Complete DNS Guide: How DNS Actually Works (Detailed Notes)

Table of Contents

---

1. DNS Architecture & Components

1.1 The Four Types of DNS Servers

DNS operates through a hierarchical system of different server types, each with specific responsibilities:

A. Recursive Resolver (DNS Resolver)

- **Role**: Acts as an intermediary between clients and authoritative DNS servers
- **Function**:
    - Receives DNS queries from client applications
    - Queries other DNS servers on behalf of the client
    - Caches responses to speed up future queries
    - Does the "heavy lifting" of the DNS lookup
- **Example Providers**: Google (8.8.8.8), Cloudflare (1.1.1.1), ISP DNS servers

- **Analogy**: Like a librarian who searches through multiple catalogs to find the book you need

B. Root Nameservers

- **Role**: The top of the DNS hierarchy
- **Function**:
    - Directs queries to appropriate TLD nameservers
    - There are 13 root server systems (labeled A through M)
    - Operated by 12 different organizations worldwide
    - Uses anycast routing for redundancy (hundreds of physical servers)
- **Example**: When looking up `www.google.com`, root servers direct to `.com` TLD servers
- **Location**: Distributed globally for resilience

C. TLD (Top-Level Domain) Nameservers

- **Role**: Manages specific top-level domains
- **Function**:
    - Stores information for all domains within a TLD
    - Points to authoritative nameservers for specific domains
- **Types of TLDs**:
    - **Generic TLDs (gTLDs)**: .com, .net, .org, .edu
    - **Country Code TLDs (ccTLDs)**: .uk, .de, .jp, .us
    - **New gTLDs**: .tech, .io, .app, .dev
- **Example**: .com TLD server knows where to find google.com's authoritative servers

D. Authoritative Nameservers

- **Role**: The final authority for a specific domain

- **Function**:
    - Holds the actual DNS records (A, AAAA, CNAME, MX, etc.)
    - Provides definitive answers to queries
    - Can answer recursively or authoritatively
- **Example**: Google's authoritative servers hold the actual IP addresses for google.com
- **Key Point**: This is where your DNS records are actually stored

1.2 Recursive vs Authoritative DNS

Feature	Recursive Resolver	Authoritative Server
Purpose	Finds information for clients	Stores actual DNS records
Caching	Yes, extensively	No caching (always fresh data)
Query Type	Makes queries to other servers	Responds with definitive answers
Data Source	Aggregates from multiple sources	Direct source of truth
Scope	Handles any domain query	Only specific domains it's authoritative for

Examples	8.8.8.8, 1.1.1.1, ISP DNS	Route53, Cloudflare DNS, your hosting provider

Cloudflare

---

2. The Complete DNS Resolution Process (Step-by-Step)

Let's trace what happens when you type `www.example.com` in your browser:

Step 1: Browser Cache Check

```
Your Browser: "Do I already know where www.example.com is?"
  Checks browser's internal DNS cache
  If found and TTL not expired → Use cached IP (DONE)
  If not found → Continue to Step 2
```

Step 2: Operating System Cache Check

```
Your OS: "Do I have this in my DNS cache?"
- Windows: Uses DNS Client service cache
- Linux/Mac: Uses nscd or systemd-resolved
- If found and TTL valid → Return cached IP (DONE)
- If not found → Continue to Step 3
```

Step 3: Query Recursive Resolver

```
Your Computer → Recursive Resolver (e.g., 8.8.8.8)
"What is the IP address for www.example.com?"

Resolver checks its cache:
  If cached and TTL valid → Return IP (DONE)
  If not cached → Start recursive lookup (Continue to Step 4)
```

Step 4: Query Root Nameserver

```
Recursive Resolver → Root Nameserver (.root)
Query: "Where can I find www.example.com?"

Root Server Response:
```

"I don't know the IP, but I can tell you where to find .com servers"
Returns: IP addresses of .com TLD nameservers

Step 5: Query TLD Nameserver

Recursive Resolver → TLD Nameserver (.com)
Query: "Where can I find www.example.com?"

TLD Server Response:
"I don't have the IP, but example.com uses these nameservers:"
Returns: NS records pointing to example.com's authoritative servers
Example: ns1.example-dns.com, ns2.example-dns.com

Step 6: Query Authoritative Nameserver

Recursive Resolver → Authoritative Nameserver (ns1.example-dns.com)
Query: "What is the IP address for www.example.com?"

Authoritative Server Response:
"Here is the definitive answer:"
Returns: A record: 93.184.216.34, TTL: 3600 seconds

Step 7: Return Result to Client

Recursive Resolver → Your Computer
"The IP address is 93.184.216.34"

Your Computer:
- Caches the result (for TTL duration)
- Passes IP to browser

Your Browser:
- Caches the result
- Initiates HTTP/HTTPS connection to 93.184.216.34

Visual Flow Diagram:

User Browser
↓
OS DNS Cache (check)
↓
Recursive Resolver (8.8.8.8)
↓
Root Server (.) → "Ask .com server"

```
↓
TLD Server (.com) → "Ask example.com's nameserver"
↓
Authoritative Server (ns1.example-dns.com) → "IP is 93.184.216.34"
↓
Result cached at each level (based on TTL)
↓
Browser connects to 93.184.216.34
```

Key Points:

- The entire process typically takes **20-120 milliseconds**
- Subsequent queries are much faster due to caching
- Each server only needs to know about the next level
- This hierarchical approach distributes the load globally

Cloudflare

---

3. DNS Hosted Zones (Public vs Private)

A **hosted zone** is a container that holds DNS records for a specific domain. Think of it as a database or configuration file for your domain's DNS information.

3.1 Public Hosted Zones

Purpose: Makes your domain accessible over the public internet

Characteristics:

- ☑ Resolves queries from anywhere on the internet
- ☑ Used for public-facing websites, applications, and services
- ☑ Records are publicly queryable
- ☑ Anyone can perform DNS lookups on your domain

Use Cases:

- Public websites (www.yoursite.com)
- Email servers (mail.yoursite.com)
- API endpoints (api.yoursite.com)
- CDN configurations

Example Configuration:

```
Hosted Zone: example.com (Public)
├── A record:          www.example.com → 203.0.113.10
├── A record:          example.com → 203.0.113.10
├── MX record:         example.com → mail.example.com (Priority: 10)
├── CNAME record: blog.example.com → example.com
└── TXT record:   example.com → "v=spf1 include:_spf.google.com ~all"
```

3.2 Private Hosted Zones

Purpose: DNS resolution within private networks only (VPC, corporate networks)

Characteristics:

- 🔒 Only accessible from within specified VPCs or private networks
- 🔒 Not resolvable from the public internet
- 🔒 Provides internal service discovery
- 🔒 Enhanced security for internal resources

Use Cases:

- Internal APIs (internal-api.company.local)
- Database servers (db-master.internal)
- Microservices communication (user-service.internal)
- Development/staging environments

Example Configuration:

```
Hosted Zone: company.internal (Private, VPC: vpc-abc123)
```

```
├── A record:          db-master.company.internal → 10.0.1.50
├── A record:          api.company.internal → 10.0.2.100
├── CNAME record: cache.company.internal → redis-cluster.company.internal
└── A record:          redis-cluster.company.internal → 10.0.3.75
```

3.3 Comparison Table

Feature	Public Hosted Zone	Private Hosted Zone
Accessibility	Internet-wide	VPC/Private network only
Query Source	Any DNS resolver	Only authorized networks
Use Case	Public websites, email	Internal services, databases
Security	Public information	Hidden from internet
Domain Names	Standard domains (.com, .net)	Can use custom TLDs (.internal, .local)
Cost	Per hosted zone + queries	Per hosted zone + VPC association
Example	www.amazon.com	db.internal.aws

3.4 Split-Horizon DNS (Hybrid Approach)

You can have **both public and private hosted zones** for the **same domain**:

Example Scenario:

```
Domain: example.com

Public Hosted Zone (for external users):
├── www.example.com → 203.0.113.10 (public web server)

Private Hosted Zone (for internal VPC):
├── www.example.com → 10.0.1.50 (internal web server)
├── admin.example.com → 10.0.1.100 (admin panel, not public)
└── db.example.com → 10.0.2.20 (database server)
```

Result:

- External users accessing `www.example.com` get the public IP
- Internal VPC resources accessing `www.example.com` get the private IP
- `admin.example.com` and `db.example.com` only resolve internally

AWS Route 53 Documentation

---

4. DNS Record Types Explained (With Examples)

DNS records are instructions that live in authoritative DNS servers and provide information about a domain.

4.1 A Record (Address Record)

Purpose: Maps a domain name to an IPv4 address

Format:

```
Host: www.example.com
Type: A
Value: 192.168.1.100
TTL: 3600
```

Example:

```
example.com          A    93.184.216.34
www.example.com      A    93.184.216.34
```

```
api.example.com      A    203.0.113.50
```

Use Cases:

- Point your domain to a web server
- Direct subdomain to specific servers
- Load balancing (multiple A records)

4.2 AAAA Record (IPv6 Address Record)

Purpose: Maps a domain name to an IPv6 address

Format:

```
Host: www.example.com
Type: AAAA
Value: 2001:0db8:85a3:0000:0000:8a2e:0370:7334
TTL: 3600
```

Example:

```
example.com          AAAA   2606:2800:220:1:248:1893:25c8:1946
www.example.com      AAAA   2606:2800:220:1:248:1893:25c8:1946
```

Note: Modern websites should have both A and AAAA records for IPv4/IPv6 dual-stack support.

4.3 CNAME Record (Canonical Name)

Purpose: Creates an alias pointing one domain to another domain

Format:

```
Host: blog.example.com
Type: CNAME
Value: example.com
TTL: 3600
```

Example:

```
www.example.com      CNAME  example.com
```

```
blog.example.com    CNAME  hosting-provider.com
shop.example.com    CNAME  shopify.com
```

Important Rules:

- ❌ Cannot use CNAME for root domain (example.com)
- ❌ Cannot coexist with other record types for the same name
- ☑ Can chain CNAMEs (but not recommended for performance)
- ☑ Perfect for pointing to CDNs or hosting services

Use Cases:

- Point subdomain to CDN: `cdn.example.com → d111111abcdef8.cloudfront.net`
- Redirect subdomain: `www.example.com → example.com`
- Service integration: `store.example.com → mystore.shopify.com`

4.4 MX Record (Mail Exchange)

Purpose: Specifies mail servers responsible for receiving email

Format:

```
Host: example.com
Type: MX
Priority: 10
Value: mail.example.com
TTL: 3600
```

Example:

```
example.com    MX    10    mail1.example.com
example.com    MX    20    mail2.example.com
example.com    MX    30    mail3.example.com
```

Priority Explanation:

- Lower number = Higher priority
- Mail servers try lowest priority first
- If it fails, try next highest priority
- Provides redundancy and load balancing

Real-World Example (Gmail):

```
example.com     MX  1    aspmx.l.google.com
example.com     MX  5    alt1.aspmx.l.google.com
example.com     MX  5    alt2.aspmx.l.google.com
example.com     MX  10   alt3.aspmx.l.google.com
example.com     MX   10   alt4.aspmx.l.google.com
```

4.5 TXT Record (Text Record)

Purpose: Stores text information, often for verification and security

Format:

```
Host: example.com
Type: TXT
Value: "v=spf1 include:_spf.google.com ~all"
TTL: 3600
```

Common Use Cases:

A. SPF (Sender Policy Framework) - Email Authentication

```
example.com     TXT  "v=spf1 ip4:192.168.1.0/24 include:_spf.google.com ~all"
```

Specifies which mail servers can send email on behalf of your domain.

B. DKIM (DomainKeys Identified Mail) - Email Signing

```
default._domainkey.example.com     TXT  "v=DKIM1; k=rsa; p=MIGfMA0GCS..."
```

Public key for verifying email signatures.

C. DMARC (Domain-based Message Authentication) - Email Policy

```
_dmarc.example.com    TXT   "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"
```

Tells receivers what to do with emails that fail SPF/DKIM.

D. Domain Verification (Google, Microsoft, etc.)

```
example.com    TXT
"google-site-verification=rXOxyZounnZasA8Z7oaD3c14JdjS9aKSWvsR1EbUSIQ"
```

E. Site Configuration

```
example.com    TXT  "v=spf1 include:_spf.google.com ~all"
example.com    TXT  "google-site-verification=..."
example.com    TXT  "facebook-domain-verification=..."
```

Note: A domain can have multiple TXT records.

4.6 NS Record (Nameserver)

Purpose: Delegates a domain or subdomain to a set of nameservers

Format:

```
Host: example.com
Type: NS
Value: ns1.example-dns.com
TTL: 172800
```

Example:

```
example.com        NS   ns1.awsdns-12.com
example.com        NS   ns2.awsdns-34.net
example.com        NS   ns3.awsdns-56.org
example.com        NS   ns4.awsdns-78.co.uk
```

Use Cases:

- **Root domain**: Points to authoritative nameservers
- **Subdomain delegation**:

```
shop.example.com    NS  ns1.shopify.com
shop.example.com    NS   ns2.shopify.com
```

-
- Delegates shop.example.com management to Shopify's DNS.

Important: NS records are usually managed by your domain registrar at the root level.

4.7 SOA Record (Start of Authority)

Purpose: Stores essential information about the domain and zone

Format:

```
example.com    SOA   ns1.example.com. admin.example.com. (
                     2024010701  ; Serial
                     7200        ; Refresh (2 hours)
                     3600        ; Retry (1 hour)
                     1209600     ; Expire (2 weeks)
                     86400 )     ; Minimum TTL (1 day)
```

Fields Explained:

- **Primary Nameserver**: ns1.example.com
- **Admin Email**: admin.example.com (admin@example.com)
- **Serial Number**: Version of zone file (YYYYMMDDNN format)
- **Refresh**: How often secondary servers check for updates
- **Retry**: How long to wait before retrying failed refresh
- **Expire**: When secondary servers stop answering if can't refresh
- **Minimum TTL**: Default TTL for records without explicit TTL

Note: Usually automatically managed by DNS providers.

4.8 PTR Record (Pointer Record)

Purpose: Reverse DNS lookup - maps IP address to domain name

Format:

```
34.216.184.93.in-addr.arpa    PTR   example.com
```

Example:

```
50.1.168.192.in-addr.arpa    PTR   mail.example.com
```

Use Cases:

- Email server verification (many mail servers check PTR)
- Security logging and forensics
- Network troubleshooting

Note: PTR records are typically managed by your hosting/ISP provider, not your DNS provider.

4.9 SRV Record (Service Record)

Purpose: Specifies location of services (host and port)

Format:

```
_service._proto.name    SRV   priority weight port target
```

Example:

```
_sip._tcp.example.com    SRV   10 60 5060 sipserver.example.com
_xmpp._tcp.example.com   SRV   5  0  5222 xmpp.example.com
```

Fields:

- **Priority**: Lower = higher priority (like MX)
- **Weight**: Load distribution among same priority
- **Port**: The port number of the service
- **Target**: Hostname providing the service

Use Cases:

- VoIP/SIP configurations

- XMPP (Jabber) chat servers
- LDAP directory services
- Microsoft Active Directory

4.10 CAA Record (Certification Authority Authorization)

Purpose: Specifies which Certificate Authorities can issue SSL certificates

Format:

```
example.com    CAA  0 issue "letsencrypt.org"
example.com    CAA  0 issuewild "letsencrypt.org"
example.com    CAA  0 iodef "mailto:security@example.com"
```

Flags:

- `issue`: Authorize CA to issue certificates for domain
- `issuewild`: Authorize CA to issue wildcard certificates
- `iodef`: Where to report certificate issuance violations

Example (Multiple CAs):

```
example.com    CAA  0 issue "letsencrypt.org"
example.com    CAA  0 issue "digicert.com"
example.com    CAA  0 issuewild "letsencrypt.org"
```

Security Benefit: Prevents unauthorized CAs from issuing certificates for your domain.

4.11 ALIAS/ANAME Record (CloudFlare & AWS Specific)

Purpose: Like CNAME but can be used at root domain

Example:

```
example.com    ALIAS  lb-123456.us-east-1.elb.amazonaws.com
```

Solves the problem:

- ❌ Can't use CNAME for root domain (example.com)
- ☑ ALIAS acts like CNAME but works at root
- Provider resolves ALIAS to A record before responding

Note: Not a standard DNS record type - implementation varies by provider.

Cloudflare DNS Record Types

---

5. TTL (Time to Live) & Caching

5.1 What is TTL?

TTL (Time to Live) is a numerical value (in seconds) that tells DNS resolvers and caches **how long to store a DNS record** before requesting fresh data.

Analogy: Think of TTL like an expiration date on milk:

- Milk with expiration date 7 days from now → You can keep using it for a week
- DNS record with TTL 3600 → Resolvers can cache it for 1 hour (3600 seconds)

5.2 How TTL Works in DNS Caching

```
Step 1: First Query
User → Recursive Resolver: "What's the IP for example.com?"
Resolver → Authoritative Server: "What's the IP for example.com?"
Authoritative Server → Resolver: "IP is 93.184.216.34, TTL is 3600 seconds"

Step 2: Caching
Resolver stores:
- Record: example.com → 93.184.216.34
- Cached at: 10:00:00 AM
- Expires at: 11:00:00 AM (10:00 + 3600 seconds)

Step 3: Subsequent Queries (before expiration)
User → Resolver: "What's the IP for example.com?"
Resolver: "I have it cached! It's 93.184.216.34" (no need to query authoritative server)
```

Step 4: After TTL Expires
User → Resolver: "What's the IP for example.com?"
Resolver: "My cache expired, let me check..."
Resolver → Authoritative Server: (queries again)

5.3 TTL Values Explained

Common TTL Values:

TTL Value (seconds)	Duration	Use Case
60	1 minute	During migrations/changes
300	5 minutes	Frequently changing records
600	10 minutes	Moderately dynamic content
1800	30 minutes	Semi-stable resources
3600	1 hour	Standard web records
7200	2 hours	Balanced performance
14400	4 hours	Stable services
43200	12 hours	Very stable records

86400	24 hours	Highly stable domains
172800	48 hours	Nameserver records

5.4 TTL Trade-offs

Short TTL (60-300 seconds)
Advantages:

- ☑ Changes propagate quickly
- ☑ Easy to switch IPs during migration
- ☑ Fast rollback if issues occur
- ☑ Better for dynamic DNS

Disadvantages:

- ❌ More load on authoritative servers
- ❌ Higher DNS query costs
- ❌ Slightly slower for users (more queries)
- ❌ More bandwidth usage

When to Use:

- During planned migrations
- Before making infrastructure changes
- Testing new configurations
- Dynamic DNS scenarios

Long TTL (3600-86400 seconds)
Advantages:

- ☑ Less load on authoritative servers
- ☑ Lower DNS query costs
- ☑ Faster for users (cached responses)
- ☑ Reduced bandwidth usage
- ☑ Better performance

Disadvantages:

- ❌ Changes take longer to propagate
- ❌ Users may see old IPs during changes
- ❌ Difficult to quickly respond to issues
- ❌ Can cause downtime during migrations

When to Use:

- Stable production environments
- Established infrastructure
- Nameserver records
- Cost optimization

5.5 Best Practices for TTL

1. Pre-Migration Strategy:

```
Step 1 (48-72 hours before): Lower TTL to 300 seconds
Step 2 (Wait for old TTL to expire): Wait for previous TTL duration
Step 3 (Make the change): Update DNS records
Step 4 (After change stabilizes): Raise TTL back to normal (3600+)
```

2. Recommended TTL Values by Record Type:

```
A/AAAA records (web servers):     3600-7200 seconds (1-2 hours)
CNAME records:                    3600 seconds (1 hour)
MX records (email):               3600-14400 seconds (1-4 hours)
TXT records (verification):       3600 seconds (1 hour)
```

```
NS records (nameservers):        172800 seconds (48 hours)
SOA minimum TTL:                 300-3600 seconds
```

3. AWS Route 53 Recommendations:

- Recommended range: **60 to 172,800 seconds**
- Default for most records: **300 seconds** (5 minutes)
- Balance between responsiveness and performance

4. Different Environments:

```
Development/Testing:  300-600 seconds (frequent changes)
Staging:              1800-3600 seconds (moderate changes)
Production:           3600-86400 seconds (stable)
```

5.6 TTL in Action: Real Example

Scenario: Migrating website from Server A to Server B

Timeline:

```
Day 1 (10:00 AM):
- Current: example.com → Server A (203.0.113.10), TTL: 3600
- Action: Lower TTL to 300 seconds
- Wait: 1 hour (for old TTL to expire everywhere)

Day 1 (11:30 AM):
- Action: Change DNS record
- New: example.com → Server B (198.51.100.20), TTL: 300
- Result: Most users switch within 5 minutes

Day 2 (After verification):
- Action: Raise TTL back to 3600 seconds
- Result: Reduced query load, better performance
```

5.7 DNS Propagation Time

Common Misconception: "DNS takes 24-48 hours to propagate"

Reality:

- DNS doesn't "propagate" - there's no master pushing to slaves
- What people call "propagation" is actually **cache expiration**
- Changes are **instant** on authoritative servers
- Delays are caused by TTL-based caching

Actual Propagation Timeline:

```
Immediate:              Authoritative servers (your DNS provider)
Within seconds:         Major DNS resolvers checking frequently
Within minutes (TTL):   Most recursive resolvers
Up to old TTL duration: Worst case (if queried just before change)
24-48 hours:            ONLY if TTL was set that high + bad luck
```

How to Check DNS Propagation:

- Tools: https://www.whatsmydns.net/
- Command line: `nslookup example.com 8.8.8.8`
- Check multiple DNS servers globally

Varonis DNS TTL Guide

---

6. Practical DNS Setup Examples

6.1 Complete DNS Configuration for a Website

Scenario: Setting up DNS for `mywebsite.com` with web hosting, email, and CDN

Step 1: Choose a DNS Provider Popular options:

- Cloudflare (Free, good performance)
- AWS Route 53 ($0.50/month per hosted zone)
- Google Cloud DNS
- Your domain registrar

Step 2: Create Hosted Zone

Using AWS Route 53:

```
Copy
# Via AWS CLI
aws route53 create-hosted-zone \
  --name mywebsite.com \
  --caller-reference $(date +%s)

# Output includes:
# - Hosted Zone ID: Z1234567890ABC
# - Nameservers: ns-123.awsdns-12.com, etc.
```

Using Cloudflare (Web UI):

- Log in to Cloudflare dashboard
- Click "Add a Site"
- Enter domain: `mywebsite.com`
- Select plan (Free tier available)
- Cloudflare scans existing DNS records
- Review imported records
- Get Cloudflare nameservers

Step 3: Update Nameservers at Domain Registrar

At your domain registrar (GoDaddy, Namecheap, etc.):

```
Old Nameservers (Registrar default):
ns1.registrar.com
ns2.registrar.com

New Nameservers (Cloudflare example):
chad.ns.cloudflare.com
rain.ns.cloudflare.com

New Nameservers (Route53 example):
ns-123.awsdns-12.com
ns-456.awsdns-34.net
```

ns-789.awsdns-56.org
ns-012.awsdns-78.co.uk

Wait: 1-2 hours for nameserver changes (can take up to 48 hours)

Step 4: Add DNS Records

A. Root Domain (mywebsite.com):

Type: A
Name: @ (or leave blank for root)
Value: 203.0.113.10 (your web server IP)
TTL: 3600

B. WWW Subdomain:

Type: CNAME
Name: www
Value: mywebsite.com
TTL: 3600

C. Email (Using Google Workspace):

Type: MX
Name: @
Value: aspmx.l.google.com
Priority: 1
TTL: 3600

Type: MX
Name: @
Value: alt1.aspmx.l.google.com
Priority: 5
TTL: 3600

(Add remaining MX records similarly)

D. SPF Record (Email Authentication):

Type: TXT
Name: @
Value: "v=spf1 include:_spf.google.com ~all"
TTL: 3600

E. CDN Setup (Cloudflare or CloudFront):

```
Type: CNAME
Name: cdn
Value: d111111abcdef8.cloudfront.net
TTL: 3600
```

F. Subdomains:

```
Type: A
Name: blog
Value: 198.51.100.20
TTL: 3600

Type: A
Name: api
Value: 198.51.100.30
TTL: 3600

Type: CNAME
Name: shop
Value: mystore.shopify.com
TTL: 3600
```

6.2 AWS Route 53 Complete Setup (CLI)

Prerequisites:

- AWS account
- AWS CLI configured
- Domain registered (can be elsewhere)

Complete Script:

```
Copy
#!/bin/bash

DOMAIN="mywebsite.com"
WEB_SERVER_IP="203.0.113.10"
MAIL_SERVER_IP="203.0.113.20"

# 1. Create Hosted Zone
```

```
ZONE_ID=$(aws route53 create-hosted-zone \
    --name $DOMAIN \
    --caller-reference $(date +%s) \
    --query 'HostedZone.Id' \
    --output text | cut -d'/' -f3)

echo "Hosted Zone Created: $ZONE_ID"

# 2. Create change batch JSON
cat > /tmp/dns-records.json <<EOF
{
  "Changes": [
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "$DOMAIN",
        "Type": "A",
        "TTL": 300,
        "ResourceRecords": [{"Value": "$WEB_SERVER_IP"}]
      }
    },
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "www.$DOMAIN",
        "Type": "CNAME",
        "TTL": 300,
        "ResourceRecords": [{"Value": "$DOMAIN"}]
      }
    },
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "$DOMAIN",
        "Type": "MX",
        "TTL": 3600,
        "ResourceRecords": [
          {"Value": "1 aspmx.l.google.com"},
          {"Value": "5 alt1.aspmx.l.google.com"}
        ]
      }
    },
    {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "$DOMAIN",
```

```
        "Type": "TXT",
        "TTL": 3600,
        "ResourceRecords": [
          {"Value": "\"v=spf1 include:_spf.google.com ~all\""}
        ]
      }
    ]
}
EOF

# 3. Apply DNS records
aws route53 change-resource-record-sets \
    --hosted-zone-id $ZONE_ID \
    --change-batch file:///tmp/dns-records.json

# 4. Get nameservers
echo "Update your domain registrar with these nameservers:"
aws route53 get-hosted-zone \
    --id $ZONE_ID \
    --query 'DelegationSet.NameServers' \
    --output table

# 5. Clean up
rm /tmp/dns-records.json
```

6.3 Cloudflare DNS Setup (API)

Using Cloudflare API:

Copy

```
#!/bin/bash

# Configuration
CF_API_KEY="your_api_key"
CF_EMAIL="your@email.com"
DOMAIN="mywebsite.com"
ZONE_ID="your_zone_id"  # Get from Cloudflare dashboard
WEB_IP="203.0.113.10"

# Function to create DNS record
create_record() {
    local type=$1
    local name=$2
    local content=$3
```

```
    local ttl=$4
    local priority=${5:-null}

    curl -X POST "https://api.cloudflare.com/client/v4/zones/$ZONE_ID/dns_records" \
         -H "X-Auth-Email: $CF_EMAIL" \
         -H "X-Auth-Key: $CF_API_KEY" \
         -H "Content-Type: application/json" \
         --data "{
           \"type\": \"$type\",
           \"name\": \"$name\",
           \"content\": \"$content\",
           \"ttl\": $ttl,
           \"priority\": $priority,
           \"proxied\": false
         }"
}

# Create A record for root domain
create_record "A" "$DOMAIN" "$WEB_IP" 3600

# Create CNAME for www
create_record "CNAME" "www" "$DOMAIN" 3600

# Create MX records
create_record "MX" "$DOMAIN" "aspmx.l.google.com" 3600 1
create_record "MX" "$DOMAIN" "alt1.aspmx.l.google.com" 3600 5

# Create TXT record for SPF
create_record "TXT" "$DOMAIN" "v=spf1 include:_spf.google.com ~all" 3600

echo "DNS records created successfully!"
```

6.4 Private Hosted Zone Setup (AWS VPC)

Scenario: Internal service discovery for microservices in AWS VPC

Step 1: Create Private Hosted Zone

Copy

```
aws route53 create-hosted-zone \
  --name internal.company.com \
  --vpc VPCRegion=us-east-1,VPCId=vpc-1a2b3c4d \
  --caller-reference $(date +%s) \
  --hosted-zone-config PrivateZone=true
```

Step 2: Add Internal DNS Records

Copy

```
# Database server
{
  "Name": "db-master.internal.company.com",
  "Type": "A",
  "TTL": 300,
  "ResourceRecords": [{"Value": "10.0.1.50"}]
}

# API Gateway
{
  "Name": "api.internal.company.com",
  "Type": "A",
  "TTL": 300,
  "ResourceRecords": [{"Value": "10.0.2.100"}]
}

# Microservices
{
  "Name": "user-service.internal.company.com",
  "Type": "A",
  "TTL": 60,
  "ResourceRecords": [{"Value": "10.0.3.10"}]
}

{
  "Name": "payment-service.internal.company.com",
  "Type": "A",
  "TTL": 60,
  "ResourceRecords": [{"Value": "10.0.3.20"}]
}
```

Step 3: Associate Additional VPCs

Copy

```
aws route53 associate-vpc-with-hosted-zone \
    --hosted-zone-id Z1234567890ABC \
    --vpc VPCRegion=us-west-2,VPCId=vpc-5e6f7g8h
```

6.5 DNS Verification Commands

Check DNS Records:

Copy

```
# Query specific DNS server
nslookup mywebsite.com 8.8.8.8

# Detailed DNS query
dig mywebsite.com @8.8.8.8

# Show all record types
dig mywebsite.com ANY

# Check MX records
dig mywebsite.com MX

# Check nameservers
dig mywebsite.com NS

# Trace full DNS resolution path
dig +trace mywebsite.com

# Check TTL
dig mywebsite.com | grep "IN"
```

Windows Commands:

Copy

```
# Basic lookup
nslookup mywebsite.com

# Query specific server
nslookup mywebsite.com 1.1.1.1

# Check MX records
nslookup -type=MX mywebsite.com

# Clear DNS cache
ipconfig /flushdns

# Display DNS cache
ipconfig /displaydns
```

Test DNS Propagation:

Copy

```
# Check multiple DNS servers
for server in 8.8.8.8 1.1.1.1 208.67.222.222 9.9.9.9; do
    echo "Checking $server:"
```

```
    dig @$server mywebsite.com +short
done
```

Summary Checklist: DNS Setup

☑ **Planning Phase:**

- Choose DNS provider
- Plan record structure
- Document required records
- Identify dependencies

☑ **Configuration Phase:**

- Create hosted zone
- Configure nameservers at registrar
- Add A/AAAA records
- Configure CNAME records
- Set up MX records (if using email)

- Add TXT records (SPF, DKIM, DMARC)
- Configure CAA records
- Set appropriate TTL values

☑ **Verification Phase:**

- Test DNS resolution
- Verify from multiple locations
- Check propagation status
- Test email delivery
- Validate DNSSEC (if enabled)

☑ **Security Phase:**

- Enable DNSSEC
- Implement CAA records
- Configure email authentication
- Set up access controls
- Enable logging

☑ **Maintenance Phase:**

- Monitor DNS performance
- Regular security audits
- Update documentation
- Test disaster recovery
- Review and optimize TTL

These comprehensive notes cover everything you need to understand and implement DNS, from the theoretical foundations to practical setup examples. The key is

understanding the resolution process, choosing appropriate record types, setting correct TTL values, and following security best practices.