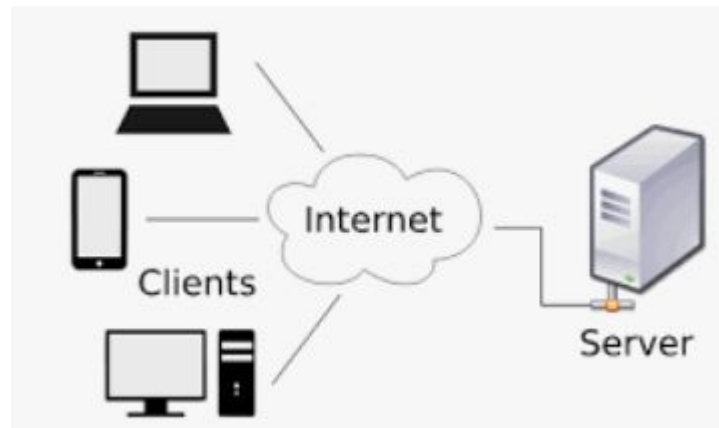

Mastering System Design

Section 2: System Design Fundamentals

— Networking & Communication —

Why Networking Matters in System Design?

- Every system relies on data exchange between components
- Networking enables scalability, reliability, and performance
- Key areas where networking plays a crucial role:
 - Communication: Ensuring smooth data transfer between clients, servers, and databases
 - Load Balancing: Distributing traffic efficiently to prevent overload on a single server
 - Security: Protecting data from unauthorized access and cyber threats
 - Efficiency: Optimizing network performance to reduce latency and improve user experience



How Networking Impacts Large-Scale Systems

- Helps handle millions of users concurrently
- Enables fast and efficient data exchange
- Reduces latency & improves system resilience
- Essential for cloud computing & distributed systems

Agenda: Key Networking Concepts in System Design

- Understanding IP Addresses
- How DNS Works
- Client-Server Model Explained
- Forward Proxy vs. Reverse Proxy
- Introduction to Load Balancing
- What is an API Gateway?
- Content Delivery Networks (CDN) in System Design

Wrap-Up & What's Next

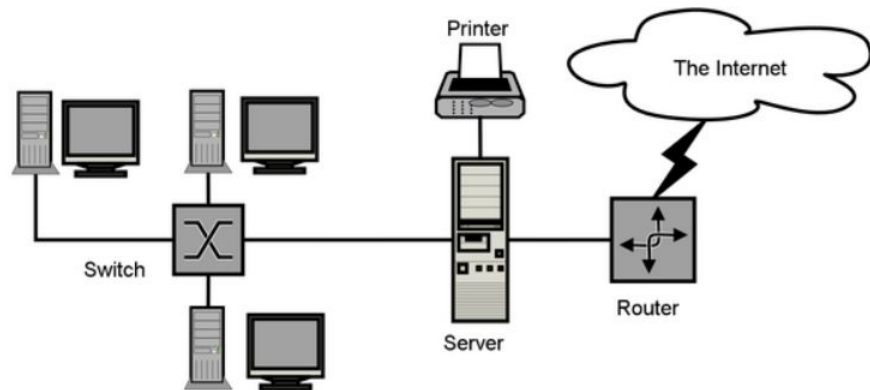
- Networking is the foundation of system design
- Next, we'll deep dive into networking components
- Coming up: IP Addressing & DNS

Understanding IP Addresses in System Design

— Networking & Communication —

Introduction to IP Addresses

- IP (Internet Protocol) addresses are unique numerical labels assigned to devices on a network.
- They enable communication between different machines, servers, and services across the internet.
- Two primary versions:
 - IPv4
 - IPv6
- Two primary categories:
 - Public
 - Private



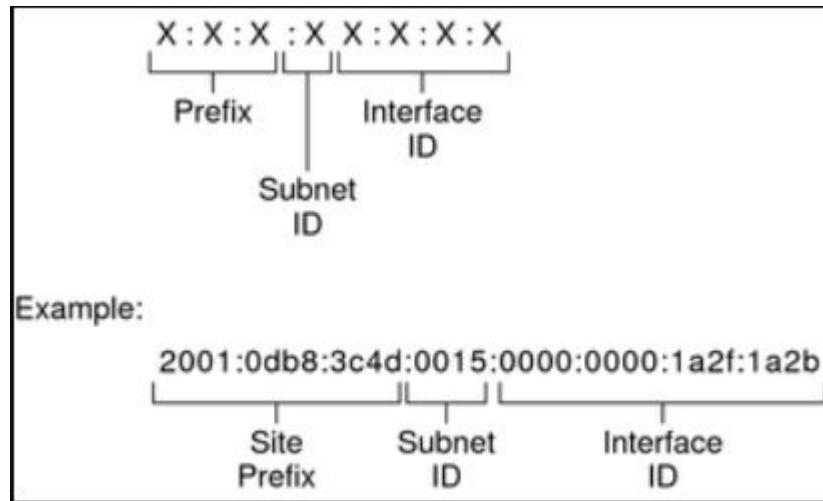
What is IPv4?

- IPv4 (Internet Protocol Version 4) is the most widely used addressing system.
- 32-bit address format (e.g., 192.168.1.1)
- Total addresses available: ~4.3 billion
- Uses: Traditional networking, web servers, and most current internet devices.
- Challenges: Limited IPs, fragmentation, and security concerns.

192.168.1.1

What is IPv6?

- IPv6 (Internet Protocol Version 6) is the next-generation IP standard.
- 128-bit address format -
2001:0db8:85a3:0000:0000:8a2e:0370:7334
- Total addresses available: 340 undecillion (virtually unlimited).
- Designed for: IoT, mobile networks, and future scalability.
- Key benefits: Larger address space, better security, and improved routing efficiency.

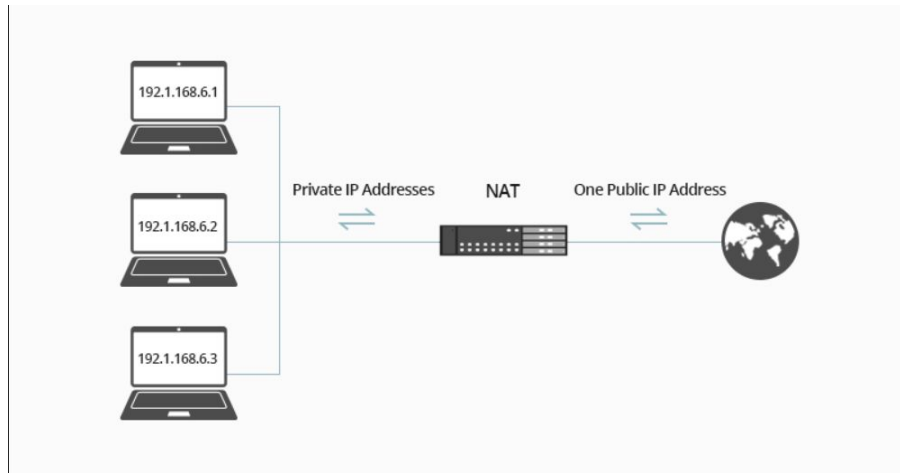


IPv4 vs. IPv6 – Key Differences

Feature	IPv4	IPv6
Address Size	32-bit	128-bit
Address Format	Decimal (e.g., 192.168.1.1)	Hexadecimal (e.g., 2001:db8::ff00:42:8329)
Address Space	~4.3 billion addresses	Virtually unlimited
Security	Relies on additional protocols	Built-in security (IPSec)
Performance	Limited due to NAT & fragmentation	More efficient routing & handling
Adoption	Still widely used	Slowly being adopted

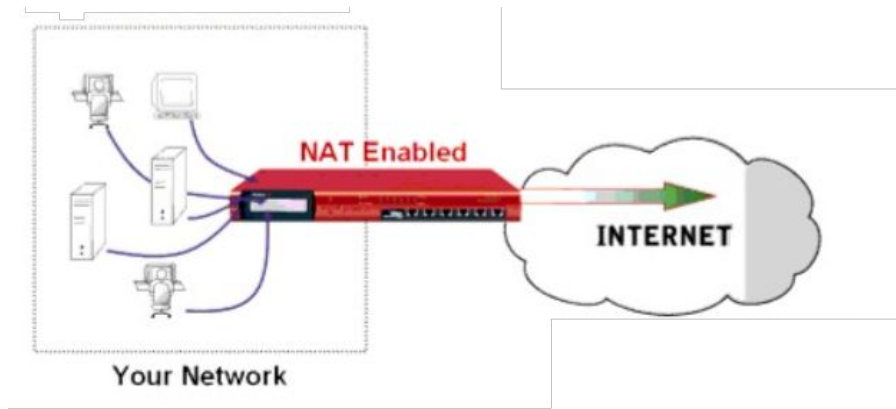
Private vs. Public IPs

- Public IPs:
 - Assigned by ISPs (Internet Service Providers).
 - Used to communicate over the internet.
 - Unique worldwide.
 - Example: 192.203.23.45.
- Private IPs:
 - Used within local networks (LANs, enterprises, homes).
 - Cannot be accessed directly from the internet.
 - examples:
 - 10.0.0.0 - 10.255.255.255
 - 172.16.0.0 - 172.31.255.255
 - 192.168.0.0 - 192.168.255.255



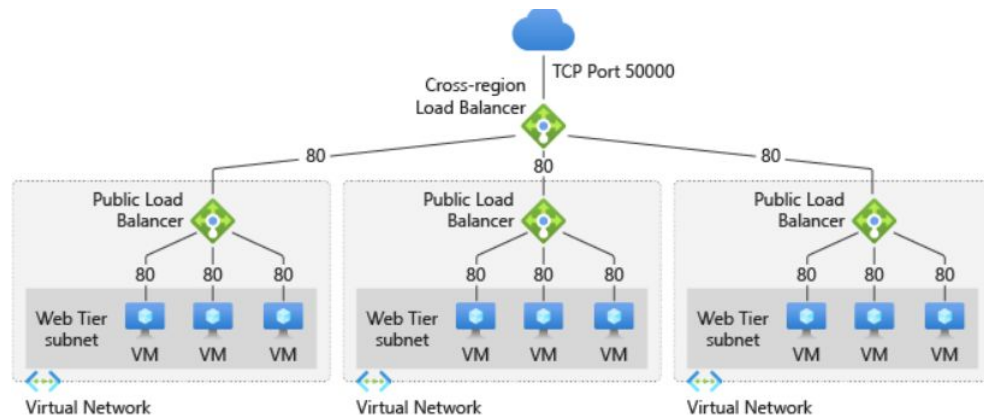
Why Do We Need Private IPs?

- Conserves public IP addresses (IPv4 limitation).
- Enhances security (private IPs are not routable over the internet).
- Enables Network Address Translation (NAT) to allow multiple devices to share a single public IP.
- Common in corporate networks, data centers, and cloud environments.



The Role of IPs in System Design

- Scalability: Helps in designing distributed, multi-region, and cloud-based architectures.
- Security: Enables firewall rules, VPNs, and private networking.
- Load Balancing: Uses IP-based traffic distribution (e.g., Round-robin DNS, Anycast IPs).
- Cloud Networking: Public, private, and hybrid cloud IP management (AWS, GCP, Azure).
- Microservices & Containers: Use internal private IPs for communication.



Common System Design Interview Questions on IPs

- How do IPv4 and IPv6 addresses differ?
- Why do we need private IPs in system design?
- How does NAT help in addressing the IPv4 shortage?
- Explain how a load balancer distributes traffic using IPs.
- How does DNS resolve IP addresses in a large-scale system?

Summary & What's Next

- IPv4 and IPv6 are the two primary internet protocols.
- Private vs. Public IPs help manage internal and external networking.
- Networking & IPs are essential for system scalability, security, and performance.
- Coming up next: How DNS works and its role in system design.

How DNS Works

— Networking & Communication —

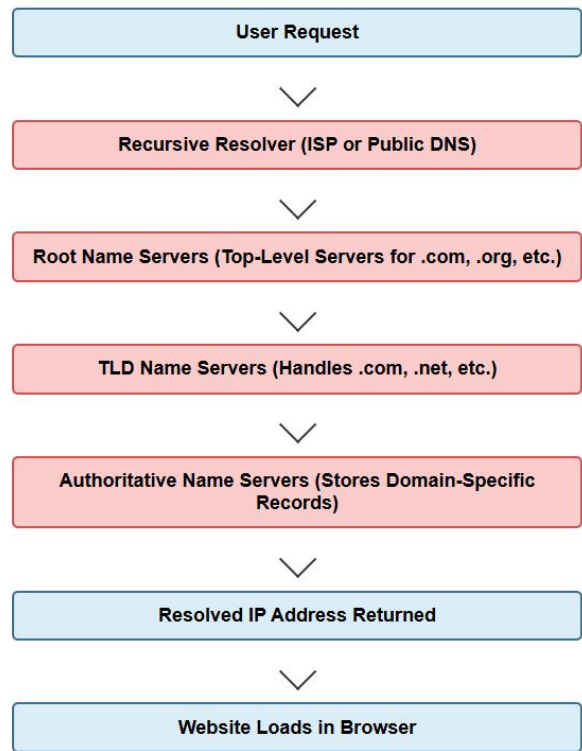
Introduction to DNS

- What is DNS?
 - The "phonebook of the internet."
 - Translates human-readable domain names (e.g., google.com) into IP addresses.
- Why is DNS important?
 - Enables accessibility of websites.
 - Supports distributed, scalable internet architecture.



Types of DNS Servers

- Root Name Servers: Top-level servers handling .com, .org, etc.
- TLD Name Servers: Specific to domains like .com, .net.
- Authoritative Name Servers: Store domain-specific records.
- Recursive Resolvers: Handle queries on behalf of users, contacting multiple DNS servers if necessary.



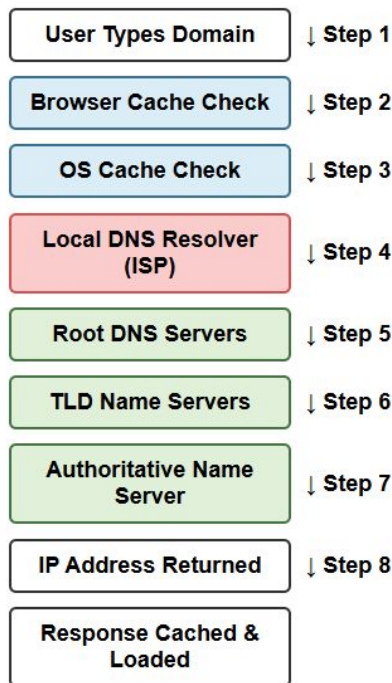
DNS Caching and Performance Optimization

- Why caching matters?
 - Reduces latency.
 - Reduces load on DNS servers.
- Where caching occurs?
 - Browser cache (stores previous lookups).
 - OS cache (/etc/hosts, Windows DNS cache).
 - Recursive Resolver cache (ISP level).
- Time-To-Live (TTL): Determines how long a cached record is valid.

The Domain Name Resolution Process

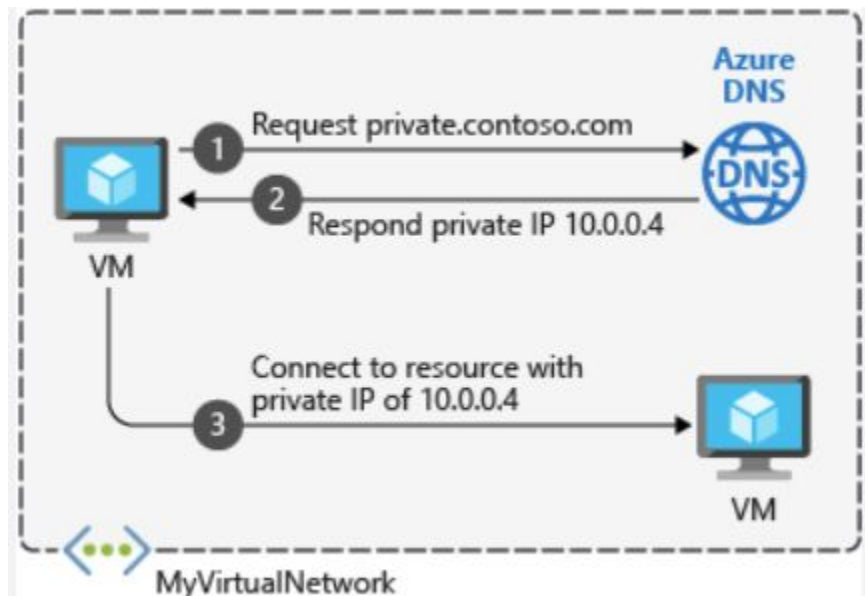
- Step-by-step breakdown of how DNS resolves a domain name:
 - a. User types a domain name in the browser.
 - b. Browser cache check (if the IP is already known).
 - c. Operating System (OS) cache check.
 - d. Query to the Local DNS Resolver (ISP or configured DNS server).
 - e. If not found, query goes to Root DNS servers.
 - f. TLD (Top-Level Domain) servers provide the Authoritative DNS server details.
 - g. Authoritative DNS server returns the IP address.
 - h. Response is cached and returned to the user.

Step-by-Step DNS Resolution



Importance of DNS in Large-Scale Systems

- Ensuring High Availability
 - Load balancing using DNS.
 - Anycast DNS for faster global resolution.
- DNS Failover Strategies
 - Primary and secondary DNS servers.
- Content Delivery Networks (CDNs) & DNS
 - How DNS directs users to the nearest CDN node for fast loading.
- DNS Security Risks
 - DNS poisoning, cache poisoning, and DDoS attacks.

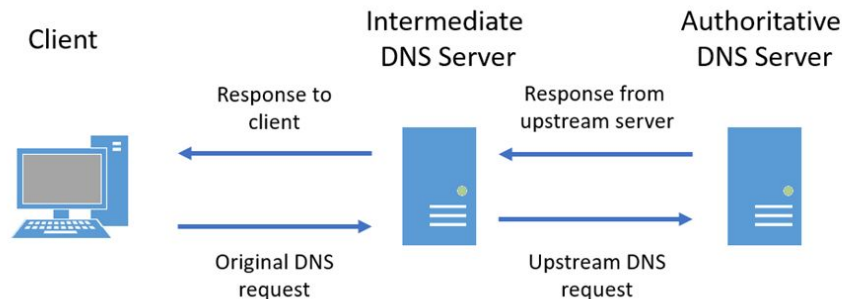


Interview Questions Related to DNS

1. Explain the DNS resolution process step by step.
2. What is the difference between authoritative and recursive DNS servers?
3. How does DNS caching improve performance? Where does it occur?
4. What is TTL in DNS, and why is it important?
5. How does DNS-based load balancing work in large-scale systems?
6. What are common DNS-related security threats, and how can they be mitigated?

Summary & Takeaways

- DNS is critical for internet functionality.
- Caching improves efficiency and reduces latency.
- DNS load balancing & failover are essential for large-scale systems.
- Security threats exist, but best practices help mitigate them.
- What's Next
 - Client-Server Model Explained

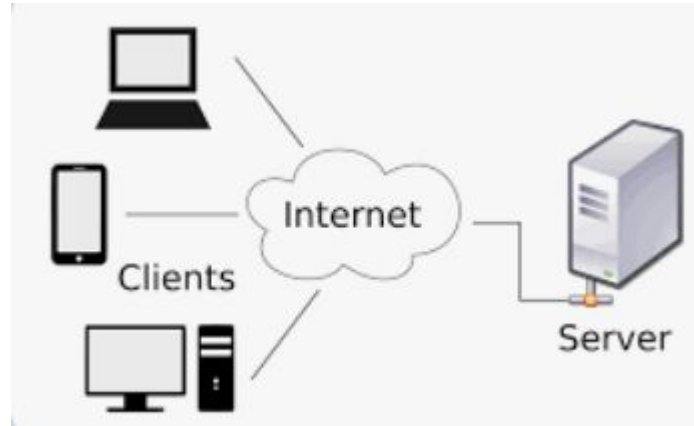


Client-Server Model Explained

— Networking & Communication —

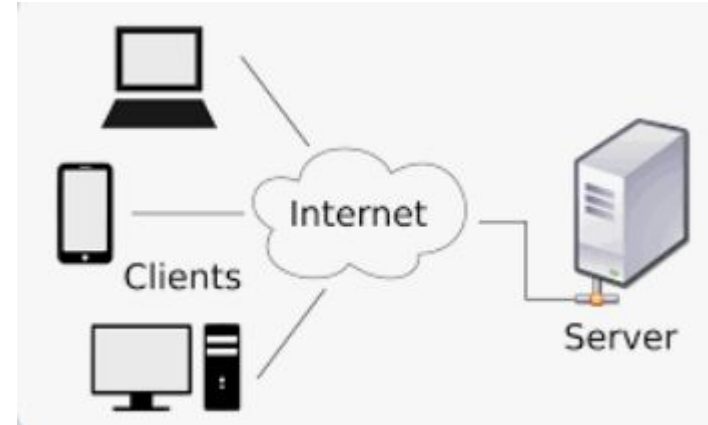
What is the Client-Server Model?

- A computing model where clients request services and servers provide them.
- Foundation of modern web, database, and application architectures.
- Why is it important?
 - a. Enables efficient resource management.
 - b. Used in web browsing, email, APIs, databases, and cloud services.
- Real-world examples:
 - a. Browsing a website (Browser = Client, Web Server = Server).
 - b. Sending an email (Email App = Client, Mail Server = Server).



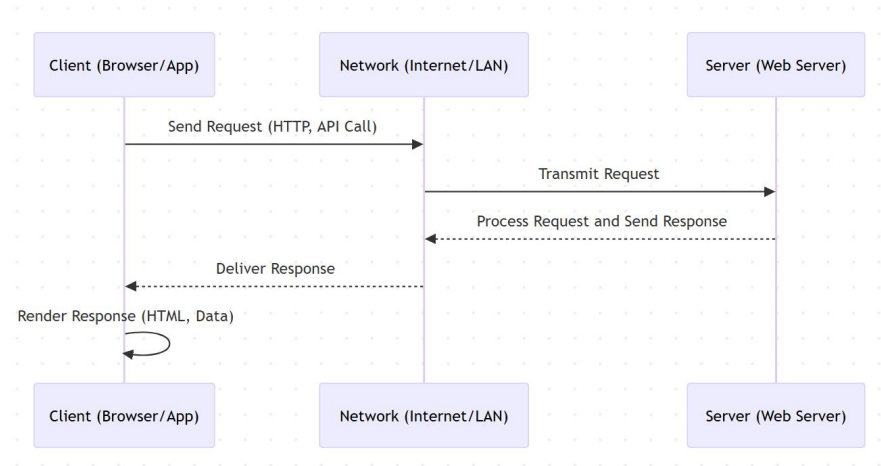
Key Components of the Client-Server Model

- **Client:**
 - The user-facing application that sends requests.
 - Examples: Web browser, mobile app, API consumer.
- **Server:**
 - A system that processes requests and returns responses.
 - Examples: Web server, database server, mail server.
- **Network:**
 - The medium that facilitates communication between clients and servers.
 - Examples: Internet, LAN, Wi-Fi, 5G.



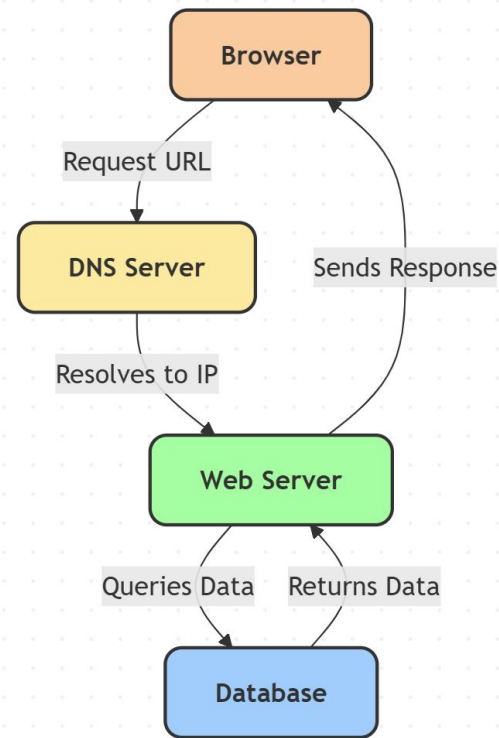
How Clients and Servers Communicate

- Basic Steps:
 - Client sends a request (HTTP, SQL query, etc.).
 - Request is transmitted over the network.
 - Server receives and processes the request.
 - Server sends a response.
 - Client receives and processes the response.
- Types of Communication:
 - Request/Response Model (e.g., HTTP, REST APIs)
 - Persistent Connections (e.g., WebSockets, FTP sessions)



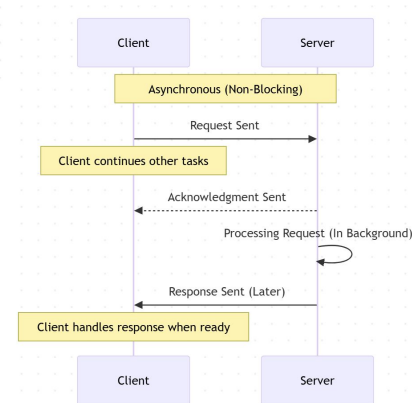
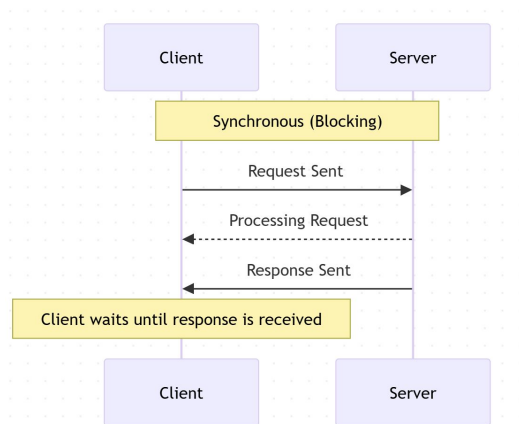
The Request-Response Cycle (HTTP Example)

1. Browser requests `https://example.com`.
2. DNS resolves domain to IP address.
3. Browser sends an HTTP GET request to the web server.
4. Web server processes request, fetches HTML, queries the database if needed.
5. Server sends an HTTP response with a status code (200 OK) and the webpage content.
6. Browser renders the webpage.



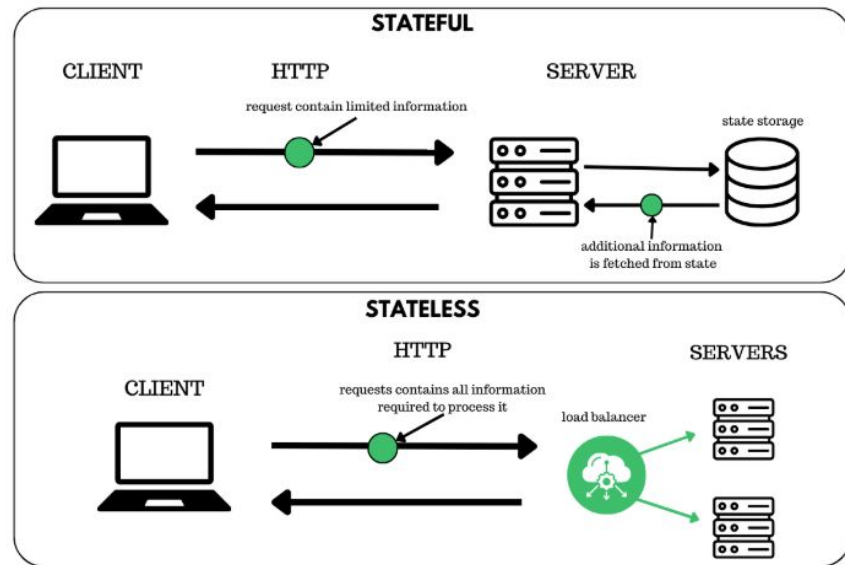
Synchronous vs Asynchronous Communication

- Synchronous Communication:
 - Client waits for a response before proceeding.
 - Example: A user submits a form and waits for a confirmation message.
 - Used in REST APIs, traditional web apps.
- Asynchronous Communication:
 - Client doesn't wait for a response and can perform other tasks.
 - Example: WebSockets for real-time chat, AJAX for background requests.
 - Used in messaging apps, live updates, IoT applications.



Stateless vs Stateful Servers

- Stateless Servers:
 - No memory of past interactions; each request is independent.
 - Example: REST APIs, HTTP servers.
 - Benefits: Scalability, easy caching, load balancing.
- Stateful Servers:
 - Maintain session information across requests.
 - Example: WebSockets, multiplayer games, banking applications.
 - Benefits: Personalization, seamless user experiences.



Real-World Examples

- Web Applications:
 - Browsing a website (Client = Browser, Server = Web Server).
- APIs & Microservices:
 - Mobile apps calling backend APIs (REST, GraphQL).
- Databases:
 - A client querying a database (SQL Client → Database Server).
- Messaging Systems:
 - WhatsApp, Slack using WebSockets for real-time communication.

Interview Questions on the Client-Server Model

- **Fundamental Questions:**
 - What is the client-server model, and how does it work?
 - How does a client communicate with a server?
 - What are some real-world examples of the client-server model?
 - What is the difference between a client and a server?
- **Request-Response Cycle Questions:**
 - Explain the HTTP request-response cycle with an example.
 - What are the key differences between synchronous and asynchronous communication?
 - How does a browser load a webpage? Walk me through the steps.
- **Architecture and Design Questions:**
 - What is the difference between stateless and stateful servers?
 - How does caching improve performance in a client-server model?
 - How do load balancers work in a client-server architecture?
- **Advanced Questions:**
 - What are some security challenges in the client-server model?
 - How does WebSockets differ from traditional request-response communication?
 - What are some limitations of the client-server model? How can they be addressed?
 - How would you design a scalable client-server system for a high-traffic application?

Wrap up and what's next....

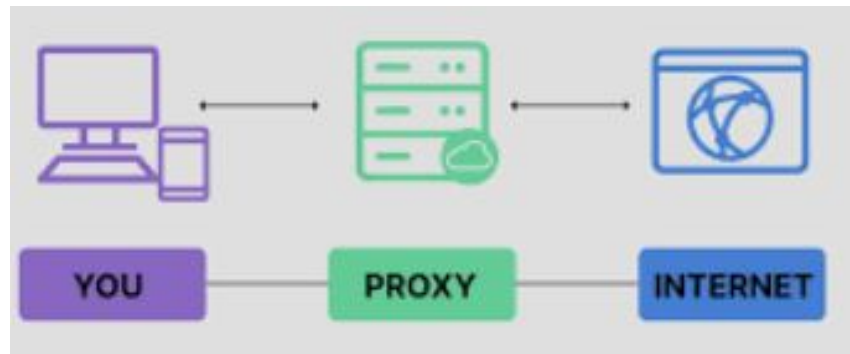
- Key Takeaways:
 - The client-server model underpins most modern computing.
 - Different communication patterns (sync vs async, stateless vs stateful) optimize performance for different use cases.
 - Understanding this model helps in system design and scalability planning.
- Next Steps:
 - Forward Proxy vs. Reverse Proxy

Forward Proxy vs. Reverse Proxy

— Networking & Communication —

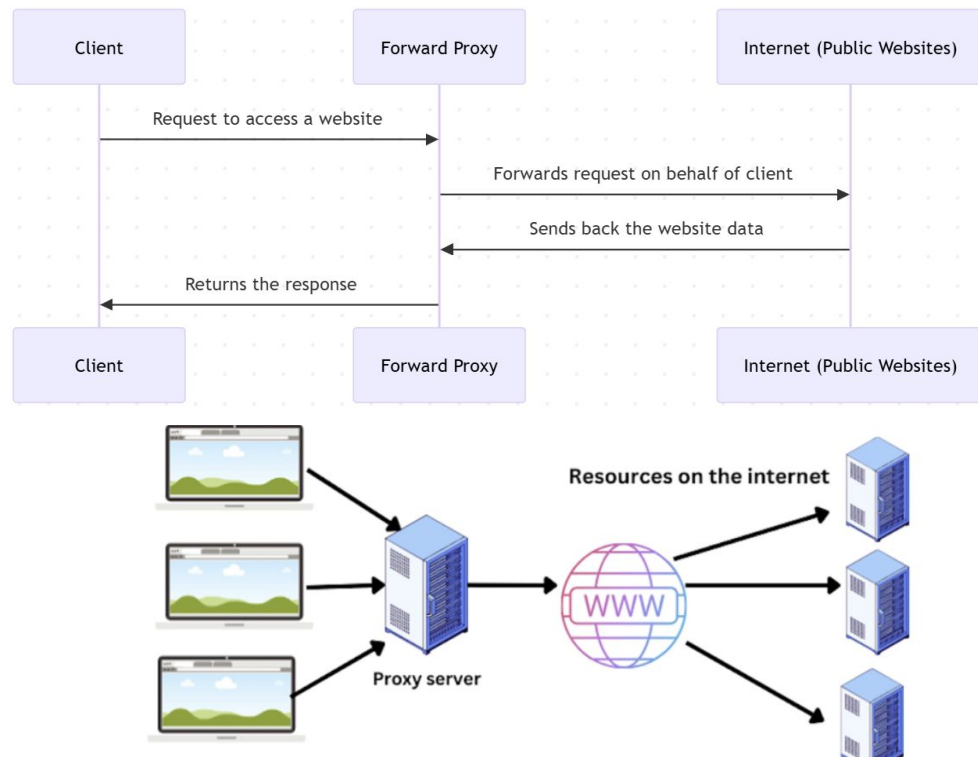
What is a Proxy?

- A proxy is an intermediary server between a client and another server.
- Proxies help with security, caching, traffic control, and anonymity.
- Two main types: Forward Proxy & Reverse Proxy.



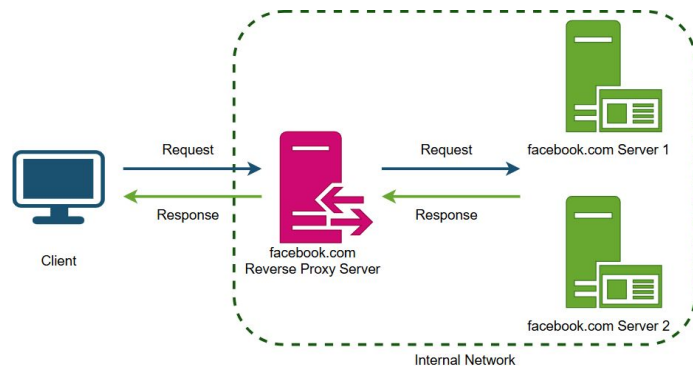
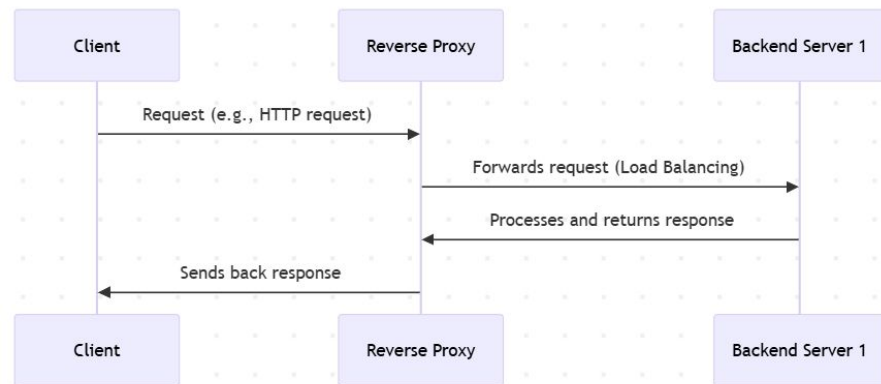
What is a Forward Proxy?

- Sits between a client and the internet (used by clients).
- Clients connect to the forward proxy instead of directly accessing websites.
- Can filter content, provide anonymity, and cache requests.
- Common use cases:
 - Bypassing geo-restrictions
 - Anonymous browsing (VPNs, TOR)
 - Caching web pages for faster access



What is a Reverse Proxy?

- Sits between users and backend servers (used by servers).
- Clients don't directly communicate with backend servers—requests go through the reverse proxy.
- Used for load balancing, caching, security, and SSL termination.
- Common use cases:
 - Load balancing across multiple servers
 - Caching content to improve speed
 - DDoS protection & security

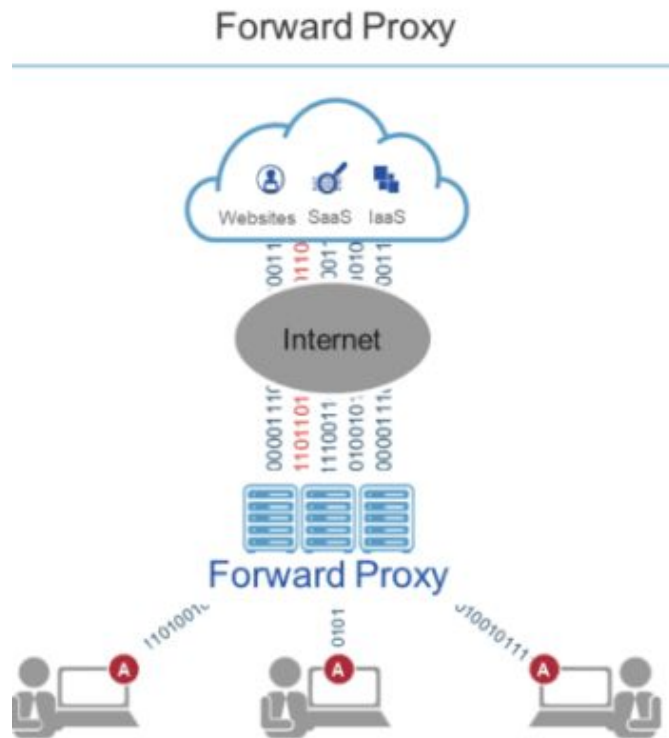


Key Differences - Forward vs. Reverse Proxy

Feature	Forward Proxy 🏆	Reverse Proxy 🚀
Position	Between client & internet	Between users & backend servers
Who Uses It?	Clients (users, browsers, apps)	Web servers, application servers
Purpose	Anonymity, filtering, access control	Load balancing, caching, security
Example	VPN, Web Proxy	Nginx, Cloudflare, AWS ALB

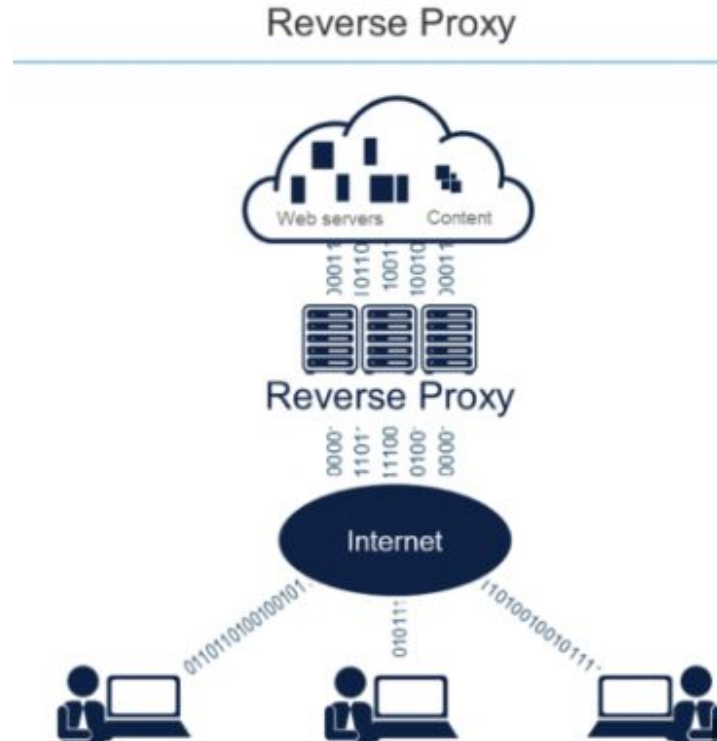
Use Cases for Forward Proxy

- Privacy & Anonymity (Hides user IP)
- Content Filtering (Restrict certain websites)
- Bypass Geo-restrictions (Access region-blocked content)
- Caching (Reduce redundant requests, speed up browsing)



Use Cases for Reverse Proxy

- Load Balancing (Distribute traffic to multiple servers)
- Security & DDoS Protection (Hide backend servers, block attacks)
- SSL Termination (Offload SSL processing to proxy)
- Caching & Compression (Reduce load on backend servers)



Interview Questions on Forward Proxy vs. Reverse Proxy

- Basic Concepts
 - What is a proxy server, and why is it used?
 - Explain the key differences between a forward proxy and a reverse proxy.
- Technical Deep Dive
 - How does a forward proxy improve security and privacy?
 - How does a reverse proxy help in load balancing and caching?
 - What are some real-world examples of forward and reverse proxies?
- Implementation & Use Cases
 - When should you use a forward proxy vs. a reverse proxy in system architecture?
 - What are some common tools and technologies used for each type of proxy?
 - How does a reverse proxy protect backend servers from DDoS attacks?
- Advanced Topics
 - How does SSL termination work in a reverse proxy?
 - What are the advantages of using Cloudflare, Nginx, or HAProxy as a reverse proxy?
 - How does a transparent proxy differ from a forward or reverse proxy?

Wrap-Up & What's Next?

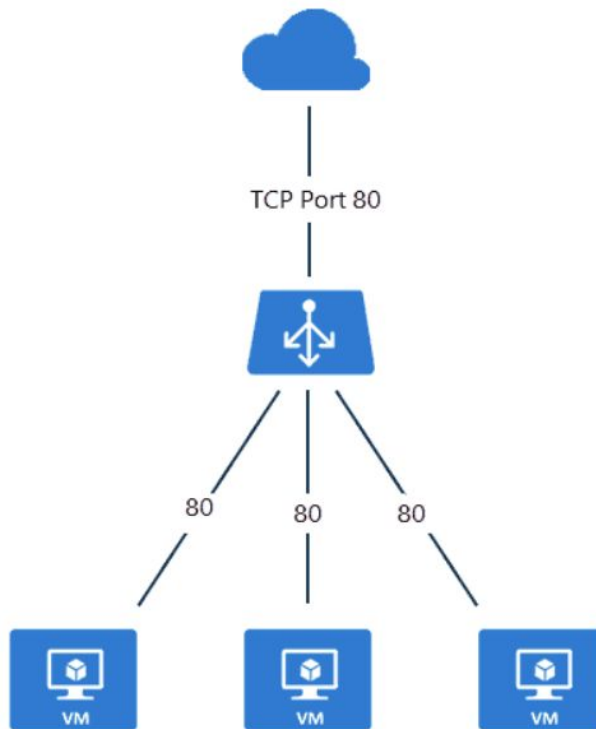
- Key Takeaways:
 - A forward proxy helps clients interact with the internet securely.
 - A reverse proxy protects and optimizes backend servers.
 - Understanding proxy usage helps in designing scalable & secure architectures.
- Next Topic:
 - Load Balancers

Introduction to Load Balancing

— Networking & Communication —

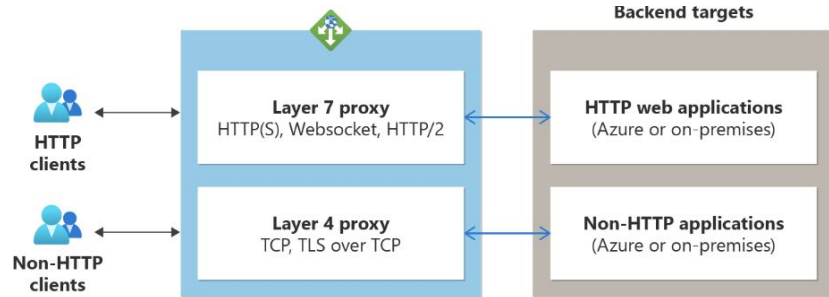
Why Load Balancing is Needed

- High Availability: Ensures system uptime even under high traffic.
- Traffic Distribution: Spreads requests evenly across servers.
- Prevents Overload: Avoids overburdening a single server.
- Improves Performance: Reduces latency and enhances response times.
- Handles Failures Gracefully: Redirects traffic in case of server failure.
- Supports Scalability: Helps scale systems efficiently.
- Example: A high-traffic e-commerce site uses load balancing to handle peak-hour requests.



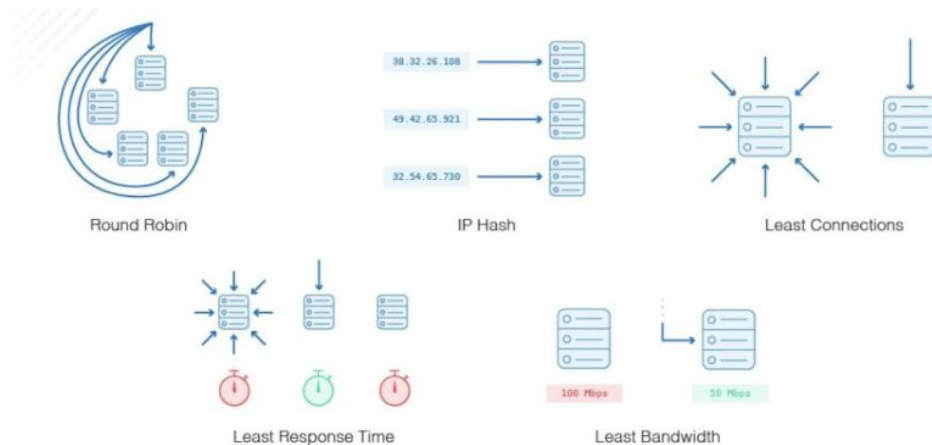
Types of Load Balancers

- Based on Layer
 - Layer 4 (Transport Layer): Operates at TCP/UDP level, distributing requests based on network-level data.
 - Layer 7 (Application Layer): Operates at HTTP/HTTPS level, making routing decisions based on request content.
- Based on Deployment
 - Hardware Load Balancers: Specialized devices (e.g., F5, Citrix NetScaler).
 - Software Load Balancers: Nginx, HAProxy, Envoy.
 - Cloud-based Load Balancers: AWS Elastic Load Balancer (ELB), Google Cloud Load Balancing.



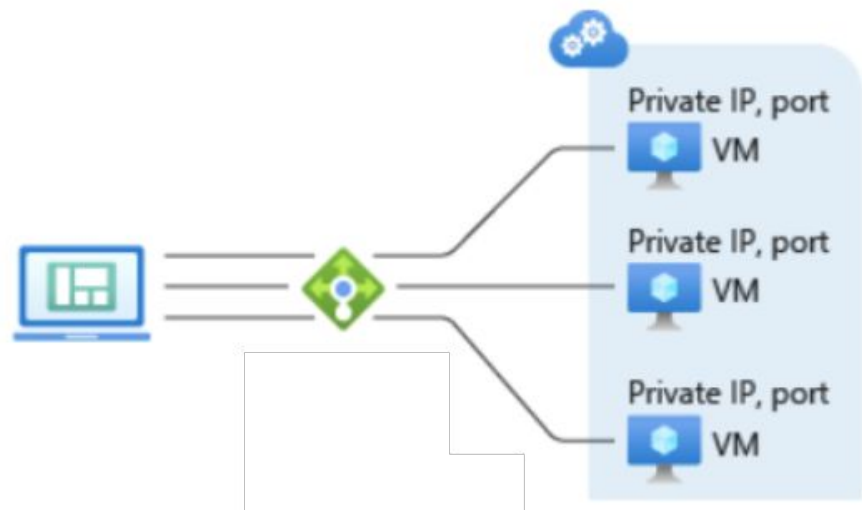
Load Balancing Strategies

- Static Load Balancing
 - Round Robin: Distributes requests sequentially to each server.
 - Least Connections: Directs traffic to the server with the fewest connections.
 - IP Hashing: Routes requests based on client IP.
- Dynamic Load Balancing
 - Least Response Time: Sends requests to the server with the fastest response.
 - Adaptive Load Balancing: Uses real-time monitoring to make decisions.
 - Weighted Load Balancing: Assigns different weights to servers based on capacity.



Load Balancer in Action

- Example Scenario:
 - A web application with multiple servers.
 - Users send requests, and the load balancer distributes them efficiently.



Choosing the Right Load Balancer

- Layer 4 vs. Layer 7: When to use each type.
- Scalability Needs: Matching the right load balancer to traffic volume.
- Security Concerns: SSL termination and DDoS protection.
- Use Cases:
 - Nginx/HAProxy for web applications.
 - AWS ELB for cloud-native applications.
 - Hardware Load Balancers for enterprise data centers.

Interview Questions on Load Balancing

- Fundamentals
 - What is load balancing, and why is it important?
 - Explain the difference between Layer 4 and Layer 7 load balancing.
 - How does a load balancer handle high availability and failover?
- Strategies & Use Cases
 - Compare Round Robin and Least Connections strategies.
 - What are the advantages of Weighted Load Balancing?
 - When would you use a software load balancer over a hardware one?
- Real-World Scenarios
 - How would you design a scalable load balancing solution for a large e-commerce site?
 - What factors should be considered when choosing a load balancing strategy?
 - How does a load balancer improve security?

Summary & Key Takeaways

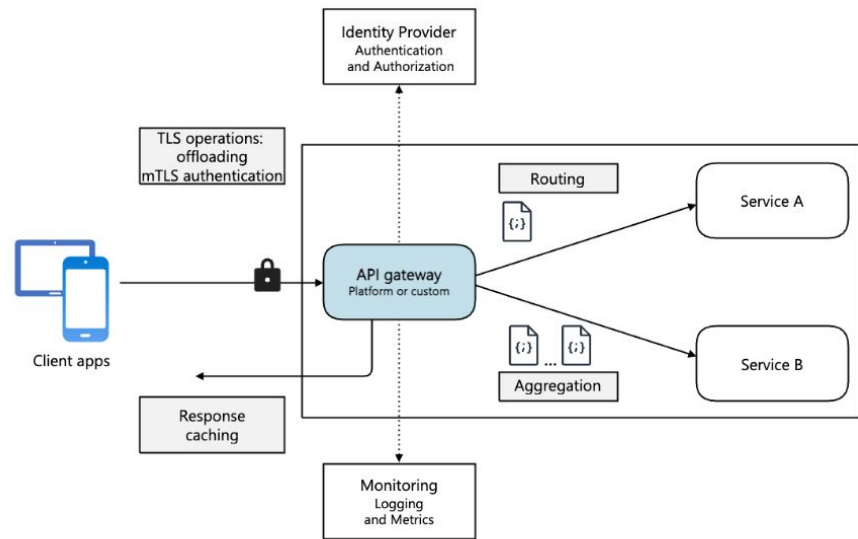
- Load balancing enhances scalability, reliability, and performance.
- Different types exist based on layers and deployment models.
- Choosing the right strategy depends on traffic patterns and system needs.
- Essential for highly available and resilient architectures.
- What's next:
 - API Gateway

What is an API Gateway?

— Networking & Communication —

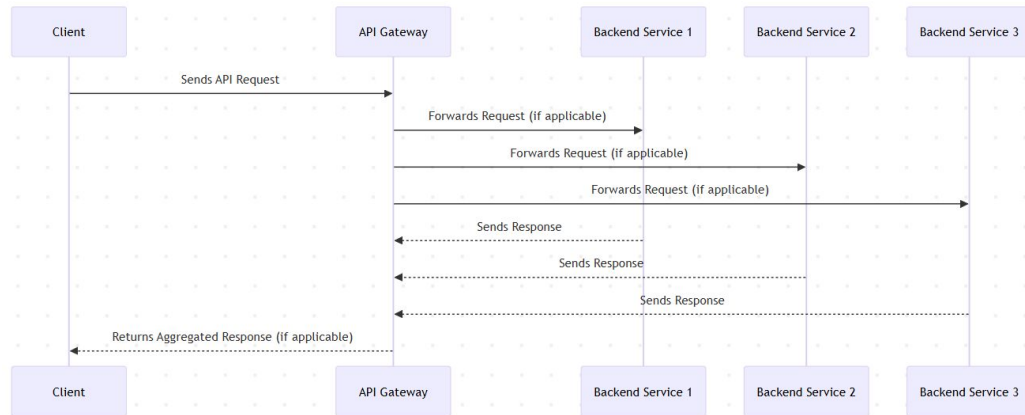
Introduction to API Gateways

- In modern system architecture, APIs connect clients (web, mobile, third-party apps) to backend services.
- Directly exposing backend services can lead to performance, security, and management challenges.
- API Gateway is a centralized entry point for all API requests, handling authentication, routing, caching, and security.



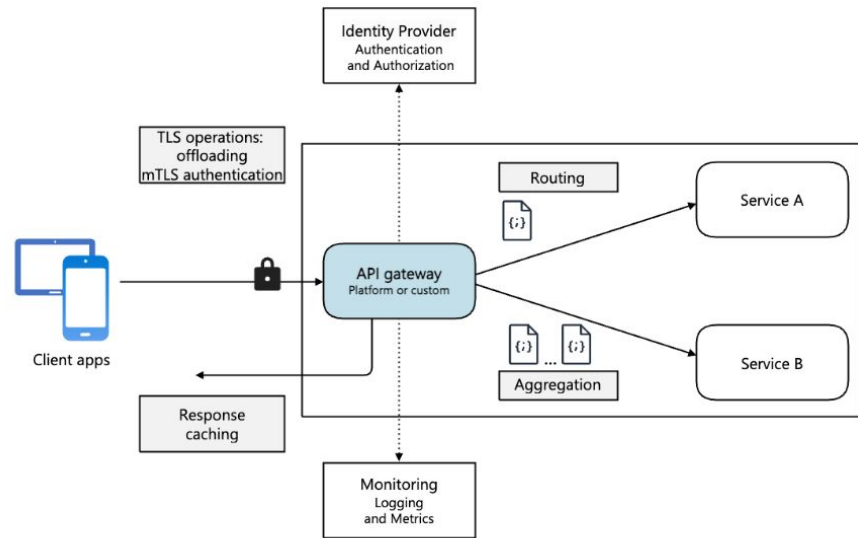
How API Gateways Work?

- API Gateway acts as a reverse proxy between clients and backend services.
- It receives API requests, processes them, and forwards them to the correct service.
- Key functions include request transformation, authentication, rate limiting, and response handling.



Benefits of Using an API Gateway

- Security: Protects backend services from direct exposure.
- Rate Limiting: Prevents abuse & DDoS attacks.
- Load Balancing: Distributes traffic efficiently.
- Caching: Speeds up response times.
- Request Transformation: Converts request formats, protocols, and headers.
- Logging & Monitoring: Tracks API performance and security events.



Security Features in API Gateways

- Authentication & Authorization: Supports OAuth, JWT, API keys.
- DDoS Protection: Blocks excessive requests from malicious sources.
- Bot Protection: Uses rate limiting & CAPTCHA to filter out bot traffic.
- TLS Termination: Handles SSL encryption to secure communication.

Rate Limiting & Throttling

- Rate Limiting: Restricts the number of API calls per user/IP per second/minute.
- Throttling: Controls traffic during peak loads to avoid system crashes.
- Helps prevent abuse, protects backend services, and ensures fair usage.

Caching for Performance Optimization

- Why Caching?: Speeds up API responses by storing frequently accessed data.
- Types of Caching:
 - In-memory caching (e.g., Redis)
 - Response caching (storing API responses for quick reuse)
 - Edge caching (CDN integration for global performance boost)

API Composition & Aggregation

- API Gateways can combine multiple API calls into a single request.
- Useful for microservices architecture where one client request may need data from multiple services.
- Example:
 - Instead of calling /user, /orders, and /cart separately,
 - API Gateway calls all 3 services internally and returns a single response.

Logging & Monitoring with API Gateways

- Helps track API performance, errors, and security threats.
- Logs can be used for debugging, auditing, and analytics.
- Supports integrations with tools like Prometheus, Grafana, and ELK Stack.

Popular API Gateway Implementations

- Open-source solutions:
 - Kong
 - NGINX
 - Traefik
- Cloud-based solutions:
 - AWS API Gateway
 - Google Apigee
 - Azure API Management

When to Use an API Gateway?

- Best suited for:
 - Microservices architectures.
 - Multi-client APIs (Web, Mobile, IoT).
 - APIs requiring security, rate limiting, and monitoring.
- Avoid if:
 - Simple monolithic apps with minimal API exposure.
 - Low-traffic, internal-only services.

API Gateway - Interview Questions

- Basic Questions:
 - What is an API Gateway, and why is it used?
 - How does an API Gateway differ from a Load Balancer?
 - What are the key benefits of using an API Gateway?
- Technical Questions:
 - How does an API Gateway handle authentication and authorization?
 - Explain rate limiting and throttling in API Gateways.
 - What caching strategies can be implemented in an API Gateway?
 - How does an API Gateway improve security against DDoS attacks?
- Scenario-Based Questions:
 - When should you use an API Gateway in a microservices architecture?
 - How would you design an API Gateway for a large-scale system with millions of users?
 - What challenges might arise when implementing an API Gateway, and how would you address them?

Conclusion & Key Takeaways

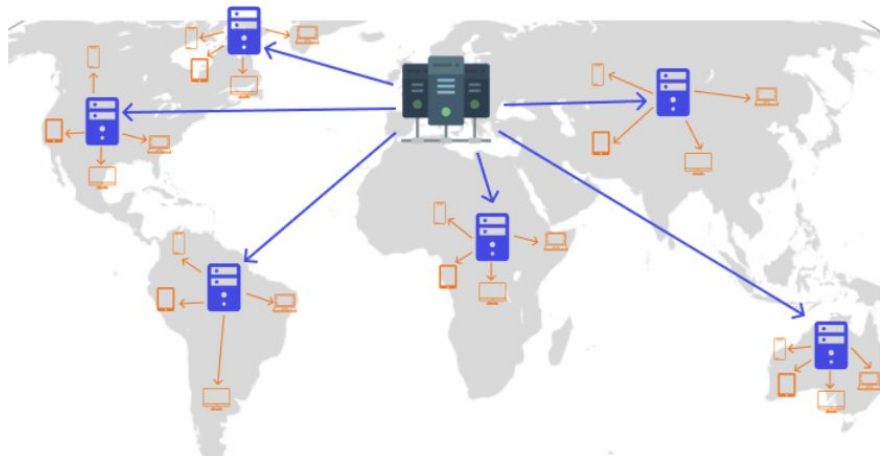
- API Gateway is a centralized entry point for managing API traffic.
- Provides security, rate limiting, caching, and logging.
- Useful for microservices & high-traffic applications.
- Choosing the right API Gateway depends on use case & scale.
- Whats next:
 - Content Delivery Networks

Content Delivery Networks (CDN) in System Design

— Networking & Communication —

Introduction to CDNs

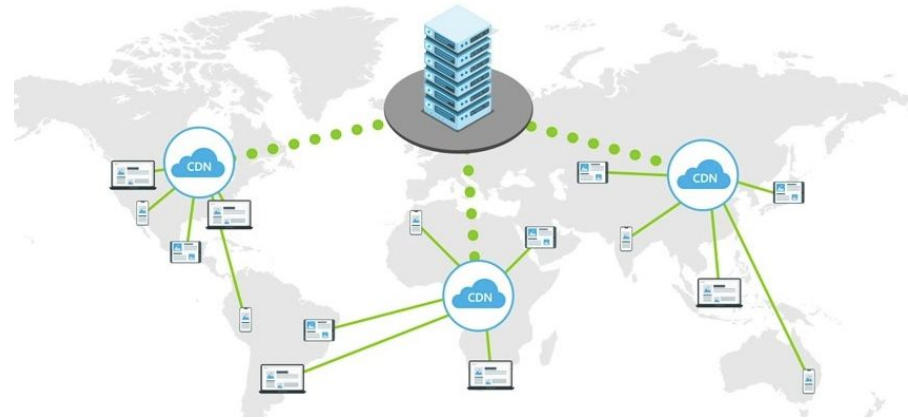
- Definition: A CDN is a globally distributed network of servers that work together to deliver content efficiently.
- Why CDNs exist:
 - Reduces latency
 - Enhances content availability
 - Improves load handling
 - Security



Why are CDNs needed?

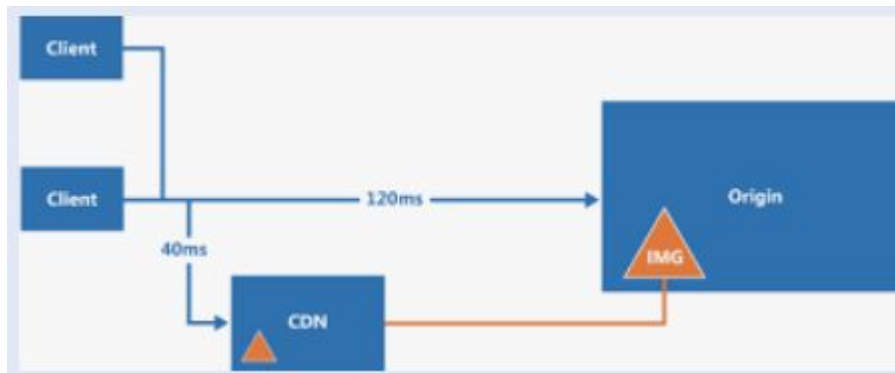
The problem without CDNs:

- High latency due to geographic distance
- Overloaded origin servers
- Bandwidth constraints & slow load times



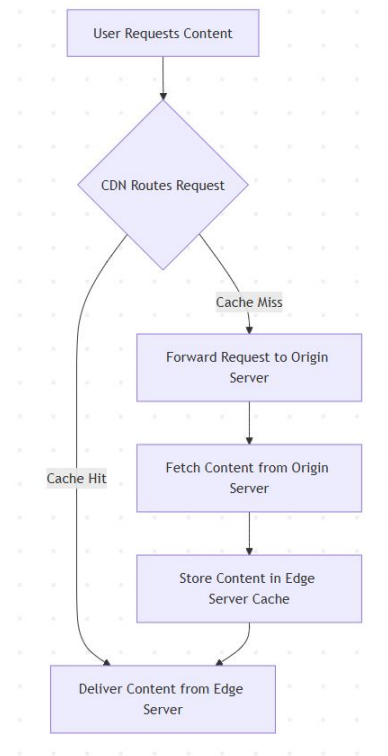
CDN Architecture Overview

- Components of a CDN:
 - Origin servers: Store the original content
 - Edge servers (PoPs): Store cached content closer to users
 - Request routing system: Directs users to the nearest PoP



How a CDN Works and Handles a User Request

1. User requests content (e.g., website, video, API data).
2. CDN directs the request to the nearest edge server based on geographic location, network latency, and server load.
3. Cache Hit: If the content is available on the edge server, it is delivered instantly.
4. Cache Miss: If the content is not cached, the request is forwarded to the origin server, fetched, and stored at the edge server for future use.

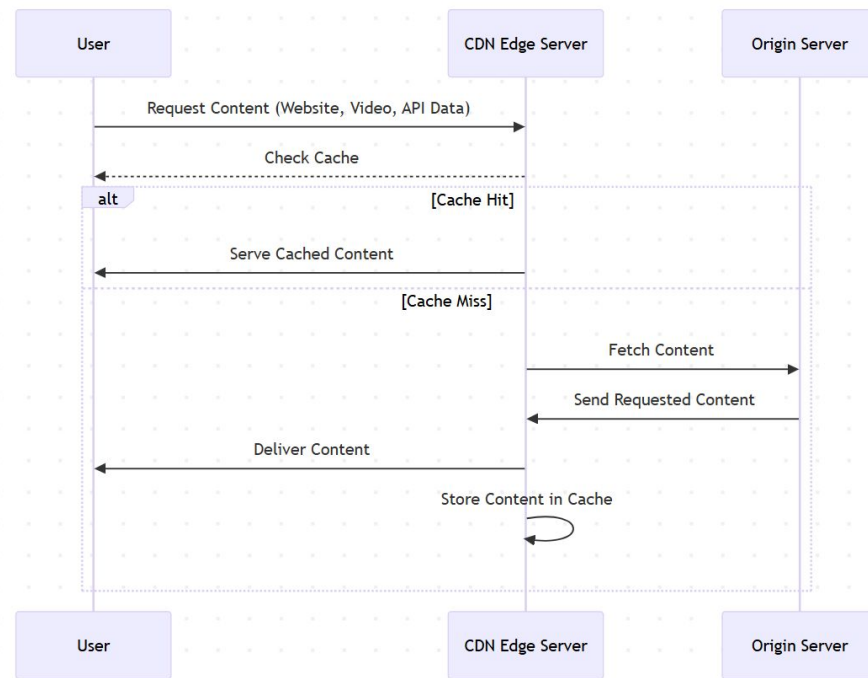


Key CDN Features/Benefits of CDN Working Mechanism

- **Caching & Replication(Reduced Latency):** Frequently accessed content is stored at edge locations to minimize origin server load.
- **Load Balancing and request routing:(Improved Availability)** Traffic is distributed across multiple PoPs to ensure reliability and prevent overload. CDN uses Geo-based, latency-based, and load-aware routing to optimize speed.
- **Compression & Optimization(Lower Bandwidth Costs):** Techniques like Gzip, Brotli, and image optimization reduce file sizes for faster delivery.
- **Security Features(Better Security):** DDoS Protection, SSL/TLS

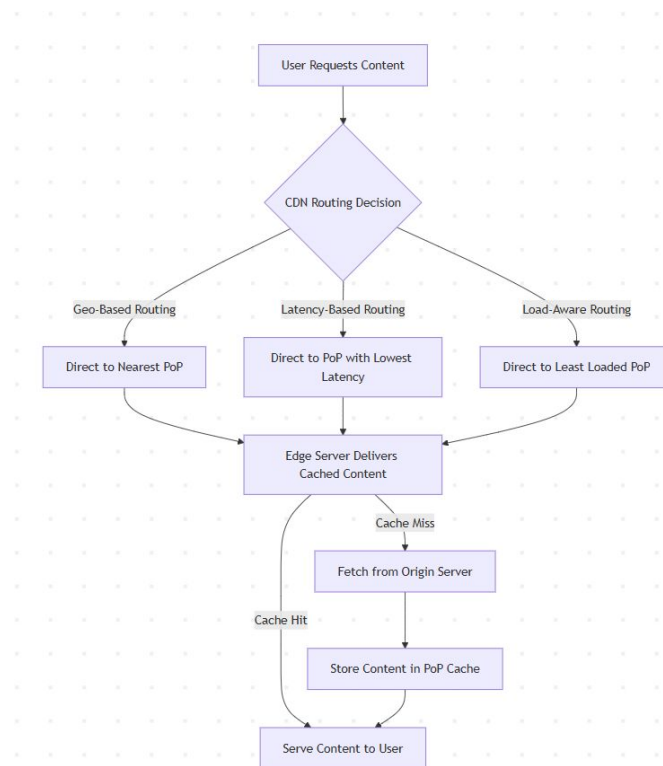
Content Caching and Replication

- How CDNs cache content:
 - Frequently accessed content is stored at the edge servers
 - Cache expiration and TTL (Time-To-Live) settings control freshness
 - Cache invalidation strategies (manual purge, stale-while-revalidate)



Load Balancing/Failover Handling and Request Routing

- Load balancing across multiple PoPs
- Failover strategies:
 - Rerouting to another PoP
 - Dynamic origin selection
- Request routing strategies:
 - Geo-based routing: Directs users to the nearest PoP
 - Latency-based routing: Chooses the PoP with the lowest latency
 - Load-aware routing: Balances traffic across PoPs



Compression & Minification

- Techniques used:
 - Gzip & Brotli compression
 - Image optimization (WebP, AVIF)
 - Minifying CSS/JS to reduce file sizes

Security Features

- DDoS mitigation:
 - Rate limiting
 - Traffic filtering
- SSL/TLS Offloading: Encrypts traffic for security

Use Cases of CDNs

- Static vs. Dynamic Content Delivery
 - Static content: Images, videos, HTML files.
 - Dynamic content: API responses, personalized pages.
- API Acceleration & Edge Computing(CDNs for APIs):
 - Reduce API response times.
 - Edge caching for frequently accessed endpoints

- Basics of CDN
 - What is a CDN, and how does it work?
 - Why do we need CDNs in system design?
 - What are the key benefits of using a CDN?
- CDN Architecture & Components
 - Explain the difference between an origin server and an edge server in a CDN.
 - What is a PoP (Point of Presence) in a CDN?
 - How does request routing work in a CDN?
- Caching Strategies & Content Delivery
 - What is a cache hit vs. cache miss, and how does a CDN handle them?
 - Explain cache expiration and TTL (Time-To-Live) in a CDN.
 - What are cache invalidation strategies, and why are they important?
 - How does a CDN handle dynamic content caching?
- Load Balancing & Failover Handling
 - How do CDNs use load balancing to improve reliability?
 - Explain different request routing strategies (Geo-based, Latency-based, Load-aware).
 - What happens if a CDN PoP fails? How does failover handling work?
- CDN Optimization Techniques
 - What compression and minification techniques do CDNs use?
 - How does a CDN optimize image and video delivery?
 - How does API acceleration work in a CDN?
- CDN Security & Challenges
 - How does a CDN protect against DDoS attacks?
 - What is SSL/TLS offloading, and why is it useful?
 - What are some challenges in CDN implementation, and how can they be mitigated?
- Advanced CDN Topics
 - How does a multi-CDN architecture work?
 - What is edge computing, and how does it relate to CDNs?
 - How would you design a CDN for a large-scale video streaming platform?
 - How do CDNs help in real-time applications like online gaming or stock trading?

Wrap-Up & Key Takeaways

- CDNs improve performance by caching and distributing content globally.
- They reduce latency, improve load times, and lower bandwidth costs.
- CDNs enhance security with DDoS protection, SSL/TLS encryption, and WAFs.
- Use cases include static content delivery, API acceleration, and edge computing.
- Next Topic:
 - Networking Fundamentals Summary & Best Practices

Summary of Covered Topics - Networking & Communication

- **Introduction to Networking in System Design** – Why networking is crucial in distributed systems.
- **Understanding IP Addresses** – Differences between IPv4 & IPv6, public vs. private IPs.
- **How DNS Works** – Domain Name Resolution, caching, and its role in large-scale systems.
- **Client-Server Model Explained** – How clients and servers interact using request-response cycles.
- **Forward Proxy vs. Reverse Proxy** – Their differences, use cases, and impact on system design.
- **Introduction to Load Balancing** – Why it's needed, types of load balancers, and balancing strategies.
- **What is an API Gateway?** – Managing API traffic, enforcing security, and enabling efficient routing.
- **Content Delivery Networks (CDN) in System Design** – How CDNs improve performance and reduce latency.
- **Whats Next**
 - Protocols