1. Introduction

This document provides the complete system design for a scalable, multi-region URL shortener with real-time analytics.
The system allows users to generate short links, track clicks, referrers, devices, geo-location, and access a dashboard for analytics.

The design covers requirements, architecture diagrams (text), engineering design, data flow, capacity planning, SLOs, and trade-offs.

---

2. Requirements Pack

2.1 Stakeholder Analysis

Primary Stakeholders

- **End Users:** Create and use short URLs, view analytics.
- **Businesses/Marketing Teams:** Require high uptime, fast redirects, accurate analytics.
- **Administrators/Engineers:** Maintain system reliability, monitor performance.

Secondary Stakeholders

- **Data Analysts:** Use analytics to study user behavior.
- **Executives/Product Owners:** Need insights for decision making.

2.2 Requirement Prioritization

P0 (Must-Have)

- Generate short URLs

- Redirect to original URL
- Track click analytics
- Store URL mappings
- Multi-region high availability

P1 (Should-Have)

- Geo-location tracking
- Device & browser analytics
- Real-time dashboard

P2 (Nice-to-Have)

- Custom short codes
- QR code generator
- Authenticated user accounts

2.3 Functional Requirements

1. Users can submit long URLs to generate short codes.
2. System redirects short URL → original URL.
3. The system logs click events:
    - timestamp
    - IP address
    - device & browser
    - referrer
    - country

4. Analytics dashboard displays:
    - total clicks
    - active links
    - CTR
    - top performing links
5. Multi-region read replicas ensure low-latency redirection.

2.4 Non-Functional Requirements

- **Availability:** 99.99% uptime
- **Latency:** Redirect < 50ms (p95)
- **Scalability:** Up to millions of redirects/day
- **Security:** Rate limiting, API keys, HTTPS
- **Durability:** No lost mappings or analytics

2.5 Constraints

- Must operate globally with a multi-region setup.
- Should minimize database load (read-heavy system).
- Must support burst traffic during marketing events.

2.6 Assumptions

- 80% of traffic are read (redirect).

- Cache will achieve at least 90–95% hit ratio.
- Users access service from multiple regions globally.

---

3. System Context Diagram (Text Representation)

Actors & Interactions:

- **User** → accesses website or short link
- **Web App** → provides UI
- **API Gateway** → manages secure API routing
- **URL Shortener Service** → stores and resolves URLs
- **Analytics Service** → processes click events
- **Data Stores** → NoSQL DB, Redis, Data Warehouse
- **CDN/Edge Nodes** → serve cached redirect responses

---

4. Use Case Diagram (Text)

Use Case 1: Shorten URL

Actor: User
System: Stores mapping and returns short code.

Use Case 2: Redirect

Actor: Anonymous client user
System: Resolves code → redirects → logs event.

Use Case 3: View Analytics

Actor: Authenticated user
System: Fetches metrics from analytics DB.

---

5. User Stories

- **As a user**, I want to shorten long URLs to share them easily.
- **As a user**, I want to track how many clicks my short URL gets.
- **As a business**, I need high uptime and low-latency redirects.
- **As an admin**, I want to monitor system health.

---

6. Core Sequence Diagrams (Text)

6.1 URL Shortening Flow

1. User → submits long URL
2. API Gateway → forwards to URL Shortener service
3. Service validates URL
4. Generates unique short code
5. Stores in ShortLink DB
6. Returns short URL

6.2 Redirection Flow

1. User clicks short URL
2. CDN checks cache →
    - if hit → redirect immediately
    - if miss → fetch from DB
3. Service publishes click event to Queue
4. System redirects user
5. Analytics service processes events

6.3 Analytics Flow

1. Event Queue receives click event
2. Analytics Worker extracts device, geo, referrer
3. Writes aggregated metrics to OLAP DB
4. Dashboard fetches analytics from analytics DB

---

7. High-Level Architecture Overview

Core Components

- **API Gateway**
- **URL Shortener Service**
- **Redirection Service**
- **Analytics Ingestion Service**
- **Message Queue (Kafka / SQS / PubSub)**
- **ShortLink DB (NoSQL)**

- **Redis Cache**
- **CDN/Edge Network**
- **Analytics OLAP Database**

Data Flow

- Shorten URL → Write DB
- Redirect → Cache → DB fallback
- Click event → Queue → Analytics DB

---

8. Engineering Notes

8.1 Capacity Planning & Sizing

Assume:

- 10 million requests/day
- 80% redirects = 8M redirect requests
- Expected cache hit ratio: 95%
- DB reads: 5% of 8M = ~400k/day
- Storage per URL:
    - short code + long URL ≈ 100 bytes
    - 10M URLs ≈ 1GB

8.2 API Specifications

POST /shorten

Request: longURL
Response: shortURL

GET /{code}

Redirects to original URL

GET /analytics/{code}

Returns metrics

8.3 Data Model

ShortLink Table

- code (primary key)
- longURL
- createdAt
- userId

Analytics Event

- eventId
- code
- timestamp
- IP
- device
- country
- referrer

8.4 Consistency Model

- **Strong consistency** for URL creation
- **Eventual consistency** for analytics

8.5 Caching Strategy

- Redis for code → URL mapping
- TTL optional
- LRU eviction

8.6 Indexing

- Primary index on code
- Secondary index: userId (optional)

8.7 Rate Limiting

- Throttle:
  - API calls per IP
  - URL creation per minute
- Prevents abuse and DDoS

8.8 Resiliency

- Retries with exponential backoff

- Timeouts on DB and queue calls
- Circuit breakers for failing services
- Multi-region failover

8.9 Observability

Logs

- Redirect logs
- Error logs

Metrics

- QPS
- latency
- cache hit ratio
- queue lag

Tracing

- API gateway traces
- DB interactions

8.10 Maintenance Plan

- Archive old analytics
- Scheduled DB backups
- Index optimization

- Auto-scaling workers and API servers

---

9. Quality Targets

9.1 SLOs

- Redirect latency: **<50ms (p95)**
- Uptime: **99.99%**
- Analytics delay: **<5 seconds**

9.2 Scalability Plan

- Stateless services → horizontal scaling
- DB sharding by:
  - shortcode prefix
  - userId (optional)
- Partitioned analytics storage

9.3 Trade-Offs

- **NoSQL vs SQL**: NoSQL chosen for scalability and fast reads.
- **Eventual consistency** for analytics improves throughput.
- **CDN caching** reduces DB load but may require invalidation.

Shortcode generation:

- Sequential IDs → fast but predictable
- Random codes → safer but more collision checks

---

10. Conclusion

This design delivers a globally distributed, highly scalable URL shortener supporting real-time analytics, high availability, and low latency. It is suitable for enterprise-grade workloads and marketing use cases.