System Design Mini Project

URL Shortener with Click Analytics (Multi-Region)

B Tech (CSE-GEN(AI))

Submitted To

Bytexl

Lovely Professional University

By

O Siva Shankar

Reg.No:12407339

Roll.No :13

Table of Contents / Index

1. Introduction

1.1 System Definition

This document describes the design of a **globally distributed web-based analytics platform for URL management** — specifically, a **URL Shortener with Click Analytics (Multi-Region)** system.

The platform enables users to convert long, complex URLs into short, easily shareable links (e.g., sho.rt/abc123) while providing **real-time analytics** on user engagement, such as **click counts**, **geographic distribution**, **referrers**, and **device information**.

The system is built for **global scale**, capable of handling billions of redirection requests per month, and aims to deliver:

- **High performance** (redirect latency < 50 ms globally)
- **High availability** (99.99 % uptime)
- **Scalable analytics processing** for billions of events
- **Secure link management and abuse prevention**
- **Maintainable and observable architecture** across multiple regions

---

1.2 Purpose of This Document

This document serves as a comprehensive **system design specification** intended for:

- **Product Managers** – to understand functional scope and business capabilities
- **Software Engineers** – to implement services, APIs, and data pipelines
- **Site Reliability Engineers (SREs)** – to monitor performance, ensure uptime, and handle scaling
- **Security Teams** – to validate privacy, access control, and compliance
- **Leadership / Stakeholders** – to assess the project's scalability, cost, and maintainability

It provides both **functional** and **non-functional** design details, including architecture diagrams, data models, scalability strategies, and operational considerations.

1.3 Problem Statement & Business Context

Problem Statement

In digital marketing, social media, and content sharing, long URLs are inconvenient and often impractical. At the same time, businesses require **actionable insights** into how links perform — where users come from, what devices they use, and how engagement varies by geography or campaign.

Existing solutions (e.g., Bitly, TinyURL) are often centralized, region-specific, or expensive for large-scale enterprise use. They also lack customization, extensibility, and granular control over analytics freshness and data retention.

Business Context

Organizations increasingly depend on link analytics for:

- **Campaign effectiveness** measurement
- **Customer engagement** tracking
- **Data-driven decision making**
- **Compliance reporting** (GDPR, data residency)

Therefore, building a **cost-efficient, privacy-preserving, and high-throughput** URL shortener with **real-time analytics** becomes critical for modern web infrastructure.

1.4 Vision and High-Level Objectives

The vision of the system is to provide **a globally available, developer-friendly link management and analytics platform** that combines performance, reliability, and insight.

Core Objectives

Goal	Target
Latency (p95)	< 50 ms redirect time globally
Availability	≥ 99.99 % for redirect path
Scalability	10 B+ redirects/month; 100 M+ links
Analytics Freshness	≤ 60 s from click to dashboard
Data Durability	99.999999999 % (via distributed replication)
Security	End-to-end encryption, RBAC, rate limiting

Supporting Objectives

- Provide both REST and future GraphQL APIs for link creation, stats retrieval, and administration.
- Support multiple custom domains and user-defined aliases.
- Enable cost-efficient operation with autoscaling and caching at multiple layers (CDN, Redis, KV store).
- Maintain compliance with privacy laws and offer configurable data retention (e.g., 90 days for raw events, 1 year for aggregates).

1.5 Out-of-Scope Items

To ensure clarity and focus, the following areas are **out of scope** for this design document:

Out-of-Scope Item	Reason
Frontend UI/UX details	This design focuses on backend architecture and APIs, not web or mobile app interfaces.
Mobile SDK development	Client-side SDKs for link creation or analytics tracking are separate deliverables.
User authentication portal or billing system	These belong to the business platform layer, not the core service.
Third-party integrations (Slack, CRM, Ads)	Future extension once core system stabilizes.
Machine learning models for trend prediction	To be introduced in later phases as part of advanced analytics.

1.6 Summary

The **URL Shortener with Click Analytics (Multi-Region)** is designed as a **mission-critical backend service** that blends **speed**, **scalability**, and **insight**. By decoupling redirect handling from analytics computation, and leveraging **edge caching** and **asynchronous streaming**, the platform can meet strict performance targets even under massive global traffic.

This document provides an end-to-end architectural blueprint detailing components, APIs, data flow, scalability mechanisms, and operational best practices, forming the foundation for a production-grade distributed system.

2. Stakeholders & Requirements

This section defines who depends on the system, what each group expects, and the technical and functional capabilities the platform must deliver.
It also clarifies the **non-functional quality attributes**—performance, reliability, consistency, and compliance—that guide engineering and operational decisions.

2.1 Stakeholder Groups

Stakeholder Group	Description / Role	Expectations	Responsibilities
End Users	Individuals clicking short links shared across the web or social media.	• Instant redirects (< 50 ms). • No downtime or broken links. • Privacy of personal data.	None (passive users).
Product Team / Business Owners	Define roadmap, features, KPIs, SLAs.	• Clear analytics on campaigns and user adoption. • Easy integration with marketing tools.	Prioritize requirements, set budgets, monitor adoption.
Backend Engineers	Design and implement core APIs, storage, and services.	• Clean architecture, reusable APIs, observability. • Strong consistency for link mapping.	Build and maintain backend microservices and databases.
Front-end Engineers / Dashboard Team	Build UI for link creation and analytics visualization.	• Reliable REST/GraphQL APIs. • Low-latency analytics queries.	Consume APIs, ensure good user experience.
Data Engineers & Analysts	Process and visualize click-stream data.	• High-quality, fresh analytics data (< 60 s delay). • Schema stability and lineage tracking.	Maintain ETL/ELT jobs, verify data accuracy.
SRE / DevOps Team	Operate and monitor the distributed system.	• 99.99 % uptime. • Predictable scaling. • Automated recovery.	Provision infrastructure, monitoring, and alerts; manage incidents.
Security / Compliance Team	Govern data protection, access control, and audit trails.	• End-to-end encryption, strong IAM. • GDPR/CCPA compliance.	Enforce policies, review audit logs, handle data-deletion requests.
Finance / Cost Management	Control cloud expenditure.	• Transparent cost model, predictable growth.	Monitor usage, enforce budgets, optimize storage and egress costs.

2.2 Functional Requirements

Functional requirements describe what the system must do—its **use cases**, their **inputs, outputs**, and **success conditions**.

Core Use Cases

ID	Use Case	Description	Inputs / Triggers	Outputs / Success Conditions
FR-1	**Create Short Link**	User or API client submits a long URL and optional alias.	Request body: long URL, alias?, TTL?.	Returns JSON {short_url, code, expiry}. Persist link mapping in KV store.
FR-2	**Redirect Link**	When user visits the short URL, system redirects to long URL.	HTTP GET /{code}.	301/302 Redirect within < 50 ms; record click event asynchronously.
FR-3	**Track Click Event**	Collect metadata (IP, UA, referrer, region) for each redirect.	Triggered by Redirect Service.	Enriched event stored in stream and OLAP; counts reflected in dashboard within ≤ 60 s.
FR-4	**View Analytics Dashboard**	Owner retrieves statistics.	GET /v1/links/{code}/stats.	Returns aggregates by time, country, device.
FR-5	**Export Reports**	Download analytics in CSV/JSON.	User action on dashboard.	File generated asynchronously, emailed or downloadable link provided.
FR-6	**Manage Links (Admin)**	Enable/disable links, set TTL, blacklist domains.	Admin POST/PATCH.	Confirmation of action; cache invalidated globally.
FR-7	**User Authentication & API Keys**	Secure API access for developers.	Login or key creation request.	JWT/API key issued; stored securely with quotas.

Derived Behavior

- **Triggers:** API calls, scheduled jobs, redirect hits, admin events.
- **Outputs:** Redirect responses, analytics aggregates, logs, alerts.
- **Success Conditions:**
  - 100 % redirect accuracy.
  - Analytics data visible ≤ 60 s after click.
  - API error rate < 0.1 %.

2.3 Non-Functional Requirements

Non-functional requirements (NFRs) ensure the system's quality and operational stability beyond pure functionality.

Performance

Metric	Target
Redirect Latency (p95)	< 50 ms (global)
API Read Latency (p95)	< 200 ms
API Write Latency (p95)	< 500 ms
Throughput	≥ 100 000 redirects/sec sustained
Analytics Freshness	≤ 60 s from click to availability

Achieved through CDN edge caching, in-memory Redis, and asynchronous streaming.

Availability & Reliability

- **Redirect path:** ≥ 99.99 % uptime.
- **Analytics API:** ≥ 99.9 %.
- **Deployment:** Multi-AZ within region + active-active multi-region.
- **Failover Goal:** RTO < 5 min, RPO < 5 s.
- **Health Checks:** HTTP 200 /healthz endpoints, automated restarts.

Consistency Model

Operation	Consistency	Justification
Create / Update Link	**Strong**	Prevent duplicate or stale mappings.
Redirect Lookup	**Read-after-write** via cache invalidation.	User should never get wrong target.
Analytics Event Processing	**Eventual**	Throughput prioritized over immediate accuracy.

This balance maximizes user-facing correctness while keeping analytics scalable.

Durability of Data

- **Replication:** Each region keeps **3 replicas** of KV data.
- **Backups:** Daily snapshots to object storage (retention = 30 days).

- **Streaming Events:** Kafka + 3x replication factor.
- **OLAP Data:** Stored on redundant disks (RAID 10 or cloud replica zones).
- **Disaster Recovery:** Cross-region replication lag ≤ 5 s.

Security & Compliance

Area	Requirement	Implementation
Encryption in Transit	All HTTP→HTTPS, TLS 1.2+	Managed certificates (ACM/Let's Encrypt).
Encryption at Rest	AES-256 for DBs and object storage	Cloud KMS managed keys.
Access Control	Role-based (RBAC)	IAM policies per microservice.
Audit Logs	Immutable 1-year retention	Cloud Logging / SIEM integration.
Privacy Compliance	GDPR, CCPA	IP truncation (/24), opt-out, deletion API.
Vulnerability Mgmt	Regular scanning	CI pipeline + Snyk/Dependabot.

Maintainability & Operability

- **Infrastructure-as-Code:** Terraform for reproducibility.
- **Continuous Delivery:** Automated tests + canary rollouts.
- **Monitoring:** Central dashboards (Grafana, Prometheus).
- **Logging Standards:** Structured JSON; correlation IDs.
- **Documentation:** OpenAPI specs and runbooks.

Scalability Targets

Scale Dimension	Baseline	Growth Plan
Short links stored	100 M	10× in 2 years
Redirect traffic	20 k RPS	Burst 100 k RPS
Analytics events/day	1 B	Linear scale via Kafka partitions
Regions	3 (initial)	Expand to 5+ (active-active)

Operational Goals

- **Monitoring Coverage:** 100 % of production services instrumented.

- **Alert MTTR:** < 15 min.
- **Change Failure Rate:** < 5 %.
- **Automation Coverage:** ≥ 90 % infra via IaC.

---

3. High-Level Architecture

3.1 Bird’s-eye view

Flow:
Clients → Edge (CDN/WAF/API GW/LB) → Stateless Microservices → Caches & Databases → Streams → Analytics Warehouse & Dashboards.

- **Hot path (latency-critical):** user click → redirect (≤ 50 ms p95).
- **Cold path (throughput-critical):** click events → streaming → enrichment/aggregation → analytics API (freshness ≤ 60 s).

3.2 C4 — Level-1 Context Diagram

Who uses the system and what it integrates with.

```
flowchart LR
  subgraph Users
    U1[End User (Browser)]
    U2[Marketer / Product (Web App)]
    U3[External App (API Client)]
  end

  U1 -->|GET /{code}| EDGE[CDN + WAF + API Gateway]
  U2 -->|REST/GraphQL| EDGE
  U3 -->|REST/Keys| EDGE

  EDGE --> APP[URL Shortener Platform]

  subgraph External Services
    SB[SafeBrowsing/Blacklist API]
    GEO[GeoIP DB]
    IAM[Email/OIDC Provider]
  end

  APP <---> SB
  APP <---> GEO
  APP <---> IAM
```

3.3 C4 — Level-2 Container Diagram

Key deployable **containers** and data stores.

```
flowchart TB
  %% Edge
  subgraph Edge Layer
    CDN[CDN/Edge POPs]
    WAF[WAF]
    APIGW[API Gateway / Global LB]
  end

  %% App Services
  subgraph Application Layer
    AUTH[Auth Service]
    LINK[Link Service (CRUD, policy)]
    REDIR[Redirect Service (hot path)]
    INGEST[Analytics Ingest (HTTP→Event)]
    ADMIN[Admin UI/API]
    ANALYTICS_API[Analytics API (OLAP queries)]
  end

  %% Data Layer
  subgraph Data Layer
    CACHE[(Redis Cluster)]
    KV[(KV Store: code→target)]
    MQ[[Kafka/PubSub]]
    OBJ[(Object Storage: raw Parquet)]
    OLAP[(OLAP Warehouse: ClickHouse/BigQuery)]
    META[(OLTP/Config DB)]
  end

  %% Stream Processing
  subgraph Analytics Processing
    ENRICH[Enricher (UA/Geo/Referrer)]
    AGG[Stream Aggregator (minute/hour/day)]
    BATCH[Batch/Backfill Jobs]
  end

  %% Edges
  CDN --> APIGW
  WAF --> APIGW

  APIGW --> AUTH
  APIGW --> LINK
  APIGW --> REDIR
  APIGW --> ANALYTICS_API
  APIGW --> ADMIN

  REDIR --> CACHE
  REDIR --> KV
  REDIR --> MQ

  LINK --> KV
  LINK --> CACHE
  LINK --> META
  LINK --> SB[SafeBrowsing]
```

```
MQ --> ENRICH --> AGG --> OLAP

AGG --> OBJ

BATCH --> OLAP

BATCH --> OBJ


ANALYTICS_API --> OLAP
```

3.4 Layered Architecture

3.4.1 Client layer

- **Web browser (end users):** hit /{code} and follow 301/302.
- **Web app (marketers):** create/manage links, view dashboards.
- **External APIs/Integrations:** server-to-server link creation, stats export.

3.4.2 Edge layer

- **CDN / Edge POPs:** cache 301 responses for hot codes; anycast routing to nearest POP.
- **WAF:** blocks malicious payloads (SQLi, XSS), IP reputation filtering.
- **API Gateway / Global LB:** TLS termination, JWT/API-key auth, quotas, request routing, canary/weighted traffic splits.

3.4.3 Application layer (stateless microservices)

- **Auth Service:** OIDC/JWT validation, key mint/rotate, scopes/quotas.
- **Link Service:** create/update/delete link mappings; custom aliases; TTL; policy checks (domain blacklist); cache purge on updates; **strong consistency** writes to KV.
- **Redirect Service:** ultra-fast read path → Redis → KV (fallback); returns 301; emits click event **asynchronously**; zero blocking on analytics.
- **Analytics Ingest:** accepts raw events (if emitted over HTTP), normalizes schema, publishes to **Kafka** (or writes directly if REDIR publishes).
- **Analytics API:** serves aggregates and time series from **OLAP** with pagination and safe query limits.
- **Admin API/UI:** abuse controls (disable links, blacklist, rate limits), ops tools, runbooks.

3.4.4 Data layer

- **Redis Cluster (cache):** hot code→target objects; negative caching for unknown/disabled; stale-while-revalidate.
- **KV Store (OLTP for mapping):** DynamoDB/Cassandra (partition by randomized code prefix). **Read-after-write** ensured via cache purge and write-through on updates.

- **Message Queue / Stream (Kafka/PubSub):** partition by region/code; replication factor 3; retention (e.g., 7–14 days) for replay.
- **Object Storage:** canonical raw events in Parquet (cold storage, reprocessing, audits).
- **OLAP Warehouse:** ClickHouse/BigQuery—daily partitions, clustered by code for fast link queries.
- **Meta/Config DB:** small OLTP (Postgres) for tenants, keys, feature flags, quotas.

3.4.5 Data processing / analytics layer

- **Enricher:** UA → device/OS/browser; GeoIP; referrer→domain; batched lookups with LRU caches.
- **Stream Aggregator:** minute/hour/day rollups; HLL for uniques; dedupe by event_id; watermarking for late events; writes to OLAP/materialized views.
- **Batch Jobs:** backfills/corrections, schema migrations, compaction, retention enforcement.

3.5 Core Flows

3.5.1 Synchronous (request/response)

A) Redirect (hot path)
Latency budget target: **Edge 5–10 ms → Cache 1–2 ms → KV miss 20–30 ms → 301**.

```
sequenceDiagram
 participant U as User
 participant E as CDN/APIGW
 participant R as Redirect Svc
 participant C as Redis
 participant K as KV Store
 U->>E: GET /{code}
 E->>R: route request
 R->>C: GET code
 alt Cache hit
   C-->>R: {target, flags}
 else Cache miss
   R->>K: GetItem(code)
   K-->>R: {target, flags}
   R->>C: SET code payload (TTL)
 end
```

```
  R-->>E: 301 Location: target
  note over R: emit ClickEvent → Kafka (async)
  E-->>U: 301 Redirect
```

B) Create link (control path)

```
sequenceDiagram
  participant C as Client
  participant G as API Gateway
  participant L as Link Service
  participant K as KV Store
  participant Cc as Redis
  C->>G: POST /v1/links {long_url, alias?}
  G->>L: Auth + forward
  L->>K: PutItem(code→target) (strong write)
  L->>Cc: PURGE/DEL cache key
  L-->>G: 201 {short_url, code}
  G-->>C: 201
```

3.5.2 Asynchronous (event-driven)

Click analytics pipeline

1. Redirect Svc publishes ClickEvent to **Kafka** (non-blocking).
2. **Enricher** consumes → UA/Geo/referrer normalization.
3. **Aggregator** produces minute/hour/day **rollups** into **OLAP**; raw kept in **Object Storage**.
4. **Analytics API** serves aggregates; dashboards poll or use cached tiles.

3.6 Reliability & Availability

Where and how we achieve robustness:

- **Redundancy everywhere**
  - **Edge:** many POPs; anycast/geo routing.
  - **API/Services:** multi-AZ deployments; HPA for autoscale; at least N+1 instances.
  - **Redis/KV/Kafka:** clustered with replication (e.g., RF=3); zone-aware placement.
- **Failover strategies**

    - **Regional HA:** health checks and **automatic failover** to nearest healthy region (Global LB).
    - **Data replication:** cross-region async for KV mappings (target **RPO ≤ 5 s**); stream replication across clusters.
    - **Graceful degradation:** if Redis down → go to KV; if OLAP slow → serve cached tiles; if stream lagging → analytics freshness temporarily > 60 s, but redirects unaffected.

- **Resilience patterns**
    - **Circuit breakers** on KV/OLAP; timeouts (e.g., 100–300 ms) and **jittered retries**.
    - **Bulkheads**: redirect plane isolated from analytics plane (separate pools/quotas).
    - **Load shedding**: drop/queue non-critical analytics when saturation is detected.
    - **Backpressure**: bounded consumer lag, DLQ for poison messages.
- **Data durability**
    - Kafka RF=3, idempotent producers; OLAP replicated shards; raw events to object storage; daily backups/snapshots; retention policies.

3.7 Latency & Freshness Budgets (guidance)

Step	**Budget (p95)**
Edge accept & route	5–10 ms
Cache hit (Redis)	1–2 ms
KV miss (point read)	20–30 ms
Response build + 301	2–5 ms
Total redirect	**≤ 50 ms**
Enrich + aggregate	≤ 45 s
OLAP visible	≤ 15 s
Analytics freshness	**≤ 60 s**

3.8 Ownership & Boundaries

- **Link Service** owns **mapping truth**; other services only read via cache/KV.
- **Redirect Service** owns **hot path SLO**; zero hard dependencies on analytics.
- **Analytics** owns **derived truth**; corrections via batch/backfill do not affect redirect.

4. Component Design

The *URL Shortener with Click Analytics (Multi-Region)* platform follows a **microservices-based architecture**, with each service owning a well-defined domain and data boundary.
All services communicate through **REST APIs** or **asynchronous event streams**, with authentication and authorization governed by a centralized Auth service.

4.1 Overview of Services

Layer	Component	Type	Responsibility
Security & Access	Authentication / Authorization Service	Control plane	Identity, API keys, JWT validation
User Management	User Profile Service	Control plane	Manage users, tenants, quotas
Core Domain	Link Management Service	Data plane	Create/update/delete short links
Core Domain	Redirect Service	Data plane	Fast lookup and redirect
Analytics	Analytics Ingest Service	Data plane	Ingest click events
Analytics	Stream Processor / Enricher	Analytics plane	Process, enrich, and aggregate click data
Analytics	Analytics API	Control plane	Query and report analytics
Admin Ops	Admin / Configuration Service	Control plane	Manage feature flags, policies, blacklists
Infrastructure	Cache / KV / Stream / OLAP	Data layer	Persistent storage and queues

---

4.2 Authentication & Authorization Service

Responsibilities

- Handle user authentication using **OAuth 2.0 / OIDC** for web clients and **API key/JWT** for server-to-server calls.
- Issue short-lived access tokens and refresh tokens.
- Validate incoming JWTs in API Gateway or internal services.
- Maintain RBAC (Role-Based Access Control): e.g., admin, developer, read-only.
- Support rate-limit scopes (e.g., 1k req/min per token).

Boundaries

- Does *not* store link or analytics data.

- Does *not* manage quotas directly — provides identity attributes used by other services.

Interfaces

- **Exposes:**
  - POST /auth/login – authenticate users.
  - POST /auth/token – issue JWT.
  - GET /auth/introspect – validate token.
- **Consumes:**
  - IAM directory (e.g., Google OIDC, Cognito).
  - User Profile Service (for user metadata).

State Management

- Database: users, roles, tokens, revoked_tokens.
- Cached sessions in Redis for fast validation.

Scaling Strategy

- Stateless; horizontally scalable behind load balancer.
- Use distributed session cache for revocation checks.

Failure Behavior

- If down, new logins fail but existing tokens continue until expiry.
- Gateway caches token introspection to minimize outage impact.

4.3 User Profile Service

Responsibilities

- Maintain user metadata (name, plan, quota, domain ownership).
- Track usage: number of links created, API calls used, storage consumed.
- Provide tenant isolation for analytics and link ownership.

Boundaries

- Only manages *user and tenant information*.
- Does not perform authentication or link operations.

Interfaces

- **Exposes:**
  - GET /users/{id} – retrieve profile.
  - PATCH /users/{id} – update quota or plan.

    - GET /users/{id}/usage – report usage stats.
- **Consumes:**
    - Auth Service for user ID validation.
    - Link Service for usage counts.

State Management

- Database: users, tenants, usage_metrics.
- Periodic sync to cache for active users.

Scaling Strategy

- Moderate QPS; horizontally scalable with read replicas.
- Sharded by tenant_id.

Failure Behavior

- If offline, user dashboards may show stale quotas.
- Does not impact core redirect operations.

4.4 Link Management Service

Responsibilities

- Create, update, disable, and delete short links.
- Validate input URLs and prevent duplicates.
- Manage metadata (TTL, owner_id, tags).
- Purge cache and CDN entries when a link changes.

Boundaries

- Owns the *truth* of the link mapping (code → target_url).
- Does not perform redirect or analytics ingestion.

Interfaces

- **Exposes:**
    - POST /v1/links – create new short link.
    - PATCH /v1/links/{code} – modify or disable.
    - GET /v1/links/{code} – retrieve link metadata.
    - DELETE /v1/links/{code} – delete.
- **Consumes:**
    - Auth (JWT validation).

    - Admin Config Service (policy checks, domain blacklist).
    - KV Store (DynamoDB, Cassandra).
    - Cache Service (Redis).

State Management

- KV Store table: Links(code PK, long_url, owner_id, created_at, ttl, enabled).
- Caches popular mappings in Redis for quick retrieval.

Scaling Strategy

- Stateless; horizontally scalable; uses DB partitioning by code prefix.
- Writes are strongly consistent.

Failure Behavior

- Temporary outage halts link creation but not redirects.
- Recovery via database replication ensures no data loss.

4.5 Redirect Service

Responsibilities

- Handle all redirect requests (GET /{code}).
- Fetch mapping from cache or KV store.
- Return 301 (permanent) or 302 (temporary) redirect.
- Emit click event to analytics stream asynchronously.

Boundaries

- Purely stateless; does not perform writes except event emit.
- Does not modify link or user data.

Interfaces

- **Exposes:**
    - GET /{code} – redirect.
- **Consumes:**
    - Redis cache.
    - KV Store (fallback on cache miss).
    - Kafka (publish event).

State Management

- No persistent local state.

- Uses Redis for hot link caching.

Scaling Strategy

- Horizontally scalable; can scale to 100k+ requests/sec per region.
- CDN terminates TLS; redirect layer only handles lookup + event push.

Failure Behavior

- If Redis fails → falls back to KV lookup.
- If stream broker fails → queues events locally or drops non-critical analytics (redirect still succeeds).

4.6 Analytics Ingest Service

Responsibilities

- Accept events from Redirect Service (Kafka producer or HTTP fallback).
- Validate and normalize payloads.
- Forward to stream broker for enrichment.

Boundaries

- Handles only ingestion and schema validation; does not aggregate.

Interfaces

- **Exposes:**
    - POST /v1/events – receive event (backup path).
- **Consumes:**
    - Kafka/PubSub topics for downstream analytics.

State Management

- Transient (stateless). Uses schema registry (e.g., Avro/JSON Schema).

Scaling Strategy

- Horizontally scalable, stateless; can run multiple consumers/producers per region.

Failure Behavior

- If ingestion down, redirect service retries or buffers events; system degrades gracefully (redirects unaffected).

4.7 Stream Processor / Enricher

Responsibilities

- Consume click events.

- Enrich each record with:
    - Geo-location (MaxMind DB).
    - Referrer domain extraction.
    - User-Agent → device/OS/browser.
- Deduplicate events and aggregate by time windows (minute/hour/day).
- Write aggregates to OLAP warehouse and store raw in object storage.

Boundaries

- Operates on event streams only; no direct API exposure.

Interfaces

- **Consumes:** Kafka topics.
- **Emits:**
    - Enriched topics → OLAP ingestion.
    - Rollup tables → Analytics API.

State Management

- Stateful (window state) managed by Flink/Spark with checkpointing.
- Checkpoints every 60 s to object storage for fault recovery.

Scaling Strategy

- Parallel consumers; scale horizontally by Kafka partition count.

Failure Behavior

- Failing instances auto-restart; unprocessed events replayed (exactly-once semantics).
- Minor analytics delay; no data loss.

4.8 Analytics API Service

Responsibilities

- Provide aggregated analytics queries for dashboards.
- Serve metrics grouped by time, region, device, and referrer.
- Offer pagination and export features (CSV/JSON).

Boundaries

- Read-only; does not modify link or event data.

Interfaces

- **Exposes:**

    - GET /v1/links/{code}/stats
    - GET /v1/stats/summary
    - GET /v1/stats/export?format=csv
- **Consumes:**
    - OLAP warehouse (ClickHouse, BigQuery).

State Management

- No write state; caches frequent queries for 1–5 min.

Scaling Strategy

- Horizontally scalable; CPU-bound by query load.
- Uses OLAP replicas to distribute queries.

Failure Behavior

- If down, dashboards unavailable but data remains safe.
- Auto-recovers; retry logic at frontend.

4.9 Admin / Configuration Service

Responsibilities

- Manage feature flags, service configuration, blacklists, and system-level policies.
- Define quotas per plan (e.g., free, premium).
- Expose admin dashboard for operational management.

Boundaries

- Does not interact with redirect path.
- Scoped to control plane only.

Interfaces

- **Exposes:**
    - GET /admin/configs
    - POST /admin/blacklist
    - PATCH /admin/flags
- **Consumes:**
    - Auth (for admin role validation).
    - KV/Meta DB.

State Management

- Database tables: feature_flags, policies, blacklist, quotas.
- Cached in Redis for fast access.

Scaling Strategy

- Low QPS; replicated for availability; read-heavy.

Failure Behavior

- If offline, link creation may skip blacklist checks (risk mitigated via cache).
- Redirects unaffected.

4.10 Cross-Cutting Infrastructure Components

Component	Role	Failure Handling
Redis Cache	Speed up link lookups	Fallback to KV store
Kafka / PubSub	Transport events	Persistent queues, replay
OLAP Warehouse	Analytics queries	Replica fallback; dashboards show "stale" data
Object Storage	Backup, raw data	Multi-region redundancy
API Gateway	Security, routing	Active-active across AZs

4.11 Scaling Summary

Service	Stateless?	Scale Strategy	Peak QPS	Notes
Auth	Yes	LB + cache tokens	1k	Cached validations
User Profile	Mostly	Read replicas	500	Moderate QPS
Link Service	Yes	Partitioned KV	5k	Strong writes
Redirect	Yes	CDN + Redis	100k+	Highest QPS
Ingest	Yes	Kafka partitions	50k	Async
Stream Processor	Stateful	Partition-based	50k events/s	Checkpointed
Analytics API	Yes	Query replicas	2k	Read-heavy
Admin Config	Yes	Small cluster	100	Control plane only

4.12 Failure Domain Isolation

Failure	Impact	Mitigation
Auth outage	New sessions fail	Cached token validation
Link Service DB down	Link creation blocked	Cached reads; existing redirects safe
Redis down	Cache miss penalty	Fallback to KV store
Kafka outage	Analytics delayed	Retry/backpressure; redirect unaffected
OLAP down	Reports unavailable	Use cached dashboards
Region outage	Local failover	Global DNS reroute, async replication

5. Data Model

5.1 Logical Data Model (Domains & Entities)

Core OLTP (operational) entities

- **User**: account identity for creators/admins.
- **Organization/Tenant**: groups users, owns links/domains/quotas.
- **ApiKey**: credentials for server-to-server integration.
- **Domain**: custom domains used for short links (e.g., sho.rt, brand.co).
- **Link**: mapping code → long_url with flags, TTL, metadata.
- **Policy/Blacklist**: domain/rule sets for abuse prevention.
- **AuditLog**: immutable record of admin/user actions.
- **Job**: background/batch exports (report generation, backfill).

Analytics/Warehouse entities

- **EventRaw**: raw click events (as-ingested) with minimal PII (IP truncated).
- **EventEnriched**: event with UA, device, OS, country, referrer_domain.
- **AggMinute / AggHour / AggDay**: rollups per (code, time_bucket, dimensions…).
- **CostMetering**: usage counters for billing/cost monitoring.

5.2 Relationships & Ownership Boundaries

- **Organization 1—N Users**
- **Organization 1—N ApiKeys**
- **Organization 1—N Domains**
- **Organization 1—N Links**
- **Link 1—N EventRaw / EventEnriched**

- **Link 1—N Agg*** (aggregate rows referenced by link)
- **User 1—N AuditLogs**
- **Organization 1—N Jobs**

Ownership: Organization owns **Users, ApiKeys, Domains, Links**. Analytics rows are derived and reference **Link.code**.

5.3 ER Diagram (Mermaid)

```
erDiagram
  ORGANIZATIONS {
    string org_id PK
    string name
    string plan
    json   settings
    datetime created_at
  }
  USERS {
    string user_id PK
    string org_id FK
    string email
    string role  // admin|editor|viewer
    datetime created_at
    datetime last_login_at
  }
  API_KEYS {
    string key_id PK
    string org_id FK
    string name
    string hashed_key
    json   scopes
    datetime created_at
    datetime expires_at
    boolean enabled
```

```
}
DOMAINS {
  string domain PK
  string org_id FK
  boolean verified
  datetime created_at
}
LINKS {
  string code PK
  string org_id FK
  string domain FK // optional custom domain
  string long_url
  boolean enabled
  datetime created_at
  datetime expires_at
  json   tags
}
AUDIT_LOGS {
  string audit_id PK
  string org_id FK
  string user_id
  string action
  json   details
  datetime created_at
}
EVENTS_RAW {
  string event_id PK
  string code FK
  datetime ts
  string ip_trunc
  string user_agent
  string referrer
  string edge_region
}
EVENTS_ENRICHED {
  string event_id PK
  string code FK
  datetime ts
  string country
  string device
  string os
  string browser
  string referrer_domain
}
AGG_DAY {
  string code FK
  date   day_bucket
  string country
  string device
  string referrer_domain
  int    clicks
  int    uniques
}

ORGANIZATIONS ||--o{ USERS : has
ORGANIZATIONS ||--o{ API_KEYS : issues
ORGANIZATIONS ||--o{ DOMAINS : owns
ORGANIZATIONS ||--o{ LINKS : owns
LINKS ||--o{ EVENTS_RAW : receives
LINKS ||--o{ EVENTS_ENRICHED : receives
LINKS ||--o{ AGG_DAY : aggregates
ORGANIZATIONS ||--o{ AUDIT_LOGS : records
```

5.4 Physical Schema (OLTP vs Analytics)

OLTP (Relational / KV-backed)

- **Purpose:** Strongly consistent writes for link mapping & admin data.
- **Stores:**
    - **KV (DynamoDB/Cassandra)** for Links (primary lookups by code).
    - **Relational (PostgreSQL)** for Organizations, Users, ApiKeys, Domains, AuditLogs, Jobs.

Tables / Keys

- links(code PK, org_id, domain, long_url, enabled, created_at, expires_at, tags)
    - **Partition key**: code (use randomized/hashed prefix to avoid hot partitions).
    - **Indexes**: GSI on (org_id, created_at) for org-level listing; GSI on domain+code for vanity domains.
- organizations(org_id PK, name, plan, settings, created_at)
- users(user_id PK, org_id FK, email UNIQUE, role, created_at, last_login_at)
    - Index: (org_id, role)
- api_keys(key_id PK, org_id FK, hashed_key, scopes, enabled, expires_at, created_at)
    - Index: (org_id, created_at), (hashed_key) unique
- domains(domain PK, org_id FK, verified, created_at)
- audit_logs(audit_id PK, org_id, user_id, action, details, created_at)
    - Partition by org_id, index on created_at DESC for paging

Analytics (Columnar + Objects)

- **Purpose:** High-volume events; fast scans & aggregations.
- **Stores:**
    - **Object Storage** (S3/GCS): events_raw & checkpoints (Parquet).
    - **OLAP** (ClickHouse/BigQuery):
        - events_enriched partitioned by event_date; clustered by code.
        - agg_minute, agg_hour, agg_day as materialized tables/views.

ClickHouse example (OLAP)

```
CREATE TABLE events_enriched (
  event_date Date,
  code String,
  ts DateTime,
  country LowCardinality(String),
  device LowCardinality(String),
  os LowCardinality(String),
  browser LowCardinality(String),
  referrer_domain LowCardinality(String),
  event_id String
) ENGINE = MergeTree()
PARTITION BY event_date
ORDER BY (code, ts);

CREATE MATERIALIZED VIEW agg_day
ENGINE = SummingMergeTree()
PARTITION BY toDate(ts)
ORDER BY (code, toDate(ts), country, device, referrer_domain)
AS
SELECT
  code,
  toDate(ts) AS day_bucket,
  country, device, referrer_domain,
  count() AS clicks,
  uniqCombined(event_id) AS uniques
FROM events_enriched
GROUP BY code, day_bucket, country, device, referrer_domain;
```

5.5 Normalization vs Denormalization

- **OLTP:** normalized enough to keep **single source of truth** (e.g., links.long_url, organizations.plan).
- **Analytics:** denormalized **wide rows** and **pre-aggregates** (minute/hour/day) to minimize query latency and cost.
- Rationale: link reads must be single-key lookups; analytics must scan/aggregate billions of rows efficiently.

5.6 SQL vs NoSQL Selection

- **Links mapping**: **KV / wide-column (DynamoDB/Cassandra)** → single-digit ms lookups by code.
- **Org/User/API keys**: **Relational** for transactional consistency and joins.
- **Events & Aggregates**: **Columnar OLAP** for scan/aggregate; **Object storage** for cheap, durable raw logs.

5.7 Partitioning & Indexing

- **KV**: hash-prefix codes to distribute load evenly; avoid hot partitions for viral links.
- **OLAP**: partition by date; cluster by code (and optionally org_id) for targeted queries.
- **Relational**: composite indexes for frequent filters (e.g., (org_id, created_at DESC)).

5.8 Data Retention & Archival

- **Raw events**: keep **90 days** (S3/GCS lifecycle: Standard → Infrequent → Glacier).
- **Aggregates**: keep **12–24 months** for trending; beyond that, keep monthly rollups.
- **Audit logs**: keep **1 year** (compliance).
- **Soft deletes**: links disabled but retained for 30 days before purge.

---

6. API Design

6.1 Principles & Conventions

- **Style:** REST (JSON). Internal services may use gRPC.
- **Versioning:** Path-based (/v1/...), reserve /v2 for breaking changes.
- **Auth:** OAuth2/OIDC for web; **API keys/JWT** for server-to-server.
- **Idempotency:** All POST writes accept Idempotency-Key header.
- **Pagination:** limit (≤1000), cursor (opaque).
- **Filtering/Sorting:** Query params (from, to, order_by=clicks_desc, country=IN).
- **Rate Limits:** Per API key & IP; headers return X-RateLimit-*.

6.2 Public Endpoints (Selected)

1) Create Short Link

Purpose: Create a mapping code → long_url.
Method/URL: POST /v1/links
Auth: API key or user JWT (role: editor/admin)
Headers: Idempotency-Key: <uuid> (recommended)

Request (JSON)

```
{
  "long_url": "https://example.com/promo?id=123",
  "custom_alias": "promo123",     // optional
  "domain": "sho.rt",             // optional (must be owned/verified)
  "expires_at": "2026-12-31T23:59:59Z",
  "tags": ["campaign-jan", "ads"]
}
```

Response 201

```
{
  "code": "promo123",
  "short_url": "https://sho.rt/promo123",
  "long_url": "https://example.com/promo?id=123",
  "expires_at": "2026-12-31T23:59:59Z",
  "enabled": true
}
```

Errors: 400 invalid URL, 401/403 auth, 409 alias taken, 422 policy violation, 429 rate limited, 500 server.

---

2) Resolve/Preview Link (Metadata)

*(For UI/admin; **not** the public redirect path)*
Method/URL: GET /v1/links/{code}
Auth: JWT or API key
Response 200

```
{
  "code": "abc12",
  "domain": "sho.rt",
  "long_url": "https://example.com",
  "enabled": true,
  "created_at": "2025-11-10T10:15:00Z",
  "expires_at": null,
  "tags": ["promo"]
}
```

Errors: 404 not found; 403 if accessing other org’s link.

3) Redirect (Public)

Method/URL: GET /{code} (edge/CDN route)

Auth: none

Response: 301 Location: <long_url> (or 302 if temporary)

Headers: Cache-Control, Surrogate-Key: link:{code} for CDN purge.

4) Update/Disable Link

Method/URL: PATCH /v1/links/{code}

Auth: editor/admin

Request

```
{
  "long_url": "https://example.com/new",
  "enabled": false,
  "expires_at": "2026-01-01T00:00:00Z",
  "tags": ["winter"]
}
```

Response 200: updated link JSON

Errors: 404, 409 conflict (concurrent update via ETag/If-Match), 422 policy.

5) Delete Link

Method/URL: DELETE /v1/links/{code}

Auth: admin

Response 204

Effect: link removed; CDN/cache purged.

6) List Links (Org scope)

Method/URL: GET /v1/links?limit=50&cursor=...&tag=campaign-jan&enabled=true

Auth: JWT/API key

Response 200

```
{
  "items": [{ "code":"a1", "long_url":"...", "enabled":true }, ...],
  "next_cursor": "eyJvZmZzZXQiOjUwLC..."
}
```

7) Analytics: Per-Link Stats

Method/URL:
GET /v1/links/{code}/stats?from=2025-11-01T00:00:00Z&to=2025-11-07T23:59:59Z&dimensions=country,device&bucket=day&limit=1000

Response 200

```
{
  "code": "promo123",
  "bucket": "day",
  "from": "2025-11-01T00:00:00Z",
  "to": "2025-11-07T23:59:59Z",
  "rows": [
    {"day":"2025-11-01","country":"IN","device":"Mobile","clicks":12450,"uniques":9876},
    {"day":"2025-11-01","country":"US","device":"Desktop","clicks":2210,"uniques":1850}
  ]
}
```

Errors: 400 invalid params, 403 cross-tenant access, 429 rate limited.

---

8) Export Report (Async Job)

Method/URL: POST /v1/reports/export
Request

```
{
  "scope": {"org_id": "org_123"},
  "from": "2025-11-01",
  "to": "2025-11-30",
  "format": "csv",
  "dimensions": ["day","country","referrer_domain"],
  "metrics": ["clicks","uniques"]
}
```

Response 202

```
{"job_id":"job_9f2a", "status":"queued"}
```

Follow-up: GET /v1/jobs/{job_id} → returns status & download URL when ready.

6.3 Admin/Operations Endpoints (Selected)

- POST /v1/admin/blacklist – add domain/host rule.
- POST /v1/admin/disable/{code} – immediate disable + purge.
- GET /v1/admin/quotas – list org quotas.
- PATCH /v1/admin/feature-flags – toggle features per org.

6.4 Error Semantics (Consistent JSON)

```
{
  "error": {
    "code": "alias_conflict",
    "message": "The custom alias is already in use",
    "details": {"alias":"promo123"}
  }
}
```

- Standard HTTP codes: 200/201/202/204 success; 400/401/403/404/409/412/422/429/5xx errors.
- Retry-After header for 429.

6.5 Versioning Strategy

- Path-based: /v1/... (frozen contracts).
- Introduce /v2 for breaking changes; **sunset headers** to communicate deprecation schedules.
- **Backward-compatible** additions (new fields) allowed within v1.

6.6 Pagination, Filtering, Sorting

- **Pagination**: limit + cursor (opaque).
- **Filtering**: e.g., ?enabled=true&tag=promo&from=...&to=....
- **Sorting**: order_by=created_at_desc (for lists) or order_by=clicks_desc (stats).

6.7 Idempotency & Concurrency

- **Idempotency-Key** required for POST creates; server stores recent keys (TTL 24h).
- **Optimistic concurrency** on updates: ETag + If-Match to prevent lost updates.

- **At-least-once** analytics ingestion with **dedupe** by event_id.

6.8 Security & Authorization

- **AuthN**: OAuth2/OIDC for UI sessions; API keys/JWT for programmatic access.
- **AuthZ**: RBAC (roles: admin, editor, viewer). Optional ABAC with org-based attributes.
- **Rate limits**: per API key/user & per endpoint class (writes stricter).
- **Scopes**: links:read, links:write, analytics:read, admin:*.
- **Auditing**: all admin and destructive actions logged in AuditLog.

6.9 SLA Headers & Caching

- **Cache-Control** on stats responses (e.g., max-age=30 for read-heavy endpoints).
- **Surrogate-Key** on link metadata so CDN/edge can purge quickly after updates.

Quick Reference: Endpoint Matrix

Endpoint	Method	Scope	Notes
/v1/links	POST	editor	Idempotent create
/v1/links/{code}	GET	viewer	Metadata
/v1/links/{code}	PATCH	editor	ETag/If-Match
/v1/links/{code}	DELETE	admin	Soft delete or hard
/v1/links	GET	viewer	List w/ filters, pagination
/{code}	GET	public	301/302 redirect
/v1/links/{code}/stats	GET	viewer	OLAP query
/v1/reports/export	POST	editor	Async job
/v1/jobs/{job_id}	GET	viewer	Job status
/v1/admin/*	mixed	admin	Policies/flags/quotas

7. Analytics Pipeline

7.1 Overview

Goal: collect every redirect as an **event**, enrich it (UA/Geo/Referrer), aggregate in real time and batch, and expose low-latency OLAP queries with **≤60s freshness**.

Stages:
Producers → Ingestion (stream) → Real-time processing → Batch/ETL → Storage (lake+warehouse) → Analytics API/Dashboards → Feedback (alerts/recos).

7.2 Event Producers

- **Redirect Service** (primary): emits ClickEvent per successful redirect.
- **Link Service**: emits LinkCreated/Updated/Disabled control events (for lineage and cache invalidation).
- **Platform metrics**: optional ServiceLatency, CacheHitRatio for operational analytics.

ClickEvent (minimal raw):

```
{
  "event_id": "uuid",
  "code": "abc12",
  "ts": "2025-11-16T12:34:56.789Z",
  "ip_trunc": "203.0.113.0/24",
  "user_agent": "Mozilla/5.0 ...",
  "referrer": "https://twitter.com/...",
  "edge_region": "BOM"
}
```

7.3 Ingestion Layer (Stream Broker)

- **Kafka / PubSub** with **N partitions** per region; **RF=3** replication.
- Topics:
  - click_events_raw (write-heavy)
  - link_control_events
  - click_events_enriched (optional)
- **Ordering:** per-partition; choose partitioner by hash(code) for locality.
- **Backpressure:** producer acks=all, linger/batch to reduce overhead.

7.4 Stream Processing (Real-time)

- **Framework:** Flink / Kafka Streams (stateful operators, checkpoints to object storage).
- **Enrichment:**

    - **GeoIP** (country) via in-memory DB with hourly refresh
    - **UA parsing** → device/OS/browser (regex DB)
    - **Referrer** → domain extraction/normalization
- **Windowed aggregation:** tumbling minute → hour → day; **materialized views**.
- **Deduplication:** keyed by event_id using compacted state store (TTL 24h) + Bloom filter guard.
- **Uniques: HyperLogLog** per (code, bucket, dimension*) for memory-efficient cardinality.

Produced outputs:

- events_enriched (optional stream → OLAP ingest)
- agg_minute (pre-aggregates to OLAP)
- Raw → object storage (Parquet) for replay.

7.5 Batch Processing (ETL)

- Nightly jobs:
    - **Backfills/corrections** (late/out-of-order events beyond watermark).
    - **Rollups** (minute→hour→day) and compaction.
    - **Data quality** checks (volume, distinct codes, HLL error bounds).
- Rebuild aggregates for impacted ranges when enrichment rules change.

7.6 Storage for Analytics

- **Data Lake:** object storage (s3://events/raw/yyyy/mm/dd/*.parquet) with lifecycle → IA/Glacier after 30/90 days.
- **Warehouse (OLAP):** ClickHouse/BigQuery
    - events_enriched partitioned by event_date, **ORDER BY (code, ts)**
    - agg_minute, agg_hour, agg_day as materialized tables/views clustered by (code, bucket)
- **Serving:** Analytics API runs parameterized queries with safe limits + result caching (30–120s).

7.7 Schema Evolution

- **Schema registry** (Avro/JSON Schema).
- **Backward-compatible** additions only in v1 (new nullable fields).
- Producers send schema_version; processors branch by version.

- OLAP uses **ADD COLUMN** (nullable/LowCardinality) to avoid rewrite.

7.8 Delivery Semantics & Dedup

- **Producers:** idempotent; acks=all, retries w/ backoff.
- **Stream:** effectively **at-least-once** end-to-end; dedup by event_id at processor & OLAP unique key.
- **Exactly-once** optional with Flink EOS (2PC sinks) if required; cost/complexity trade-off.
- **Late events:** watermark = event time − 5m; late data lands in **corrections** tables merged nightly.

7.9 Product Feedback Loops

- **Dashboards** (owner/org views, per-code insights).
- **Alerts** (e.g., sudden traffic spike or unusual referrer → notify owner).
- **Abuse signals** (bot-like patterns) feed **Admin Service** to auto-throttle/disable links.
- **Recommendations** (future): smart TTL suggestions, best posting times by geography.

---

8. Caching & CDN Strategy

8.1 Goals

- **Reduce latency** (p95 redirect < 50 ms).
- **Offload databases** (≥95% hits from CDN/Redis).
- **Improve resilience** (serve cached when origins throttle).

8.2 CDN (Global Edge) vs Application Cache (Redis)

CDN / Edge

- Caches **HTTP 301/302** responses for hot codes.
- Anycast routing to nearest POP; **Surrogate-Key: link:{code}** for precise purge.
- Cacheable metadata GETs (e.g., /v1/links/{code} if allowed).

Application Cache (Redis/Memcached)

- **Cache-aside** on redirect path (GET code first).
- **Negative caching** for 404/disabled codes (short TTL).
- **Write-through** on link updates (optional) + **explicit purge** to keep strong RYW.

8.3 Cache Keys & Namespacing

- **Keys:** link:{code} → { target_url, enabled, expires_at, policy_rev }
- **Variant keys:** linkmeta:{org_id}:{code}, stats:{code}:{bucket}:{from}:{to}:{dims_hash}
- **Versioning:** include policy_rev or schema_rev to invalidate old payloads safely.

8.4 TTL Strategy

- **CDN:** 300s default; stale-while-revalidate=60; purge on link change/disable.
- **Redis:** 5–10 minutes for positives; **60–120s** for negative cache entries.
- **Stats responses:** 30–120s to cap OLAP load.

Trade-offs: higher TTL → higher hit rate but more staleness risk; mitigated via targeted purges and SWR.

8.5 Stampede Prevention

- **Single-flight locks** (mutex per code) on cache miss.
- **Jittered TTLs** (±10–20%) to avoid thundering herd.
- **Refresh-ahead** for very hot keys when TTL < T (e.g., 15% of TTL).

8.6 Warm-up & Pre-population

- **Preload** viral links on deploy or when traffic predictor flags spikes.
- **Batch hydrate** top N codes hourly by recent clicks.
- **Region-aware warming** to nearest POP/Redis shard.

8.7 Failure & Degradation

- If Redis is degraded: increase per-instance L1 cache; rely on KV with stricter timeouts.
- If CDN purge fails: fall back to **short TTLs** temporarily.
- Always prefer **serving a redirect** (even with slightly stale target) over 5xx; audit such cases.

---

9. Scalability & Sharding

9.1 Expected Scale (initial → 12 months)

- **Active links:** 100M → 300M
- **Redirects:** avg 20k RPS → peak 100k RPS (global), bursts during campaigns

- **Events/day:** 0.5–1.5B
- **Analytics storage growth:** ~60–120 GB compressed/day (raw), aggregates 10–20% of raw

Strategy: Horizontal scaling first (more instances/partitions/nodes), vertical scaling only for specialized OLAP nodes.

9.2 Database Sharding Strategies

KV Store (code → target)

- **Primary: Hash-based sharding** on **randomized code prefix** (base-62 shuffled) → even distribution, avoids hotspots.
- **Pros:** simple, uniform; **Cons:** range scans by code impossible (not needed).
- **Hotspot mitigation:** detect viral codes → **promote to edge/L1 cache**, optionally pin to in-memory map with micro-TTL.

Relational (Users/Orgs/Keys)

- **Org-based sharding** (range/hash on org_id) with read replicas.
- **Directory-based** mapping to shards for future rebalancing.

OLAP (events & aggregates)

- **Partition by date** (day) and **cluster by (code, ts)**.
- **Add replicas** per region for read scaling.

9.3 Choosing Shard Keys & Consequences

- **code** (randomized) spreads reads evenly; great for point lookups.
- **org_id** helps list org's links; keep GSI/secondary index for this path.
- **Pitfall:** vanity aliases concentrated by certain prefixes → add **prefix randomization** or bucket map to avoid hot shards.

9.4 Re-sharding & Online Migration

- **Plan:** introduce **routing service / shard map** (versioned) and **dual-write** during transition.
- Steps:
    1. Create new shard set.
    2. **Backfill** data (change data capture).
    3. **Dual-read** (prefer new, fallback old) + **dual-write**.
    4. Cut over by shard-map version; monitor; decommission old.

- **Zero-downtime** via feature flags and progressive traffic shifting.

9.5 Application Layer Scalability

- **Stateless services** (Redirect, Link, Auth, Analytics API) behind **L4/L7 load balancers**.
- **Autoscaling** via p95 latency/RPS.
- **No sticky sessions**: JWT tokens for auth; session data in Redis only if absolutely needed (e.g., rate-limit counters).

9.6 Session & Rate State

- **Sessions:** JWT (self-contained), short TTL; refresh tokens in Auth store.
- **Rate limits:** token bucket counters in Redis (per key/IP), sharded by key hash.

9.7 Capacity Guardrails

- **Redis**: ≤75% memory, eviction LRU; shards sized for peak +30%.
- **Kafka**: partitions sized for **2×** peak throughput; segment/retention configured to avoid broker GC stalls.
- **KV**: provisioned capacity for **miss QPS** with 2× headroom; monitor p95 read.

10. Rate Limiting & Resilience

10.1 Purpose of Rate Limiting

Rate limiting protects the platform from:

- **Abuse** (bots, spam, malicious scripts creating thousands of links).
- **Accidental overload** (clients retrying aggressively or loops).
- **Fair resource allocation** among tenants.
- **Cost control** (prevent excessive OLAP/analytics queries).

It ensures that system performance and availability remain stable even during traffic spikes.

10.2 Granularity of Limits

Granularity	Typical Use	Example Limit
Per IP	Prevent DDOS or scraper abuse	100 req/min
Per User	Account-level quota	1000 req/min
Per API Key	Integration quota	5000 req/min

Granularity	Typical Use	Example Limit
Per Organization	Tenant fairness	100k req/hour
Per Endpoint	Critical path control	/v1/links POST: 60/min /v1/stats GET: 10/sec

Implementation:

- Use a **token bucket** algorithm in Redis.
- Key pattern: ratelimit:{scope}:{id}:{endpoint}.
- Fields: remaining tokens, last refill timestamp.
- Leaky bucket or fixed window for backup in low-traffic APIs.

Headers returned:

X-RateLimit-Limit: 100

X-RateLimit-Remaining: 72

X-RateLimit-Reset: 1731762000

10.3 Resilience Patterns

1. Timeouts

- Prevent hung threads and request pile-up.
- Default values:
    - Cache (Redis): 50–100 ms
    - KV Store: 300 ms
    - Kafka Produce: 200 ms
    - OLAP Query: 400–1000 ms (depends on query complexity)
- Each service enforces both **client-side** and **server-side** timeouts.

2. Retries with Backoff and Jitter

- Retry **only safe, idempotent operations** (GET, certain POST with Idempotency-Key).
- Backoff = exponential with random jitter (±10–20%) to prevent synchronized storms.
- Example: wait 100 ms, 300 ms, 900 ms → stop after 3 attempts.

3. Circuit Breakers

- Stop cascading failures when dependencies fail.
- Implement using a **half-open** pattern:
    - Trip if >50% failures or latency exceeds threshold over 30 s.

    - Remain open for 60 s, then test requests gradually.
- Example: Redirect service isolates Redis/KV calls via breaker.

4. Bulkheads

- Isolate resources by:
    - **Feature:** redirect vs analytics.
    - **Tenant:** large organizations in separate thread pools.
    - **Region:** active-active clusters (US/EU/APAC) segregated.
- Prevents "noisy neighbor" effects.

10.4 Graceful Degradation

When a dependency is slow or unavailable:

- **Redis failure:** fallback to KV lookups (slower but functional).
- **Analytics lag:** dashboards show partial data with "Data delayed" badge.
- **OLAP down:** serve cached report or simplified aggregates.
- **Admin service offline:** disable policy checks temporarily (with alerts).

Goal: **redirects always succeed** (even if analytics lag).

10.5 Load Shedding

When system load exceeds safe thresholds (CPU > 85%, queue length > limit):

- Drop low-priority traffic (analytics queries, admin jobs).
- For redirect path:
    - Serve cached/stale data or static response before dropping.
    - Return HTTP 429 Too Many Requests for repeated overload.
- Log dropped requests in monitoring system for root-cause review.

10.6 Recovery Strategy

- Services auto-restart on crash (Kubernetes liveness probes).
- Stuck queues drained gradually.
- Circuit breakers auto-close after successful health checks.
- Postmortems required if SLO breach > 5% of error budget.

---

11. Observability & SLOs

11.1 Overview

Observability = *metrics + logs + traces*, providing end-to-end visibility.

Component	Purpose
Monitoring	Quantitative health (latency, error rate, throughput).
Logging	Qualitative insight (context, error causes).
Tracing	Distributed flow (cross-service latency attribution).

11.2 Metrics

RED Metrics (APIs)

Metric	Definition	Example Target
Requests	Total per endpoint	10k/s
Errors	4xx+5xx %	< 0.1%
Duration (p95)	Response latency	< 50 ms redirect

USE Metrics (Resources)

Metric	Component	Target
Utilization	CPU, memory, disk	< 75%
Saturation	Queue length, thread pool	< 80%
Errors	Cache miss, broker retry	< 1% sustained

11.3 Logging

- **Structured JSON logs:** {timestamp, trace_id, span_id, service, message, error, latency_ms}
- Log levels: INFO, WARN, ERROR, CRITICAL.
- Correlation IDs injected at gateway (trace propagation header).
- Sensitive data (e.g., URLs, IPs) redacted or hashed.
- Centralized ingestion: ELK (Elasticsearch–Logstash–Kibana) or Cloud Logging.

11.4 Distributed Tracing

- Framework: **OpenTelemetry** or **Jaeger**.

- Trace spans: Gateway → Redirect → Cache → KV → Stream emit.
- Each span logs:
  - latency
  - upstream/downstream dependencies
  - response codes
- Enables root-cause analysis of slow requests.

11.5 Dashboards

Dashboard	Key Panels
Redirect Service	p50/p95/p99 latency, cache hit %, 5xx rate
Link Service	create/update latency, DB write throughput
Analytics Stream	lag (seconds), events processed/s, DLQ rate
OLAP	query latency, freshness lag
Global	per-region error heatmap, uptime summary

11.6 SLOs (Service Level Objectives)

SLI	Target	Period
Redirect availability	≥ 99.99%	30 days
Redirect latency p95	≤ 50 ms	30 days
Analytics freshness	≤ 60 s	rolling hour
API availability	≥ 99.9%	30 days
Data loss (events)	< 0.01%	continuous

Error Budget:
If 99.99% SLO → 0.01% error budget (≈ 4.3 min downtime/month).
Used to control deployment velocity — if budget exhausted, freeze releases until recovery.

11.7 Alerting

- **Multi-window, multi-burn-rate** alerts (fast + slow detection).
- Example:
  - 2% error budget in 1h → page SRE.

  - 5% error budget in 6h → incident review.
- **Synthetic canaries:** periodic simulated redirects for every region.

---

12. Infrastructure & Deployment

12.1 Deployment Model

Cloud-native (IaaS/PaaS)

- Hosted on **AWS / GCP**.
- Core compute in **Kubernetes (EKS/GKE)** clusters per region.
- Managed services for databases: DynamoDB, Redis (Elasticache), Kafka (MSK), ClickHouse/BigQuery.

12.2 Multi-AZ & Multi-Region Setup

Component	Resilience Strategy
Kubernetes nodes	Multi-AZ worker pools
Redis / KV	Replicated across 3 AZs
Kafka	3 brokers/region, RF=3
OLAP	Cross-region replica sets
Object Storage	Multi-region versioning enabled

- **Failover:** Anycast DNS or Global Load Balancer routes traffic to nearest healthy region.
- **Replication lag:** < 5 s between active regions.
- **Isolation:** analytics and control planes are separate.

12.3 Network Topology

```
VPC (per region)
 ├── Public Subnets
│   ├── Load Balancers (ALB/NLB)
│   ├── Bastion hosts
│
 ├── Private Subnets
│   ├── App Services (EKS nodes)
```

```
|   ├── Redis clusters
|   ├── Kafka brokers
|   ├── OLAP and databases
|
└── Security Groups:
   - Allow 443 inbound from CDN
   - Least privilege east-west (service mesh mTLS)
```

- **Peering / Transit Gateway** connects regional VPCs.
- **NAT Gateways** for outbound internet access.
- **WAF** and **API Gateway** in front of all public endpoints.

12.4 Environments

Environment	Purpose	Isolation
Development	Local testing; mock dependencies	Separate credentials
Staging	Pre-production; full load simulation	Replica data, no production creds
Production	Live traffic	Strict access, encrypted secrets

Each environment isolated via dedicated Kubernetes namespaces, VPCs, and IAM policies.

12.5 CI/CD Pipeline

Stage	Description
Source Control	GitHub / GitLab; feature branches with PRs
Build	Docker images built via CI (GitHub Actions, Jenkins)
Test	Unit, integration, security (SAST), and load tests
Artifact Storage	Container registry + Helm chart repo
Deploy	ArgoCD or FluxCD syncs manifests to K8s
Release Strategy	Blue/Green for control plane; Canary for redirect service
Rollback	Helm version rollback or Argo “undo”

Automated gates prevent deployment if error budget exceeded.

12.6 Infrastructure as Code (IaC)

Tool: Terraform (optionally Pulumi or CloudFormation).

Managed via GitOps:

- Each environment's infra defined declaratively (main.tf per region).
- Terraform state in remote backend (S3 + DynamoDB lock).
- Code reviewed via PRs → ensures peer review for all infra changes.

Benefits:

- **Reproducibility:** identical environments.
- **Auditability:** version control for infra.
- **Traceability:** link every change to a commit and owner.
- **Rollback:** reapply old state in minutes.

12.7 Security and Deployment Policies

- **mTLS** within cluster (service mesh).
- **Image signing** and **vulnerability scans** before deploy.
- **Secrets** in managed vault (AWS Secrets Manager / HashiCorp Vault).
- **Least privilege IAM** per service account.
- **Network policies** restrict lateral movement.

12.8 Deployment Example Timeline

1. Developer pushes code → GitHub triggers CI.
2. Docker image built → scanned → pushed to registry.
3. Helm chart updated → merged → ArgoCD syncs.
4. Canary deployment (10%) monitored for 10 min.
5. If SLOs hold → full rollout (100%).
6. Metrics + traces confirm stability → close release.

12.9 Disaster Recovery & Backups

Component	Backup Frequency	Retention	Recovery
KV Store	Daily snapshot	30 days	<15 min restore

Component	Backup Frequency	Retention	Recovery
OLAP	Daily export	7 days	<1 h rebuild
Object Storage	Versioned	90 days	Cross-region restore
Redis	RDB every 6h	24h	Reload snapshot

13. Security & Privacy

13.1 Defense-in-Depth Security Model

The platform follows a **defense-in-depth** approach, layering protection across the **network**, **application**, and **data** tiers.
Every component assumes upstream breaches are possible and validates input and permissions locally.

Layer	Controls
Network	VPC isolation, private subnets, WAF, API Gateway auth, mTLS between services
Application	Input validation, RBAC/ABAC, CSRF protection, rate limiting
Data	Encryption in transit & at rest, strict IAM, row-level access, audit logs

---

13.2 Authentication

- **Identity Providers:** Google, Microsoft, GitHub (via **OIDC / OAuth 2.0**)
- **Formats:**
  - JWT (JSON Web Token) for service-to-service auth
  - API Keys for programmatic clients (scoped + expiring)
- **Session Management:**
  - UI uses short-lived access tokens (15 min) + refresh tokens (8 h)
  - Token introspection via Auth Service
  - Single Sign-On (SSO) for enterprise tenants
- **Passwordless option:** via email magic link for end users
- **Replay protection:** nonce + token expiry validation

13.3 Authorization

- **Model:** hybrid **RBAC + ABAC**

    - RBAC: Roles = admin, editor, viewer
    - ABAC: Policies evaluated on resource attributes (e.g., link.org_id == user.org_id)
- **Tenant Isolation:**
    - Org ID enforced at JWT claims level
    - DB queries filtered automatically via ORM policies
- **Fine-grained scopes:**
    - links:read, links:write, analytics:read, admin:*

13.4 Data Protection

Category	**Protection Mechanism**
In Transit	TLS 1.3 for all client/server & inter-service traffic; HSTS headers; perfect-forward secrecy
At Rest	AES-256 encryption on DBs, caches, object storage; managed KMS rotation every 90 days
Backups	Encrypted snapshots + checksum verification
Logs	Tokenization of PII; retention 30 days
Secrets Management	Vault / AWS Secrets Manager — secrets never stored in code or config maps; rotated automatically

---

13.5 Privacy & Compliance

- **Data Minimization:** store only truncated IP (e.g., /24) and coarse location.
- **Purpose Limitation:** analytics only for link-owner dashboards, not resale.
- **User Consent:** banners for cookies/analytics, opt-out at account level.
- **Data Subject Rights:**
    - *Export:* JSON/CSV of personal data via API /v1/me/export
    - *Delete:* hard-delete requests processed in ≤ 30 days (GDPR Art. 17)
- **Compliance:** follows **GDPR**, **CCPA**, and **ISO 27001** best practices.
- **Audit Logs:** immutable table audit_logs records every admin action (who, what, when, origin IP).

---

14. Testing & Maintenance

14.1 Testing Strategy

Test Type	Scope	Tools	Frequency
Unit Tests	Pure logic (URL parser, ID gen, validators)	PyTest / Jest	On commit
Integration Tests	API + DB + Cache flows	Postman / TestContainers	On merge
Contract Tests	Between microservices (Link↔Redirect↔Analytics)	Pact	CI stage
End-to-End (E2E)	Full scenario (create→click→stats)	Cypress / Playwright	Pre-release
Performance Tests	Load, stress, soak (10× peak)	k6 / Locust	Weekly
Security Tests	SAST/DAST scans, dependency audit	OWASP ZAP, Trivy	Per build
Chaos Tests	Failure injections (kill Redis/Kafka nodes)	Gremlin / Litmus	Monthly

---

14.2 Test Coverage & Critical Paths

Coverage Target: ≥ 80 % overall, 100 % on core modules.
Critical paths:

1. Link creation → persistence.
2. Redirect lookup → cache → DB fallback.
3. Analytics ingestion → aggregation.
4. Auth token validation.

14.3 Test Environments & Data

- Staging DB seeded with **sanitized production-like** datasets.
- Synthetic traffic generator simulates 50 k RPS.
- Data masking removes emails, IPs before test import.
- Isolated VPC for perf tests to avoid prod impact.

14.4 Load & Stress Testing

- Load Profile: baseline 50 k RPS, ramp up to 100 k.
- Metrics tracked: p95 latency, error %, CPU, memory, cache hit %.
- Goal: system degrades gracefully (< 1 % errors under 2× peak).

- Long-run (soak) tests validate memory leaks & connection recycling.

14.5 Maintenance & Operations

- **Release Management:** semantic versioning (v1.2.3); changelogs auto-generated.
- **Deprecation Policy:** announce ≥ 6 months before removal; dual-support old/new API.
- **Runbooks:** stored in Confluence/Notion, covering:
  - Incident classification & escalation (P1–P3).
  - Redis/Kafka recovery steps.
  - Cache purge scripts.
  - Manual failover procedures.
- **Operational Tasks:** health checks, cert renewals, index maintenance, log rotation.

---

15. Cost & Capacity Planning

15.1 Major Cost Drivers

Category	Sub-components	Optimization Strategy
Compute	Kubernetes nodes, autoscaled pods	Right-size CPU/mem; spot instances for analytics
Storage	KV (DB), OLAP, S3 (raw data), Redis	Data lifecycle; compression; TTL for caches
Network Egress	CDN→Client, Inter-region replication	Regional serving; compress payloads
Managed Services	Kafka, ClickHouse, Monitoring	Reserved capacity; usage alerts
Observability	Logs + metrics retention	Shorter log retention (30 days)

15.2 Cost vs Architecture Choices

- **Multi-Region:** +20 % infra cost ↔ 99.99 % availability.
- **Longer Retention:** raw events 90 → 30 days saves ≈ 40 %.
- **OLAP Replication:** +15 % cost ↔ faster analytics SLOs.
- **Serverless Kafka vs Self-hosted:** managed cost ↑ 25 %, ops load ↓ 80 %.

15.3 Capacity Planning Process

1. **Estimate Demand:** based on active links × avg redirects/link/day.
2. **Model Workload:**
   - Redirect = read-heavy (Redis/KV I/O bound).
   - Analytics = write-heavy (Kafka + OLAP CPU bound).
3. **Baseline Resources:**
   - Redirect svc: 1 pod = 2 vCPU/2 GB ≈ 10 k RPS.
   - Redis shard = 16 GB ≈ 1 M keys.
   - Kafka broker = 5 k msg/s per partition.
4. **Provision Headroom:** 30–40 % above peak.
5. **Auto-Scaling Rules:**
   - HPA triggers > 70 % CPU for 2 min.
   - Queue lag > 10 k messages → scale processors.
6. **Forecasting:** quarterly trend analysis; “what-if” traffic 2×/5×.

15.4 Optimization Examples

- **Redis Hit Rate:** raise TTL → reduce DB reads by 40 %.
- **Kafka Partition Tuning:** 1 partition per 1 k RPS.
- **OLAP Compression:** ZSTD reduces storage ~3×.
- **Cold Storage:** move old aggregates to S3 Glacier (1/10 cost).

15.5 Headroom & Simulation

Metric	Target	Behavior
CPU Utilization	60–70 % avg	ensures burst capacity
Memory Usage	< 80 %	avoid swap thrash
Queue Lag	< 5 s	maintain analytics freshness
Disk IOPS	< 70 %	steady writes without latency

Chaos-style spike tests: simulate 2× traffic; verify auto-scale response < 60 s.

---

16. Constraints & Assumptions

16.1 Explicit Constraints

Type	Constraint
Regulatory	Must comply with GDPR and CCPA; data retention ≤ 12 months for raw PII
Tech Stack	Python + Flask/FastAPI (backend); Redis, Kafka, ClickHouse; Kubernetes (GKE/EKS)
Time / Team	6 engineers, 4 months MVP timeline
Budget	≤ $5 k monthly OPEX (excluding CDN)
Availability SLO	99.99 % redirect path; 99.9 % control plane
Data Consistency	Strong for link mapping; eventual for analytics
Regions	Initially 2 (US-EAST, INDIA); expand to EU in phase 2
Dependencies	Managed services only (no self-hosted DBs in MVP)

16.2 Assumptions

Area	Assumption
Traffic Growth	10 % month-over-month; peak burst = 5× avg
Org Size	≤ 100 users per org in MVP
Link Lifetime	avg 6 months; 10 % links expire monthly
Analytics Freshness	1 min lag acceptable
Error Tolerance	≤ 0.1 % redirect failures tolerated
Cloud Limits	Redis ≤ 256 GB RAM per cluster
Latency Budget	Global CDN edge adds ≤ 10 ms
PII Storage	IP anonymized; no cookies beyond session scope

17 Future Enhancements

The current architecture is designed for extensibility: clean APIs, event-driven data pipelines, and infrastructure hooks that enable future capabilities without major redesign.
Below are **planned and potential extensions** beyond the MVP phase.

17.1 Advanced Analytics & Recommendation Engine

Goal: Provide actionable insights instead of raw stats.

- **Planned features:** trend prediction, best posting times by region, anomaly detection.

- **Architecture hooks:**
  - Enriched event schema already stores country, device, referrer.
  - OLAP warehouse partitioned by date → ready for time-series ML.
  - Feature-store bucket in object storage reserved for model inputs.
- **Tech options:** Prophet, BigQuery ML, or PyTorch forecasting jobs.

17.2 ML-Based Anomaly Detection

Purpose: Detect abnormal click bursts, fraud, or bot behavior.

- Streaming jobs can apply **statistical thresholds** or **isolation forests** per link ID.
- Outliers publish to an **"alerts" topic**, consumed by Admin Service for flagging/disabling links.
- Architectural readiness: Kafka topics and DLQ (DL Queue) already exist.

17.3 Expanded Multi-Region Deployment

Goal: Serve users with < 30 ms latency worldwide.

- Future regions: EU-WEST, APAC-SOUTHEAST, ME-CENTRAL.
- **Enhancements:**
  - Global database replication (Active-Active).
  - DNS-based Geo-Routing.
  - Cross-region stream mirroring for Kafka/OLAP.
- Hooks: anycast DNS and region labels in config map already implemented.

17.4 Better Multi-Tenancy Isolation

Current: logical tenant IDs + row-level filters.
Future: physical isolation per large enterprise (org-specific schemas or namespaces).

- Support "bring-your-own-domain" security keys.
- Fine-grained tenant-specific rate limits and data encryption keys.

17.5 Self-Service Admin and Configuration

Enable organizations to manage their own feature flags, quotas, and analytics retention.

- Add a "Tenant Control Panel" micro-frontend connected to Admin API.
- Uses RBAC hooks already available in Auth Service.

17.6 Automation & Auto-Remediation

- **Auto-tuning:** scale Kafka partitions and Redis nodes based on lag metrics.
- **Auto-remediation:** detect stalled pods or high latency → trigger restarts or cache flushes.
- Integrate with Kubernetes Operators and Prometheus Alertmanager.

17.7 Marketplace Integrations

Future SaaS plugins: Slack, HubSpot, Google Analytics, Zapier.

- Expose webhooks (POST /v1/webhooks/events).
- Provide SDKs in Python, Node, Go for third-party usage.
- Architecture ready via event bus and API Gateway extensibility.

17.8 Developer Platform

Expose open API spec (Swagger / GraphQL schema) for custom analytics dashboards.
Long-term goal: “Shortener as a Service” model.

18 Diagrams

18.1 Diagram Inventory

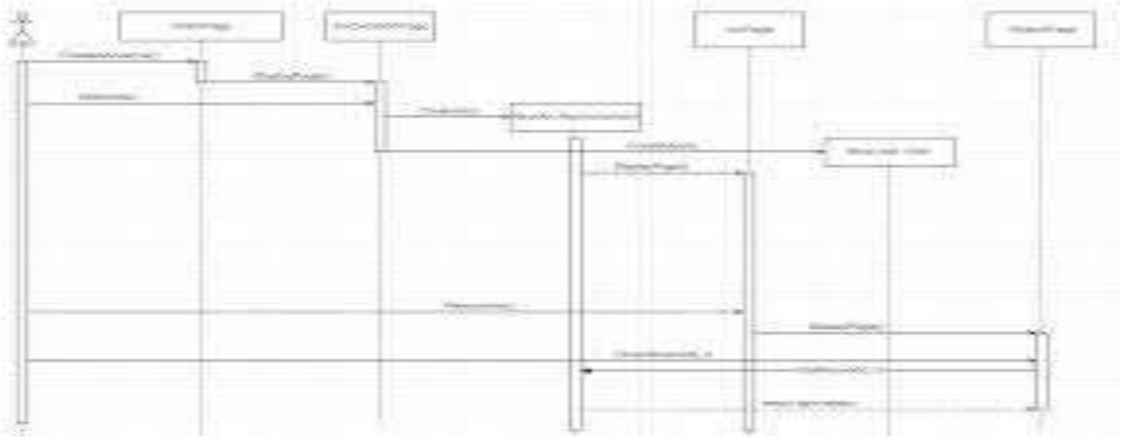
Diagram Type	Purpose	Key Insight
C4 Context	Shows system and external actors	Boundaries of responsibility
C4 Container	Depicts microservices + datastores	Request flow and dependencies
Component Diagram (Auth Service)	Zoom into auth logic and data flows	JWT lifecycle, scope validation
Component Diagram (Analytics Pipeline)	Stream and batch processors	Event flow + checkpointing
Sequence Diagram (User Redirect)	Step-by-step 301 redirect	Cache hit vs DB miss
Sequence Diagram (Analytics Ingest)	Click → Kafka → OLAP	Async event processing
Deployment Diagram	Regions, VPCs, subnets, services	Network topology & HA

Diagram Type	**Purpose**	**Key Insight**
Data Model (ERD)	Entities and relationships	Logical schema overview
API Flow Diagram	Frontend → Gateway → Services	Auth and rate limit middleware
Ops View Diagram	Monitoring & alert pipelines	Observability architecture

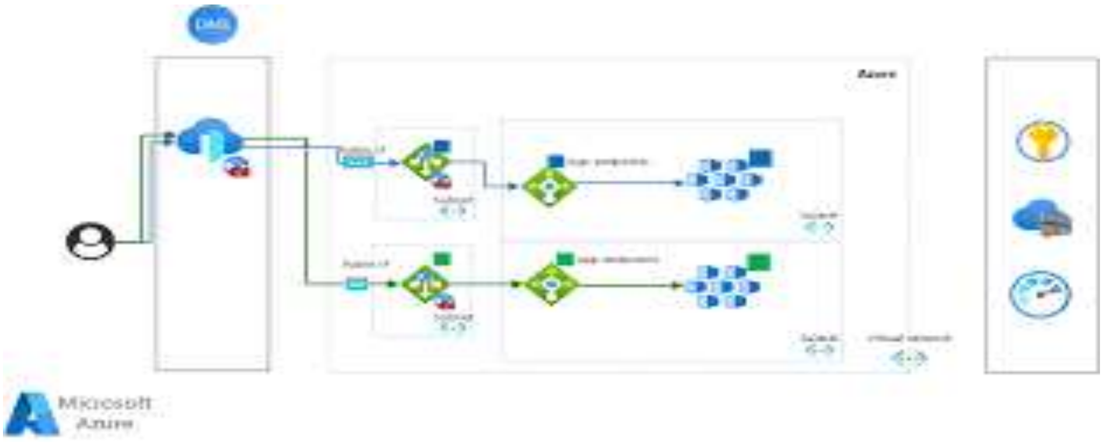
Each diagram includes captions explaining data flows, resilience mechanisms, and trade-offs.
For example:

• Sequence Diagram

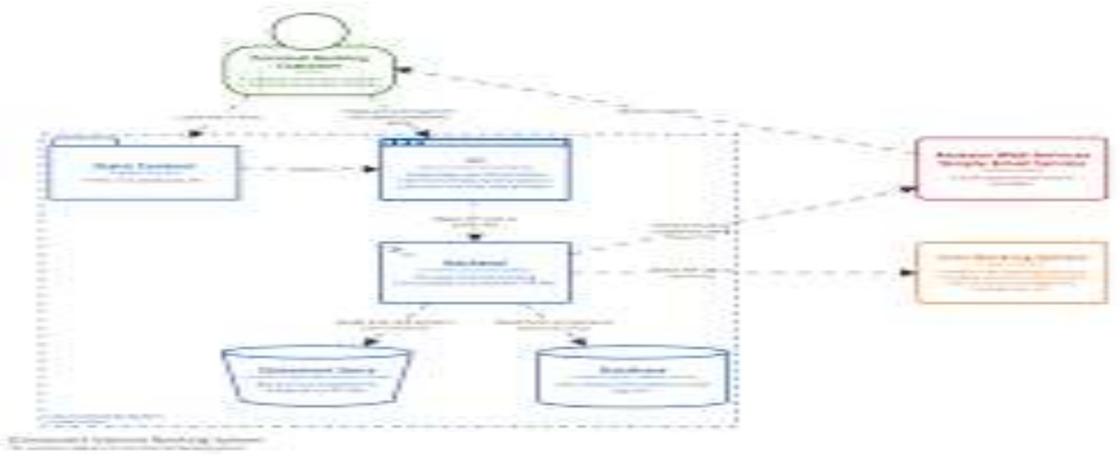
Sequence Diagram — Redirect: emphasizes async analytics emission to keep p95 < 50 ms.



- **Deployment Diagram:** highlights multi-AZ design for 99.99 % availability.Deployment Diagram:



- **C4 Container:** shows clear boundaries between redirect and analytics planes.C4 Container:



19 References

1. *Google Cloud Architecture Framework – Scalability & Resilience.*
2. AWS Well-Architected Framework, 2024.
3. Martin Kleppmann, *Designing Data-Intensive Applications,* O'Reilly.
4. ByteByteGo System Design Playbook.
5. Uber Engineering Blog – Kafka for Analytics Pipelines.
6. Cloudflare Tech Blog – Edge Caching Patterns.
7. ClickHouse Docs – AggregatingMergeTree Engine.
8. OWASP Top 10 Security Risks, 2025 Edition.
9. OpenTelemetry Instrumentation Guide.
10. Google SRE Book – Error Budgets & SLOs.
11. Redis Labs – High-Availability Best Practices.
12. Netflix Chaos Engineering Principles.
13. Flink Streaming Architecture Whitepaper.
14. Terraform Enterprise Documentation.
15. ISO 27001 and GDPR Guidelines for Cloud Services.

20 Conclusion

The *URL Shortener with Click Analytics (Multi-Region)* design achieves a balance between **scalability, performance, and resilience**, aligning with the project's core objectives:

Goal	**Achieved By**
Scalability	Horizontally scalable microservices, hash-sharded datastores, Kafka partitioning

Goal	Achieved By
Low Latency	CDN edge caching + Redis cache-aside; < 50 ms p95 redirects
High Availability	Multi-region Kubernetes deployments, auto-failover, replication
Maintainability	Modular services, IaC (Terraform), observability dashboards
Security	Defense-in-depth, RBAC/ABAC, TLS 1.3, KMS-based encryption
Cost Control	Autoscaling, data retention policies, optimized OLAP storage

Trade-offs

- Adopted **eventual consistency** for analytics to meet latency and cost targets.
- Chose **managed cloud services** to reduce ops burden at slightly higher cost.
- Emphasized **horizontal scale out** instead of vertical hardware upgrades.
- Kept redirect plane stateless for resilience; analytics plane stateful but async.

Next Steps

1. **Proof of Concept (PoC):**
   Deploy minimal stack (redirect + link svc + Redis + Kafka) to validate p95 latency and event flow.
2. **Phase 1 Release:**
   Add analytics ingestion, OLAP integration, and dashboards.
3. **Validation:**
   Measure SLO compliance (latency ≤ 50 ms, availability ≥ 99.99 %, freshness ≤ 60 s).
4. **Phase 2:**
   Introduce ML features and multi-region replication.
5. **Post-Launch:**
   Monitor cost, optimize infra usage, iterate on features based on telemetry.

Closing Statement

This architecture lays a robust foundation for a **global, low-latency, privacy-compliant URL shortening and analytics platform**.
It balances **simplicity, performance, and extensibility**, enabling future growth toward a multi-tenant, intelligent analytics ecosystem.