API Gateway vs CDN, Replication Mechanisms, and GraphQL Deep Dive

Part 1: API Gateway vs CDN
API Gateway: Acts as the front door for APIs, managing routing, authentication, throttling, and monitoring.
CDN (Content Delivery Network): Distributes static content globally for low-latency delivery by caching content near users.
They complement each other — CDN serves static assets, while API Gateway manages dynamic API traffic.

Part 2: Multi-Region Architecture
Each region (A, B) includes API Gateway, Microservices, Cache, and Database. A Global Load Balancer and CDN Edge provide resilience and fast delivery.
- **Cache Replication:** Asynchronous for eventual consistency.
- **Database Replication:** Synchronous or asynchronous with leader election for strong consistency.

Part 3: Failover Patterns
- **Active-Active:** Both regions serve live traffic; data replicates bidirectionally.
- **Active-Passive:** One region is standby; replication is unidirectional for failover readiness.

Part 4: Replication Mechanisms
- Database replication can be multi-master or single-master with WAL/binlog streaming.
- Cache replication can use CRDTs, async sync, or read-through on failover.

Consistency achieved via quorum/leader election for databases and eventual consistency for caches.

Part 5: GraphQL Deep Dive

What is GraphQL?
GraphQL is a query language and runtime for APIs that lets clients request exactly the data they need. Developed by Facebook, it provides flexibility, efficiency, and strong typing for modern web and mobile applications.

Architecture:
GraphQL has three main components:
1. **Schema** – Defines available types and relationships between data objects.
2. **Query** – Client specifies what data it wants.
3. **Resolver** – Server-side logic that fetches data from backend systems like databases or microservices.

Execution Flow:
Client → GraphQL Server → Schema Validation → Resolvers → Data Sources → Aggregated Response.

Advantages:
- Eliminates over-fetching and under-fetching of data.
- Reduces number of network calls with a single endpoint.
- Strongly typed schema allows predictable responses.
- Versionless evolution of APIs.
- Ideal for mobile and data-intensive frontend apps.

Data Types:
GraphQL defines types such as Query, Mutation, Subscription, Scalar, Enum, and Object types.
- **Query**: Read data.
- **Mutation**: Write or modify data.
- **Subscription**: Real-time updates over WebSocket.

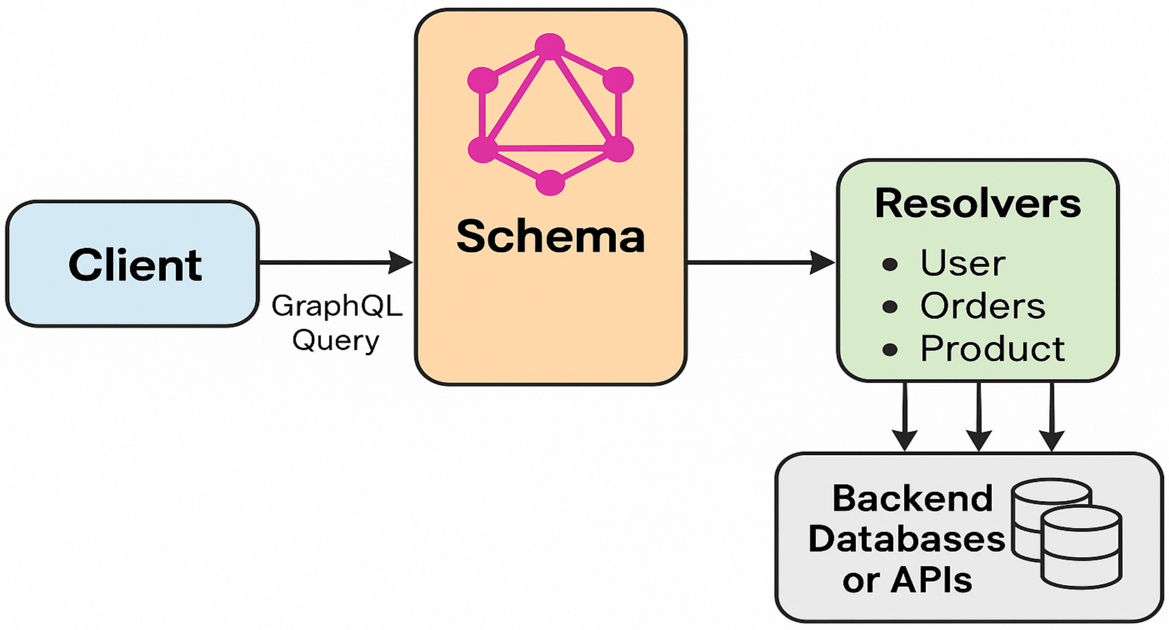
Performance Considerations:
Challenges include deep nested queries and the N+1 problem. Mitigation techniques include:
- Depth limiting and cost analysis.
- Caching or batching using DataLoader.
- Schema federation and query persistency.

Real-World Use Cases:
- **GitHub API v4**: Exposes repositories, commits, and issues via GraphQL.
- **Shopify**: Storefront and Admin APIs for flexible e-commerce data.
- **Netflix**: Federation to unify microservice data.
- **Amazon AppSync**: Managed GraphQL service for serverless apps.

REST vs GraphQL Summary:

Feature	REST	GraphQL
Endpoints	Multiple	Single
Data Shape	Fixed	Flexible (Client-defined)
Over-fetching	Common	None
Under-fetching	Common	None
Versioning	Required	Versionless
Best For	Simple APIs	Complex data aggregation

When Not to Use GraphQL:
Avoid GraphQL when your system handles flat, static data, heavy CDN caching, or minimal inter-service data joins.


Client
GraphQL Query
Schema
Resolvers
• User
• Orders
• Product
Backend Databases or APIs