Contents

Network Infrastructure & Traffic Management

This document covers network infrastructure components and traffic management strategies for system design.

Components

Load Balancing

- **Layer 4 (Transport):** Routes based on IP and port
- **Layer 7 (Application):** Routes based on content (HTTP headers, URLs)
- **Algorithms:** Round-robin, least connections, weighted, IP hash
- **Types:** Hardware vs software, active-passive vs active-active

Proxies

- **Forward Proxy**
  - Client-side proxy that hides the identity of the client from the server (corporate firewalls, caching). The server doesn't know which specific client made the request.
- **Reverse Proxy**
  - Server-side proxy that conceals the identity of the server from the client (load balancing, SSL termination). The client doesn't know which specific server handled the request.
- **Examples**
  - Nginx, HAProxy, CloudFlare

DNS & Content Delivery

- **DNS Basics**
  - **Root Servers:** Top level of DNS hierarchy, know addresses of TLD servers for all domains (.com, .org, etc.)
  - **TLD (Top-Level Domain) Servers:** Manage specific domains like .com, .org, know authoritative servers for domains within their TLD
  - **Authoritative DNS Servers:** Hold actual DNS records for a domain

    - **Recursive DNS Resolvers:** Query authoritative servers on behalf of clients
    - **DNS Flow:** Client → Recursive Resolver → Root → TLD → Authoritative → Response
- **Anycast Routing**
    - **Multiple servers share same IP address**, BGP routes users to nearest/best server
    - **Benefits:** Reduced latency, improved availability, DDoS mitigation
- **Content Delivery Networks (CDNs)**
    - **Geographically distributed edge servers** cache content closer to users
    - **Process:** User request → DNS resolves to nearest edge → Cache hit/miss → Serve content
    - **Benefits:** Reduced latency, bandwidth savings, origin protection

API Gateway

- **Purpose:** Centralized entry point for microservices, providing cross-cutting concerns and API management
- **Features:** Authentication/authorization, rate limiting, request/response transformation, API versioning, analytics, routing
- **Benefits:** Simplified client integration, centralized security, operational control
- **Trade-offs:** Additional latency hop, potential single point of failure, increased complexity
- **Examples:** Amazon API Gateway, Kong, Zuul, Istio Gateway

Service Exposure Patterns

Direct Service Exposure

- **Approach:** Clients communicate directly with individual services
- **Benefits:** Minimal latency, simple networking, no additional infrastructure
- **Challenges:** Client complexity (service discovery, auth, routing), security concerns

Gateway/Proxy Layer

- **Approach:** Centralized entry point handles cross-cutting concerns
- **Benefits:** Service abstraction, centralized auth/logging/rate limiting, operational control
- **Challenges:** Additional latency, potential bottleneck, infrastructure complexity

Related Trade-offs

Load Balancer vs. API Gateway

- **Summary:** Load balancers distribute traffic across multiple servers to ensure availability and performance. API gateways provide a centralized entry point with additional features like authentication, rate limiting, and request transformation. Both can distribute traffic but serve different architectural purposes.
- **Trade-off:** Simple traffic distribution vs. comprehensive API management and control.
- **Component Comparison:**
    - **Load Balancer:** Focuses on traffic distribution, health checks, and high availability. Simple, fast, and reliable for basic routing needs
    - **API Gateway:** Provides traffic routing plus authentication, authorization, rate limiting, request/response transformation, analytics, and API versioning. More comprehensive but adds complexity and latency

  - **Hybrid Approach:** Use API Gateway for external traffic (client-facing) and load balancers for internal service-to-service communication
- **Questions to Ask:**
  - Do you need API management features beyond basic traffic distribution?
  - Are you exposing APIs to external clients or just internal services?
  - What authentication and authorization requirements exist?
  - Do you need rate limiting, request transformation, or API analytics?
  - How important is minimizing latency vs. having centralized control?
  - Are you building a microservices architecture that needs service discovery?

Direct Service Exposure vs. Gateway/Proxy Layer

- **Summary:** Direct service exposure allows clients to communicate directly with individual services, maximizing performance and simplicity. Gateway/proxy layers provide centralized control, security, and abstraction but add network hops and complexity.
- **Trade-off:** Performance and simplicity vs. centralized control and security.
- **Architecture Comparison:**
  - **Direct Exposure:** Clients connect directly to services, minimal latency, simple networking, but requires clients to handle service discovery, authentication, and routing logic
  - **Gateway/Proxy Layer:** Centralized entry point handles cross-cutting concerns (auth, logging, rate limiting), service abstraction, and routing, but adds latency and becomes a potential single point of failure
  - **Hybrid Approach:** Direct access for internal/trusted services, gateway for external clients or services requiring additional controls
- **Questions to Ask:**
  - What are the security requirements and trust boundaries?
  - Do clients need to know about individual service locations and protocols?
  - How many different types of clients will access the services?
  - What cross-cutting concerns (authentication, logging, rate limiting) need to be applied?
  - How critical is minimizing latency vs. having operational control?
  - Do services need to be independently deployable and discoverable?
  - What's the tolerance for client complexity vs. infrastructure complexity?

API Gateway vs. Reverse Proxy

- **Summary:** While both API Gateways and Reverse Proxies manage traffic, they cater to different needs. An API Gateway is more about managing, routing, and orchestrating API calls in a microservices architecture, whereas a Reverse Proxy is about general server efficiency, security, and network traffic management.
- **Trade-off:** Application-specific API management vs. general-purpose traffic and security management.
- **Component Comparison:**
  - **API Gateway:** Focuses on API lifecycle management, request/response transformation, rate limiting, authentication/authorization, API versioning, and microservices orchestration. Application-aware with rich API management features
  - **Reverse Proxy:** Handles general traffic routing, load balancing, SSL termination, caching, and security filtering. Network-level focus with high performance and broad

    protocol support
  - **Hybrid Approach:** Use both together - Reverse Proxy for general traffic management and security, API Gateway for application-specific API orchestration and management
- **Questions to Ask:**
  - Are you primarily managing APIs or general web traffic?
  - Do you need application-aware features like API versioning and transformation?
  - What's more important: high-performance traffic handling or rich API management?
  - Are you building a microservices architecture that needs API orchestration?
  - Do you need protocol translation or just traffic forwarding?
  - What security requirements exist at the network vs. application layer?
  - Can you benefit from using both components in a layered approach?