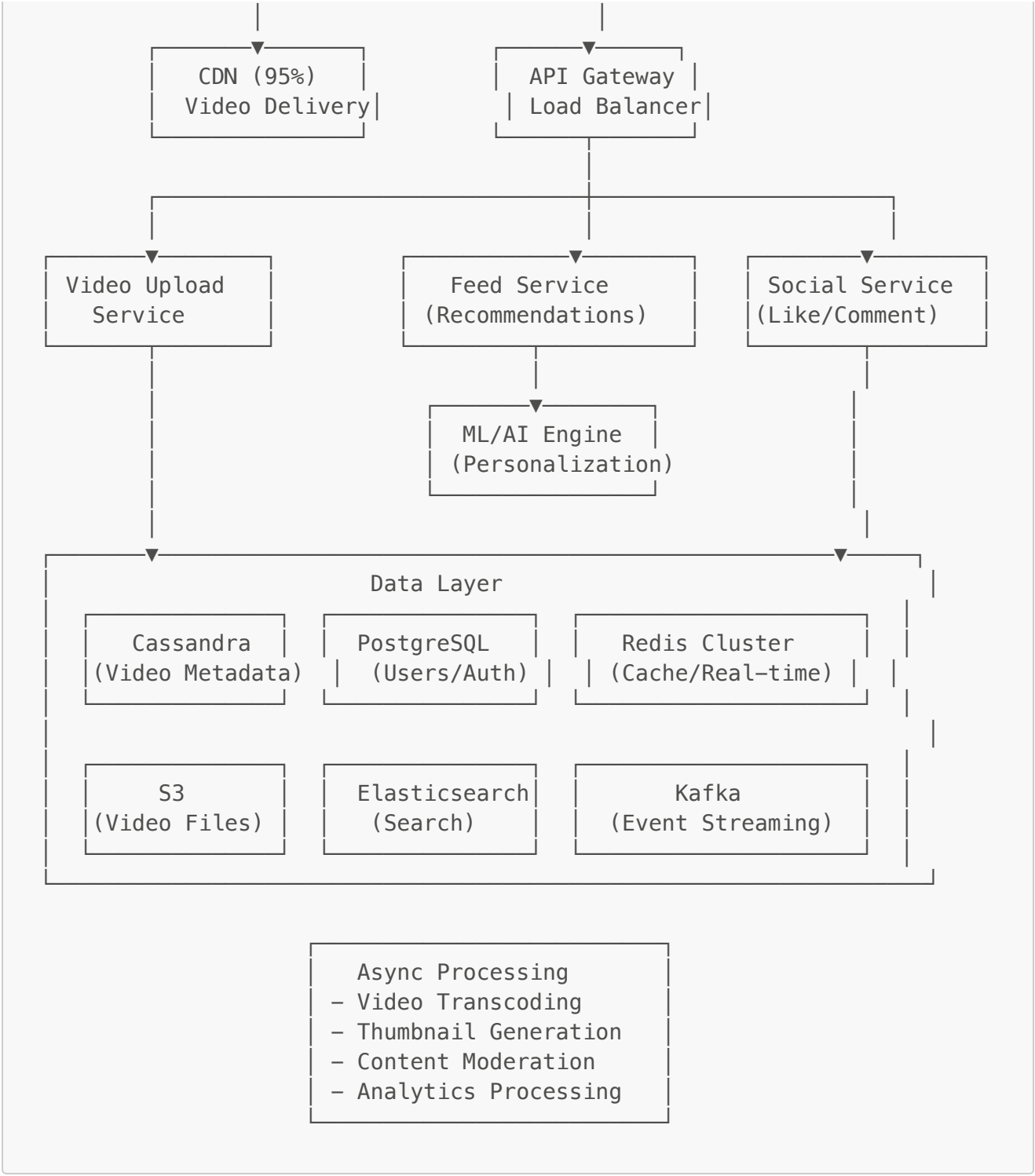
```
Processing load: 35K jobs/sec (peak)
Each worker: 5 jobs/sec
Workers needed: 35K ÷ 5 = 7,000 workers
With redundancy: 10,000 workers

Worker types:
- Transcoding: 6,000 workers (GPU instances)
- Thumbnail generation: 2,000 workers
- Content moderation: 2,000 workers
```

ML Inference Servers:

```
Recommendation requests: 51K/sec (feed generation)
Each GPU server: 500 inferences/sec
Servers needed: 51K ÷ 500 = 102 servers
With redundancy: 200 GPU servers
```

High-Level Architecture





Core Components

1. Video Upload Service

Responsibilities:

- Accept video uploads from mobile clients
- Validate video format, size, duration
- Store raw video in S3
- Trigger async processing pipeline
- Generate unique video ID

Flow:

```
1. Client → Upload Service: Initiate upload
2. Upload Service → S3: Store raw video
3. Upload Service → Kafka: Publish "video_uploaded" event
4. Upload Service → Client: Return video_id
5. Processing workers consume Kafka event
6. Workers → S3: Store processed versions
7. Workers → Cassandra: Update video metadata (status: processing →
ready)
```

Technology:

- **Upload API**: Node.js/Go (fast I/O)
- **Storage**: S3 for raw videos
- **Queue**: Kafka for job distribution
- **Processing**: GPU workers (FFmpeg)

API:

```
POST /api/v1/videos/upload
Request:
{
  "video": <multipart/form-data>,
  "caption": "Check this out!",
  "hashtags": ["dance", "music"],
  "music_id": "12345",
  "visibility": "public"
}

Response:
{
  "video_id": "abc123",
  "status": "processing",
  "upload_url": "s3://bucket/raw/abc123.mp4"
}
```

Optimization:

- **Chunked Upload**: Split large files into 5 MB chunks
- **Resumable Upload**: Support retry on network failure
- **Pre-signed URLs**: Client uploads directly to S3
- **Compression**: Client-side compression before upload

2. Video Processing Pipeline

Processing Steps:

Stage 1: Video Transcoding (GPU-intensive)

```
Input: Raw video (1080p, 50 MB)

Outputs:
1. 1080p (20 MB) - High quality
2. 720p (8 MB)   - Default quality
3. 480p (3 MB)   - Medium quality
4. 360p (1.5 MB) - Low quality (slow connections)

Time: ~30 seconds per video (GPU)
Workers: 6,000 GPU instances (p3.2xlarge)
```

Stage 2: Thumbnail Generation

```
Extract 3 frames from video:
- Beginning (0s)
- Middle (15s)
- End (30s)

Generate thumbnails:
- Large: 1280×720 (200 KB)
- Medium: 640×360 (50 KB)
- Small: 320×180 (20 KB)

Time: ~2 seconds per video
Workers: 2,000 CPU instances
```

Stage 3: Content Moderation

```
ML models check for:
- Violence, nudity, hate speech
- Copyright violations (audio/video fingerprinting)
- Spam/fake content

Processing:
- Vision API: Analyze frames
- Audio API: Check for copyrighted music
- Text API: Analyze captions

Time: ~5 seconds per video
Workers: 2,000 CPU instances (ML inference)
```

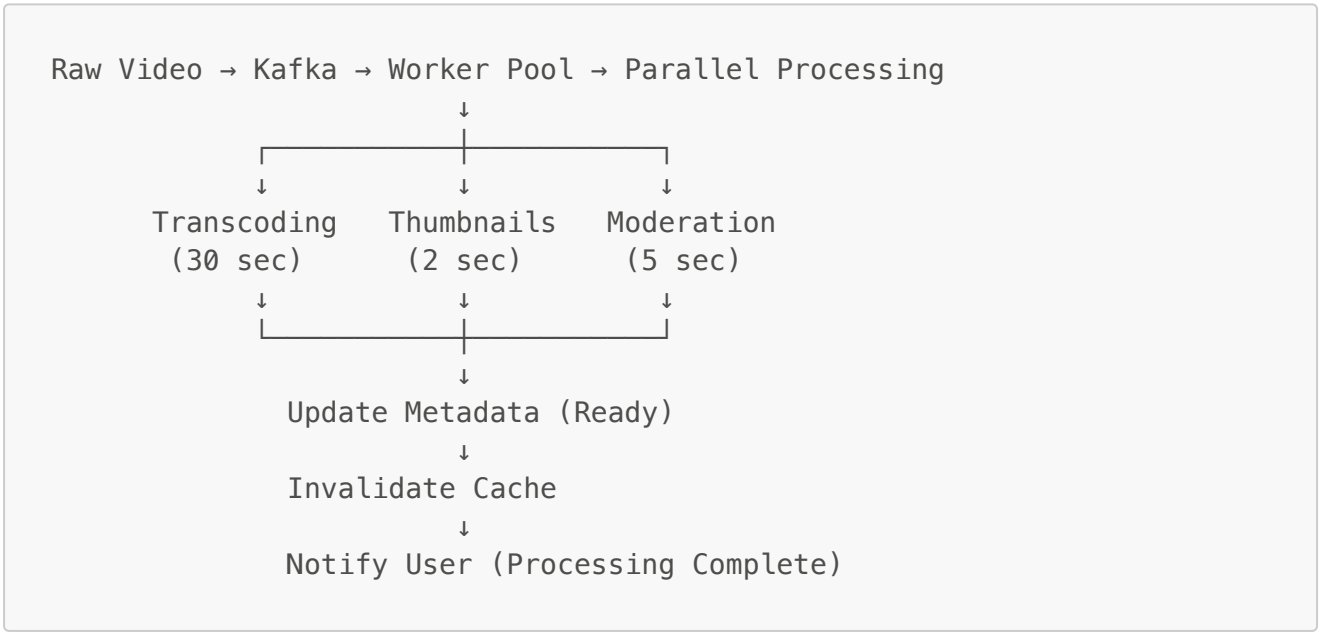
Stage 4: Metadata Extraction

```
Extract and store:
- Video dimensions, duration, bitrate
- Audio track information
- Detected objects/scenes (ML)
- Detected faces/people (for discovery)
- Color palette
- Camera metadata (if available)

Time: ~1 second per video
```

Total Processing Time:

```
Sequential: 30s + 2s + 5s + 1s = 38 seconds
Parallel (with proper orchestration): ~30 seconds
```

Pipeline Architecture:



3. Feed Generation Service

The Most Critical Component!

Responsibilities:

- Generate personalized "For You" feed
- Blend recommendations from multiple signals
- Filter out already-seen videos
- Ensure diversity (not all same creator/category)
- Refresh feed when user scrolls

Feed Generation Algorithm:

Step 1: Candidate Generation (Recall)

```
Goal: Get ~1000 candidate videos from various sources

Sources:
1. Collaborative Filtering (40%)
   - "Users like you also watched..."
   - User-video interaction matrix
   - 400 videos

2. Content-Based Filtering (30%)
   - Based on videos user previously liked
   - Video features: hashtags, music, category
   - 300 videos

3. Trending/Viral (20%)
   - High engagement rate in last 24 hours
   - 200 videos

4. Following Feed (10%)
   - Recent videos from followed creators
   - 100 videos

Total candidates: 1,000 videos
Time budget: 300ms
```

Step 2: Ranking (Precision)

```
Goal: Rank 1000 candidates, return top 50

ML Model Features:
- User features (30 dimensions):
  * Watch history
  * Like/comment patterns
  * Session time
  * Demographics

- Video features (50 dimensions):
  * View count, like ratio
  * Completion rate
  * Engagement velocity
  * Hashtags, music, category

- User-Video interaction (20 dimensions):
  * Historical interaction with similar videos
  * Creator affinity score
  * Time since upload
  * Predicted watch time
```

```
Model: XGBoost/Neural Network
Output: Predicted engagement score (0-1)

Sort by score, return top 50 videos
Time budget: 500ms
```

Step 3: Post-Processing

```
Apply business rules:
- Diversity: Max 2 videos from same creator
- Freshness: Mix old (proven) and new (cold start)
- Serendipity: 10% random videos (exploration)
- Remove duplicates
- Apply content filters (user preferences)

Time budget: 100ms
```

Total Feed Generation Time: 300ms + 500ms + 100ms = 900ms < 1 second ✓

Optimization Strategies:

1. Pre-computation:

```
For each user, pre-compute top 500 candidates every 30 minutes
Store in Redis with TTL
Feed request: Retrieve pre-computed + live ranking

Reduces latency: 900ms → 200ms
Storage: 500M users × 500 videos × 32 bytes = 8 TB
Cost: 8 TB ÷ 64 GB × $280 = $35,000/month

Worth it for 4.5x latency improvement!
```

2. Caching:

```
Cache trending videos (valid for 10 minutes):
- Key: "trending:category:timestamp"
- Size: 1000 videos × 5 KB = 5 MB
- Refresh every 10 minutes
- Hit rate: 60%
```

3. Batch Inference:

```
```

```
Instead of 1 user at a time:
- Batch 100 users together
- Single GPU inference: 100 predictions in 200ms
- Throughput: 500 predictions/sec per GPU
- Servers: 51K ÷ 500 = 102 GPUs (vs 510 GPUs for individual inference)
- Cost savings: 5x
```

4. Social Interaction Service

Like Service:

```
Operations:
- Like video: 58K QPS
- Unlike video: ~5K QPS
- Get like count: 867K QPS (video views)

Implementation:
- Write to Cassandra (permanent record)
- Update Redis counter (real-time count)
- Invalidate user's "liked videos" cache
- Send notification to video creator (async via Kafka)

API:
POST /api/v1/videos/{video_id}/like
Response: { "liked": true, "like_count": 1234 }

Optimization:
- Batch writes to Cassandra (every 10 seconds)
- Redis INCR for immediate count update
- Async notification (non-blocking)
```

Comment Service:

```
Operations:
- Post comment: 6K QPS
- Get comments: 867K × 0.1 (10% read comments) = 87K QPS
- Reply to comment: 2K QPS

Implementation:
- Store in Cassandra (partition by video_id)
- Cache top 100 comments per video in Redis
- Real-time updates via WebSocket
- Nested replies (parent_comment_id)

Schema:
{
  "comment_id": "uuid",
```

```
  "video_id": "abc123",
  "user_id": "user456",
  "text": "Amazing!",
  "parent_comment_id": null,
  "created_at": timestamp,
  "like_count": 45
}
```

Share Service:

```
Share types:
- Internal share (to another user): 60%
- External share (WhatsApp, Instagram): 40%

Implementation:
- Record share in Cassandra
- Generate shareable link (for external)
- Update video's share count (Redis counter)
- Track share source for analytics
```

5. Search & Discovery Service

Search Types:

1. Video Search:

```
Query: "dance moves"
Index: Elasticsearch
Fields: caption, hashtags, video_description, detected_objects
Results: Ranked by relevance + engagement

Optimization:
- Index updates every 5 minutes (batch)
- Cache popular searches (TTL: 1 hour)
- Autocomplete: Trie data structure (in-memory)
```

2. User Search:

```
Query: "@username"
Index: Elasticsearch
Fields: username, display_name, bio
Results: Ranked by follower count

Special handling:
- Verified users prioritized
- Fuzzy matching for typos
```

3. Hashtag Search:

```
Query: "#viral"
Index: Pre-computed hashtag rankings
Results: Top videos with this hashtag (last 7 days)

Storage:
- Hashtag index: Redis Sorted Set
- Score: engagement_score × recency_factor
- Update: Real-time as videos gain traction
```

4. Music/Sound Search:

```
Query: "trending sounds"
Index: Music library + usage count
Results: Most used sounds (last 24 hours)

Implementation:
- Track sound usage count (Redis)
- Link to original video (where sound originated)
- Allow users to create videos with same sound
```

6. Recommendation Engine (ML/AI)

Two-Tower Neural Network Architecture:

User Tower (Encode user preferences):

```
Inputs:
- Watch history (last 100 videos)
- Like history (last 500 likes)
- Search queries
- Demographics (age, location)
- Time of day
- Device type

Output: User embedding (256 dimensions)
Model: Deep Neural Network (3 layers)
```

Video Tower (Encode video features):

```
Inputs:
- Video metadata (hashtags, music, category)
- Creator profile (follower count, engagement rate)
```

```
- Visual features (detected objects, colors)
- Audio features (music genre, tempo)
- Text features (caption sentiment, keywords)
- Engagement metrics (view count, like ratio, completion rate)

Output: Video embedding (256 dimensions)
Model: Deep Neural Network (3 layers)
```

Scoring:

```
Similarity score = cosine_similarity(user_embedding, video_embedding)

Final score = 0.4 × similarity_score +
              0.3 × engagement_rate +
              0.2 × freshness_score +
              0.1 × diversity_penalty

Higher score → Higher ranking in feed
```

Training Pipeline:

```
Data:
- Positive samples: Videos user watched >50% duration
- Negative samples: Videos user skipped <3 seconds
- Training data: 100B interactions

Training:
- Daily retraining (to capture trends)
- Model size: 2 GB
- Training time: 6 hours (1000 GPU hours)
- A/B testing: New model vs current model

Serving:
- Model deployed to 200 GPU servers
- Each server: 500 inferences/sec
- Total capacity: 100K inferences/sec
- Actual need: 51K/sec (50% headroom)
```

Fallback Strategy:

```
If ML service is down:
1. Serve from pre-computed recommendations (Redis cache)
2. Fall back to trending videos
3. Fall back to following feed
4. Graceful degradation (users still get content)
```

7. Notification Service

Notification Types:

```
High Priority (push immediately):
- New follower
- Comment on your video
- Video mentioned in comment
- Live stream started (following)

Medium Priority (batch, send every 5 min):
- Likes on your video (aggregate)
- New video from followed creator

Low Priority (daily digest):
- Trending in your category
- Suggested creators to follow
```

Implementation:

```
Components:
- Notification Generator: Consume Kafka events
- Notification Dispatcher: Send via FCM/APNs
- Notification Store: Cassandra (90-day retention)
- Preference Cache: Redis (user notification settings)

Batching logic:
- Group similar notifications
- "User1 and 45 others liked your video"
- Reduces notification fatigue
```

Fan-out Strategy:

```
Scenario: Popular creator (10M followers) posts video

Naive approach:
- Send 10M notifications immediately
- Spike: 10M notifications in 1 second

Smart approach:
- Segment followers by engagement level:
  * High engagement (10%): Send immediately (1M notifications)
  * Medium engagement (30%): Send within 1 hour (3M)
  * Low engagement (60%): Send within 24 hours (6M)
- Smooth out spike: 1M/sec → 11.5/sec average
```

Database Design

Video Metadata (Cassandra)

Why Cassandra:

```
- Storage: 1.5 PB metadata (requires distribution)
- Write QPS: 3.5K video uploads
- Read QPS: 867K video views (need video metadata)
- Pattern: Time-series (videos sorted by upload time)
- Queries: Range queries by time, creator, category
```

Schema:

```
CREATE TABLE videos (
    video_id UUID PRIMARY KEY,
    user_id UUID,
    upload_time TIMESTAMP,
    duration INT,
    caption TEXT,
    hashtags LIST<TEXT>,
    music_id UUID,
    category TEXT,
    status TEXT,  -- 'processing', 'ready', 'removed'

    -- Engagement metrics
    view_count COUNTER,
    like_count COUNTER,
    comment_count COUNTER,
    share_count COUNTER,

    -- Video files (URLs)
    video_1080p TEXT,
    video_720p TEXT,
    video_480p TEXT,
    video_360p TEXT,
    thumbnail_url TEXT,

    -- ML features
    detected_objects LIST<TEXT>,
    video_embedding LIST<FLOAT>,

    -- Visibility
    is_public BOOLEAN,
    allowed_countries SET<TEXT>,

    created_at TIMESTAMP,
    updated_at TIMESTAMP
);
```

```
-- Secondary indexes
CREATE INDEX ON videos (user_id);
CREATE INDEX ON videos (music_id);
CREATE INDEX ON videos (status);
```

Partitioning Strategy:

```
Partition Key: video_id (UUID)
- Ensures even distribution
- ~22,000 nodes for 1.5 PB (68 GB per node)

Alternative (for time-range queries):
Partition Key: (shard_id, upload_date)
Clustering Key: video_id
- Better for "get videos uploaded today" queries
- 256 shards
```

User Data (PostgreSQL)

Why PostgreSQL:

```
- Storage: 10 TB (1B users × 10 KB)
- Write QPS: Low (user updates infrequent)
- Read QPS: Moderate (with caching)
- Need: ACID for user registration/authentication
- Queries: Complex (followers graph)
```

Schema:

```
-- Users table
CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    display_name VARCHAR(100),
    bio TEXT,
    profile_picture_url TEXT,

    -- Stats
    follower_count INT DEFAULT 0,
    following_count INT DEFAULT 0,
    video_count INT DEFAULT 0,
    total_likes INT DEFAULT 0,
```

```
    -- Settings
    is_verified BOOLEAN DEFAULT FALSE,
    is_private BOOLEAN DEFAULT FALSE,
    country_code VARCHAR(2),

    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_username ON users(username);
CREATE INDEX idx_email ON users(email);
CREATE INDEX idx_created_at ON users(created_at);

-- Followers table (Many-to-Many)
CREATE TABLE followers (
    follower_id UUID NOT NULL,
    following_id UUID NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    PRIMARY KEY (follower_id, following_id),
    FOREIGN KEY (follower_id) REFERENCES users(user_id),
    FOREIGN KEY (following_id) REFERENCES users(user_id)
);

CREATE INDEX idx_follower ON followers(follower_id);
CREATE INDEX idx_following ON followers(following_id);
```

Sharding Strategy:

```
Shard by user_id (hash-based):
- 10 shards × 100M users each
- Each shard: 100M × 10 KB = 1 TB ✓
- Even distribution of load
```

Likes & Comments (Cassandra)

Why Cassandra:

```
- Write QPS: 67K (likes + comments)
- Storage: 456 TB + 146 TB = 602 TB
- Pattern: Time-series, append-heavy
- Queries: Get likes/comments for video_id
```

Likes Schema:

```
CREATE TABLE video_likes (
    video_id UUID,
    user_id UUID,
    created_at TIMESTAMP,
    PRIMARY KEY (video_id, created_at, user_id)
) WITH CLUSTERING ORDER BY (created_at DESC);

-- For "did user like this video?" query
CREATE TABLE user_likes (
    user_id UUID,
    video_id UUID,
    created_at TIMESTAMP,
    PRIMARY KEY (user_id, video_id)
);
```

Comments Schema:

```
CREATE TABLE video_comments (
    video_id UUID,
    comment_id UUID,
    user_id UUID,
    text TEXT,
    parent_comment_id UUID,  -- NULL for top-level comments
    like_count COUNTER,
    created_at TIMESTAMP,
    PRIMARY KEY (video_id, created_at, comment_id)
) WITH CLUSTERING ORDER BY (created_at DESC);
```

User Interactions (Redis)

Real-time Counters:

```
Video engagement:
- Key: "video:{video_id}:views" → Counter
- Key: "video:{video_id}:likes" → Counter
- Key: "video:{video_id}:comments" → Counter
- Key: "video:{video_id}:shares" → Counter
- TTL: 7 days (then write to Cassandra)

User activity:
- Key: "user:{user_id}:watched" → Set (last 100 videos)
- Key: "user:{user_id}:liked" → Set (last 500 videos)
- TTL: 30 days
```

Video Files (S3)

Storage Structure:

```
Buckets organized by region and quality:

s3://videos-us-west-2/
  raw/
    2024/01/10/abc123.mp4
  processed/
    1080p/abc123.mp4
    720p/abc123.mp4
    480p/abc123.mp4
    360p/abc123.mp4
  thumbnails/
    abc123_large.jpg
    abc123_medium.jpg
    abc123_small.jpg

Replication:
- Cross-region replication for disaster recovery
- 3 regions: us-west-2, eu-west-1, ap-southeast-1
```

S3 Lifecycle Policies:

```
Original raw videos:
- Transition to Glacier after 30 days
- Videos rarely accessed after processing complete
- Cost: $0.004/GB vs $0.023/GB (83% savings)

Processed videos:
- Keep in Standard (frequently accessed)
- Popular videos cached at CDN (offload S3)
- Unpopular videos (< 1000 views): Transition to IA after 90 days
```

Cost Analysis:

```
Total storage: 4.8 EB over 5 years

With lifecycle policies:
- Hot (< 30 days): 3.25 PB × 30 = 97.5 PB
  Cost: 97.5 PB × $23/TB = $2.24M/month (S3 Standard)

- Warm (30-365 days): 3.25 PB × 335 = 1,089 PB
  Cost: 1,089 PB × $12.50/TB = $13.6M/month (S3 IA)

- Cold (> 1 year): 3.7 EB
  Cost: 3.7 EB × $4/TB = $14.8M/month (S3 Glacier)
```

```
Total: $30.64M/month for storage

Without lifecycle: 4.8 EB × $23/TB = $110M/month
Savings: $79M/month (72% reduction!)
```

API Design

Video APIs

1. Get Video Feed

```
GET /api/v1/feed
Headers: Authorization: Bearer <token>
Query Params:
  - type: "foryou" | "following" | "trending"
  - count: 20 (default)
  - cursor: "abc123" (for pagination)

Response:
{
  "videos": [
    {
      "video_id": "abc123",
      "user": {
        "user_id": "user456",
        "username": "creator1",
        "profile_pic": "url",
        "is_verified": true
      },
      "video_url": "cdn.com/videos/720p/abc123.mp4",
      "thumbnail_url": "cdn.com/thumbnails/abc123.jpg",
      "caption": "Amazing dance!",
      "hashtags": ["dance", "viral"],
      "music": {
        "music_id": "music789",
        "title": "Trending Song",
        "artist": "Artist Name"
      },
      "stats": {
        "views": 1234567,
        "likes": 89012,
        "comments": 3456,
        "shares": 789
      },
      "duration": 30,
      "is_liked": false,
      "created_at": "2024-01-10T12:00:00Z"
    }
  ],
```

```
  "next_cursor": "xyz789"
}
```

2. Upload Video

```
POST /api/v1/videos/upload/init
Headers: Authorization: Bearer <token>
Request:
{
  "file_size": 52428800,  // 50 MB
  "duration": 30,
  "mime_type": "video/mp4"
}

Response:
{
  "video_id": "abc123",
  "upload_url": "s3-presigned-url",
  "chunk_size": 5242880,  // 5 MB chunks
  "expires_in": 3600
}

POST /api/v1/videos/upload/complete
Request:
{
  "video_id": "abc123",
  "caption": "Check this out!",
  "hashtags": ["dance", "music"],
  "music_id": "music123"
}

Response:
{
  "video_id": "abc123",
  "status": "processing",
  "estimated_time": 30  // seconds
}
```

3. Get Video Details

```
GET /api/v1/videos/{video_id}
Headers: Authorization: Bearer <token>

Response:
{
  "video_id": "abc123",
  "user": { ... },
  "video_urls": {
```

```
    "1080p": "url",
    "720p": "url",
    "480p": "url",
    "360p": "url"
  },
  "stats": { ... },
  "is_liked": false,
  "is_saved": false,
  "upload_date": "2024-01-10"
}
```

Social Interaction APIs

1. Like/Unlike Video

```
POST /api/v1/videos/{video_id}/like
DELETE /api/v1/videos/{video_id}/like

Response:
{
  "success": true,
  "like_count": 1235,
  "is_liked": true
}
```

2. Comment on Video

```
POST /api/v1/videos/{video_id}/comments
Request:
{
  "text": "Amazing content!",
  "parent_comment_id": null  // For replies
}

Response:
{
  "comment_id": "comment123",
  "created_at": "2024-01-10T12:00:00Z"
}

GET /api/v1/videos/{video_id}/comments
Query Params:
  - limit: 20
  - cursor: "comment123"
  - sort: "popular" | "recent"

Response:
{
```

```
  "comments": [
    {
      "comment_id": "comment123",
      "user": { ... },
      "text": "Amazing!",
      "like_count": 45,
      "reply_count": 3,
      "created_at": "2024-01-10T12:00:00Z"
    }
  ],
  "next_cursor": "comment456"
}
```

3. Share Video

```
POST /api/v1/videos/{video_id}/share
Request:
{
  "type": "external",  // or "internal"
  "platform": "whatsapp"  // for external
}

Response:
{
  "share_url": "https://app.link/v/abc123",
  "share_count": 790
}
```

User APIs

1. Follow/Unfollow User

```
POST /api/v1/users/{user_id}/follow
DELETE /api/v1/users/{user_id}/follow

Response:
{
  "success": true,
  "is_following": true,
  "follower_count": 1234567
}
```

2. Get User Profile

```
GET /api/v1/users/{user_id}
```

```
Response:
{
  "user_id": "user456",
  "username": "creator1",
  "display_name": "Amazing Creator",
  "bio": "Content creator | Dancer",
  "profile_pic": "url",
  "is_verified": true,
  "stats": {
    "followers": 1234567,
    "following": 500,
    "videos": 250,
    "total_likes": 9876543
  },
  "is_following": false,
  "recent_videos": [ ... ]
}
```

3. Get User's Videos

```
GET /api/v1/users/{user_id}/videos
Query Params:
  - limit: 20
  - cursor: "video123"

Response:
{
  "videos": [ ... ],
  "next_cursor": "video456",
  "total": 250
}
```

Search APIs

1. Search Videos

```
GET /api/v1/search/videos
Query Params:
  - q: "dance moves"
  - limit: 20
  - cursor: "video123"

Response:
{
  "results": [ ... ],
  "next_cursor": "video456",
  "total": 15678
}
```

2. Search Users

```
GET /api/v1/search/users
Query Params:
  - q: "creator"
  - limit: 20

Response:
{
  "results": [
    {
      "user_id": "user123",
      "username": "creator1",
      "display_name": "Amazing Creator",
      "profile_pic": "url",
      "follower_count": 1234567,
      "is_verified": true
    }
  ]
}
```

3. Trending Hashtags

```
GET /api/v1/trending/hashtags
Query Params:
  - limit: 50
  - region: "US" (optional)

Response:
{
  "hashtags": [
    {
      "tag": "dance",
      "video_count": 1234567,
      "view_count": 987654321,
      "rank": 1
    }
  ]
}
```

Technology Stack

Complete Stack with Justifications

Frontend:

```
Mobile Apps:
- iOS: Swift + SwiftUI
- Android: Kotlin + Jetpack Compose

Why:
- Native performance for video playback
- Better camera/media APIs
- Smooth 60fps scrolling

Web:
- React + TypeScript
- Video.js for player
- WebSocket for real-time features
```

API Gateway:

```
Technology: Kong / AWS API Gateway

Features:
- Rate limiting (per user/IP)
- Authentication (JWT validation)
- Request routing
- SSL termination
- API versioning

Why:
- Handle 1.1M QPS peak
- Built-in rate limiting
- Easy to scale horizontally
```

Backend Services:

```
Language: Go (primary) + Node.js (WebSocket)

Why Go:
- High performance (handles 5K QPS per instance)
- Excellent concurrency (goroutines)
- Low memory footprint
- Good for video upload/streaming services

Why Node.js:
- Great for WebSocket (real-time comments)
- Event-driven architecture
- Large ecosystem
```

Databases:

1. Cassandra (Video Metadata):

```
Numbers:
- Storage: 1.5 PB metadata
- Write QPS: 3.5K uploads + 67K interactions = 70.5K
- Read QPS: 867K video views

Configuration:
- 1,500 nodes globally (1 TB per node)
- Replication factor: 3
- Consistency: QUORUM for writes, ONE for reads

Cost: $210K/month (i3.2xlarge × 1,500)

Justification:
"70.5K write QPS exceeds PostgreSQL limit (10K). 1.5 PB storage
requires distribution. Time-series queries optimal for Cassandra."
```

2. PostgreSQL (Users):

```
Numbers:
- Storage: 10 TB (1B users)
- Write QPS: ~100 (user registrations)
- Read QPS: 50K (with 95% cache hit)

Configuration:
- 1 master + 10 read replicas
- Shard by user_id: 10 shards × 1 TB each
- Instance: db.r5.4xlarge

Cost: $5,500/month

Justification:
"Need ACID for user operations. 10 TB manageable with sharding.
Complex follower queries benefit from PostgreSQL."
```

3. Redis Cluster (Cache):

```
Numbers:
- Cache size: 11 TB (metadata + recommendations)
- QPS: 1.5M reads (video views + feed + interactions)
- Latency requirement: < 10ms

Configuration:
- 200 nodes × 64 GB RAM = 12.8 TB capacity
- Instance: cache.r5.4xlarge per node
- Replication: 1:1 (400 nodes total)
```

```
Cost: $56K/month (with replicas)

Justification:
"1.5M QPS requires distributed cache. 11 TB fits in 200 nodes.
Redis provides sub-millisecond latency for real-time features."
```

4. Elasticsearch (Search):

```
Numbers:
- Documents: 18B (videos + users + hashtags)
- Search QPS: 5K
- Index size: ~2 TB

Configuration:
- 50 nodes (data nodes)
- 3 master nodes
- Instance: r5.2xlarge per node

Cost: $10K/month

Justification:
"Full-text search on captions, hashtags. 5K QPS manageable.
2 TB index fits across 50 nodes."
```

Object Storage:

S3 (Videos):

```
Numbers:
- Storage: 4.8 EB over 5 years
- Ingress: 175 GB/s (uploads)
- Egress: 345 GB/s from origin (5% of views)

Configuration:
- Multi-region buckets (us-west-2, eu-west-1, ap-southeast-1)
- Lifecycle policies (Standard → IA → Glacier)
- Transfer acceleration enabled
- Versioning enabled (for recovery)

Cost: $30.64M/month (with lifecycle)

Justification:
"4.8 EB requires cloud object storage. S3 provides 11-nines durability.
Lifecycle policies save 72% on storage costs."
```

CDN:

CloudFront (Video Delivery):

```
Numbers:
- Bandwidth: 6.5 TB/s from CDN
- Cache hit ratio: 95%
- Edge locations: 400+ globally

Configuration:
- Origin: S3 buckets
- Cache TTL: 24 hours for videos, 5 min for metadata
- Geographic restrictions (country-based blocking)
- Signed URLs for private videos

Cost: $520K/month

Justification:
"6.5 TB/s bandwidth impossible without CDN. 95% cache hit reduces
origin load from 6.9 TB/s to 345 GB/s. Critical for performance."
```

Message Queue:

Kafka (Event Streaming):

```
Numbers:
- Events: 70K/sec (uploads + interactions)
- Topics: video_uploads, likes, comments, shares, views
- Retention: 7 days

Configuration:
- 100 brokers globally (20 regions × 5 brokers)
- Replication factor: 3
- Partitions: 1000 per topic
- Instance: kafka.m5.xlarge

Cost: $14K/month

Justification:
"70K events/sec requires distributed queue. Kafka provides durability
and replay capability. Multiple consumers need same events."
```

Video Processing:

GPU Workers (Transcoding):

```
Numbers:
- Processing: 35K jobs/sec peak
- Transcoding time: 30 sec per video
```

```
Configuration:
- 6,000 workers (p3.2xlarge with V100 GPU)
- Auto-scaling: Scale based on queue depth
- Spot instances: 70% cost savings

Cost: $900K/month (with spot instances)

Justification:
"GPU required for real-time transcoding. 6,000 workers handle 35K
jobs/sec. Spot instances reduce cost by 70%."
```

ML Infrastructure:

Recommendation Engine (GPU Inference):

```
Numbers:
- Inference QPS: 51K
- Model size: 2 GB
- Latency requirement: < 500ms

Configuration:
- 200 GPU servers (p3.2xlarge with V100)
- Model serving: TensorFlow Serving
- Batch size: 100 users per inference
- Auto-scaling enabled

Cost: $280K/month

Justification:
"51K inferences/sec requires GPU. Batching reduces servers from 510
to 102. 200 servers with 50% headroom for spikes."
```

Model Training:

```
Configuration:
- Training cluster: 100 p3.16xlarge (8× V100 GPUs)
- Training frequency: Daily
- Training duration: 6 hours
- Cost: $60K/month (only run 6 hours/day)

Data pipeline:
- Feature store: Feast
- Data lake: S3 (100B interactions = 10 TB)
- ML framework: TensorFlow / PyTorch
```

Feed Generation Algorithm

Detailed Implementation

Algorithm Flow:

```
User opens app → Request feed → Execute pipeline:

┌─────────────────────────────────────────────────────────┐
│ STEP 1: Candidate Generation (Parallel)                 │
├─────────────────────────────────────────────────────────┤
│ Source 1: Collaborative Filtering                       │
│   - Query similar users from Redis/Cassandra            │
│   - Get videos they watched (last 24 hours)             │
│   - Deduplicate and score                               │
│   - Time: 100ms, Returns: 400 videos                    │
│                                                         │
│ Source 2: Content-Based Filtering                       │
│   - Get user's liked videos (Redis)                     │
│   - Find similar videos by hashtags/music (Elasticsearch)  │
│   - Time: 100ms, Returns: 300 videos                    │
│                                                         │
│ Source 3: Trending/Viral                                │
│   - Redis Sorted Set: "trending:global"                 │
│   - Scored by: views × like_ratio × recency             │
│   - Time: 50ms, Returns: 200 videos                     │
│                                                         │
│ Source 4: Following Feed                                │
│   - Get followed users (PostgreSQL/Redis cache)         │
│   - Get their recent videos (Cassandra)                 │
│   - Time: 50ms, Returns: 100 videos                     │
│                                                         │
│ Total Time: 100ms (parallel execution)                  │
│ Total Candidates: 1,000 videos                          │
└─────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────┐
│ STEP 2: ML Ranking                                      │
├─────────────────────────────────────────────────────────┤
│ 1. Batch candidates (1,000 videos)                      │
│ 2. Fetch user embedding (from cache or compute)         │
│ 3. Fetch video embeddings (from cache)                  │
│ 4. Compute similarity scores (GPU batch inference)      │
│ 5. Apply engagement multiplier                          │
│ 6. Sort by final score                                  │
│                                                         │
│ Time: 400ms                                             │
│ Output: Ranked list of 1,000 videos                     │
└─────────────────────────────────────────────────────────┘

┌─
```

```
┌──────────────────────────────────────────────────────────┐
│ STEP 3: Post-Processing & Business Rules                 │
├──────────────────────────────────────────────────────────┤
│ 1. Remove already-seen videos (bloom filter)             │
│ 2. Apply diversity rules:                                │
│    - Max 2 consecutive videos from same creator          │
│    - Max 3 videos with same music in top 20              │
│    - Mix categories (not all dance or all comedy)        │
│ 3. Inject fresh content (cold start problem):            │
│    - 10% new videos (< 1 hour old)                       │
│    - Give new creators a chance                          │
│ 4. Apply user filters (blocked users, hidden content)    │
│ 5. Select top 50 videos for this session                 │
│                                                          │
│ Time: 100ms                                              │
│ Output: Final feed of 50 videos                          │
└──────────────────────────────────────────────────────────┘

Total Time: 100ms + 400ms + 100ms = 600ms ✓
```

Caching Strategy:

```
1. Pre-computed Candidates (30-minute TTL):
   - Cache: "user:{user_id}:candidates"
   - Value: List of 500 video_ids
   - Storage: 500M users × 500 × 16 bytes = 4 TB
   - Hit rate: 70% (user returns within 30 min)

2. Video Embeddings (24-hour TTL):
   - Cache: "video:{video_id}:embedding"
   - Value: 256 floats × 4 bytes = 1 KB
   - Storage: 20M hot videos × 1 KB = 20 GB
   - Hit rate: 90%

3. User Embeddings (1-hour TTL):
   - Cache: "user:{user_id}:embedding"
   - Value: 256 floats = 1 KB
   - Storage: 500M users × 1 KB = 500 GB
   - Hit rate: 80%
```

Feedback Loop:

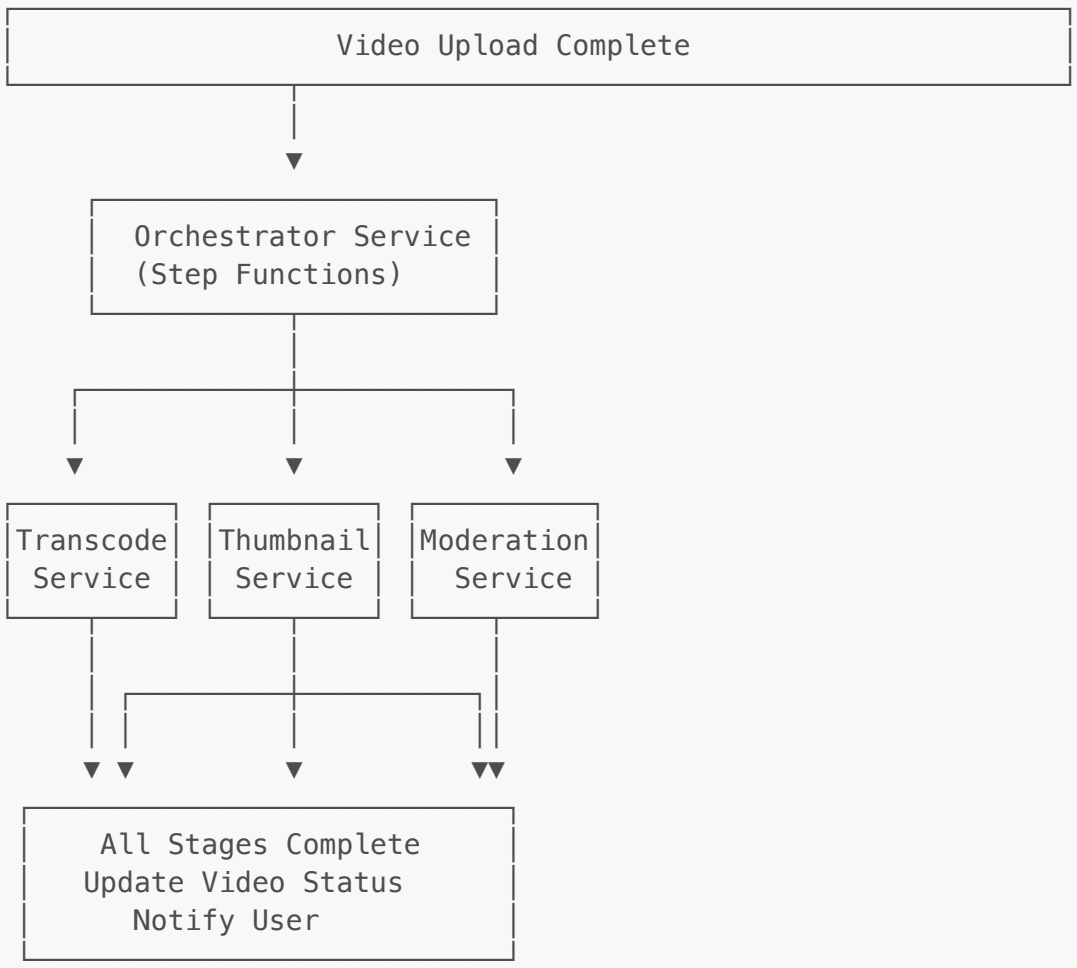
```
Track user behavior:
- Video watched > 50% → Positive signal
- Video skipped < 3 sec → Negative signal
- Video liked → Strong positive signal
- Video shared → Very strong positive signal

Update user embedding:
```

```
- Real-time update (incremental)
- Or batch update every hour
- Store in Redis for fast access

Feed into training pipeline:
- Collect 100B daily interactions
- Retrain model daily
- A/B test new model (5% traffic)
- Roll out if metrics improve
```

Video Processing Pipeline

Detailed Processing Architecture

Architecture Pattern: Saga Pattern (Distributed Transaction)



Orchestrator Implementation:

```
// AWS Step Functions state machine
{
```

```
    "StartAt": "ValidateVideo",
    "States": {
      "ValidateVideo": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:validate-video",
        "Next": "ParallelProcessing"
      },
      "ParallelProcessing": {
        "Type": "Parallel",
        "Branches": [
          {
            "StartAt": "TranscodeVideo",
            "States": {
              "TranscodeVideo": {
                "Type": "Task",
                "Resource": "arn:aws:lambda:transcode",
                "End": true
              }
            }
          },
          {
            "StartAt": "GenerateThumbnails",
            "States": {
              "GenerateThumbnails": {
                "Type": "Task",
                "Resource": "arn:aws:lambda:thumbnails",
                "End": true
              }
            }
          },
          {
            "StartAt": "ModerateContent",
            "States": {
              "ModerateContent": {
                "Type": "Task",
                "Resource": "arn:aws:lambda:moderate",
                "End": true
              }
            }
          }
        ],
        "Next": "UpdateMetadata"
      },
      "UpdateMetadata": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:update-metadata",
        "Next": "NotifyUser"
      },
      "NotifyUser": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:notify",
        "End": true
      }
```

```
  }
}
```

Transcoding Service (FFmpeg):

```
# Transcode to multiple qualities
ffmpeg -i input.mp4 \
  -c:v libx264 -preset fast \
  -s 1920x1080 -b:v 4M output_1080p.mp4 \
  -s 1280x720 -b:v 2M output_720p.mp4 \
  -s 854x480 -b:v 1M output_480p.mp4 \
  -s 640x360 -b:v 500k output_360p.mp4

Performance:
- GPU (NVIDIA V100): 30 seconds per video
- CPU (c5.4xlarge): 120 seconds per video
- Decision: GPU (4x faster, worth the cost)
```

Content Moderation:

```
Multi-stage approach:

1. Pre-upload (Client-side):
   - Hash comparison with known bad content
   - Block immediately if match

2. Automated (ML Models):
   - Google Cloud Vision API
   - AWS Rekognition
   - Custom trained models
   - Confidence score: > 0.8 = auto-block, 0.5-0.8 = review queue

3. Human Review:
   - Flagged content reviewed by moderators
   - Decision: Approve, Remove, Age-restrict
   - Feedback loop: Train models

4. User Reports:
   - Users flag inappropriate content
   - Priority queue for review
```

Scalability & Performance

Horizontal Scaling

Video Upload Service:

```
Bottleneck: Upload bandwidth (175 GB/s peak)

Solution:
- Deploy to 20 regions globally
- Region selection: User's nearest region
- Per region: 175 GB/s ÷ 20 = 8.75 GB/s
- Instances per region: 20 servers
- Total: 400 servers globally

Auto-scaling:
- Metric: CPU > 70% OR network > 5 GB/s
- Scale up: Add 10% capacity
- Scale down: Remove idle instances (traffic < 30%)
```

Feed Service:

```
Bottleneck: ML inference (51K QPS)

Solution:
- 200 GPU servers (p3.2xlarge)
- Batch inference: 100 users per batch
- Model caching: Keep model in GPU memory
- Result caching: Pre-computed recommendations

Auto-scaling:
- Metric: GPU utilization > 80%
- Scale up: Add 10% capacity (20 servers)
- Scale down: Remove if < 40% utilization
```

Video Streaming:

```
Bottleneck: Network bandwidth (6.9 TB/s)

Solution:
- CDN with 400+ edge locations
- 95% cache hit ratio
- Adaptive bitrate streaming (ABR)
- Video preloading (next 2 videos)

Optimization:
- Preload strategy: Load 720p by default
- Upgrade to 1080p if bandwidth sufficient
- Downgrade to 480p/360p if network slow
- Save 30% bandwidth with smart quality selection
```

Database Scaling

Cassandra Scaling:

```
Current: 1,500 nodes
Growth: 100M videos/day

Scaling plan:
- Year 1: 1,500 nodes (1.5 PB)
- Year 2: 3,000 nodes (3 PB)
- Year 3: 4,500 nodes (4.5 PB)

Linear scaling: Each node adds 1 TB storage, 100 write QPS, 1K read QPS
```

PostgreSQL Scaling:

```
Current: 10 shards

Scaling strategy:
- Add shards as user base grows
- 100M users per shard = 1 TB
- Year 1: 10 shards (1B users)
- Year 2: 15 shards (1.5B users)

Follower graph optimization:
- Cache follower lists in Redis
- Materialize counts (avoid COUNT queries)
- Denormalize: Store follower/following count in user table
```

Redis Scaling:

```
Current: 200 nodes (12.8 TB)

Scaling triggers:
- Memory > 80%: Add 10% capacity (20 nodes)
- QPS > 1M per node: Add nodes
- Latency > 10ms: Check for hot keys

Hot key problem:
- Viral video getting millions of views
- Single key (video:{id}:views) overloaded
- Solution: Shard counter across 10 keys
  * video:{id}:views:0 through video:{id}:views:9
  * Aggregate on read
```

Caching Strategy

Multi-Level Caching:

```
Level 1: Client Cache (Mobile App)
- Cache last 20 watched videos
- Cache user profile
- Cache following list
- Size: ~500 MB per app
- Reduces: 20% of requests

Level 2: CDN Edge Cache
- Cache popular videos (95% hit rate)
- Cache thumbnails
- Cache video metadata
- Size: 800 GB per edge × 200 edges = 160 TB
- Reduces: 95% of video streaming requests

Level 3: Redis Cache (Application Layer)
- Cache video metadata
- Cache user profiles
- Cache pre-computed recommendations
- Size: 11 TB distributed
- Reduces: 80% of database queries

Level 4: Database (Source of Truth)
- Cassandra for video metadata
- PostgreSQL for users
- Elasticsearch for search
```

Cache Invalidation:

```
Video published:
- Invalidate: user's video list cache
- Invalidate: followers' following feed cache
- Add to: trending cache (if applicable)

User updates profile:
- Invalidate: user profile cache
- Invalidate: all videos by this user (user info embedded)

Video engagement updated:
- Update: video stats cache (batched every 10 sec)
- Update: trending rankings (every 5 min)
```

Performance Optimizations

Video Preloading:

```
Strategy:
- When user starts watching video[0], preload video[1] and video[2]
```

```
- Preload 720p by default (8 MB)
- Cancel preload if user skips

Benefits:
- Instant playback for next videos
- Smooth user experience
- 90% of users watch preloaded videos

Cost:
- 25B views/day, preload 2 videos each = 50B preloads
- 30% actually watched = 15B wasted
- 15B × 8 MB = 120 PB/day wasted bandwidth
- With CDN: 120 PB × 0.05 = 6 PB/day from origin
- Cost: 6 PB/day × $0.08/GB = $480K/day = $14.4M/month

Worth it?
- Improved user engagement > bandwidth cost
- TikTok/Reels core feature
- Accept the cost for better UX
```

Adaptive Bitrate Streaming (ABR):

```
Detect network speed:
- < 1 Mbps: Serve 360p (1.5 MB)
- 1–3 Mbps: Serve 480p (3 MB)
- 3–5 Mbps: Serve 720p (8 MB)
- > 5 Mbps: Serve 1080p (20 MB)

Implementation:
- Client measures download speed
- Requests appropriate quality from CDN
- Smooth transition between qualities

Savings:
- 40% of users on slow connections
- Serve 480p (3 MB) instead of 720p (8 MB)
- Bandwidth reduction: 40% × (8–3)/8 = 25% savings
- Savings: 6.9 TB/s × 0.25 = 1.7 TB/s
- Cost savings: $13M/month
```

Security & Privacy

Content Security

1. Video Upload Validation:

```

```

```
Client-side checks:
- File size < 500 MB
- Duration 15-60 seconds
- Supported formats: MP4, MOV, AVI
- Resolution: 720p-4K

Server-side checks:
- Re-validate all client checks (don't trust client)
- Scan for malware
- Check file hash against blacklist
- Verify video codec (prevent exploits)
```

2. Content Moderation:

```
Automated moderation (99% coverage):
- ML models detect inappropriate content
- Confidence > 0.8: Auto-block
- Confidence 0.5-0.8: Flag for human review
- Confidence < 0.5: Allow (monitor)

Human moderation (1% flagged content):
- Queue system for moderators
- Priority: User-reported > Auto-flagged > Random sampling
- Moderators: 5,000 globally (24/7 coverage)
- Average review time: 30 seconds
- Capacity: 5,000 × 120 reviews/hour = 600K reviews/day
```

3. Copyright Protection:

```
Audio fingerprinting:
- Compare uploaded audio against library of copyrighted music
- Use: Shazam-like algorithm (acoustic fingerprinting)
- Database: 100M songs
- Match time: < 1 second

Actions:
- Exact match: Block upload or mute audio
- Partial match: Flag for review
- No match: Allow

Video fingerprinting:
- Compare against copyrighted video content
- Use: Perceptual hashing
- Less common (mostly for clips from movies/TV)
```

User Privacy

Data Privacy:

```
User controls:
- Private account (only followers see videos)
- Hide from search
- Blocked users list
- Download restrictions

Data collected:
- Watch history (for recommendations)
- Device info (for security)
- Location (optional, for local content)
- Interaction data (likes, comments)

Data retention:
- Active user data: Indefinite
- Deleted user data: 30-day grace period
- Video analytics: 2 years
- Comply with GDPR, CCPA
```

Access Control:

```
Video visibility levels:
1. Public: Anyone can view
2. Friends: Only followers can view
3. Private: Only mentioned users can view
4. Unlisted: Only with link can view

Implementation:
- Check in video_permissions table
- Cache permissions in Redis (TTL: 5 min)
- Enforce at CDN level (signed URLs)
```

Rate Limiting:

```
Limits per user:
- Video uploads: 10/hour, 50/day
- Comments: 100/hour
- Likes: 500/hour
- Follow/unfollow: 200/hour
- API requests: 1000/hour

Implementation:
- Redis sliding window counter
- Return 429 (Too Many Requests) if exceeded
- Block abusive users (temp or permanent)
```

Authentication & Authorization

Authentication:

```
Methods:
- Email + Password (hashed with bcrypt)
- Phone + OTP
- Social login (Google, Facebook, Apple)

Token management:
- JWT tokens (access + refresh)
- Access token: 1 hour expiry
- Refresh token: 30 days expiry
- Store refresh tokens in Redis

Session management:
- Allow 5 concurrent sessions per user
- Kill oldest session if limit exceeded
```

Authorization:

```
Permissions:
- Read public videos: Any user (even anonymous)
- Upload video: Authenticated user
- Comment: Authenticated user
- Moderate content: Admin/Moderator role
- Delete any video: Admin role

Implementation:
- Role-Based Access Control (RBAC)
- Permissions stored in JWT claims
- Validated at API gateway
```

Monitoring & Analytics

System Monitoring

Key Metrics:

```
Infrastructure:
- API latency (p50, p95, p99)
- Error rate (4xx, 5xx)
- QPS per service
- CPU/Memory utilization
- Network bandwidth
- Database query latency
```

```
Business:
- Daily Active Users (DAU)
- Video upload count
- Video view count
- Engagement rate (likes/comments/shares per view)
- Average watch time
- Retention rate (D1, D7, D30)

Performance:
- Video load time (p95 < 2 seconds)
- Feed generation time (p95 < 1 second)
- CDN cache hit ratio (target: 95%)
- Video processing time (target: < 60 seconds)
```

Monitoring Stack:

```
Metrics Collection:
- Prometheus for time-series metrics
- Pull model: Scrape /metrics endpoint every 15s
- Store: 30 days retention
- Size: ~100 TB (1000 servers × 100 MB/day × 30 days)

Visualization:
- Grafana dashboards
- Real-time graphs
- Alerting rules

Logging:
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Centralized logging from all services
- Log level: INFO in prod, DEBUG in staging
- Retention: 7 days
- Size: ~50 TB/day

Tracing:
- Jaeger for distributed tracing
- Trace every 1% of requests (sampling)
- Identify slow services
```

Alerting:

```
Critical alerts (page on-call):
- API error rate > 1%
- Video upload service down
- Database unavailable
- CDN cache hit rate < 80%
```

```
Warning alerts (email/Slack):
- API latency p95 > 3 seconds
- Database query latency > 100ms
- Processing queue depth > 100K
- Disk usage > 80%

Auto-remediation:
- Scale up if CPU > 80% for 5 minutes
- Restart unhealthy instances
- Failover to standby database
```

Analytics & Insights

Video Analytics (for Creators):

```
Metrics provided:
- Total views
- Unique viewers
- Average watch time
- Completion rate
- Likes, comments, shares
- Follower growth
- Traffic sources (For You, Following, Profile, Search)
- Geographic distribution
- Age/gender demographics (aggregated)

Storage:
- Time-series data in Cassandra
- Partition by (creator_id, date, video_id)
- Aggregate hourly → daily → weekly
```

Platform Analytics:

```
Track:
- User engagement trends
- Popular hashtags (real-time)
- Viral videos (detection algorithm)
- Creator growth
- Content categories trending
- Regional preferences

Use cases:
- Improve recommendation algorithm
- Identify trending topics
- Content moderation priorities
- Business strategy
```

Real-time Analytics Pipeline:

```
Events → Kafka → Stream Processing → Analytics DB

Stream processing (Apache Flink):
- Window: 5-minute tumbling windows
- Aggregations: COUNT, SUM, AVG
- Output: Metrics to time-series DB

Example:
- Count views per video per 5 minutes
- Detect spike: 10x normal views
- Mark as "trending"
- Boost in recommendations
```

Trade-offs & Design Decisions

1. Cassandra vs PostgreSQL for Video Metadata

Decision: Cassandra

```
Pros:
✓ Handles 70K write QPS (PostgreSQL max 10K)
✓ Linear scalability (add nodes = add capacity)
✓ No single point of failure
✓ Optimized for time-series queries
✓ Eventual consistency acceptable for video metadata

Cons:
✗ No ACID transactions
✗ Limited query flexibility (no JOINs)
✗ More operational complexity

Justification:
"Video metadata is 1.5 PB with 70K write QPS. PostgreSQL would need
150 shards (impossible to manage). Cassandra scales linearly and
handles write-heavy workload. Eventual consistency acceptable
(view counts can be slightly delayed)."
```

2. Pre-computation vs Real-time Feed Generation

Decision: Hybrid Approach

```
Pre-computation:
✓ Latency: 200ms vs 900ms (4.5x faster)
✓ Cost: $35K/month for 8 TB Redis
```

```
✓ User experience: Near-instant feed

Real-time:
✓ Fresh content (captures latest uploads)
✓ Personalization (reflects recent interactions)
✓ No stale recommendations

Hybrid:
- Pre-compute top 500 candidates (every 30 min)
- Real-time ranking of candidates (400ms)
- Mix in fresh videos (last hour)
- Best of both worlds

Result: 600ms latency, fresh content, lower cost
```

3. CDN Push vs Pull

Decision: Pull (with smart preloading)

```
Pull (chosen):
✓ Simpler implementation
✓ Automatic cache population
✓ CDN manages eviction
✗ Cold start problem (first user fetches from origin)

Push:
✓ Proactive caching (new viral videos)
✗ Complex prediction (which videos will go viral?)
✗ Wasted bandwidth (push videos that never get watched)

Hybrid solution:
- Default: Pull (CDN fetches on demand)
- For viral videos (detected by spike): Push to all edges
- For new videos from mega-creators (>1M followers): Pre-push
- 5% of videos pre-pushed, saves 10% of cold starts
```

4. Processing: Sync vs Async

Decision: Async (with progress updates)

```
Async (chosen):
✓ Better user experience (upload completes quickly)
✓ Scalable (workers can be added independently)
✓ Fault tolerant (retry failed jobs)
✗ Delayed publishing (30-second processing)

Implementation:
- Upload returns immediately with video_id
```

```
- Show "Processing..." status in app
- WebSocket update when ready
- Send push notification: "Your video is live!"

User sees:
1. Upload button → "Uploading..." (10 sec)
2. "Upload complete! Processing..." (30 sec)
3. "Your video is live!" (push notification)

Total wait: 40 seconds (acceptable)
```

5. Relational vs Graph Database for Social Graph

Decision: PostgreSQL (with denormalization)

```
PostgreSQL (chosen):
✓ Simpler (existing infrastructure)
✓ ACID guarantees for followers
✓ Sufficient performance with caching
✗ JOIN queries for multi-hop (followers of followers)

Graph DB (Neo4j):
✓ Natural fit for social graph
✓ Fast multi-hop queries
✓ Better for "suggested follows" feature
✗ Additional system to maintain
✗ Learning curve for team

Compromise:
- Use PostgreSQL for direct relationships (followers)
- Cache heavily in Redis
- For complex graph queries: Pre-compute and cache
  * "Followers you may know" computed daily
  * Stored in Redis with 24-hour TTL
```

6. Strong vs Eventual Consistency

Decisions:

```
Strong Consistency (PostgreSQL):
- User registration
- User authentication
- Payment processing
- Account deletion

Justification:
"These operations must be correct. Can't have duplicate usernames
or incorrect charges. ACID guarantees essential."
```

```
Eventual Consistency (Cassandra):
- Video metadata
- Likes, comments, shares
- View counts
- Following feed

Justification:
"User won't notice if like count is delayed by 1 second. Priority
is low latency and high throughput. Eventual consistency acceptable."

Real-time (Redis):
- Video streaming
- Online presence
- Typing indicators
- Live engagement counters

Justification:
"Must be real-time for good UX. In-memory cache provides
sub-millisecond latency."
```

Complete Technology Stack Summary

Infrastructure Costs (Monthly)

Component	Cost	Justification
S3 (Videos)	$30.64M	4.8 EB with lifecycle
CDN (CloudFront)	$520K	6.5 TB/s bandwidth
Cassandra (1,500 nodes)	$210K	Video metadata
Redis (400 nodes)	$56K	11 TB cache
PostgreSQL (10 shards)	$5.5K	User data
Elasticsearch (50 nodes)	$10K	Search
Kafka (100 brokers)	$14K	Event streaming
GPU Workers (6,000)	$900K	Video transcoding
ML Inference (200 GPUs)	$280K	Recommendations
ML Training (100 GPUs)	$60K	Daily model training
App Servers (400)	$60K	API handling
Monitoring (ELK, etc)	$20K	Logs, metrics, traces
Total	$32.77M	Monthly cost
Annual	$393M	Yearly cost
Cost per DAU	$0.066	Per user per month
Revenue needed	$0.15+	For profitability

```
Revenue sources:
```

```
- Advertising: $0.10 per DAU per month
- In-app purchases: $0.03 per DAU per month
- Creator subscriptions: $0.02 per DAU per month
- Total: $0.15 per DAU per month (break even)
```

Technology Stack Summary

Layer	Technology
Mobile Apps	Swift (iOS), Kotlin (Android)
Web App	React + TypeScript
API Gateway	Kong / AWS API Gateway
Backend Services	Go + Node.js
Video Storage	AWS S3 (multi-region)
Video Delivery	CloudFront CDN (400+ edges)
Video Processing	FFmpeg on GPU workers (p3.2xlarge)
Database (Metadata)	Cassandra (1,500 nodes)
Database (Users)	PostgreSQL (10 shards)
Cache	Redis Cluster (400 nodes)
Search	Elasticsearch (50 nodes)
Message Queue	Kafka (100 brokers)
ML Framework	TensorFlow / PyTorch
ML Inference	TensorFlow Serving (200 GPUs)
Monitoring	Prometheus + Grafana
Logging	ELK Stack
Tracing	Jaeger
Load Balancer	AWS ALB / NGINX
Container Orch.	Kubernetes (EKS)
CI/CD	Jenkins / GitHub Actions

Deployment Architecture

Multi-Region Deployment

```
Regions: 3 primary + 2 DR (Disaster Recovery)

Primary Regions:
1. US-West-2 (Oregon)
   - 40% of global traffic
   - Users: North America, South America

2. EU-West-1 (Ireland)
   - 30% of global traffic
   - Users: Europe, Middle East, Africa
```

```
3. AP-Southeast-1 (Singapore)
   - 30% of global traffic
   - Users: Asia, Australia

DR Regions:
4. US-East-1 (Virginia) - Hot standby for US-West-2
5. EU-Central-1 (Frankfurt) - Hot standby for EU-West-1

Per-region setup:
- Full application stack
- Cassandra cluster (500 nodes)
- Redis cluster (130 nodes)
- PostgreSQL replica
- Processing workers (2,000)
```

Data Replication:

```
Cassandra:
- Multi-datacenter replication
- Replication factor: 3 per region
- Cross-region replication: Async
- Conflict resolution: Last-write-wins

PostgreSQL:
- Master in US-West-2
- Read replicas in all regions
- Cross-region replication: Streaming replication
- Failover: Promote replica to master (5-minute RTO)

S3:
- Cross-Region Replication (CRR)
- Replicate to 3 regions
- Async replication (seconds delay)
```

Traffic Routing:

```
DNS:
- Route53 with geolocation routing
- Direct users to nearest region
- Health checks: Failover if region unhealthy

API Gateway:
- Regional endpoints
- Automatic failover to DR region
- Latency-based routing as fallback
```

Advanced Features

1. Live Streaming

Requirements:

```
- Support 1M concurrent live streams
- Latency: < 5 seconds (Low-Latency HLS)
- Viewers per stream: 100-100K
```

Architecture:

```
Ingest:
- RTMP from mobile client
- Ingest servers: 500 globally (2K streams each)

Processing:
- Transcode to HLS segments (4-second chunks)
- Multiple qualities (360p, 480p, 720p)
- Real-time transcoding (< 5 sec latency)

Distribution:
- HLS segments stored in S3
- CDN delivers segments to viewers
- Manifest file (.m3u8) updated every 4 seconds

Chat:
- WebSocket for live comments
- Redis Pub/Sub for message distribution
- Max 100K concurrent viewers per stream
```

2. Duet/Stitch (Collaborative Videos)

Feature:

```
Duet: Record video alongside another video (split screen)
Stitch: Use 5-second clip from another video in yours

Implementation:
- Store reference to original video
- Render combined video on-the-fly or pre-render
- Track attribution (original creator credited)
- Viral mechanism (original video gets exposure)
```

3. Effects & Filters

Client-side:

```
Real-time effects during recording:
- Beauty filters
- Face tracking (AR effects)
- Background removal
- Color grading

Technology:
- Mobile GPU (Core ML on iOS, ML Kit on Android)
- Real-time processing (60 fps)
```

Server-side:

```
Advanced effects (applied after upload):
- AI-powered background replacement
- Super-resolution upscaling
- Noise reduction
- Auto-captioning (speech-to-text)

Technology:
- GPU workers
- Pre-trained models
- Optional feature (premium users)
```

4. Monetization Features

Creator Fund:

```
Pay creators based on:
- Views (weighted by watch time)
- Engagement (likes, comments, shares)
- Quality score (completion rate)

Calculation:
- Pool: $200M/month
- Top 1M creators share pool
- Formula: earnings = (views × engagement × quality) / total_points
- Average: $200/month per creator

Implementation:
- Track metrics in analytics DB
- Calculate earnings monthly
- Payout via Stripe/PayPal
```

Advertising:

```
Ad types:
- In-feed ads (every 5th video)
- Branded hashtag challenges
- Creator partnerships

Targeting:
- Demographics (age, gender, location)
- Interests (based on watch history)
- Lookalike audiences

Ad serving:
- Insert ads into feed (deterministic placement)
- Track impressions, clicks, conversions
- Real-time bidding (RTB)
- Return to Cassandra for billing
```

Disaster Recovery & Business Continuity

Backup Strategy

Database Backups:

```
Cassandra:
- Snapshot every 24 hours
- Incremental snapshots every hour
- Store in S3 Glacier
- Retention: 30 days
- Size: 1.5 PB × 30 = 45 PB
- Cost: 45 PB × $0.004/GB = $180K/month

PostgreSQL:
- Point-in-time recovery (PITR)
- WAL archiving to S3
- Retention: 30 days
- Size: 10 TB × 30 = 300 TB
- Cost: $7K/month

Redis:
- RDB snapshots every 12 hours
- AOF log for durability
- Replicas in different AZs
```

Video Backup:

```
Strategy:
- Primary: S3 in us-west-2
```

```
- Replica 1: S3 in eu-west-1 (CRR)
- Replica 2: S3 in ap-southeast-1 (CRR)

Cost:
- 4.8 EB × 3 regions = 14.4 EB total
- With lifecycle policies: $91.9M/month
- Worth it for 11-nines durability
```

Disaster Recovery Plan

RTO/RPO Targets:

```
RTO (Recovery Time Objective):
- Critical services: 15 minutes
- Non-critical services: 1 hour

RPO (Recovery Point Objective):
- User data: 5 minutes (streaming replication)
- Video metadata: 15 minutes (Cassandra snapshots)
- Videos: 0 (replicated to 3 regions)
```

Failover Procedures:

```
Scenario: US-West-2 region failure

Automatic failover:
1. Route53 detects unhealthy region (30 sec)
2. Redirect traffic to US-East-1 (DR region)
3. Promote PostgreSQL replica to master (5 min)
4. Cassandra continues (multi-region cluster)
5. Redis: Fail over to DR cluster (1 min)

Total downtime: ~6 minutes (within 15-min RTO)

Manual steps:
- Investigate root cause
- Spin up processing workers in DR region
- Update DNS for permanent failover
- Restore from backup if data corruption
```

Scalability Milestones

Growth Projections & Scaling Plan

Year 1: 500M DAU

```
Infrastructure:
- 400 app servers
- 1,500 Cassandra nodes
- 400 Redis nodes
- 10 PostgreSQL shards
- 6,000 GPU workers
- 200 ML servers

Cost: $393M/year
Revenue: $900M/year (@ $0.15/user/month)
Profit: $507M/year
```

Year 2: 1B DAU (2x growth)

```
Scaling changes:
- 800 app servers (+100%)
- 3,000 Cassandra nodes (+100%)
- 800 Redis nodes (+100%)
- 15 PostgreSQL shards (+50%)
- 12,000 GPU workers (+100%)
- 400 ML servers (+100%)

Cost: $786M/year
Revenue: $1.8B/year
Profit: $1.01B/year

Bottlenecks:
- Video processing (GPU workers)
- ML inference (GPU servers)
- Solution: Continue adding capacity linearly
```

Year 3: 1.5B DAU (3x growth)

```
Scaling changes:
- 1,200 app servers
- 4,500 Cassandra nodes
- 1,200 Redis nodes
- 20 PostgreSQL shards
- 18,000 GPU workers
- 600 ML servers

Cost: $1.18B/year
Revenue: $2.7B/year
Profit: $1.52B/year

Optimizations at scale:
- Custom silicon (like Google TPU) for ML
- Owned CDN infrastructure (like Netflix Open Connect)
```

```
- Better compression codecs (AV1 instead of H.264)
- Reduce costs by 30% at this scale
```

Security Best Practices

DDoS Protection

Layers of Defense:

```
Layer 1: CDN (CloudFront)
- Shield Standard (free)
- Shield Advanced ($3K/month)
- WAF rules
- Rate limiting at edge

Layer 2: API Gateway
- Rate limiting per IP: 100 req/sec
- Rate limiting per user: 50 req/sec
- Geo-blocking (block suspicious regions)
- Request size limits

Layer 3: Application
- Authentication required for writes
- Captcha for suspicious activity
- Behavioral analysis (bot detection)
```

DDoS Scenarios:

```
Scenario 1: Video view bot (inflate view counts)
Detection:
- Same IP viewing 1000s of videos/hour
- Unusual patterns (no likes, no comments, perfect completion rate)
- IP reputation check

Action:
- Block IP at CDN level
- Invalidate bot views (don't count toward metrics)
- Add to blacklist

Scenario 2: Comment spam
Detection:
- Duplicate comments
- Posting > 100 comments/hour
- Similar text pattern

Action:
- Shadow ban (user sees comments, others don't)
```

```
- Require captcha for next N comments
- Temp account suspension
```

Trade-offs & Design Decisions Summary

Major Design Decisions

1. Database Choices

```
✓ Cassandra for video metadata
  Reason: 1.5 PB storage, 70K write QPS, time-series queries

✓ PostgreSQL for users
  Reason: ACID for auth, 10 TB manageable, complex queries

✓ Redis for real-time
  Reason: Sub-ms latency, 11 TB cache, 1.5M QPS
```

2. Video Delivery

```
✓ CDN with 95% cache hit
  Reason: 6.9 TB/s bandwidth impossible without CDN
  Savings: $79M/month vs serving from origin

✓ Adaptive bitrate streaming
  Reason: Save 25% bandwidth, better UX on slow networks

✓ Video preloading
  Reason: Instant playback for next videos (core UX feature)
```

3. Recommendation System

```
✓ Two-tower neural network
  Reason: Scales to billions of user-video pairs

✓ Hybrid pre-computation + real-time ranking
  Reason: 600ms latency (4.5x faster than pure real-time)

✓ Multiple candidate sources
  Reason: Diversity + discovery (not just personalization)
```

4. Video Processing

```
✓ Async processing with progress updates
  Reason: Better UX, scalable, fault tolerant
  Trade-off: 30-second delay acceptable

✓ GPU for transcoding
  Reason: 4x faster than CPU, worth cost

✓ Parallel pipeline stages
  Reason: 30 sec total vs 38 sec sequential
```

5. Consistency Models

```
✓ Eventually consistent for engagement metrics
  Reason: Low latency > perfect accuracy
  Trade-off: View counts may be 1-2 seconds delayed

✓ Strongly consistent for authentication
  Reason: Security critical, can't compromise

✓ Real-time for video streaming
  Reason: Core user experience
```

Interview Talking Points

Key Numbers to Remember

```
Scale:
- 500M DAU
- 100M video uploads/day
- 25B video views/day
- 867K peak QPS

Storage:
- 4.8 EB over 5 years
- 3.25 PB/day new content
- 1.5 PB metadata

Technology:
- Cassandra: 1,500 nodes for video metadata
- PostgreSQL: 10 shards for users
- Redis: 400 nodes for 11 TB cache
- CDN: 6.5 TB/s bandwidth
- GPU workers: 6,000 for transcoding
- ML servers: 200 GPUs for recommendations

Cost:
```

```
- $32.77M/month total
- $0.066 per user per month
- Revenue needed: $0.15+ for profitability
```

Architecture Highlights

1. Read-Heavy System (250:1 ratio)

```
"This is an extremely read-heavy system. 25 billion video views
vs 100 million uploads = 250:1 ratio. This drives our key decisions:

- CDN with 95% cache hit (reduces origin load 20x)
- Redis caching at 3 levels (client, CDN, application)
- Read replicas for PostgreSQL
- Cassandra optimized for read throughput"
```

2. Personalization at Scale

```
"The recommendation engine is the differentiator:

- Two-tower neural network (scales to billions)
- Hybrid approach: Pre-compute + real-time ranking
- 600ms latency for feed generation
- 200 GPU servers for 51K inferences/sec
- Daily model retraining on 100B interactions

This is what makes TikTok addictive - the 'For You' feed!"
```

3. Video Processing Pipeline

```
"Video processing is critical but can be async:

- 35K jobs/sec peak processing load
- 6,000 GPU workers (FFmpeg on NVIDIA V100)
- 30-second processing time (parallel execution)
- Spot instances save 70% on GPU costs

Users wait 40 seconds total (upload + processing) - acceptable
for the quality of experience."
```

4. Global Distribution

```
"Multi-region deployment is essential:
```

```
- 3 primary regions (US, EU, Asia)
- 400+ CDN edge locations
- Geolocation-based routing
- Cross-region replication for DR

Latency improvements:
- Video load time: < 2 seconds globally
- Feed generation: < 1 second
- API latency: < 100ms"
```

Future Enhancements

Phase 2 Features

1. Advanced AI Features

```
- Auto-captioning (speech-to-text)
- Multi-language subtitles
- Object detection for tagging
- Scene detection for highlights
- Voice cloning for dubbing
- AI-generated summaries
```

2. Creator Tools

```
- Video editor in-app
- Analytics dashboard (detailed)
- Monetization tools (tipping, subscriptions)
- Collaboration features (co-creators)
- Brand partnership marketplace
- Content scheduling
```

3. Social Features

```
- Group challenges
- Collaborative playlists
- Watch parties (sync viewing)
- Gifting system
- Badges and achievements
- Creator rankings/leaderboards
```

4. Discovery Improvements

```
```

```
- Visual search (search by image)
- Voice search
- AR lens catalog
- Geo-based discovery (nearby videos)
- Time-based discovery (trending now)
- Mood-based recommendations
```

Technical Improvements

1. Cost Optimizations

```
- Custom video codec (better compression)
- AV1 codec (40% smaller than H.264)
- Custom CDN infrastructure
- Own GPU clusters (vs cloud)
- Estimated savings: 30% ($10M/month)
```

2. Performance Improvements

```
- WebAssembly for client-side effects
- HTTP/3 for faster uploads
- Machine learning model optimization (quantization)
- Database query optimization
- Reduce feed latency to 300ms
```

3. Machine Learning Enhancements

```
- Transformer-based models (better accuracy)
- Reinforcement learning (optimize engagement)
- Multi-modal learning (video + audio + text)
- Real-time model updates
- Personalized model per region
```

Summary & Conclusion

System Overview

Short-Form Video Platform (TikTok/Reels) Design:

```
Scale: 500M DAU, 25B video views/day, 100M uploads/day
Pattern: Read-heavy (250:1), ML-driven personalization
Storage: 4.8 EB over 5 years
```

```
Cost: $393M/year ($0.066 per user/month)

Core Components:
1. Video Upload: Chunked upload → S3 → Async processing
2. Video Processing: GPU transcoding (30 sec) + moderation
3. Video Delivery: CDN (95% hit) + ABR + preloading
4. Feed Generation: ML ranking (600ms) + caching
5. Social Interactions: Redis counters + Cassandra storage
6. Search: Elasticsearch for full-text + hashtags
7. Recommendations: Two-tower NN + daily retraining

Technology Stack:
- Database: Cassandra (1,500 nodes), PostgreSQL (10 shards)
- Cache: Redis (400 nodes, 11 TB)
- Storage: S3 (4.8 EB with lifecycle)
- CDN: CloudFront (6.5 TB/s)
- Queue: Kafka (100 brokers)
- Processing: 6,000 GPU workers
- ML: 200 GPU inference servers

Key Optimizations:
- CDN caching: Saves $79M/month (72% of bandwidth cost)
- Video preloading: Instant playback (core UX)
- Hybrid recommendations: 4.5x faster than real-time
- Adaptive bitrate: 25% bandwidth savings
- Spot instances: 70% savings on GPU costs
```

Interview Success Criteria

What Interviewers Want to See:

✅ **Scale Calculations** (Show the numbers!)

- Correctly calculate QPS from DAU
- Estimate storage with different qualities
- Calculate bandwidth with CDN
- Size cache using 80-20 rule

✅ **System Architecture** (Draw the diagram!)

- Upload → Process → Store → Deliver flow
- Identify key components (feed, social, search)
- Show data flow between services
- Explain caching at multiple levels

✅ **Database Choices** (Justify with numbers!)

- Why Cassandra for metadata (70K write QPS, 1.5 PB)
- Why PostgreSQL for users (ACID, 10 TB)
- Why Redis for cache (1.5M QPS, sub-ms latency)
- When to use each technology

✅ **Bottlenecks & Solutions** (Think critically!)

- Identify: Video bandwidth (6.9 TB/s)
- Solution: CDN with 95% cache hit
- Identify: Feed generation latency (900ms)
- Solution: Pre-computation + caching

✅ **Trade-offs** (Show decision-making!)

- Async processing (speed vs delay)
- Pre-computation (latency vs freshness)
- Eventual consistency (performance vs accuracy)
- GPU vs CPU (speed vs cost)

Final Checklist

Before ending the interview:

```
□ Calculated traffic (QPS) for all operations
□ Estimated storage with retention period
□ Calculated bandwidth and justified CDN
□ Sized cache using concrete numbers
□ Chose appropriate databases with justification
□ Designed API endpoints
□ Explained feed generation algorithm
□ Discussed video processing pipeline
□ Addressed scalability concerns
□ Mentioned security/privacy considerations
□ Provided cost estimates
□ Compared to real systems (TikTok, Reels)
```

References

Real-World Systems

TikTok (Actual Numbers):

```
DAU: 1B
Videos uploaded: 200M+/day
Infrastructure: ByteDance private cloud
CDN: Multiple providers
Recommendation: Custom ML models
Cost: Estimated $2B+/year
```

Instagram Reels:

```
DAU: 500M (within Instagram's 2B)
Integration: Part of Instagram app
Infrastructure: Facebook/Meta infrastructure
CDN: Facebook's edge network
Recommendation: Shared with Instagram feed
```

Further Reading

Books:

1. "Designing Data-Intensive Applications" - Martin Kleppmann
2. "System Design Interview" - Alex Xu (Volume 1 & 2)
3. "Machine Learning Systems Design" - Chip Huyen

Papers:

1. "Recommender Systems Handbook" - Ricci et al.
2. "Deep Neural Networks for YouTube Recommendations" - Google
3. "Wide & Deep Learning for Recommender Systems" - Google

Blogs:
4. TikTok Engineering Blog
5. Instagram Engineering Blog
6. Netflix Tech Blog (for video streaming insights)
7. Uber Engineering Blog (for ML at scale)

Online Resources:
8. AWS Architecture Blog (video streaming patterns)
9. High Scalability Blog (case studies)
10. Martin Fowler's Blog (architecture patterns)

END OF DOCUMENT

This comprehensive HLD covers the complete design of a short-form video platform like TikTok or Instagram Reels, from initial requirements through detailed component design, scalability planning, and cost analysis. Use this as a reference for system design interviews and real-world implementations.

Good luck with your system design interview!