Netflix Video Streaming System Design - High-Level Design (HLD)

Table of Contents

Problem Statement

Design a global video streaming platform like Netflix that allows users to:

- Browse and search for movies, TV shows, documentaries
- Stream video content with adaptive quality based on network conditions
- Download content for offline viewing
- Get personalized recommendations
- Resume watching across multiple devices
- Watch with subtitles in multiple languages
- Support profiles for different family members
- Handle live streaming events

Scale Requirements

- **200 million subscribers worldwide**
- **125 million hours watched per day**
- **15,000+ titles in catalog**
- **200 countries/regions served**
- **Peak traffic: 3x average (weekend evenings)**
- **Support 4K HDR streaming**

Functional Requirements

Must Have (P0)

1. **User Management**

- User registration and subscription management
- Multiple profiles per account (up to 5)
- Parental controls and content ratings
- Watch history and continue watching

2. **Content Browsing**
    - Browse by category, genre, trending
    - Search for content by title, actor, director
    - Content details (synopsis, cast, ratings, reviews)
    - Personalized homepage with recommendations
3. **Video Streaming**
    - Adaptive bitrate streaming (ABR)
    - Multiple quality levels (360p, 480p, 720p, 1080p, 4K)
    - Support various codecs (H.264, H.265, VP9, AV1)
    - Resume playback across devices
    - Skip intro/credits functionality
4. **Content Management (Admin)**
    - Upload and encode video content
    - Manage metadata (title, description, cast, genre)
    - Content scheduling and regional availability
    - A/B testing for thumbnails and recommendations
5. **Download for Offline**
    - Download content to mobile devices
    - Manage storage and expiration
    - DRM protection for downloads

Nice to Have (P1)

- Live streaming support
- Interactive content (choose your own adventure)
- Watch party (synchronized viewing with friends)
- User reviews and ratings
- Content recommendations via email/push notifications
- Behind-the-scenes bonus content
- Multiple audio tracks and subtitle languages

Non-Functional Requirements

Performance

- **Video start time**: < 2 seconds (p99)
- **Rebuffering ratio**: < 0.5% of play time

- **Search latency**: < 300ms
- **Homepage load time**: < 1 second
- **Video quality**: Adapt within 10 seconds of bandwidth change

Scalability

- Handle 200M+ subscribers
- Support 100M+ concurrent streams at peak
- Store 100+ petabytes of encoded video
- Process 1000+ hours of new content per week
- Support global traffic distribution

Availability

- **99.99% uptime** for streaming service
- **99.9% uptime** for catalog/browse features
- Graceful degradation during regional failures
- Multi-region active-active deployment

Consistency

- **Eventual consistency** for watch history (acceptable)
- **Strong consistency** for payment/subscription
- **Causal consistency** for recommendations

Security

- **DRM** (Digital Rights Management) for content protection
- **SSL/TLS** for all communications
- **Token-based authentication** with expiration
- **Geographic content restrictions** (licensing)
- **Concurrent stream limits** based on subscription tier

Quality of Experience (QoE)

- **98%+ video start success rate**
- **Adaptive bitrate** to network conditions
- **Preloading** for seamless playback
- **Intelligent buffering** strategy

Capacity Estimation

Traffic Estimates

```
Total Subscribers: 200M
Daily Active Users (DAU): 100M (50% engagement)
Average viewing time per user: 1.25 hours/day
```

```
Total hours watched per day: 125M hours

Concurrent viewers at peak: 30M (30% of DAU)
Average video bitrate: 3 Mbps (adaptive)
Peak bandwidth: 30M × 3 Mbps = 90 Tbps (11.25 TB/s)
```

Storage Estimates

```
Total content: 15,000 titles
Average movie length: 2 hours
Average TV series: 10 episodes × 45 min = 7.5 hours

Total content hours: ~100,000 hours

Encoding formats (per video):
- 4K (25 Mbps): 11.25 GB/hour
- 1080p (5 Mbps): 2.25 GB/hour
- 720p (2.5 Mbps): 1.125 GB/hour
- 480p (1 Mbps): 450 MB/hour
- 360p (0.5 Mbps): 225 MB/hour

Storage per hour: ~15 GB (all formats)
Total storage: 100,000 hours × 15 GB = 1.5 PB

With 5 years of content and multiple language tracks:
Total storage: ~10 PB
```

Bandwidth Estimates

```
Peak concurrent streams: 30M
Average bitrate: 3 Mbps

Peak outgoing bandwidth: 30M × 3 Mbps = 90 Tbps

Daily bandwidth:
- Average streams: 10M concurrent
- 10M × 3 Mbps × 24 hours = 32 PB/day
```

CDN & Caching

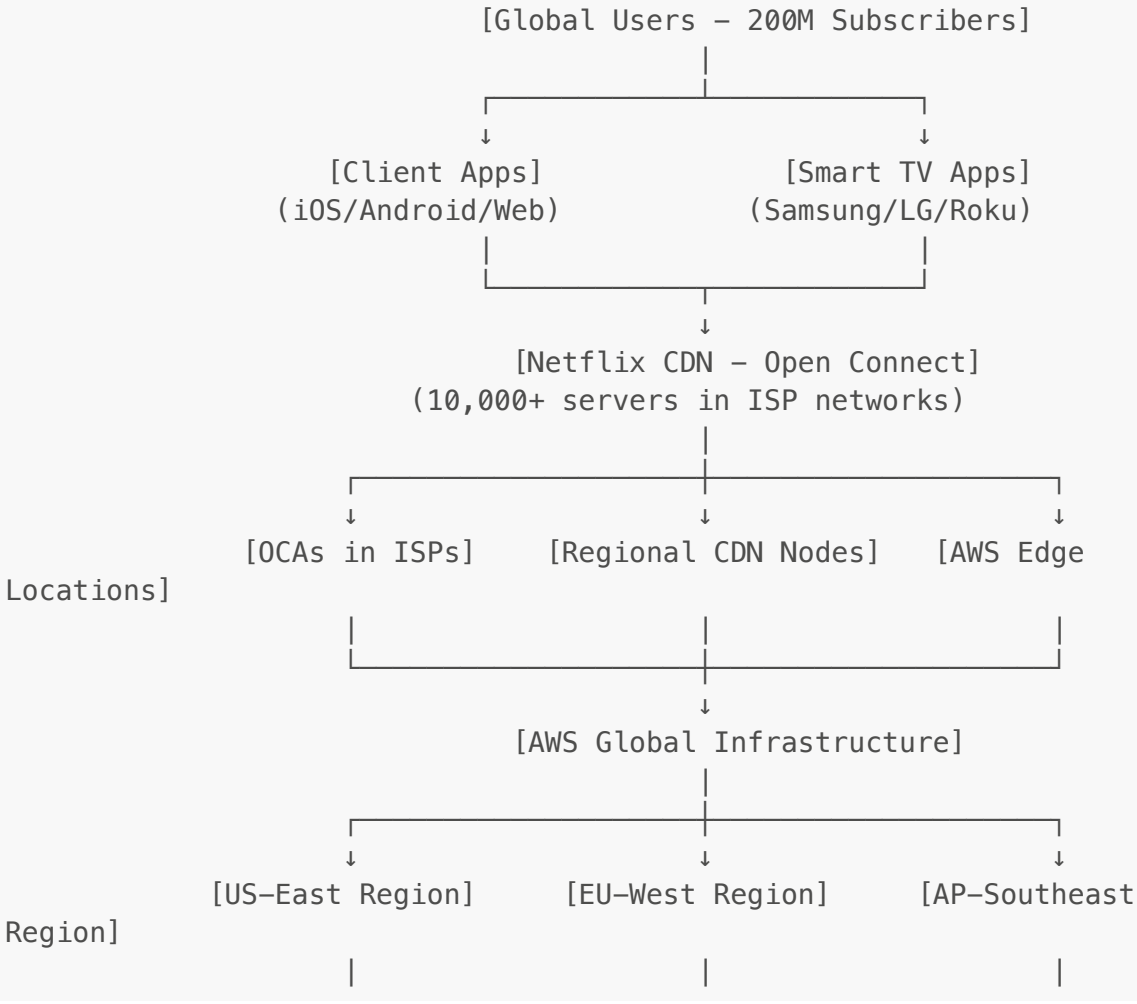
```
Popular content (top 20%): 80% of views (Pareto principle)
Cache hit ratio target: 95%

Cache storage per region:
- Top 20% of content = 2 PB
```

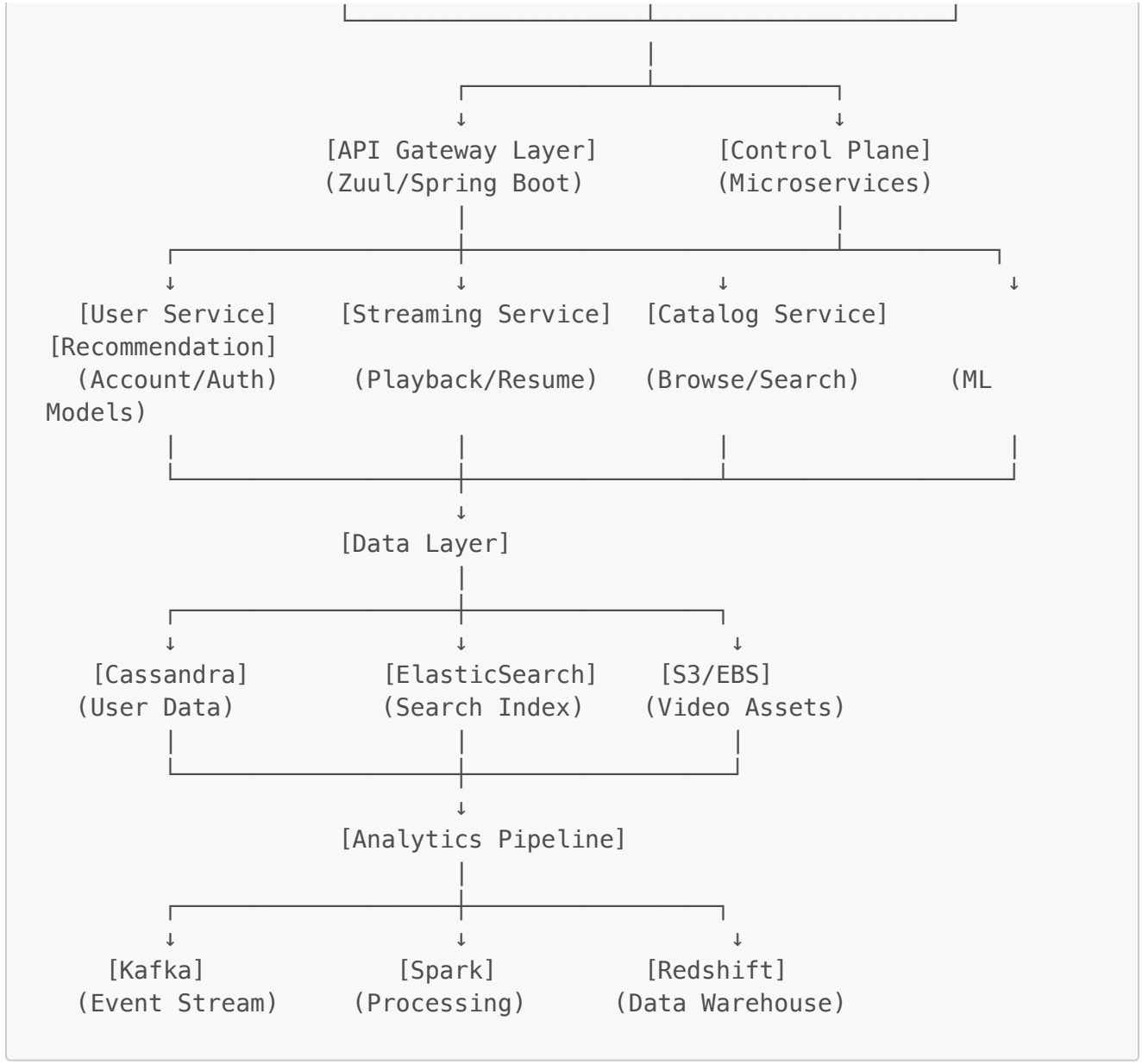
```
- Distribute across 100 regions
- 20 TB per region

Open Connect Appliances (OCAs):
- ~10,000 OCAs globally
- 100 TB storage each
- Serve 90% of traffic from ISP caches
```

Encoding Pipeline

```
New content per week: 1000 hours
Encoding time per hour (all formats): 4 hours
Parallelization factor: 20x

Required encoding capacity:
- 1000 hours × 4 hours = 4000 compute hours/week
- With 20x parallelization: 200 hours/week
- ~30 hours/day continuous encoding
```

High-Level Architecture





Core Components

1. Netflix Open Connect (CDN)

Purpose: Global content delivery network for video streaming

Architecture:

- **Open Connect Appliances (OCAs)**: Custom servers deployed in ISP networks
- **10,000+ OCAs** worldwide in 1000+ locations
- **100 TB storage** per OCA
- **Serve 90%+ traffic** from ISP-level caches

Benefits:

- **Reduced latency**: Content served from nearest ISP
- **Cost savings**: Minimize internet transit costs
- **Better QoE**: Higher cache hit ratios

- **ISP partnerships**: Reduced network congestion

How it works:

1. ISPs host Netflix OCAs in their networks
2. Popular content pre-positioned on OCAs daily
3. User streams are served from nearest OCA
4. Cache misses filled from regional AWS locations
5. Fill requests during off-peak hours

Technology: Custom Linux-based appliances with FreeBSD, NGINX

2. API Gateway (Zuul)

Purpose: Edge service for routing, authentication, and filtering

Responsibilities:

- **Request routing** to appropriate microservices
- **Authentication & authorization** via JWT tokens
- **Rate limiting** per user and device
- **Dynamic routing** based on A/B tests
- **Request/response transformation**
- **Circuit breaking** for fault isolation
- **Metrics collection** and logging

Features:

- **Multi-region active-active** deployment
- **Intelligent failover** to backup regions
- **Origin shield** to protect backend services
- **Request validation** and sanitization

Technology: Netflix Zuul 2 (async non-blocking), Spring Boot

3. Microservices Architecture

Core Services:

A. User Service

- User registration and authentication
- Profile management (5 profiles per account)
- Subscription management and billing
- Parental controls and content restrictions
- Watch history and viewing preferences

Database: Cassandra (user profiles), MySQL (billing)

B. Streaming Service

- Video playback session management
- Adaptive bitrate (ABR) logic
- Resume playback across devices
- Concurrent stream limit enforcement
- Playback quality monitoring
- Subtitle and audio track selection

Technology: Node.js, Go for high-performance streaming logic

C. Catalog Service

- Content metadata management
- Browse and discovery APIs
- Genre categorization
- Content availability by region
- Release date and scheduling
- Artwork and thumbnail management

Database: ElasticSearch (search), Cassandra (metadata)

D. Recommendation Service

- Personalized recommendations per profile
- Similar content suggestions
- Trending content by region
- Continue watching suggestions
- "Because you watched X" recommendations
- ML model serving infrastructure

Technology: Python (ML models), Java (serving), TensorFlow

E. Encoding Service

- Video transcoding pipeline
- Generate multiple bitrates and resolutions
- Apply DRM protection
- Generate thumbnails and preview clips
- Quality validation and artifact detection
- Parallel encoding for faster processing

Technology: AWS Elemental MediaConvert, FFmpeg, custom encoders

F. Search Service

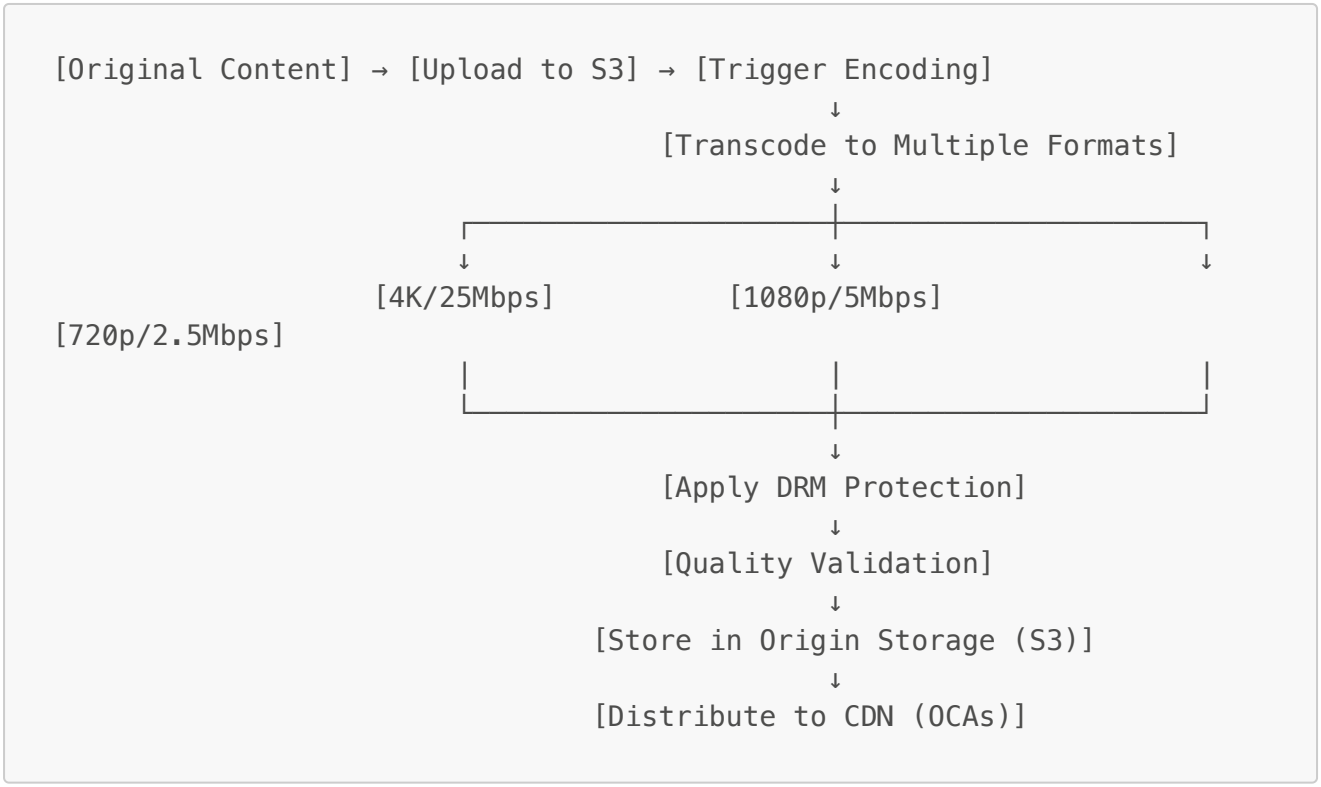
- Full-text search across titles
- Actor, director, genre search
- Fuzzy matching and typo tolerance
- Search suggestions and autocomplete

- Personalized search ranking

Database: ElasticSearch cluster

4. Video Encoding Pipeline

Architecture:



Encoding Formats:

- **Video codecs**: H.264 (compatibility), H.265 (4K), VP9, AV1 (future)
- **Audio codecs**: AAC, Dolby Digital, Dolby Atmos
- **Container format**: MP4, CMAF (Common Media Application Format)
- **Streaming protocol**: HLS, DASH

Quality Levels:

```
4K:    3840×2160 @ 25 Mbps (HDR10, Dolby Vision)
1080p: 1920×1080 @ 5 Mbps
720p:  1280×720  @ 2.5 Mbps
480p:  854×480   @ 1 Mbps
360p:  640×360   @ 0.5 Mbps

Audio tracks: Multiple languages, 5.1 surround, Atmos
Subtitles: 20+ languages per title
```

Encoding Process:

1. Content uploaded by studios to S3

2. Metadata extraction (resolution, bitrate, codec)
3. Parallel encoding jobs for each quality level
4. Quality validation (VMAF score, artifact detection)
5. DRM encryption (Widevine, PlayReady, FairPlay)
6. Generate thumbnails (every 5 seconds)
7. Store encoded assets in origin storage
8. Trigger CDN distribution to OCAs
9. Update catalog database with asset URLs
10. Notify content team of completion

Technology:

- AWS Elemental MediaConvert
- FFmpeg for custom processing
- Netflix's internal encoding tools (Archer, Cosmos)
- Distributed task processing with Titus (container platform)

5. Adaptive Bitrate (ABR) Streaming

Purpose: Deliver optimal video quality based on network conditions

How ABR Works:

1. Video encoded in multiple bitrates (ladder)
2. Client measures available bandwidth
3. Client requests appropriate bitrate segment
4. Switch bitrate mid-stream based on buffer level
5. Prioritize avoiding rebuffering over quality

Bitrate Ladder:

```
Resolution  Bitrate    Use Case
──────────────────────────────────
4K          25 Mbps    Premium, fast networks
1080p       5 Mbps     HD streaming
720p        2.5 Mbps   Standard quality
480p        1 Mbps     Mobile, slow networks
360p        0.5 Mbps   Very slow networks
```

ABR Algorithm:

```
def select_bitrate(bandwidth, buffer_level, current_bitrate):
    """
    Select optimal bitrate based on network conditions
    """
    # Conservative: keep buffer above 10 seconds
    safe_buffer_threshold = 10  # seconds
```

```
    # Aggressive upshift if buffer > 20 seconds
    aggressive_threshold = 20   # seconds

    if buffer_level < safe_buffer_threshold:
        # Downshift to prevent rebuffering
        return max(current_bitrate / 2, MIN_BITRATE)

    elif buffer_level > aggressive_threshold and bandwidth >
current_bitrate * 1.5:
        # Upshift if bandwidth allows
        return min(current_bitrate * 2, MAX_BITRATE)

    else:
        # Maintain current bitrate
        return current_bitrate
```

Netflix's Innovations:

- **Per-title encoding**: Custom bitrate ladder per title
- **Dynamic optimizer**: ML-based bitrate selection
- **Buffer-based ABR**: Prioritize rebuffer avoidance
- **Network quality prediction**: Anticipate bandwidth changes

6. Content Recommendation Engine

Purpose: Personalize homepage and suggest relevant content

Recommendation Types:

1. **Personalized Rows**: "Top Picks for You", "Trending Now"
2. **Similar Content**: "Because you watched X"
3. **Genre-based**: "Action Movies", "Korean Dramas"
4. **Continue Watching**: Resume partially watched content
5. **New Releases**: Recently added content
6. **Popular in Your Region**: Localized recommendations

ML Models:

Collaborative Filtering

```
Find similar users based on viewing history
Recommend content that similar users enjoyed
```

Content-Based Filtering

```

```

```
Analyze content features (genre, cast, director)
Recommend similar content
```

Deep Learning Models

```
Neural networks trained on:
- User viewing history
- Engagement metrics (completion rate, rewatch)
- Time of day, device type
- Search history, rating history
- A/B test performance data
```

Features Used:

- User demographics (age, location, language)
- Viewing history and watch time
- Rating and review history
- Search queries
- Abandoned content (didn't finish)
- Time of day and day of week
- Device type (mobile, TV, desktop)
- Social features (watch party participation)

Model Training Pipeline:

```
[User Events] → [Kafka] → [Spark Processing] → [Feature Store]
                                                      ↓
                                                [Model Training]
                                                (TensorFlow/PyTorch)
                                                      ↓
                                                [Model Evaluation]
                                                (Offline A/B Tests)
                                                      ↓
                                                [Model Deployment]
                                                (Online A/B Tests)
                                                      ↓
                                                [Production Serving]
```

Ranking Algorithm:

```
Score = weighted_sum([
    predicted_rating × 0.3,
    completion_probability × 0.2,
    engagement_likelihood × 0.2,
```

```
    novelty_score × 0.15,
    recency_factor × 0.1,
    popularity_score × 0.05
])
```

Personalization at Scale:

- Pre-compute recommendations hourly
- Cache top 100 recommendations per user
- Real-time updates for critical events (new content added)
- Different models for different regions
- Continuous A/B testing of algorithms

7. Data Storage Layer

A. Cassandra

Use Cases:

- User profiles and preferences
- Viewing history (time-series data)
- Device registration
- Watch position (resume data)
- Subscription information

Why Cassandra:

- **High write throughput**: Millions of playback events/sec
- **Time-series data**: Natural fit for viewing history
- **Linear scalability**: Add nodes horizontally
- **Multi-region replication**: Global active-active
- **No single point of failure**: Peer-to-peer architecture

Data Model:

```
Table: viewing_history
Partition Key: user_id
Clustering Key: timestamp
Columns: title_id, watch_duration, quality_level, device_id
```

B. ElasticSearch

Use Cases:

- Content search and discovery
- Title, actor, director search
- Autocomplete suggestions

- Faceted search (filter by genre, year, rating)

Index Structure:

```
{
  "title_id": "tt1234567",
  "title": "Stranger Things",
  "type": "series",
  "genres": ["sci-fi", "horror", "thriller"],
  "cast": ["Millie Bobby Brown", "Finn Wolfhard"],
  "director": "Duffer Brothers",
  "release_year": 2016,
  "rating": 8.7,
  "synopsis": "...",
  "available_regions": ["US", "UK", "CA"]
}
```

C. MySQL/Aurora

Use Cases:

- Billing and payment processing
- Subscription management
- Transaction records
- Financial reporting

Why SQL:

- **ACID transactions**: Critical for payments
- **Complex queries**: Reporting and analytics
- **Data integrity**: Foreign key constraints

D. S3/EBS

Use Cases:

- Encoded video files
- Thumbnails and artwork
- Subtitles and metadata
- Original content (archival)

Storage Structure:

```
s3://netflix-content/
├── originals/
│   └── title_id/
│       └── master.mp4
├── encoded/
```

```
│   └── title_id/
│       ├── 4k/
│       ├── 1080p/
│       ├── 720p/
│       └── 480p/
├── thumbnails/
│   └── title_id/
│       ├── thumbnail_1.jpg
│       └── thumbnail_2.jpg
└── subtitles/
    └── title_id/
        ├── en.vtt
        └── es.vtt
```

8. Message Queue (Kafka)

Purpose: Event streaming and asynchronous processing

Use Cases:

- **Viewing events**: Play, pause, stop, seek
- **Quality metrics**: Rebuffering, bitrate changes
- **Recommendation events**: Click, impression, rating
- **Encoding jobs**: Trigger and status updates
- **Analytics pipeline**: Feed data warehouse

Topics:

```
playback-events: Video playback lifecycle
quality-metrics: QoE measurements
recommendation-events: User interactions
search-queries: Search behavior
encoding-jobs: Content processing pipeline
```

Consumers:

- Real-time analytics dashboard
- Recommendation model training
- Data warehouse loading
- Alerting and monitoring
- A/B test analysis

9. Analytics & Data Science Platform

Architecture:

```

```

```
[Client Events] → [Kafka] → [Spark Streaming] → [Real-time Dashboards]
                              ↓
                        [HDFS/S3 Data Lake]
                              ↓
                        [Spark Batch Jobs]
                              ↓
                        [Redshift/Snowflake]
                              ↓
                        [BI Tools (Tableau)]
```

Metrics Tracked:

- **Engagement**: Hours watched, completion rate, rewatch rate
- **Quality**: Rebuffering ratio, video start time, playback failures
- **Content**: Most watched titles, trending content, regional preferences
- **Business**: Subscriber growth, churn rate, revenue
- **Technical**: API latency, error rates, CDN performance

Real-time Dashboards:

- Concurrent viewers globally
- Stream quality metrics by region
- Error rates and alerts
- CDN cache hit ratios
- Top trending content

Database Design

User Profile Schema (Cassandra)

```
CREATE TABLE user_profiles (
    user_id UUID PRIMARY KEY,
    email TEXT,
    subscription_tier TEXT,  -- basic, standard, premium
    billing_date TIMESTAMP,
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE profiles (
    profile_id UUID PRIMARY KEY,
    user_id UUID,
    profile_name TEXT,
    avatar_url TEXT,
    is_kids_profile BOOLEAN,
    language_preference TEXT,
    created_at TIMESTAMP
```

```
);

CREATE TABLE viewing_history (
    profile_id UUID,
    timestamp TIMESTAMP,
    title_id TEXT,
    watch_duration INT,  -- seconds
    total_duration INT,
    device_id TEXT,
    quality_level TEXT,
    PRIMARY KEY (profile_id, timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);

CREATE TABLE resume_positions (
    profile_id UUID,
    title_id TEXT,
    position_seconds INT,
    updated_at TIMESTAMP,
    PRIMARY KEY (profile_id, title_id)
);
```

Content Catalog Schema (ElasticSearch + Cassandra)

```
-- Cassandra for transactional data
CREATE TABLE titles (
    title_id TEXT PRIMARY KEY,
    title TEXT,
    type TEXT,  -- movie, series, documentary
    release_year INT,
    duration_minutes INT,  -- for movies
    num_seasons INT,  -- for series
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE episodes (
    series_id TEXT,
    season_number INT,
    episode_number INT,
    episode_id TEXT,
    title TEXT,
    duration_minutes INT,
    PRIMARY KEY (series_id, season_number, episode_number)
);

CREATE TABLE title_metadata (
    title_id TEXT PRIMARY KEY,
    synopsis TEXT,
    director TEXT,
    cast LIST<TEXT>,
```

```
    genres LIST<TEXT>,
    content_rating TEXT,  -- G, PG, PG-13, R, TV-MA
    available_regions SET<TEXT>,
    audio_languages LIST<TEXT>,
    subtitle_languages LIST<TEXT>
);

CREATE TABLE video_assets (
    title_id TEXT,
    quality_level TEXT,  -- 4k, 1080p, 720p, 480p, 360p
    video_url TEXT,
    bitrate INT,  -- kbps
    codec TEXT,  -- h264, h265, vp9
    file_size_bytes BIGINT,
    PRIMARY KEY (title_id, quality_level)
);
```

Recommendation Schema

```
CREATE TABLE user_recommendations (
    profile_id UUID,
    recommendation_type TEXT,  -- top-picks, trending, similar
    position INT,
    title_id TEXT,
    score DOUBLE,
    generated_at TIMESTAMP,
    PRIMARY KEY (profile_id, recommendation_type, position)
) WITH CLUSTERING ORDER BY (position ASC);

CREATE TABLE similar_titles (
    title_id TEXT,
    similar_title_id TEXT,
    similarity_score DOUBLE,
    PRIMARY KEY (title_id, similarity_score, similar_title_id)
) WITH CLUSTERING ORDER BY (similarity_score DESC);
```

Subscription & Billing (MySQL/Aurora)

```
CREATE TABLE subscriptions (
    subscription_id VARCHAR(36) PRIMARY KEY,
    user_id VARCHAR(36) NOT NULL,
    tier ENUM('basic', 'standard', 'premium'),
    status ENUM('active', 'cancelled', 'suspended'),
    start_date DATETIME,
    end_date DATETIME,
    auto_renew BOOLEAN,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
```

```
);

CREATE TABLE payments (
    payment_id VARCHAR(36) PRIMARY KEY,
    user_id VARCHAR(36) NOT NULL,
    amount DECIMAL(10,2),
    currency VARCHAR(3),
    payment_method VARCHAR(50),
    status ENUM('pending', 'completed', 'failed', 'refunded'),
    transaction_date DATETIME,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

API Design

RESTful Endpoints

Authentication APIs

```
POST   /api/v1/auth/register
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh-token
GET    /api/v1/auth/validate-token
```

User & Profile APIs

```
GET    /api/v1/users/{user_id}
PUT    /api/v1/users/{user_id}
GET    /api/v1/users/{user_id}/profiles
POST   /api/v1/users/{user_id}/profiles
PUT    /api/v1/users/{user_id}/profiles/{profile_id}
DELETE /api/v1/users/{user_id}/profiles/{profile_id}
GET    /api/v1/users/{user_id}/subscription
PUT    /api/v1/users/{user_id}/subscription
```

Browse & Discovery APIs

```
GET    /api/v1/browse/home?profile_id={profile_id}
GET    /api/v1/browse/genre/{genre_id}?page=1&limit=20
GET    /api/v1/browse/trending?region={region}
GET    /api/v1/browse/new-releases?page=1
GET    /api/v1/search?q={query}&type=title,actor,director
```

```
GET    /api/v1/titles/{title_id}
GET    /api/v1/titles/{title_id}/similar
```

Playback APIs

```
POST   /api/v1/playback/start
Body: {
  "profile_id": "uuid",
  "title_id": "tt1234567",
  "device_id": "device-123",
  "quality": "auto"
}

Response: {
  "playback_token": "jwt-token",
  "manifest_url": "https://cdn.netflix.com/.../manifest.mpd",
  "license_url": "https://drm.netflix.com/license",
  "subtitle_tracks": [...],
  "audio_tracks": [...]
}

POST   /api/v1/playback/heartbeat
Body: {
  "playback_token": "jwt-token",
  "position_seconds": 1234,
  "quality_level": "1080p",
  "buffer_level": 10
}

POST   /api/v1/playback/stop
DELETE /api/v1/playback/{playback_token}
```

Watch History & Resume APIs

```
GET    /api/v1/profiles/{profile_id}/history?limit=50
GET    /api/v1/profiles/{profile_id}/continue-watching
PUT    /api/v1/profiles/{profile_id}/resume-position
Body: {
  "title_id": "tt1234567",
  "position_seconds": 1234
}
```

Recommendation APIs

```
GET    /api/v1/recommendations/home/{profile_id}
GET    /api/v1/recommendations/similar/{title_id}?profile_id=
{profile_id}
GET    /api/v1/recommendations/trending?region={region}
POST   /api/v1/recommendations/feedback
Body: {
  "profile_id": "uuid",
  "title_id": "tt1234567",
  "interaction": "thumbs_up|thumbs_down|not_interested"
}
```

Download APIs (Mobile)

```
POST   /api/v1/downloads/authorize
GET    /api/v1/downloads/{download_id}/status
GET    /api/v1/downloads/{download_id}/manifest
DELETE /api/v1/downloads/{download_id}
```

API Response Format

```
{
  "status": "success",
  "data": {
    "title_id": "tt1234567",
    "title": "Stranger Things",
    "type": "series",
    "seasons": 4,
    "episodes": 34,
    "rating": 8.7,
    "genres": ["sci-fi", "horror", "drama"],
    "synopsis": "When a young boy vanishes, a small town uncovers a
mystery...",
    "cast": ["Millie Bobby Brown", "Finn Wolfhard", "Winona Ryder"],
    "director": "Duffer Brothers",
    "release_year": 2016,
    "available_regions": ["US", "UK", "CA", "AU"],
    "thumbnail_url": "https://cdn.netflix.com/images/...",
    "trailer_url": "https://cdn.netflix.com/videos/..."
  },
  "metadata": {
    "timestamp": "2025-01-08T12:00:00Z",
    "request_id": "req-abc123"
  }
}
```

Deep Dives

1. Video Playback Flow (End-to-End)

Complete Playback Sequence:

```
1. User Selection
   ↓
[User clicks on "Stranger Things S01E01"]
   ↓
[Client App] → POST /api/v1/playback/start

2. Backend Authorization
   ↓
[Streaming Service]
   - Validates user subscription status
   - Checks concurrent stream limit
   - Verifies content availability in region
   - Checks parental control settings
   ↓
[Generate Playback Token (JWT)]
   - Embed: user_id, profile_id, title_id, device_id
   - Expiration: 6 hours
   ↓
[Query CDN for Manifest URL]
   - Select nearest OCA/CDN node
   - Generate manifest URL (DASH/HLS)
   ↓
[Return Response to Client]

3. Client-Side Streaming
   ↓
[Client receives manifest URL]
   ↓
[Parse Manifest File]
   - Available bitrates: 360p, 480p, 720p, 1080p, 4K
   - Audio tracks: English, Spanish, French
   - Subtitle tracks: 20+ languages
   ↓
[Initial Bitrate Selection]
   - Measure network bandwidth (first 5 seconds)
   - Start with conservative bitrate (720p)
   ↓
[Request Video Segments]
   - Segment size: 4 seconds each
   - Request from CDN/OCA
   ↓
[Adaptive Bitrate Logic]
   - Monitor buffer level
   - Measure download speed
   - Switch bitrate dynamically
```

```
   ↓
[DRM License Request]
   - POST to license server
   - Decrypt segments on-device
   ↓
[Video Playback Begins]

4. Ongoing Monitoring
   ↓
[Heartbeat Events (every 30 seconds)]
   - Current position
   - Quality level
   - Buffer health
   - Rebuffering events
   ↓
[Send to Kafka for Analytics]

5. Playback Completion
   ↓
[User finishes/pauses video]
   ↓
[POST /api/v1/playback/stop]
   - Save resume position
   - Update viewing history
   - Increment watch count
```

ABR Decision Flow:

```
class AdaptiveBitrateController:
    def __init__(self):
        self.buffer_threshold_low = 10  # seconds
        self.buffer_threshold_high = 20  # seconds
        self.bandwidth_history = []

    def select_bitrate(self, current_bitrate, buffer_level,
measured_bandwidth):
        # Add to bandwidth history (last 10 measurements)
        self.bandwidth_history.append(measured_bandwidth)
        if len(self.bandwidth_history) > 10:
            self.bandwidth_history.pop(0)

        # Calculate average bandwidth
        avg_bandwidth = sum(self.bandwidth_history) /
len(self.bandwidth_history)

        # Emergency downshift if buffer critically low
        if buffer_level < 5:
            return self.get_lowest_bitrate()

        # Conservative downshift if buffer low
        if buffer_level < self.buffer_threshold_low:
```

```
            return self.downshift(current_bitrate)

        # Aggressive upshift if buffer healthy and bandwidth allows
        if buffer_level > self.buffer_threshold_high:
            if avg_bandwidth > current_bitrate * 1.5:
                return self.upshift(current_bitrate)

        # Maintain current bitrate
        return current_bitrate

    def downshift(self, current_bitrate):
        bitrate_ladder = [500, 1000, 2500, 5000, 25000]  # kbps
        current_idx = bitrate_ladder.index(current_bitrate)
        if current_idx > 0:
            return bitrate_ladder[current_idx - 1]
        return current_bitrate

    def upshift(self, current_bitrate):
        bitrate_ladder = [500, 1000, 2500, 5000, 25000]  # kbps
        current_idx = bitrate_ladder.index(current_bitrate)
        if current_idx < len(bitrate_ladder) - 1:
            return bitrate_ladder[current_idx + 1]
        return current_bitrate
```

2. Content Distribution Strategy

OCA Deployment Strategy:

```
1. Content Popularity Analysis
   ↓
[Analyze viewing patterns]
   - Top 20% content = 80% of views
   - Regional preferences (K-dramas popular in Asia)
   - Time-based trends (new releases spike)
   ↓
[Generate Distribution Plan]

2. Pre-positioning (Off-Peak Hours)
   ↓
[3 AM - 7 AM Local Time]
   ↓
[Fill OCAs with Popular Content]
   - New releases (100% distribution)
   - Regional favorites (80% distribution)
   - Long-tail content (20% distribution)
   ↓
[Verify integrity]
   - Checksum validation
   - Segment completeness check
```

```
3. Cache Eviction Policy
   ↓
[LRU with Predictive Prefetching]
   - Evict least recently used content
   - Predict upcoming trends (ML-based)
   - Reserve space for scheduled releases
```

Multi-Tier Caching:

```
Layer 1: OCA at ISP (90% hit ratio)
   - 100 TB storage
   - Popular content only
   - Updated nightly
   ↓ (10% miss)
Layer 2: Regional CDN (8% hit ratio)
   - Full catalog
   - Multiple regions
   - S3-backed
   ↓ (2% miss)
Layer 3: Origin Storage (2% hit ratio)
   - Complete catalog
   - S3 across multiple regions
   - Rarely accessed directly
```

3. DRM Implementation

Purpose: Protect content from piracy and unauthorized distribution

Supported DRM Systems:

- **Widevine** (Google - Android, Chrome)
- **PlayReady** (Microsoft - Windows, Xbox, Edge)
- **FairPlay** (Apple - iOS, Safari, Apple TV)

DRM Workflow:

```
1. Content Encryption
   ↓
[Encoding Pipeline]
   - Encrypt video segments with AES-128
   - Generate unique key per title
   - Store keys in secure key server
   ↓
[Encrypted content distributed to CDN]

2. License Request
   ↓
[User initiates playback]
```

```
   ↓
[Client requests DRM license]
POST /drm/license
{
  "playback_token": "jwt-token",
  "device_id": "device-123",
  "drm_system": "widevine"
}
   ↓
[License Server validates]
   - Check subscription status
   - Verify device authorization
   - Check concurrent streams
   - Validate geo-location
   ↓
[Return license with decryption key]
{
  "license": "encrypted-key",
  "expiration": "2025-01-08T18:00:00Z",
  "hdcp_required": true
}

3. Client-Side Decryption
   ↓
[Client decrypts segments]
   - Use hardware-backed security (TEE)
   - Decrypt in memory only
   - Never persist decrypted content
   ↓
[Display video]
```

Security Measures:

- **HDCP** (High-bandwidth Digital Content Protection) for 4K
- **TEE** (Trusted Execution Environment) on devices
- **Watermarking**: Forensic watermarks to trace leaks
- **Device binding**: License tied to specific device
- **License renewal**: Short-lived licenses (6 hours)

4. Offline Download System

Purpose: Allow mobile users to download content for offline viewing

Download Flow:

```
1. Download Request
   ↓
[User selects "Download"]
   ↓
POST /api/v1/downloads/authorize
```

```
{
  "profile_id": "uuid",
  "title_id": "tt1234567",
  "quality": "720p"  // or "auto" based on storage
}
   ↓
[Backend validates]
   - Check subscription tier (downloads allowed?)
   - Verify content is downloadable (licensing)
   - Check device download slots (limit per device)
   ↓
[Generate download manifest]
{
  "download_id": "dl-123abc",
  "segments": [
    "https://cdn.netflix.com/.../segment_001.m4s",
    "https://cdn.netflix.com/.../segment_002.m4s",
    ...
  ],
  "drm_license": "encrypted-license",
  "expiration": "7 days from now"
}

2. Client Downloads
   ↓
[Download manager]
   - Parallel segment downloads (5 concurrent)
   - Resume support on connection failure
   - Queue management
   ↓
[Store encrypted segments]
   - Internal app storage
   - Cannot be accessed by user/other apps
   ↓
[Store DRM license]
   - Encrypted with device key
   - Expiration enforced

3. Offline Playback
   ↓
[User plays downloaded content]
   ↓
[Validate license]
   - Check expiration (7 days)
   - Verify device binding
   - Check offline playback count limit
   ↓
[Decrypt and play]
   - Same as online playback
   - Track offline viewing in queue
   ↓
[Sync watch position when online]
```

Download Limits:

```
Subscription Tier   | Devices | Concurrent Downloads | Expiration
─────────────────────────────────────────────────────────────────
Basic (1 screen)    | 1       | 1                    | 7 days
Standard (2 screens)| 2       | 2                    | 7 days
Premium (4 screens)| 4       | 4                    | 7 days
```

Storage Management:

```python
class DownloadManager:
    def calculate_storage_needed(self, title_id, quality):
        """Calculate storage for download"""
        duration = get_title_duration(title_id)  # minutes

        bitrates = {
            '360p': 0.5,  # Mbps
            '480p': 1.0,
            '720p': 2.5,
            '1080p': 5.0
        }

        bitrate = bitrates[quality]
        storage_mb = (bitrate * 60 * duration) / 8  # Convert to MB

        return storage_mb

    def auto_select_quality(self, title_id, available_storage_mb):
        """Auto-select quality based on available storage"""
        duration = get_title_duration(title_id)

        for quality in ['1080p', '720p', '480p', '360p']:
            needed = self.calculate_storage_needed(title_id, quality)
            if needed <= available_storage_mb:
                return quality

        return None  # Not enough storage
```

5. Search & Discovery Architecture

Search Pipeline:

```
1. User Query
   ↓
[User types "stranger"]
   ↓
```

```
GET /api/v1/search?q=stranger&type=title

2. Query Processing
   ↓
[Search Service]
   - Tokenize query
   - Apply stemming (stranger → strange, strangers)
   - Generate synonyms
   - Detect language
   ↓
[Query ElasticSearch]

3. ElasticSearch Query
   ↓
{
  "query": {
    "multi_match": {
      "query": "stranger",
      "fields": [
        "title^3",       // Boost title matches
        "cast^2",        // Boost cast matches
        "director^2",
        "synopsis"
      ],
      "fuzziness": "AUTO"  // Handle typos
    }
  },
  "filter": {
    "terms": {
      "available_regions": ["US"]
    }
  }
}

4. Personalized Ranking
   ↓
[Re-rank results based on user profile]
   - User viewing history
   - Genre preferences
   - Recent searches
   ↓
Score = elasticsearch_score × 0.5 + personalization_score × 0.5

5. Return Results
   ↓
[Top 20 results returned]
```

Autocomplete Implementation:

```
{
  "suggest": {
```

```
    "title_suggest": {
      "prefix": "stra",
      "completion": {
        "field": "title.completion",
        "size": 5,
        "contexts": {
          "region": ["US"]
        }
      }
    }
  }
}
```

Search Optimizations:

- **Edge n-grams** for partial matching
- **Completion suggester** for autocomplete
- **Fuzzy matching** for typo tolerance (edit distance ≤ 2)
- **Boosting** by popularity and recency
- **Caching** popular queries in Redis (5 min TTL)

6. Live Streaming Support

Architecture:

```
[Live Event] → [Encoder] → [Origin Server] → [CDN/OCA] → [Users]
                                    ↓
                            [DVR Storage (S3)]
```

Challenges:

1. **Low Latency**: Target < 10 seconds delay
2. **Scale**: Millions of concurrent viewers
3. **Quality**: Maintain quality despite network variations
4. **Global**: Simultaneous viewing across time zones

Implementation:

```
1. Ingestion
   ↓
[Live broadcast feed]
   ↓
[Encoder (AWS Elemental Live)]
   - Encode in real-time
   - Multiple bitrates (ABR ladder)
   - Generate 4-second segments
   ↓
[Origin Server]
```

```
   - Receive encoded segments
   - Serve to CDN

2. Distribution
   ↓
[CDN/OCA]
   - Cache live segments
   - Serve to users
   - Low TTL (4 seconds)
   ↓
[Edge servers globally]
   - Minimize propagation delay
   - Regional failover

3. DVR Support
   ↓
[Store segments in S3]
   - Keep last 4 hours
   - Allow rewind/pause
   - Time-shift viewing
   ↓
[Generate on-demand manifest]
   - Convert live to VOD
```

Protocol: HLS or DASH with low-latency extensions (LL-HLS, LL-DASH)

Synchronization:

- Use **NTP** for clock sync across servers
- **Presentation timestamps** in manifest
- Client-side buffer management

Scalability & Reliability

Horizontal Scaling Strategies

1. Application Servers

```
Auto-Scaling Configuration:
- Min instances: 50 per region
- Max instances: 500 per region
- Scale up: CPU > 70% for 3 minutes
- Scale down: CPU < 30% for 10 minutes
- Health check: /health endpoint every 10 seconds
```

2. Database Scaling

Cassandra Cluster:

```
Configuration:
- 100 nodes per region (3 regions)
- Replication factor: 3
- Consistency level: LOCAL_QUORUM
- Partition key: user_id, profile_id
- Compaction strategy: Time-Window (viewing history)

Scaling:
- Add nodes to ring
- Automatic data rebalancing
- No downtime for scaling
```

ElasticSearch Cluster:

```
Configuration:
- 30 data nodes per region
- 3 master nodes per region
- 10 shards per index
- 1 replica per shard

Scaling:
- Add data nodes for storage/query capacity
- Rebalance shards automatically
- Rolling upgrades
```

MySQL/Aurora:

```
Configuration:
- 1 writer instance
- 5 reader instances per region
- Auto-scaling read replicas (based on CPU)
- Failover: < 30 seconds

Scaling:
- Add read replicas for read capacity
- Vertical scaling for writer (rare)
- Database proxy (RDS Proxy) for connection pooling
```

3. CDN Scaling

Open Connect:

- **10,000+ OCAs globally**
- **Automatic traffic distribution** based on load

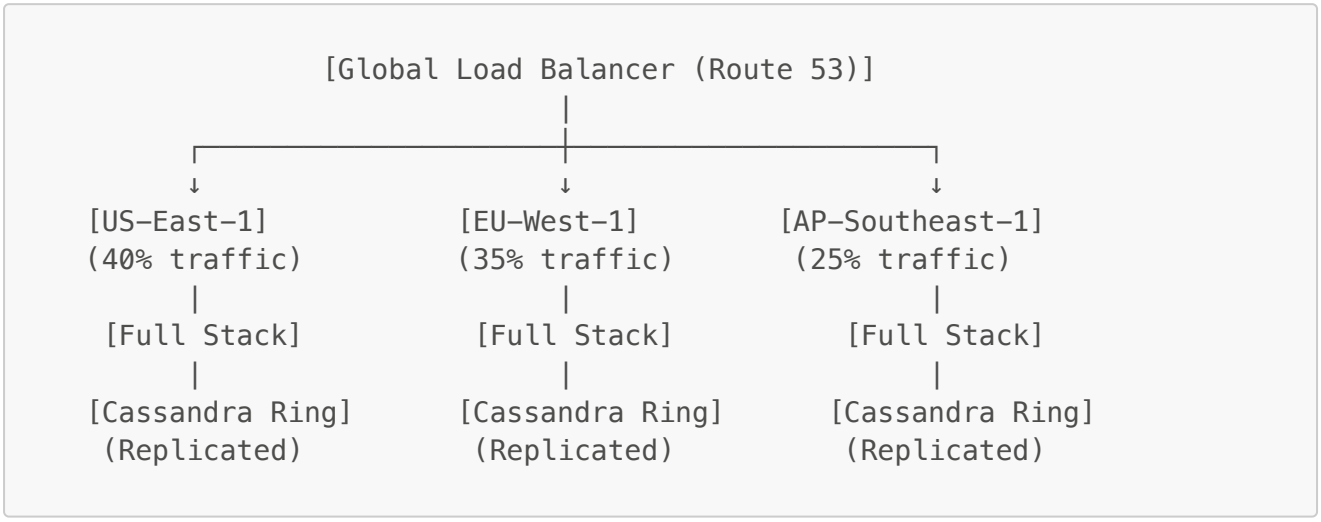
- **Health monitoring** and failover
- **Capacity planning**: Add OCAs based on regional growth

Regional CDN:

- **CloudFront** distribution per region
- **Multiple origins** for redundancy
- **Auto-scaling origin** servers

High Availability

Multi-Region Active-Active



Benefits:

- **Low latency**: Users routed to nearest region
- **High availability**: Continue serving if region fails
- **Disaster recovery**: Data replicated across regions

Consistency:

- **Cassandra**: Multi-datacenter replication
- **S3**: Cross-region replication
- **Cache**: Region-local (Redis)

Circuit Breaker Pattern

```
class CircuitBreaker:
    def __init__(self, failure_threshold=5, timeout=60):
        self.failure_count = 0
        self.failure_threshold = failure_threshold
        self.timeout = timeout
        self.last_failure_time = None
        self.state = 'CLOSED'  # CLOSED, OPEN, HALF_OPEN

    def call(self, func, *args, **kwargs):
```

```
        if self.state == 'OPEN':
            if time.time() - self.last_failure_time > self.timeout:
                self.state = 'HALF_OPEN'
            else:
                raise CircuitBreakerOpenError("Circuit breaker is OPEN")

        try:
            result = func(*args, **kwargs)
            if self.state == 'HALF_OPEN':
                self.state = 'CLOSED'
                self.failure_count = 0
            return result
        except Exception as e:
            self.failure_count += 1
            self.last_failure_time = time.time()

            if self.failure_count >= self.failure_threshold:
                self.state = 'OPEN'

            raise e
```

Rate Limiting

```
class TokenBucketRateLimiter:
    def __init__(self, capacity, refill_rate):
        self.capacity = capacity
        self.refill_rate = refill_rate  # tokens per second
        self.tokens = capacity
        self.last_refill = time.time()

    def allow_request(self, user_id):
        self.refill()

        if self.tokens >= 1:
            self.tokens -= 1
            return True
        return False

    def refill(self):
        now = time.time()
        elapsed = now - self.last_refill
        refill_amount = elapsed * self.refill_rate

        self.tokens = min(self.capacity, self.tokens + refill_amount)
        self.last_refill = now

# Usage
limiter = TokenBucketRateLimiter(capacity=100, refill_rate=10)

if limiter.allow_request(user_id):
```

```
    # Process request
    process_request()
else:
    # Return 429 Too Many Requests
    return error_response(429)
```

Rate Limits:

```
Per User:
- API requests: 1000/hour
- Video playback starts: 100/hour
- Search queries: 500/hour

Per IP (unauthenticated):
- Registration: 10/hour
- Login attempts: 20/hour
```

Monitoring & Alerting

Key Metrics

RED Metrics (Request, Error, Duration):

```
- Request Rate: Requests per second
- Error Rate: Percentage of failed requests
- Duration: Response time (p50, p95, p99)
```

System Metrics:

```
- CPU utilization: < 70% normal
- Memory utilization: < 80% normal
- Disk I/O: < 80% capacity
- Network bandwidth: < 70% capacity
```

Business Metrics:

```
- Concurrent streams
- Video start success rate: > 98%
- Rebuffering ratio: < 0.5%
- Average bitrate delivered
- Content popularity trends
```

Alerting Rules:

```
Critical (PagerDuty):
- API error rate > 5% for 5 minutes
- Video start success rate < 95% for 5 minutes
- Database primary failure
- CDN cache hit ratio < 80%

Warning (Slack/Email):
- API latency p99 > 2s for 10 minutes
- Disk space > 85%
- CPU > 80% for 15 minutes
- Unusual traffic spike (> 2x normal)
```

Observability Stack

```
Metrics: Prometheus + Grafana
- Time-series metrics
- Custom dashboards
- Real-time alerting

Logs: ELK Stack (Elasticsearch, Logstash, Kibana)
- Centralized logging
- Log aggregation from all services
- Full-text search on logs
- Log retention: 30 days

Tracing: Jaeger
- Distributed tracing
- Request flow visualization
- Performance bottleneck identification
- Service dependency mapping

APM: Datadog / New Relic
- Application performance monitoring
- User experience monitoring
- Infrastructure monitoring
```

Trade-offs & Alternatives

1. Self-Hosted CDN (Open Connect) vs. Third-Party CDN

Chose: Self-Hosted (Open Connect)

Pros:

- **Cost savings**: Eliminate transit costs (90% of bandwidth)

- **Better control**: Full control over caching behavior
- **ISP partnerships**: Improved relationships, reduced peering costs
- **Custom optimizations**: Tailored for video streaming

Cons:

- **High upfront cost**: Hardware, deployment, maintenance
- **Operational complexity**: Manage 10,000+ servers globally
- **Slower rollout**: Takes time to deploy globally

Alternative: CloudFront/Akamai

- Lower upfront cost
- Faster global deployment
- Less operational overhead
- Higher ongoing costs at scale

2. Microservices vs. Monolith

Chose: Microservices

Pros:

- **Independent scaling**: Scale services independently
- **Technology diversity**: Use best tool for each service
- **Team autonomy**: Teams own services end-to-end
- **Fault isolation**: Failure in one service doesn't bring down system

Cons:

- **Operational complexity**: More services to manage
- **Network overhead**: Inter-service communication
- **Distributed tracing needed**: Harder to debug
- **Data consistency challenges**: Eventual consistency

Alternative: Modular Monolith

- Simpler deployment
- Easier to debug
- Better performance (no network calls)
- Difficult to scale independently

3. Cassandra vs. MySQL for User Data

Chose: Hybrid (Cassandra + MySQL)

Cassandra for viewing history:

- High write throughput
- Time-series data natural fit
- Linear scalability

MySQL for billing:

- ACID transactions critical
- Complex queries needed
- Data integrity important

Alternative: Pure Cassandra or Pure MySQL

- Simpler architecture
- Loss of specialized optimizations

4. Pre-compute Recommendations vs. Real-time

Chose: Pre-computed with Real-time Updates

Approach:

- Pre-compute recommendations hourly
- Cache top 100 recommendations
- Real-time updates for critical events

Pros:

- Fast response time
- Reduced compute cost
- Predictable performance

Cons:

- Slightly stale recommendations
- Higher storage cost (cache)

Alternative: Pure Real-time

- Always up-to-date
- Higher latency
- More expensive compute

5. Push-Based vs. Pull-Based ABR

Chose: Pull-Based (Client-side ABR)

Pros:

- Client has best view of network conditions
- No server-side state needed
- Works with standard CDNs
- Flexible per-device optimization

Cons:

- Logic duplicated across clients

- Harder to update algorithm

Alternative: Push-Based (Server-side ABR)

- Centralized logic
- Easier to update
- Requires stateful servers
- Additional latency

Security & Compliance

Content Protection

Multi-Layered DRM:

```
Layer 1: Encryption
- AES-128 encryption
- Unique keys per title
- Key rotation

Layer 2: DRM Systems
- Widevine (Google)
- PlayReady (Microsoft)
- FairPlay (Apple)

Layer 3: Forensic Watermarking
- Invisible watermarks
- User/session-specific
- Trace leaked content

Layer 4: HDCP
- Hardware-level protection
- Required for 4K content
- Display device verification
```

User Data Privacy

GDPR Compliance:

- Right to access: Export user data API
- Right to be forgotten: Delete user data permanently
- Data portability: JSON export of viewing history
- Consent management: Explicit opt-in for marketing

Data Encryption:

```
In Transit:
- TLS 1.3 for all communications
```

```
- Certificate pinning on mobile apps

At Rest:
- S3 server-side encryption (AES-256)
- Database encryption for PII
- Encrypted backups
```

Access Control:

```
- IAM roles with least privilege
- MFA for production access
- Audit logging of all data access
- Regular access reviews
```

Geo-Blocking & Content Licensing

Challenge: Content licensing varies by region

Implementation:

```python
def is_content_available(title_id, user_region):
    """Check if content is available in user's region"""
    licensing_db = get_licensing_info(title_id)

    if user_region in licensing_db['blocked_regions']:
        return False

    if licensing_db['available_regions'] == 'ALL':
        return True

    return user_region in licensing_db['available_regions']

def detect_user_region():
    """Detect user region via multiple methods"""
    # Priority order:
    # 1. Billing address (most reliable)
    # 2. GPS location (mobile apps)
    # 3. IP geolocation
    # 4. Account settings

    return most_reliable_region
```

VPN Detection:

- IP reputation databases
- DNS leak detection
- WebRTC leak detection

- Known VPN provider blocking
- Behavioral analysis

Cost Optimization

Storage Costs

Optimization Strategies:

1. **S3 Lifecycle Policies**:

```
Hot Tier (Standard): Recent content (< 90 days) - $0.023/GB
Warm Tier (IA): Older content (90-365 days) - $0.0125/GB
Cold Tier (Glacier): Archive (> 365 days) - $0.004/GB
```

2. **Per-Title Encoding**:

- Analyze content complexity
- Simpler content: Use lower bitrates
- Complex content: Use higher bitrates
- Save 20-30% on storage

3. **Compression**:

- H.265 vs H.264: 50% bitrate savings
- AV1 codec: Additional 30% savings
- Gradual migration to newer codecs

Bandwidth Costs

Strategies:

1. **Open Connect**: Save 90% on transit costs
2. **Regional Caching**: Cache popular content closer to users
3. **Compression**: Reduce data transferred
4. **Off-Peak Prefetching**: Fill caches during low-traffic hours

Compute Costs

1. **Spot Instances**: Use for encoding pipeline (60-80% savings)
2. **Reserved Instances**: For baseline capacity (30-50% savings)
3. **Auto-Scaling**: Scale down during off-peak
4. **Right-Sizing**: Match instance types to workload

Estimated Monthly Costs (200M subscribers):

```
Infrastructure:
- Compute (EC2/ECS): $5M/month
- Storage (S3): $2M/month
- Database (Cassandra, MySQL): $3M/month
- CDN (Open Connect): $1M/month (vs $20M for third-party)
- Bandwidth: $2M/month
- Total: ~$13M/month

Per Subscriber: $0.065/month for infrastructure
Revenue: $15/month avg subscription
Infrastructure Cost: 0.43% of revenue
```

Future Enhancements

Phase 1 (Months 1-6) - MVP

- [x] User registration and authentication
- [x] Basic video streaming (ABR)
- [x] Content catalog and search
- [x] Simple recommendations
- [x] Resume playback
- [ ] Mobile apps (iOS, Android)

Phase 2 (Months 7-12) - Growth

- [ ] Personalized recommendations (ML)
- [ ] Download for offline viewing
- [ ] Multiple profiles per account
- [ ] Parental controls
- [ ] Advanced search and filters
- [ ] A/B testing framework

Phase 3 (Year 2) - Scale

- [ ] Live streaming support
- [ ] Interactive content (choose your own adventure)
- [ ] Watch party (synchronized viewing)
- [ ] AI-powered content analysis
- [ ] Advanced analytics dashboard
- [ ] Multi-language support (20+ languages)

Phase 4 (Year 3+) - Innovation

- [ ] VR/AR streaming
- [ ] Cloud gaming integration
- [ ] AI-generated preview trailers

- [ ] Real-time content translation
- [ ] Blockchain-based content rights management

Technology Stack Summary

Layer	Technology	Purpose
CDN	Open Connect (10K+ OCAs)	Video delivery
Client	iOS/Android (Swift/Kotlin), Web (React)	User interfaces
API Gateway	Netflix Zuul 2, Spring Cloud Gateway	Routing, auth, rate limiting
Load Balancer	AWS ELB/ALB, NGINX	Traffic distribution
App Servers	Node.js, Go, Java (Spring Boot)	Business logic
Streaming	HLS, DASH, ABR algorithms	Adaptive streaming
Encoding	AWS Elemental, FFmpeg	Video transcoding
Caching	Redis, Memcached	Performance optimization
SQL Database	MySQL, Aurora	Billing, transactions
NoSQL Database	Cassandra	User data, viewing history
Search	ElasticSearch	Content discovery
Object Storage	Amazon S3	Video files, media assets
Message Queue	Apache Kafka	Event streaming
Analytics	Spark, Hadoop, Flink	Data processing
Data Warehouse	Redshift, Snowflake	Business intelligence
ML Platform	TensorFlow, PyTorch	Recommendations
Monitoring	Prometheus, Grafana, Datadog	Metrics & alerts
Logging	ELK Stack (Elasticsearch, Logstash, Kibana)	Centralized logs
Tracing	Jaeger, AWS X-Ray	Distributed tracing
Container	Docker, Kubernetes, ECS	Orchestration
DRM	Widevine, PlayReady, FairPlay	Content protection

Interview Talking Points

Key Design Decisions

1. **Why Open Connect (custom CDN)?**
    - Cost savings: Reduce 90% of transit costs (billions per year)

- Better control over caching and delivery
- ISP partnerships reduce network congestion
- Custom optimizations for video streaming workload

2. **Why Cassandra for viewing history?**

   - High write throughput (millions of events per second)
   - Time-series data is natural fit for Cassandra's data model
   - Linear scalability by adding nodes
   - Multi-region replication for global availability
   - No single point of failure

3. **Why client-side ABR?**

   - Client has best visibility into network conditions
   - No server-side state needed (simpler architecture)
   - Works with any standard CDN
   - Device-specific optimizations possible
   - Reduces server load

4. **Why microservices architecture?**

   - Independent scaling of services (streaming vs recommendation)
   - Technology diversity (Node.js for I/O, Python for ML)
   - Team autonomy (small teams own services)
   - Fault isolation (one service failure doesn't crash system)
   - Gradual rollout of features

5. **Why pre-compute recommendations?**

   - Fast response time (< 100ms)
   - Predictable performance under load
   - Reduced compute costs
   - Can batch process during off-peak hours

6. **Why multi-tier caching?**

   - 90% served from ISP caches (OCA)
   - 8% from regional CDN
   - 2% from origin (S3)
   - Minimizes latency and bandwidth costs

7. **Why Kafka for events?**

   - High throughput for millions of events per second
   - Durable storage for replay capability
   - Multiple consumers (analytics, recommendations, monitoring)
   - Real-time stream processing

Bottlenecks & Solutions

Bottleneck	Solution
Video encoding time	Parallel encoding, spot instances, pre-encoding
CDN capacity	Add OCAs, predictive caching, off-peak fills
Database writes	Cassandra sharding, batch writes, eventual consistency
Recommendation latency	Pre-compute, cache, incremental updates
Search performance	ElasticSearch with proper indexing, caching
Authentication overhead	JWT tokens, cache sessions, token refresh
Network bandwidth	ABR, compression, regional caching
Storage costs	Lifecycle policies, per-title encoding, codec optimization

Scale Calculations You Should Know

```
Daily Active Users (DAU): 100M
Hours watched per day: 125M
Peak concurrent streams: 30M

Bandwidth at peak:
30M streams × 3 Mbps = 90 Tbps = 11.25 TB/s

Storage for 10 years:
19 TB/day × 365 days × 10 years ≈ 69 PB

QPS at peak:
- Playback API: 30M concurrent / 7200 sec avg session = 4,200 QPS
- Browse API: 100M DAU × 20 requests/day / 86400 = 23,000 QPS
- Search API: 100M DAU × 5 searches/day / 86400 = 5,800 QPS
```

Common Interview Questions & Answers

Q: How do you handle a celebrity posting content to millions of followers?
A: Use fan-out on read for high-follower users. Their posts are fetched on-demand rather than pre-computed, avoiding write amplification.

Q: How do you ensure video starts in under 2 seconds?
A:

- Pre-position popular content on OCAs near users
- Start with lower bitrate initially (fast start)
- Use adaptive bitrate streaming
- Pre-fetch and buffer ahead
- Optimize manifest size

Q: How would you handle a region-wide AWS outage?

A:

- Multi-region active-active deployment
- Route53 health checks automatically failover
- Cassandra cross-region replication maintains data
- OCAs continue serving cached content
- Expected impact: minimal for streaming, some degradation for catalog

Q: How do you prevent content piracy?

A:

- Multi-DRM (Widevine, PlayReady, FairPlay)
- Forensic watermarking to trace leaks
- HDCP for 4K content
- Device binding for downloads
- Short-lived licenses (6 hours)
- VPN detection and blocking

Q: How do you decide what content to cache on OCAs?

A:

- Analyze viewing patterns (Pareto: 20% content = 80% views)
- Regional preferences (K-dramas in Asia)
- Time-based trends (new releases)
- ML models for prediction
- Off-peak hours for fills

Q: How would you reduce costs by 20%?

A:

- Optimize encoding (per-title, newer codecs like AV1)
- S3 lifecycle policies (move old content to Glacier)
- Spot instances for encoding pipeline
- Increase cache hit ratio (save bandwidth)
- Right-size compute instances
- Negotiate better CDN rates

Q: How do recommendations scale to 200M users?

A:

- Pre-compute recommendations hourly (batch job)
- Cache top 100 recommendations per user
- Distribute across 100s of cache servers
- Incremental updates for real-time events
- Different models per region

Q: How do you ensure 99.99% uptime?

A:

- Multi-region active-active (no single point of failure)
- Circuit breakers prevent cascading failures
- Rate limiting protects from overload
- Auto-scaling handles traffic spikes
- Graceful degradation (serve cached data)
- Chaos engineering tests failure scenarios

System Design Patterns Used

1. **CQRS (Command Query Responsibility Segregation)**

- Separate read and write paths
- Write to Cassandra master
- Read from replicas and cache
- Optimize each path independently

2. **Event Sourcing**

- Store all user actions as events (Kafka)
- Rebuild state from event log
- Enable analytics and ML training
- Audit trail for compliance

3. **Circuit Breaker**

- Prevent cascading failures
- Fast fail when service is down
- Auto-recovery when service returns
- Implemented with Hystrix/Resilience4j

4. **Bulkhead Pattern**

- Isolate resources for different features
- Separate thread pools for streaming vs browsing
- Prevent one feature from starving others
- Better fault isolation

5. **Cache-Aside (Lazy Loading)**

- Application manages cache
- Check cache first, then database
- Write to database, invalidate cache
- Simple and predictable

6. **Strangler Fig**

- Gradually migrate from monolith to microservices
- Route traffic to new services incrementally

- Reduce risk of big-bang migration
- Maintain old system during transition

7. **Saga Pattern**

- Distributed transactions across services
- Compensating transactions for rollback
- Used for subscription/payment workflows
- Eventual consistency

8. **API Gateway Pattern**

- Single entry point for all clients
- Authentication, authorization, rate limiting
- Request routing to microservices
- Response aggregation

References & Further Reading

Netflix Engineering Blog Posts

1. **Open Connect** - Netflix's Content Delivery Network
    - https://openconnect.netflix.com/
2. **Netflix's Recommendation System** - Collaborative filtering at scale
3. **Chaos Engineering** - Breaking things on purpose
4. **A/B Testing at Scale** - Data-driven product development
5. **Dynamic Adaptive Streaming over HTTP (DASH)** - Streaming protocol
6. **Per-Title Encode Optimization** - Save bandwidth without sacrificing quality

Academic Papers

1. **Adaptive Bitrate Streaming** - Principles and implementations
2. **Content Delivery Networks** - Architecture and optimization
3. **Recommendation Systems** - Collaborative filtering algorithms
4. **Distributed Systems** - CAP theorem, consistency models

Industry Resources

1. **System Design Primer** - https://github.com/donnemartin/system-design-primer
2. **High Scalability Blog** - Real-world architecture examples
3. **AWS Architecture Center** - Best practices and patterns
4. **Google SRE Book** - Site reliability engineering principles

Video Streaming Standards

1. **HLS (HTTP Live Streaming)** - Apple's streaming protocol
2. **DASH (Dynamic Adaptive Streaming over HTTP)** - MPEG standard
3. **CMAF (Common Media Application Format)** - Universal format

4. **WebRTC** - Real-time communication

Books

1. **Designing Data-Intensive Applications** by Martin Kleppmann
2. **System Design Interview** by Alex Xu (Vol 1 & 2)
3. **Building Microservices** by Sam Newman
4. **Site Reliability Engineering** by Google
5. **Streaming Systems** by Tyler Akidau

Appendix

Latency Numbers Every Programmer Should Know

```
L1 cache reference:                   0.5 ns
Branch mispredict:                    5 ns
L2 cache reference:                   7 ns
Mutex lock/unlock:                    25 ns
Main memory reference:                100 ns
Compress 1KB with Snappy:             3,000 ns = 3 µs
Send 1KB over 1 Gbps network:         10,000 ns = 10 µs
Read 4KB randomly from SSD:           150,000 ns = 150 µs
Read 1 MB sequentially from memory:   250,000 ns = 250 µs
Round trip within datacenter:         500,000 ns = 500 µs
Read 1 MB sequentially from SSD:      1,000,000 ns = 1 ms
Disk seek:                            10,000,000 ns = 10 ms
Read 1 MB sequentially from disk:     20,000,000 ns = 20 ms
Send packet CA → Netherlands → CA:    150,000,000 ns = 150 ms
```

Video Streaming Terminology

ABR (Adaptive Bitrate Streaming): Dynamically adjusting video quality based on network conditions

Bitrate: Amount of data transmitted per second (measured in Mbps)

Buffer: Pre-loaded video data stored locally on device

CDN (Content Delivery Network): Distributed servers that deliver content to users

Codec: Algorithm for encoding/decoding video (H.264, H.265, VP9, AV1)

DRM (Digital Rights Management): Technology to prevent unauthorized copying

HLS (HTTP Live Streaming): Apple's streaming protocol

Manifest: File listing available video segments and qualities

OCA (Open Connect Appliance): Netflix's custom CDN servers

Rebuffering: Playback pause due to insufficient buffer

Segment: Small chunk of video (typically 2-10 seconds)

Transcoding: Converting video from one format to another

Capacity Planning Formulas

Storage:

```
Total Storage = Content Hours × Formats × Bitrate × Duration
              = 100,000 hrs × 5 formats × avg 3 Mbps × 3600 sec/hr
              ≈ 10 PB
```

Bandwidth:

```
Peak Bandwidth = Concurrent Users × Average Bitrate
               = 30M users × 3 Mbps
               = 90 Tbps
```

QPS (Queries Per Second):

```
Peak QPS = (DAU × Actions per User) / (86400 sec × Peak Factor)
         = (100M × 30) / (86400 × 0.3)
         ≈ 115,000 QPS
```

Cache Size:

```
Cache Size = Hot Data Percentage × Total Data × Cache Ratio
           = 0.2 × 10 PB × 0.95
           = 1.9 PB (distributed across regions)
```

Common QPS Benchmarks

```
Small service:       100 - 1,000 QPS
Medium service:      1,000 - 10,000 QPS
Large service:       10,000 - 100,000 QPS
Very large service:  100,000 - 1,000,000+ QPS

Netflix scale:       ~500,000 QPS (all APIs combined)
```

Quick Interview Checklist

Before the Interview

- [ ] Review Netflix's actual architecture from blog posts
- [ ] Understand trade-offs: Microservices vs Monolith, SQL vs NoSQL
- [ ] Practice capacity estimation (storage, bandwidth, QPS)
- [ ] Know common system design patterns
- [ ] Review CAP theorem and consistency models
- [ ] Understand CDN architecture and caching strategies
- [ ] Know video streaming protocols (HLS, DASH)
- [ ] Review database sharding and replication

During the Interview

1. Clarify Requirements (5 min)

- [ ] Functional requirements (must-have vs nice-to-have)
- [ ] Non-functional requirements (scale, performance, availability)
- [ ] Scale numbers (users, videos, concurrent streams)
- [ ] Read-heavy or write-heavy workload?

2. Back-of-Envelope Estimation (5 min)

- [ ] Storage calculation (video files, metadata)
- [ ] Bandwidth calculation (peak concurrent streams)
- [ ] QPS estimation (API endpoints)
- [ ] Cache size needed

3. High-Level Design (10 min)

- [ ] Draw architecture diagram
- [ ] Identify major components (CDN, API Gateway, databases)
- [ ] Show data flow
- [ ] Explain technology choices

4. Deep Dives (20 min)

- [ ] Video playback flow
- [ ] Adaptive bitrate streaming
- [ ] Content recommendation
- [ ] Database design
- [ ] Caching strategy
- [ ] DRM and security

5. Scalability & Reliability (5 min)

- [ ] Horizontal scaling approach

- ☐ High availability strategy
- ☐ Disaster recovery plan
- ☐ Monitoring and alerting

6. Trade-offs & Bottlenecks (5 min)

- ☐ Discuss alternatives considered
- ☐ Identify potential bottlenecks
- ☐ Propose solutions

Document Version: 1.0
Last Updated: January 8, 2025
Author: System Design Interview Prep
Status: Complete & Interview-Ready ✅

Key Takeaways:

1. **Netflix's secret sauce** is Open Connect - custom CDN deployed in ISP networks
2. **90% of traffic** served from ISP-level caches (OCAs)
3. **Adaptive bitrate streaming** is key to handling varying network conditions
4. **Microservices architecture** enables independent scaling and fault isolation
5. **Cassandra** for high-write throughput time-series data
6. **Pre-computed recommendations** balance freshness with performance
7. **Multi-region active-active** ensures global availability
8. **DRM protection** with Widevine, PlayReady, FairPlay
9. **Cost optimization** through per-title encoding and codec selection
10. **99.99% uptime** through redundancy, circuit breakers, and graceful degradation

Good luck with your system design interview! 🎬 🍿