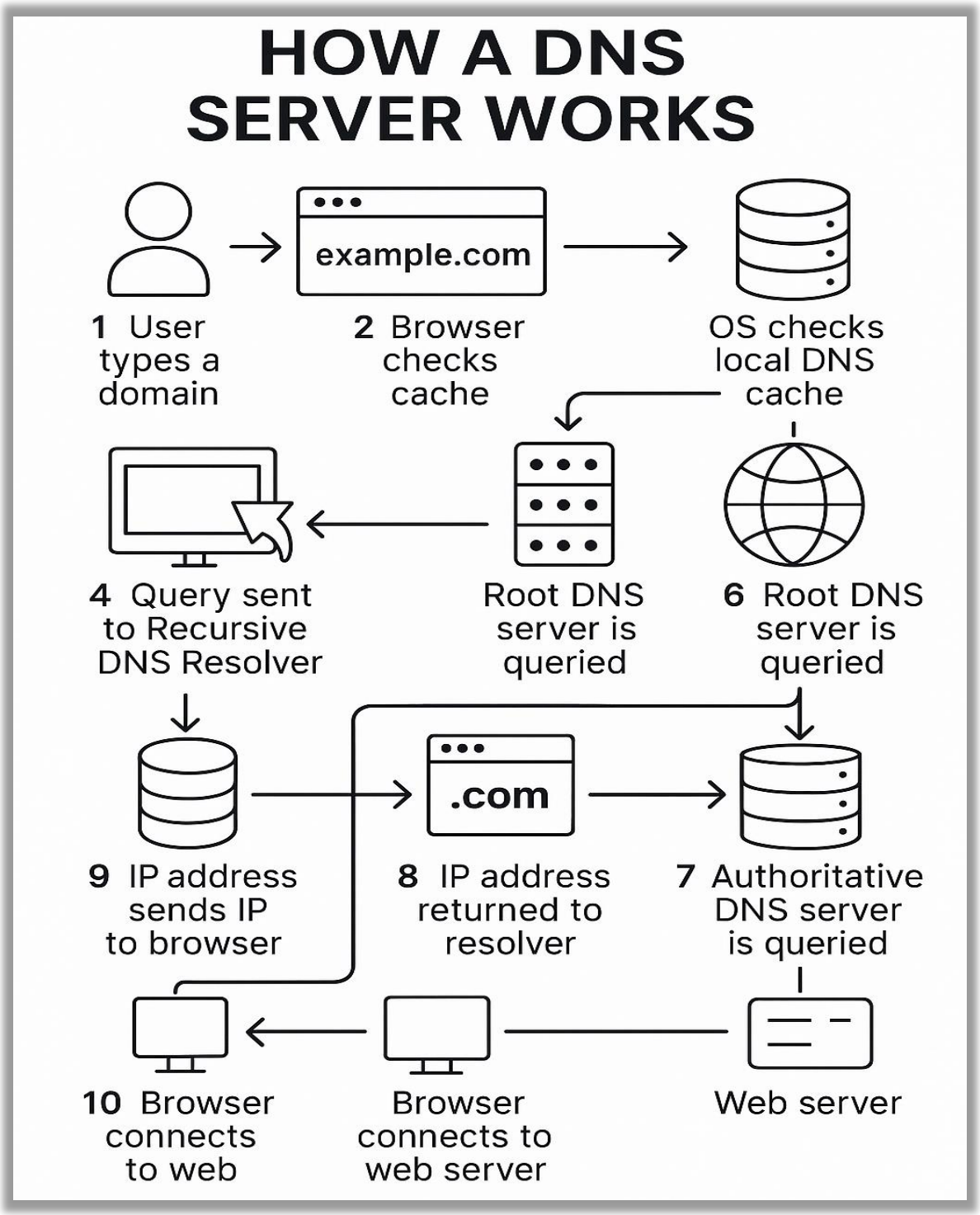
All About DNS

How It Works, Why It Matters, and Real-World DevOps Use Cases

Venkatesh Jilakarra

What is DNS (Domain Name System)?

DNS is a fundamental component of the Internet. It translates human-readable domain names (www.google.com) into IP addresses (142.250.190.36) that computers use to identify each other on the network. Without DNS, users would have to memorize complex IP addresses to access websites.



How DNS Works – Step-by-Step

When a user types a domain name into a web browser, the browser must find the corresponding IP address to establish a connection. This process involves multiple steps and components:

1. User Types a Domain

- The user opens a browser and enters a domain name like www.example.com.
- **Example**: You type www.google.com into the Chrome browser.

2. Browser Checks DNS Cache

- The browser checks its local cache to see if it already knows the IP address for www.example.com.
- If the record exists and is not expired, it uses it immediately.
- **Example**: If you visited Google recently, your browser might still remember its IP (e.g., 142.250.190.68).

3. OS Checks Local DNS Cache

- If the browser cache doesn't have the info, the operating system (Windows/Linux/macOS) checks its own cache.
- **Example**: The OS stores recent DNS responses to avoid querying again.

4. Query Sent to Recursive DNS Resolver

- If neither cache has the IP, the request is sent to a **recursive DNS resolver**.

- This is typically provided by your ISP or a third-party (e.g., Google DNS: 8.8.8.8).
- **Example**: Your device sends a DNS query to 8.8.8.8.

5. Recursive Resolver Checks Its Own Cache

- The resolver first checks if it has the requested IP cached from previous queries.
- If not, it begins a full DNS resolution process.

6. Root DNS Server is Queried

- The resolver contacts a **root DNS server**.
- Root servers don't know the full address but respond with the location of the Top-Level Domain (TLD) server.
- **Example**: For www.google.com, it directs the resolver to .com TLD servers.

7. TLD Server is Queried

- The resolver then queries the **TLD DNS server** (e.g., .com, .org, .net) for the domain.
- It responds with the location of the **authoritative DNS server** for example.com.
- **Example**: .com server responds with the authoritative name server for google.com.

8. Authoritative DNS Server is Queried

- This server holds the actual DNS records for the domain.
- It provides the final **IP address**.
- **Example**: The authoritative server for google.com returns 142.250.190.68.

9. IP Address is Returned to Resolver

- The recursive resolver receives the IP address and caches it for future requests.
- It forwards the IP to the user's device.

10. Browser Connects to Web Server

- The browser uses the IP to connect directly to the web server and load the website.
- **Example**: The browser establishes a connection to 142.250.190.68, and Google's homepage loads.

Types of DNS Records

DNS records are stored in **zone files** on authoritative name servers. Different types of records serve different purposes:

- **A (Address) Record**: Maps a domain to an IPv4 address.
- **AAAA Record**: Maps a domain to an IPv6 address.
- **CNAME (Canonical Name) Record**: Creates an alias for a domain.
- **MX (Mail Exchange) Record**: Specifies the mail servers for email handling.
- **NS (Name Server) Record**: Indicates the authoritative name servers for the domain.
- **TXT Record**: Holds human-readable text for various uses, including domain verification and email security (e.g., SPF, DKIM).
- **SOA (Start of Authority) Record**: Contains administrative information about the zone.

DNS Hierarchy and Structure

DNS is hierarchical and organized into different levels:

- **Root Level**: Represented by a dot (.) and managed by root name servers.
- **Top-Level Domains (TLDs)**: These include generic TLDs like .com, .org, .net, and country code TLDs like .in, .uk.
- **Second-Level Domains**: These come directly under TLDs. For instance, in example.com, "example" is the second-level domain.
- **Subdomains**: Parts of the domain added before the second-level domain, such as blog.example.com.

This hierarchy allows for distributed control and scalability across the global internet.

DNS Caching

To enhance performance and reduce latency, DNS responses are cached at several levels:

- **Browser Cache**: Stores recently resolved addresses for quick retrieval.
- **Operating System Cache**: Intermediate cache on the user's device.

- **Recursive Resolver Cache**: Stores DNS responses for a Time-To-Live (TTL) period to prevent repetitive queries to root and authoritative servers.

Caching dramatically reduces DNS lookup times and decreases traffic on upstream DNS infrastructure.

DNS Security Considerations

While DNS is essential for internet functionality, it is also a target for various attacks:

Common Vulnerabilities

- **DNS Spoofing/Cache Poisoning**: Attackers inject false DNS records into a resolver's cache, directing users to malicious sites.
- **Man-in-the-Middle Attacks**: Intercept DNS requests to serve altered or misleading responses.

Security Measures

- **DNSSEC (DNS Security Extensions)**: Adds digital signatures to DNS responses to ensure they are not tampered with.
- **DNS over HTTPS (DoH)** and **DNS over TLS (DoT)**: Encrypt DNS queries and responses to prevent third-party observation or manipulation.

Real-World scenarios

1. CI/CD Pipeline Failing on External Dependencies

Scenario: A Jenkins job intermittently fails while pulling code from GitHub.

Relevance:
The Jenkins agent cannot resolve the domain due to a DNS timeout or misconfigured resolver. The SRE team investigates and identifies issues with the internal DNS forwarders or recursive resolvers.

2. Infrastructure Deployment with Terraform and DNS (AWS Route 53)

Scenario: You are automating the creation of DNS records for a new application.

Relevance:
Using Terraform, you configure a Route 53 zone and set up A/AAAA records to point to a new ALB or EC2 instance, allowing applications to be accessible immediately after deployment.

3. Blue-Green or Canary Deployment via DNS Switching

Scenario: A new version of an application is deployed to a parallel environment (green) for testing and gradual rollout.

Relevance:
DNS records are updated to shift traffic to the new version. Rollback is performed by reverting DNS to the previous version. DNS TTL configuration becomes crucial for responsiveness.

4. High Latency in Multi-Region Applications

Scenario: Users in a specific region (e.g., Asia) are experiencing high latency.

Relevance:
A misconfigured DNS routing policy is sending requests to a server in another region (e.g., the US). Correcting geo-based DNS routing improves performance and user experience.

5. Monitoring and Alerting Failures Due to DNS Issues

Scenario: Prometheus cannot scrape targets or Alertmanager becomes unreachable.

Relevance:
These services rely on internal DNS (e.g., alertmanager.service.cluster.local). DNS misconfigurations or failures (such as in CoreDNS) disrupt observability.

6. Security Incident via DNS Spoofing or Cache Poisoning

Scenario: A user unknowingly visits a malicious website that looks like an internal tool.

Relevance:
DNS cache poisoning redirected the domain to an attacker-controlled server. This scenario highlights the importance of DNSSEC, secure resolvers, and DNS monitoring.

7. VPC Peering and Private DNS Resolution Across AWS Accounts

Scenario: A database hosted in one VPC is accessed from another via VPC peering.

Relevance:
Proper DNS configuration is essential for internal services to resolve names across VPCs using Route 53 private hosted zones and conditional forwarding rules.

8. Rollbacks Delayed Due to High DNS TTLs

Scenario: A faulty deployment is live, and the rollback strategy depends on DNS record changes.

Relevance:
Due to a high TTL (e.g., 1 hour), clients continue accessing the broken service. Using a lower TTL in production environments allows faster DNS propagation for quick recovery.

9. External API Timeouts Caused by Slow DNS Resolution

Scenario: Application logs show timeouts when connecting to an external API.

Relevance:
DNS queries for api.paymentprovider.com take several seconds. Switching to a more reliable DNS provider (like Cloudflare or Google DNS) resolves the issue.

10. Local Development with Fake Domains

Scenario: Development environment uses custom domains like service.local.dev.

Relevance:
Local DNS resolution is configured using tools like DNSMasq or through /etc/hosts. Misconfigurations or stale entries can break communication between services in the local environment.