

DNS A Visual Guide

Understand Name Resolution from First Lookup to Real World Operations

Reiner Braun

Sample Book — Chapters 1 to 2

Introduction

This book is designed for readers who are taking their first steps with DNS, including systems engineers who want to build the right foundation from the start.

Rather than asking you to memorize more than you need, the book focuses on the essential concepts that will support your future study and day-to-day work. More advanced topics are included only when they are likely to become useful to many readers, and even then they are introduced only far enough to explain their purpose, context, and overall role.

Processes, relationships, and structures that can be difficult to picture from text alone are supported by diagrams. By moving between explanations, visuals, and examples, you can build a clearer and more practical understanding of the subject.

Product names, service names, logos, icons, and other identifying marks mentioned in this book are trademarks or registered trademarks of their respective owners. This book is not an official publication, certified training resource, or endorsed learning material of any company or organization.

How to Use This Book

Read the chapters in order when you are new to DNS. Each chapter assumes the mental model and vocabulary established earlier, so following the sequence will help you build the necessary background knowledge step by step.

Before reading a chapter, review its **What You'll Be Able to Do** section. These outcomes identify the practical knowledge you should expect to gain and help you distinguish the chapter's essential ideas from supporting detail.

Use the diagrams to clarify relationships, processes, boundaries, and structures that are difficult to hold in mind from prose alone. The diagrams support the explanation; they do not replace the surrounding text.

Run the commands and work through the hands-on examples yourself whenever possible. Use a lab, test environment, or documentation-only address and domain when a command could affect a real system.

At the end of each chapter, use the **Chapter Summary** and **Review Questions** to confirm that you understand and can recall the fundamentals before moving on.

Advanced-topic overviews do not need to be memorized at this stage. Treat them as signposts that will help you recognize subjects, tools, and design choices you may need to study in greater depth later.

Table of Contents

Introduction

How to Use This Book

Part I — Core Concepts and Safe Day-to-Day Operation

Chapter 1 — DNS in the System: The Big Picture

- The main idea

- Names, DNS data, addresses, and connections are different things

- Where DNS fits in a normal application request

- Why DNS is distributed

- A high-level lookup for `app.example.com`

- DNS is more than address lookup

- What DNS does not do

 - DNS is not an HTTP redirect

 - DNS is not a firewall rule

 - DNS is not a certificate

 - DNS is not a health guarantee

- A practical way to classify a connection failure

- The three conclusions to carry forward

Chapter 2 — DNS Names, Hierarchy, Domains, and Zones

- Read a DNS name from right to left

- The root and fully qualified names

- The levels of the hierarchy

- Domain and zone are not synonyms

- The zone apex

- Zone cuts and delegation boundaries

- Absolute and relative names

- Breaking down `api.prod.example.com.`

- Hostnames are a practical subset of DNS names

- Naming conventions that reduce operational risk

- Existing zone or delegated subdomain?

 - Add records to the existing parent zone

 - Delegate `dev.example.com` as a child zone

- Registration and DNS hosting are different services

- The model to retain

Part I — Core Concepts and Safe Day-to-Day Operation

DNS is the system that lets applications use stable names while the systems behind those names change. A systems engineer needs more than the idea that DNS “turns names into IP addresses.” DNS is a distributed database, a delegation system, a cache, and a critical dependency that sits in front of most network connections.

The first half of this guide builds the working model needed before deeper troubleshooting, security, cloud, and container topics. It explains:

- how a name is resolved from a client to an authoritative source;
- how domains, zones, and delegations divide responsibility;
- how DNS data is represented and which record types are used most often;
- how authoritative zones are operated;
- why changes remain visible at different times in different places; and
- how operating systems and applications decide which resolver path to use.

The examples use names under `example.com` and IP address ranges reserved for documentation. They describe realistic patterns without referring to a specific production environment.

Chapter 1 — DNS in the System: The Big Picture

What You'll Be Able to Do

After completing this chapter, you will be able to:

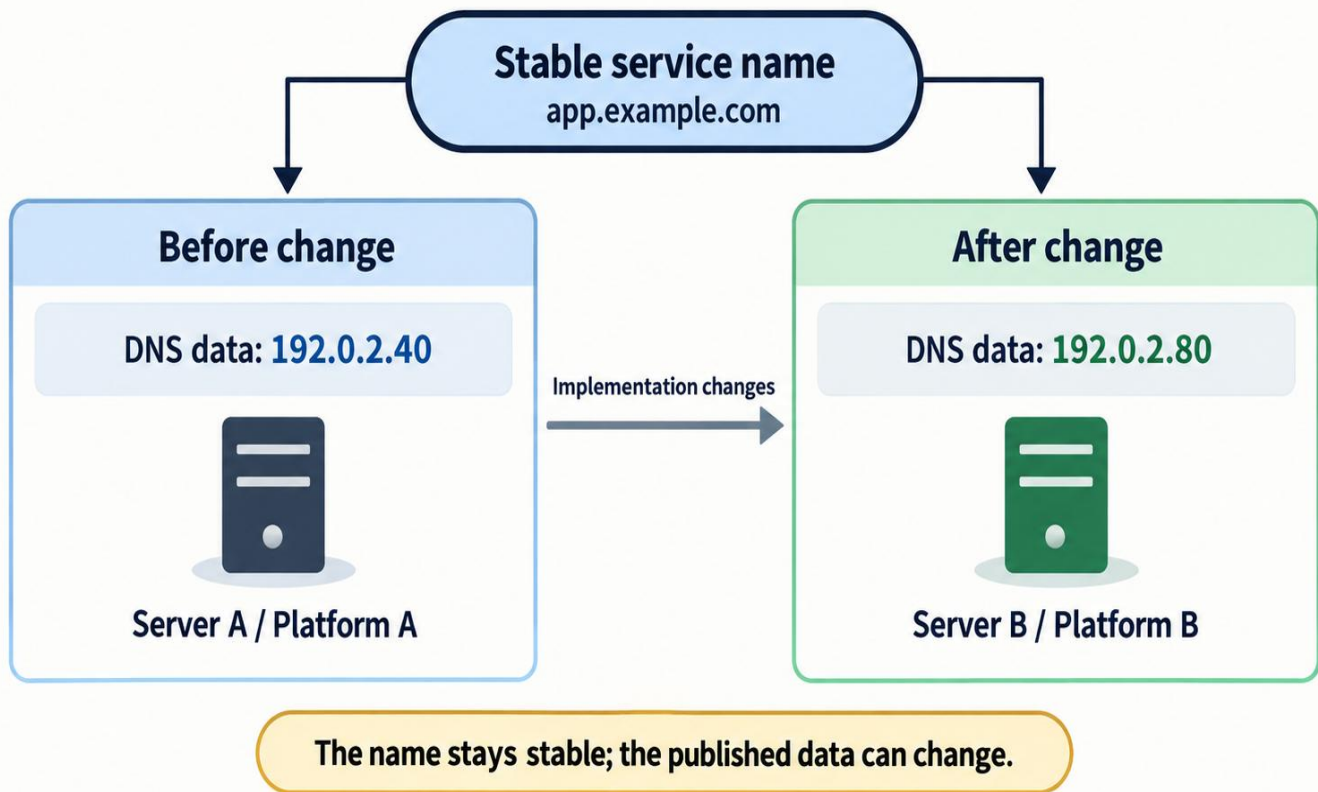
- Explain DNS as a distributed naming system rather than a single server or a simple address converter.
- Trace the major stages from an application name lookup to a working application exchange.
- Separate DNS success from routing, firewall, transport, TLS, and application health.

The main idea

DNS, the Domain Name System, is a distributed naming system that helps clients find services by name. Service discovery is the process of locating the network endpoint that provides a needed service. DNS allows an application to refer to `app.example.com` even when the server address, hosting platform, or network design changes.

Chapter 1: DNS in the System: The Big Picture

1-1 Stable Name, Changing Implementation



001

Figure 1-1: A stable service name with changing implementation data

A **cache** is a temporary store of data that can be reused to avoid repeating a lookup. DNS caching improves speed and resilience, but it also allows a previous answer to remain in use for a limited time after authoritative data changes.

That statement contains the central idea for this entire guide: **a name is a stable point of reference, while the data associated with the name can change.**

A user may think, "I am connecting to `app.example.com`." The computer must perform several distinct tasks:

1. obtain DNS data for `app.example.com`;
2. choose an address from the result;
3. open a network connection to that address;
4. negotiate any required security protocol; and
5. exchange application data.

DNS is mainly involved in the first task. It does not complete the others.

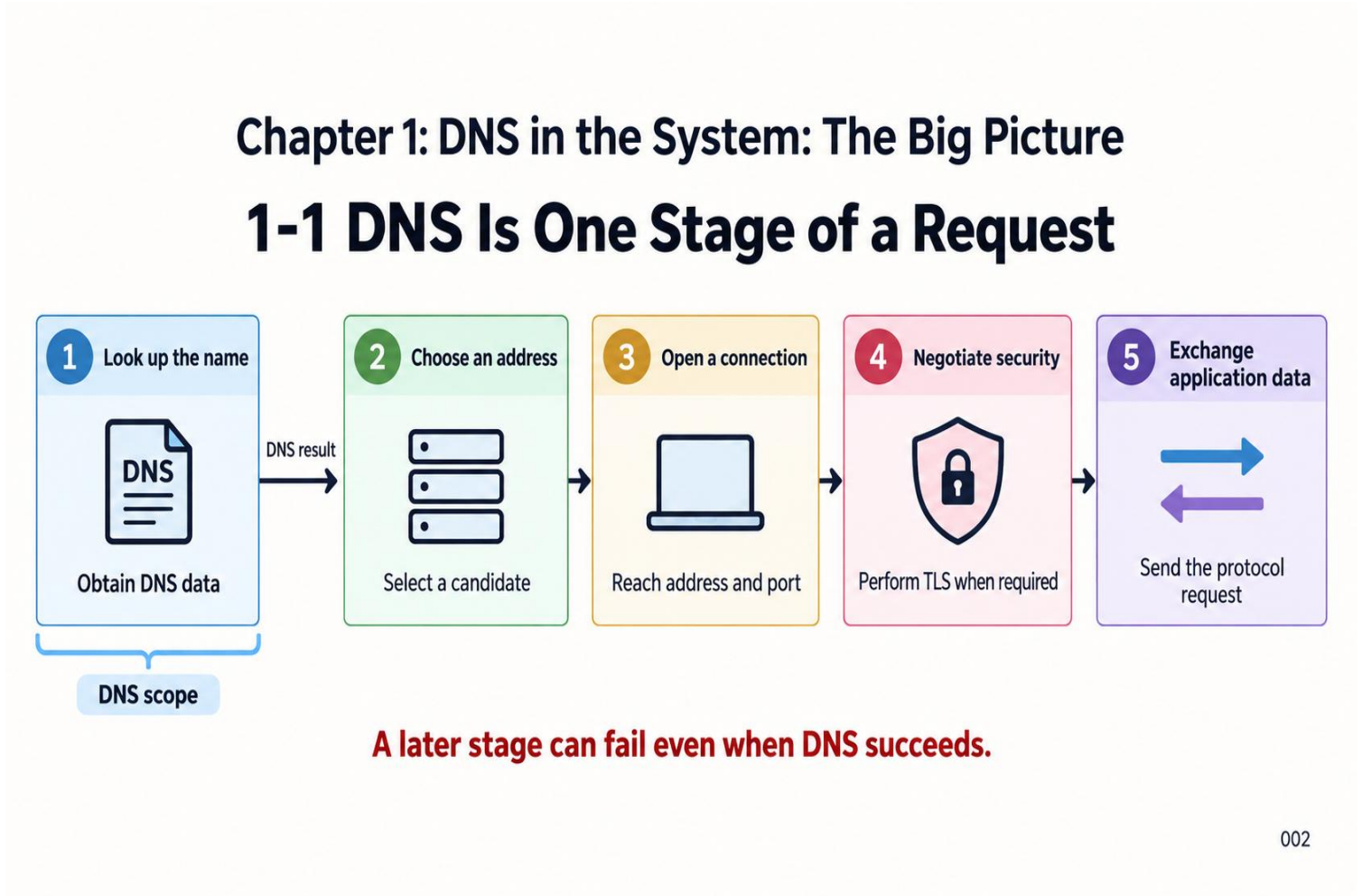


Figure 1-2: The five tasks from name lookup to application exchange

Names, DNS data, addresses, and connections are different things

A **DNS name** is a sequence of labels such as `app`, `example`, and `com`, written as `app.example.com`.

A **resource record** is one typed item of DNS data associated with a name. For example, an address record can associate `app.example.com` with an IP address.

An **Internet Protocol address**, usually shortened to **IP address**, is a numeric address used to deliver packets to a network interface. A **packet** is a unit of data carried across a packet-switched network. IPv4 addresses look like `192.0.2.40`; IPv6 addresses look like `2001:db8::40`.

A **network connection** is an active communication path between endpoints, normally identified by addresses, transport protocols, and port numbers. A **transport protocol** carries application data between endpoints, and a **port number** identifies a particular service endpoint on an address. The fact that DNS returned an address does not mean that a connection to that address will succeed.

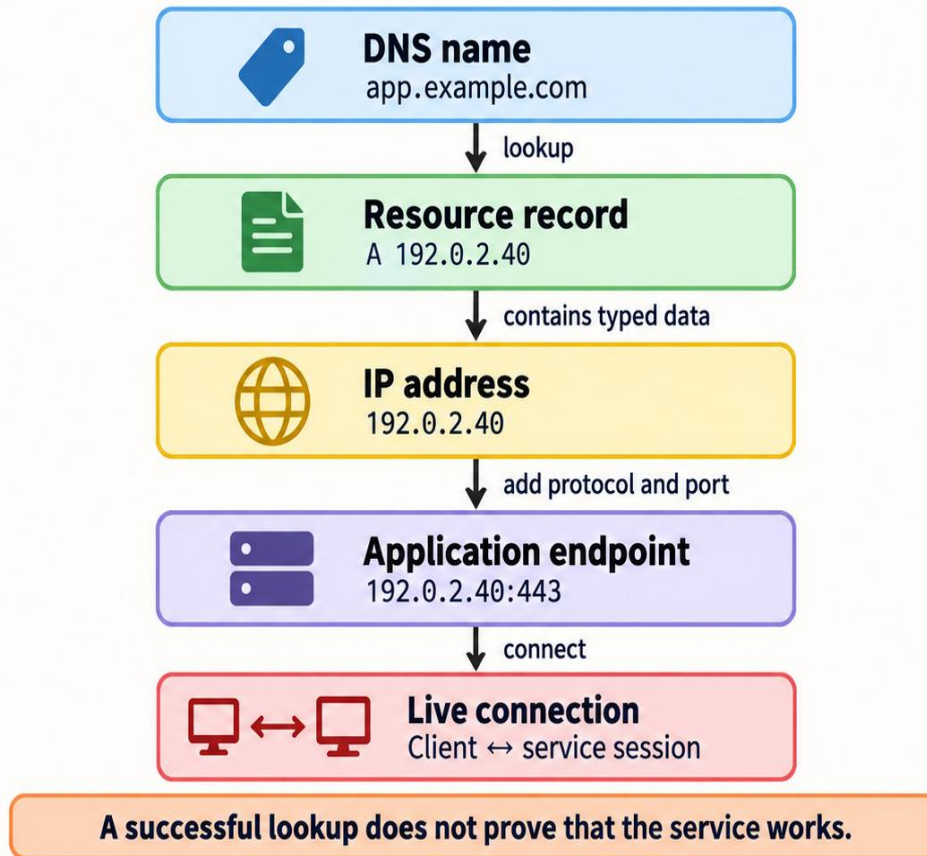
These concepts are related, but they are not interchangeable:

Item	Example	What it represents
DNS name	<code>app.example.com</code>	A name that can be looked up
DNS record	<code>app.example.com. 300 IN A 192.0.2.40</code>	Typed data published for that name
IP address	<code>192.0.2.40</code>	A network-layer destination
Application endpoint	<code>192.0.2.40:443</code>	An address plus a service port
Application connection	A client connected to the service on port 443	A live communication session

A successful lookup proves only that DNS returned a usable answer. It does not prove that routing works, a firewall permits traffic, the service is listening, the application is healthy, or a security certificate is valid.

Chapter 1: DNS in the System: The Big Picture

1-2 Name, Record, Address, Endpoint, Connection



003

Figure 1-3: Name, resource record, IP address, endpoint, and live connection

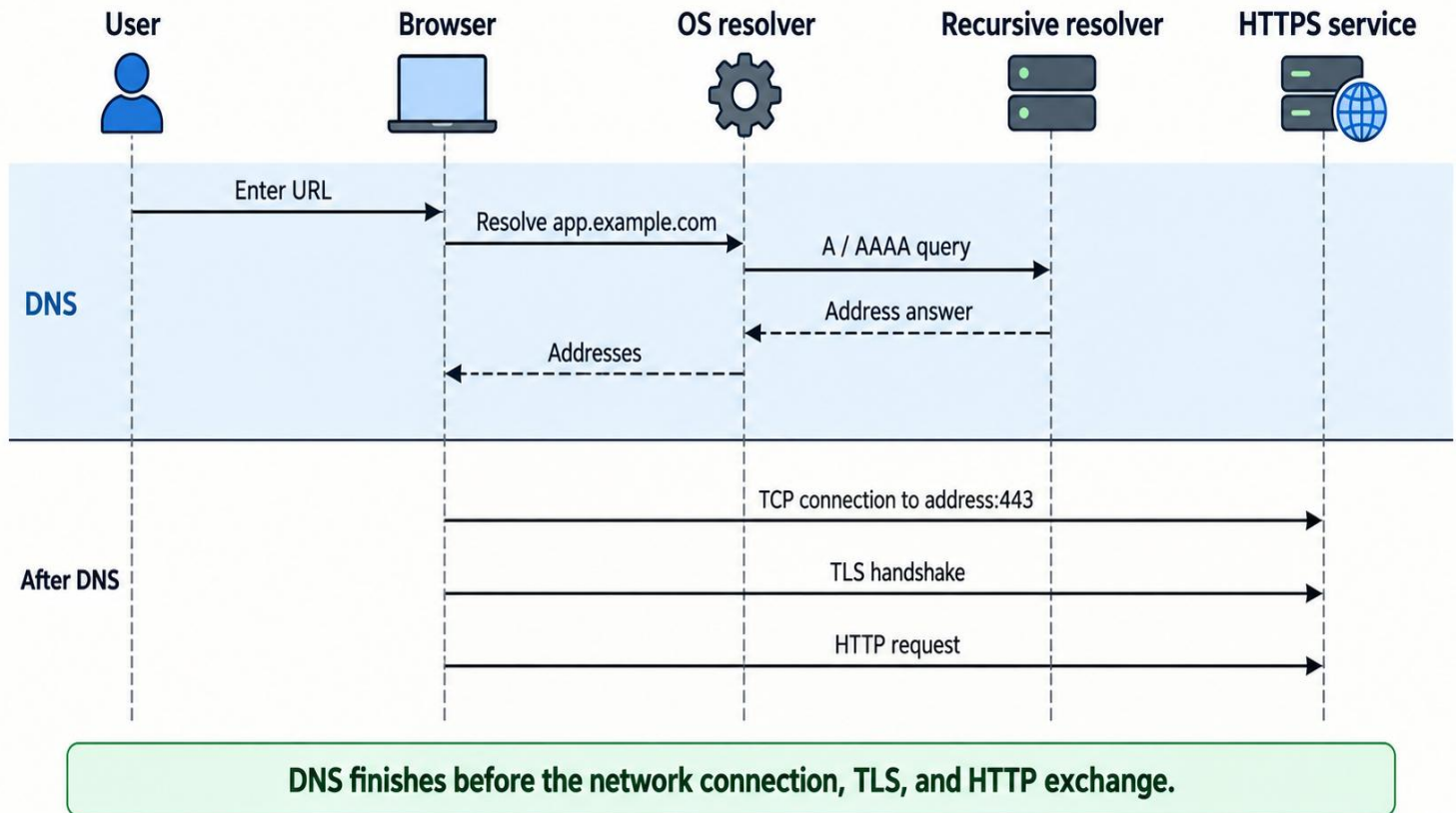
Where DNS fits in a normal application request

Consider a user opening `https://app.example.com/` in a browser. A **Uniform Resource Locator (URL)** identifies the scheme, destination name, and optional path of a network resource. **HTTPS** is the secure form of the Hypertext Transfer Protocol used by web clients and servers.

A simplified sequence is:

```
User enters a URL
|
v
Browser asks for app.example.com
|
v
Operating system or application uses a DNS resolver
|
v
DNS answer supplies one or more addresses
|
v
Browser connects to an address on the HTTPS port
|
v
Security negotiation and HTTP request occur
```

1-3 DNS in an HTTPS Request



004

Figure 1-4: Where DNS fits in an HTTPS request

DNS happens before the browser can contact the destination by address. The same dependency exists for many other activities:

- a remote administration tool connecting to a server name;
- an application connecting to a database name;
- a mail server locating the server responsible for a domain;
- an internal service finding another service by name; and
- a monitoring system checking a named endpoint.

When several unrelated applications fail at once, DNS may be the common dependency. Conversely, when DNS works but one application fails, the failure may be after name resolution.

Why DNS is distributed

A single global database operated by one organization would be difficult to scale, secure, and manage. DNS instead divides responsibility across a hierarchy.

An everyday analogy is a large organization's directory:

- the corporate directory tells you which department owns a branch;
- the department directory tells you which team owns a sub-branch;
- the team publishes the final contact information; and
- local users may keep recently used entries in a cache.

No single directory needs to hold every detail. Each level knows either the answer or where to look next.

DNS follows the same broad pattern. The top of the hierarchy identifies the servers responsible for top-level domains such as `.com`. The `.com` servers identify the servers responsible for `example.com`. The `example.com` servers publish records such as the address for `app.example.com`.

This hierarchy distributes both data and authority. Different organizations can manage different parts of the namespace without copying the complete global database into one place.

Chapter 1: DNS in the System: The Big Picture

1-4 Distributed DNS Authority

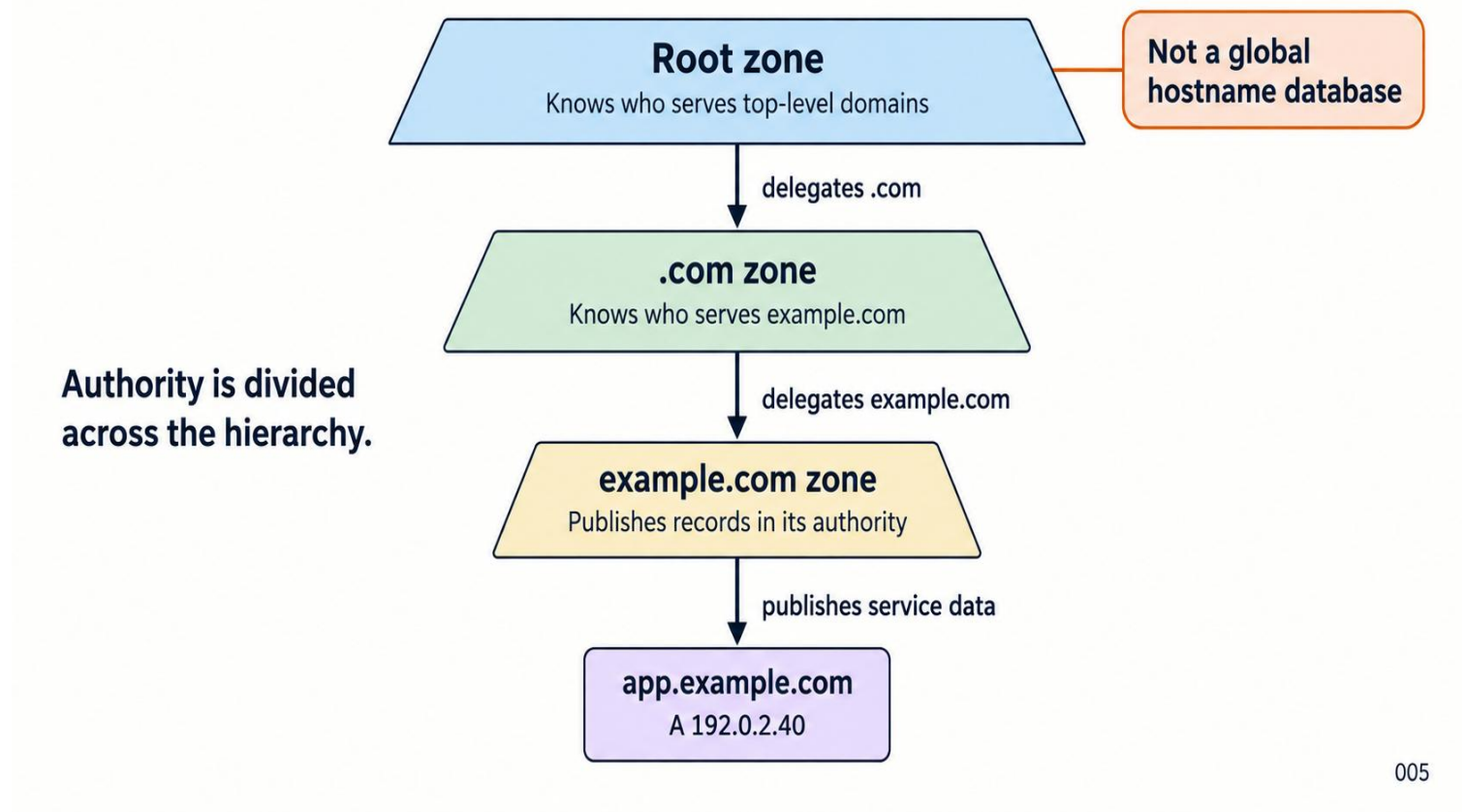


Figure 1-5: Distributed hierarchy and delegated authority

A high-level lookup for `app.example.com`

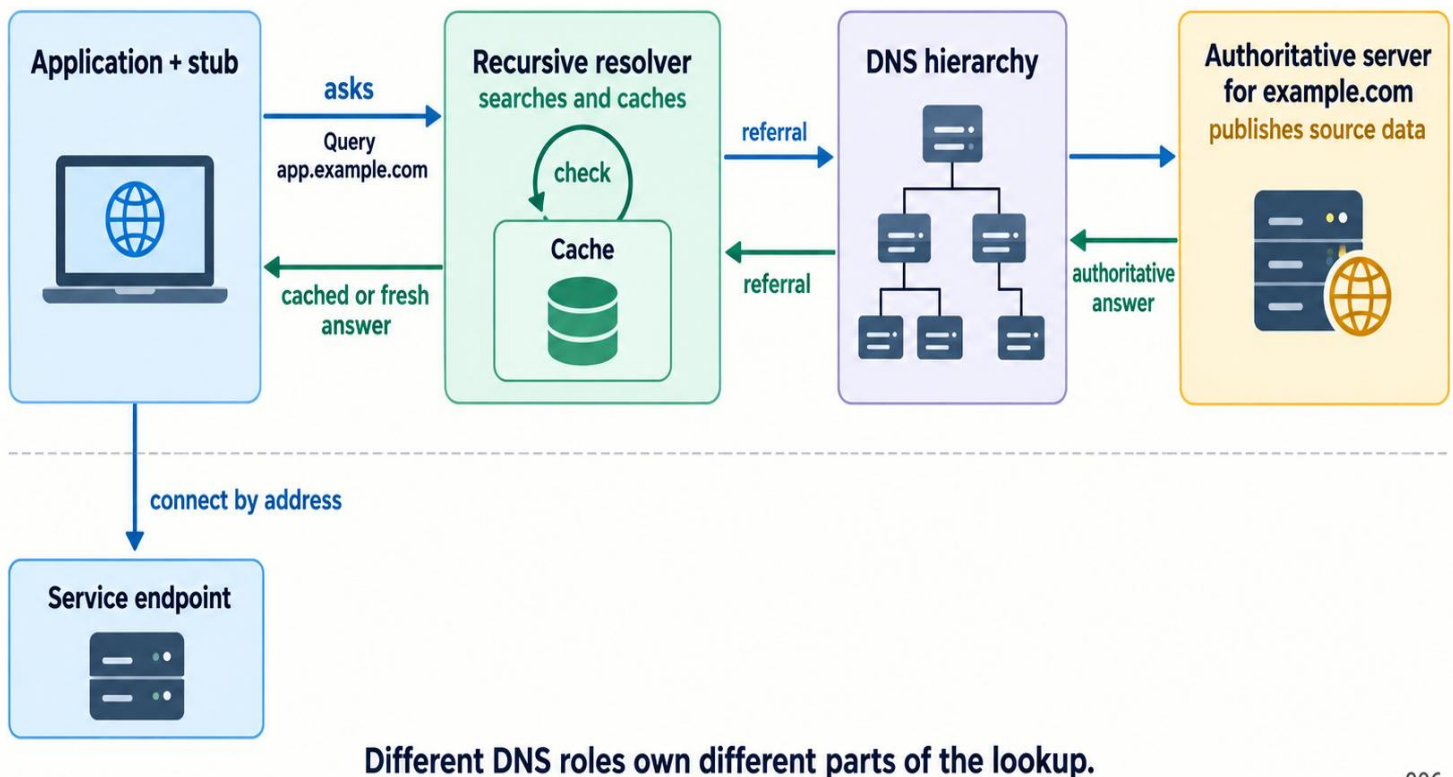
The exact path depends on caches and local configuration, but a first lookup commonly looks like this:

1. The application asks a local resolver component for `app.example.com`.
2. The request reaches a **recursive resolver**, which is a server that finds the answer on behalf of the client.
3. If the recursive resolver has a valid cached answer, it returns that answer immediately.
4. If it does not, it follows the DNS hierarchy to locate the servers authoritative for `example.com`.
5. An **authoritative server**, which publishes the official DNS data for a zone, returns the record for `app.example.com`.
6. The recursive resolver caches the answer for a limited period and returns it to the client.
7. The application uses the returned address to begin a separate network connection.

Later chapters examine each role precisely. For now, retain the division of responsibility:

The client asks, the recursive resolver searches and caches, and the authoritative server publishes the source data.

1-5 Lookup Roles for app.example.com



006

Figure 1-6: High-level lookup roles for app.example.com

DNS is more than address lookup

Address records are common, but DNS can publish several kinds of information. It can identify:

- the servers authoritative for a zone;
- the mail servers for a domain;
- the location and port of a named service;
- certificate-authority policy;
- text used by another protocol; and
- aliases from one DNS name to another.

This is why “DNS converts a hostname to an IP address” is useful as a first sentence but incomplete as a working definition. **DNS is a typed, hierarchical data system used for naming and service discovery.**

What DNS does not do

Several technologies are often confused with DNS because they appear in the same user-visible workflow.

DNS is not an HTTP redirect

An **HTTP redirect** is a web-server response that tells a browser to request a different URL. DNS can make two names lead to the same server, but it does not change the path, scheme, or URL displayed by the browser.

For example, making `old.example.com` and `new.example.com` resolve to the same address does not by itself redirect:

`https://old.example.com/reports`

to:

`https://new.example.com/archive/reports`

That behavior must be implemented at the HTTP layer after a connection has been established.

DNS is not a firewall rule

A **firewall** is a control that permits or blocks network traffic according to policy. DNS may return the correct address while a firewall blocks the subsequent connection.

DNS is not a certificate

A **digital certificate** is a signed statement used to associate a public key with an identity, commonly for Transport Layer Security. **Transport Layer Security (TLS)** is the protocol that protects HTTPS and many other application connections.

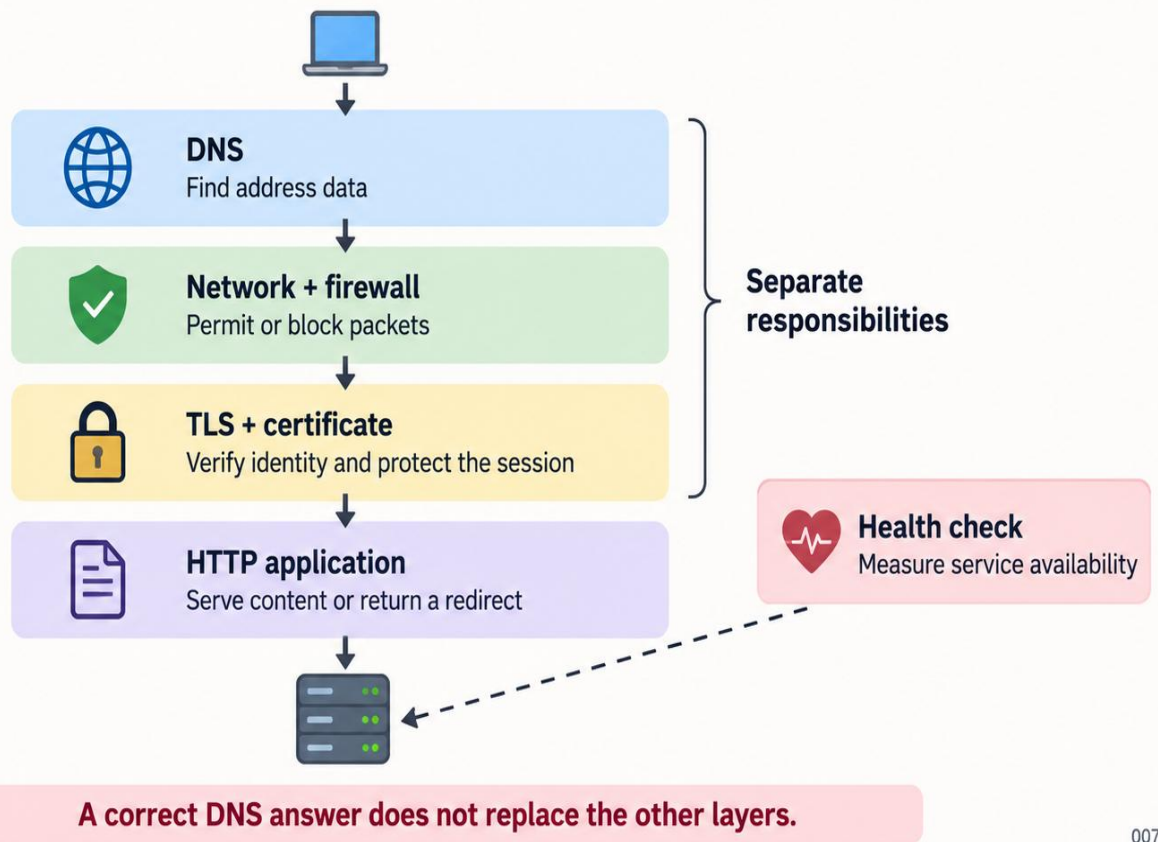
DNS naming and certificate validation are related, but separate. A client can resolve `app.example.com` correctly and still reject the connection because the certificate is expired or does not cover that name.

DNS is not a health guarantee

A DNS answer can remain cached while the destination service is unavailable. Multiple address records can also include an address whose application is unhealthy. DNS-based traffic policies can add health checks in some managed platforms, but a normal record alone does not test the application.

Chapter 1: DNS in the System: The Big Picture

1-7 DNS and Neighboring Responsibilities



007

Figure 1-7: DNS versus redirect, firewall, certificate, and health check responsibilities

A practical way to classify a connection failure

When an application reports “cannot connect,” first separate the stages conceptually:

1. **Name stage:** Did the application obtain an address or other required DNS data?
2. **Network stage:** Can packets reach the destination?
3. **Transport stage:** Can a connection be opened to the required port?
4. **Security stage:** Does protocol negotiation and certificate validation succeed?
5. **Application stage:** Does the service return the expected response?

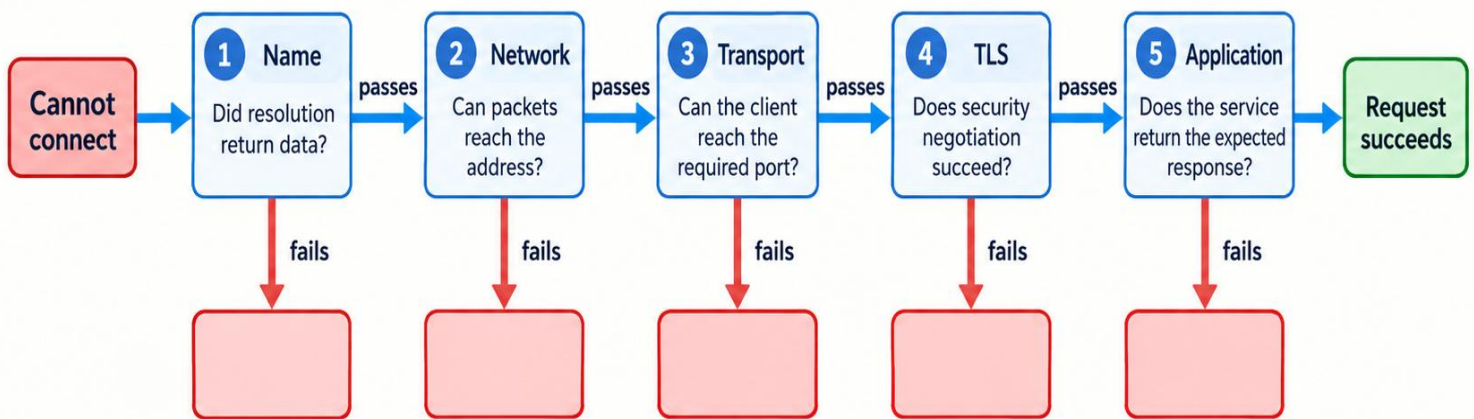
This is not yet a full troubleshooting procedure. It is a way to avoid treating every connection problem as a DNS problem.

For example:

- “Name or service not known” strongly suggests a name-resolution problem.
- “Connection timed out” may indicate routing, firewall, service, or DNS transport issues; it does not prove that the name is absent.
- “Connection refused” usually means an address was reached but nothing accepted the requested connection.
- “Certificate name mismatch” happens after name resolution and connection establishment.

Chapter 1: DNS in the System: The Big Picture

1-8 Classifying a Connection Failure



Find the first failing stage.

008

Figure 1-8: Connection-failure stages from name to application

The three conclusions to carry forward

DNS is distributed. Responsibility is divided among many independently operated servers.

DNS is a dependency, not the complete application path. A lookup result and a working service are different outcomes.

DNS data is cached. A published change does not become visible to every client at the same instant.

Those conclusions explain much of the behavior studied in the remaining chapters.

Chapter Summary

- DNS gives applications stable names while the data and infrastructure behind those names can change.
- A lookup depends on several distributed systems and occurs before an application can connect to an endpoint.
- A correct DNS answer confirms name resolution only; later connection and application stages can still fail.

Review Questions

1. What remains stable when the server or address behind a service changes?
2. Which stages occur after DNS returns an address?

3. Why does a successful lookup not prove that an HTTPS service is healthy?
4. Why is DNS divided among multiple administrative authorities?

Chapter 2 — DNS Names, Hierarchy, Domains, and Zones

What You'll Be Able to Do

After completing this chapter, you will be able to:

- Break a fully qualified domain name into labels and read its hierarchy correctly.
- Distinguish a domain, a zone, a zone apex, and a delegation boundary.
- Decide whether a team needs records in an existing zone or authority over a delegated child zone.

Read a DNS name from right to left

A **DNS name** identifies a position in a hierarchy, and the hierarchy becomes broader as you move from left to right. In `api.prod.example.com`, `api` is the most specific label and `com` is the broadest visible label.

A **label** is one component of a DNS name between dots. The name below contains four visible labels:

```
api.prod.example.com
|   |       |   |
api prod example com
```

The hierarchy actually continues one level higher to the DNS root.

The root and fully qualified names

The **DNS root** is the top of the DNS namespace. It is represented in textual DNS form by an empty label after a final dot.

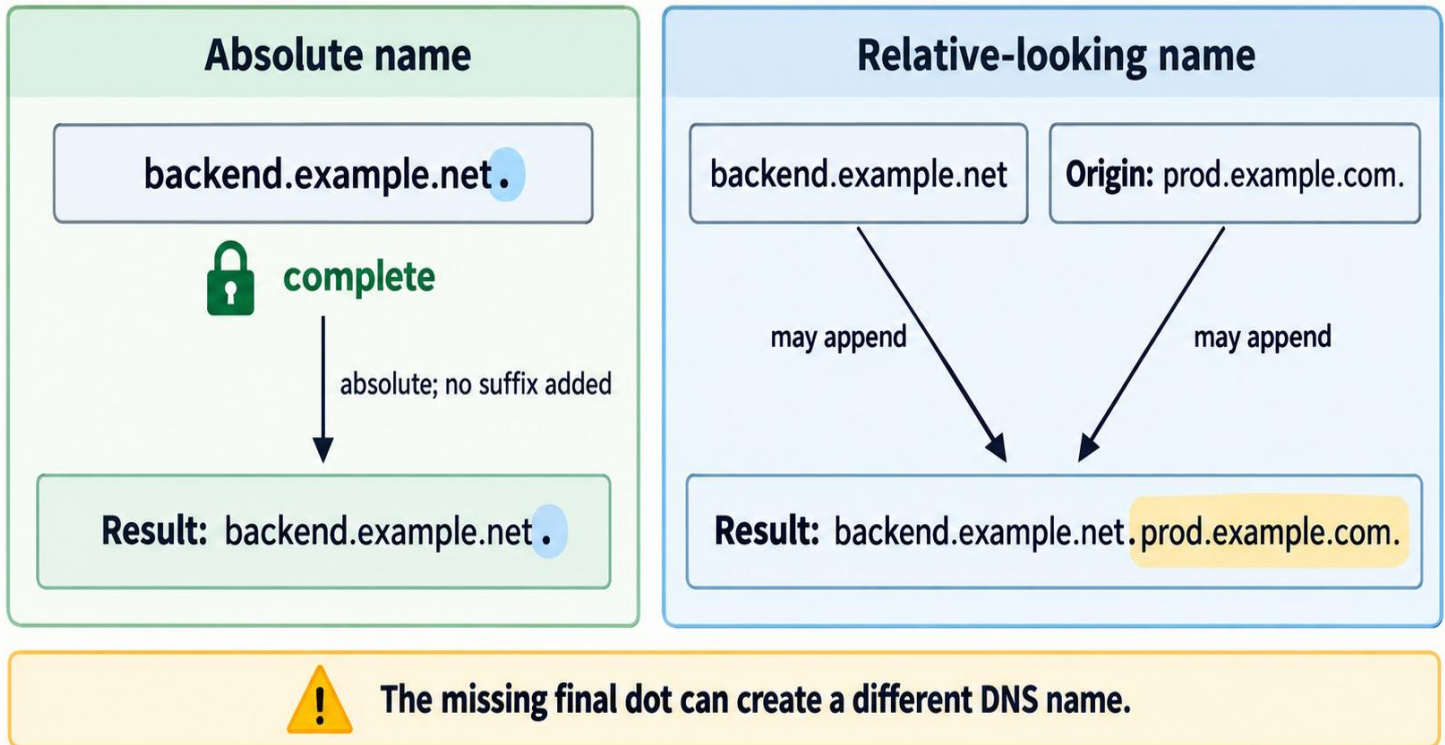
Therefore, the complete form of the example name is:

```
api.prod.example.com.
```

A **fully qualified domain name (FQDN)** is a complete DNS name whose position is unambiguous all the way to the root. In DNS configuration and diagnostic output, the trailing dot explicitly marks a name as fully qualified.

Users and applications normally omit the final dot, so `api.prod.example.com` is commonly treated as the same intended name. In configuration files, however, omitting the dot may cause software to treat the name as relative and append another domain. That distinction can create serious mistakes.

2-2 The Trailing Dot Completes an FQDN



010

Figure 2-2: Trailing dot and FQDN completion

The levels of the hierarchy

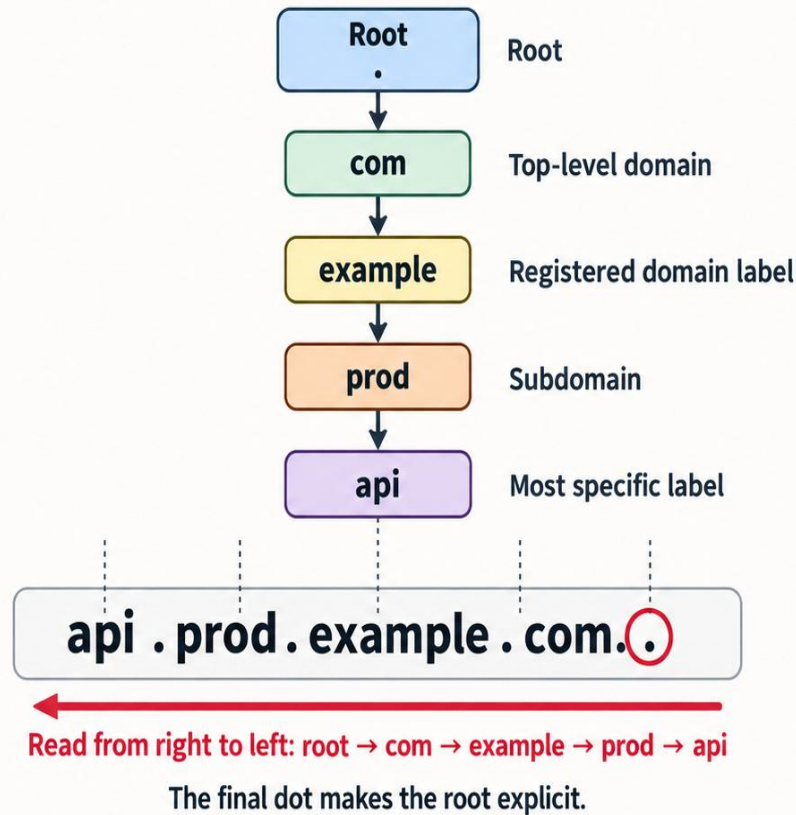
For `api.prod.example.com.`:

- `.` is the root;
- `com.` is a **top-level domain (TLD)**, meaning a domain immediately below the root;
- `example.com.` is a registered domain in this example;
- `prod.example.com.` is a subdomain; and
- `api.prod.example.com.` is a more specific name, perhaps used for a host or service.

A **subdomain** is any domain below another domain in the hierarchy. `prod.example.com` is a subdomain of `example.com`, and `api.prod.example.com` is also within the `example.com` domain.

The visible labels do not state how the organizations involved have divided operational control. For that, DNS uses zones and delegation.

2-1-2-3 Reading a DNS Name as a Hierarchy



009

Figure 2-1: Reading a DNS name from the root toward the most specific label

Domain and zone are not synonyms

A **domain** is a branch of the DNS namespace: one name and all names below it.

A **zone** is the portion of that namespace whose authoritative data is managed together by a particular administrative authority.

This difference matters because a domain can contain a delegated child zone.

Assume the `example.com` organization delegates `prod.example.com` to a platform team. Then:

```
example.com domain
|
+-- www.example.com      managed in the example.com zone
+-- mail.example.com     managed in the example.com zone
+-- prod.example.com     delegated to another zone
    |
    +-- api.prod.example.com managed in the prod.example.com zone
    +-- db.prod.example.com  managed in the prod.example.com zone
```

The domain `example.com` still includes every name below it conceptually. The `example.com` zone, however, stops carrying ordinary authoritative data below the delegated `prod.example.com` boundary.

A **subdomain does not automatically become a zone**. It becomes a separate zone only when it is delegated or otherwise served as a distinct authoritative data set.

The zone apex

The **zone apex** is the top name of a zone. If the zone is `example.com`, then `example.com` is its apex. If `prod.example.com` is delegated as a separate zone, `prod.example.com` is the apex of that child zone.

The apex has structural records required for zone operation. In particular, it has:

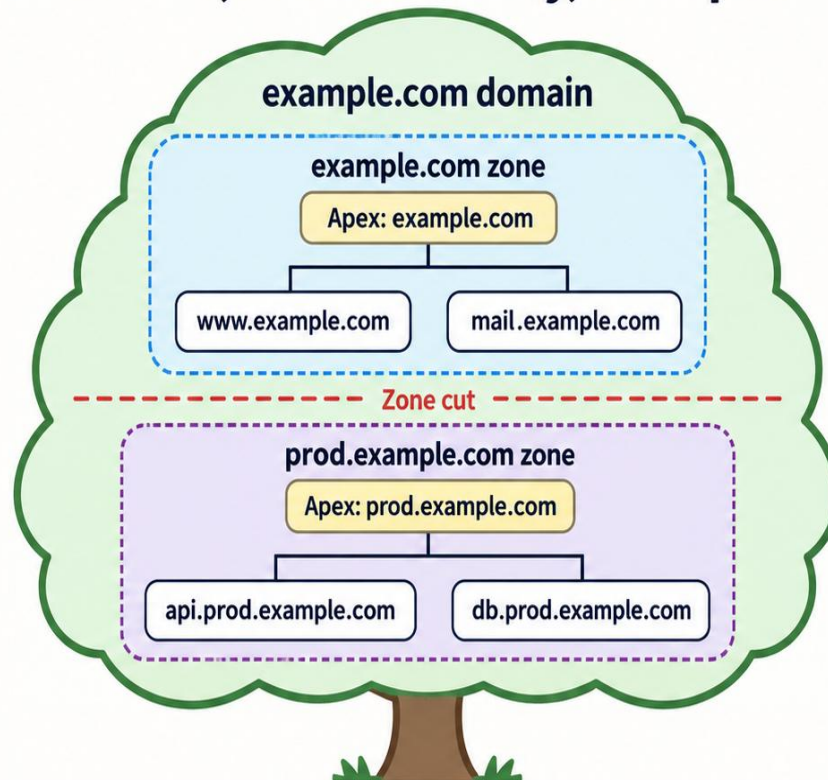
- an `SOA` record describing core zone-management information; and
- `NS` records identifying the zone's authoritative name servers.

These record types are introduced in Chapter 5 and examined operationally in Chapter 6.

The apex is not the same as `www`. A website may be available at both `example.com` and `www.example.com`, but those are two distinct DNS names.

Chapter 2: DNS Names, Hierarchy, Domains, and Zones

2-4-2-5 Domain Branch, Zone Boundary, and Apex



A domain includes descendants; zones divide administrative authority.

011

Figure 2-3: Domain branch, zone boundary, and zone apex

Zone cuts and delegation boundaries

A **zone cut** is the boundary at which authority for a child name is delegated to another zone.

In the earlier example, the zone cut is at `prod.example.com`. The parent zone does not normally publish the application records below that boundary. Instead, it publishes information that tells resolvers which servers are authoritative for the child.

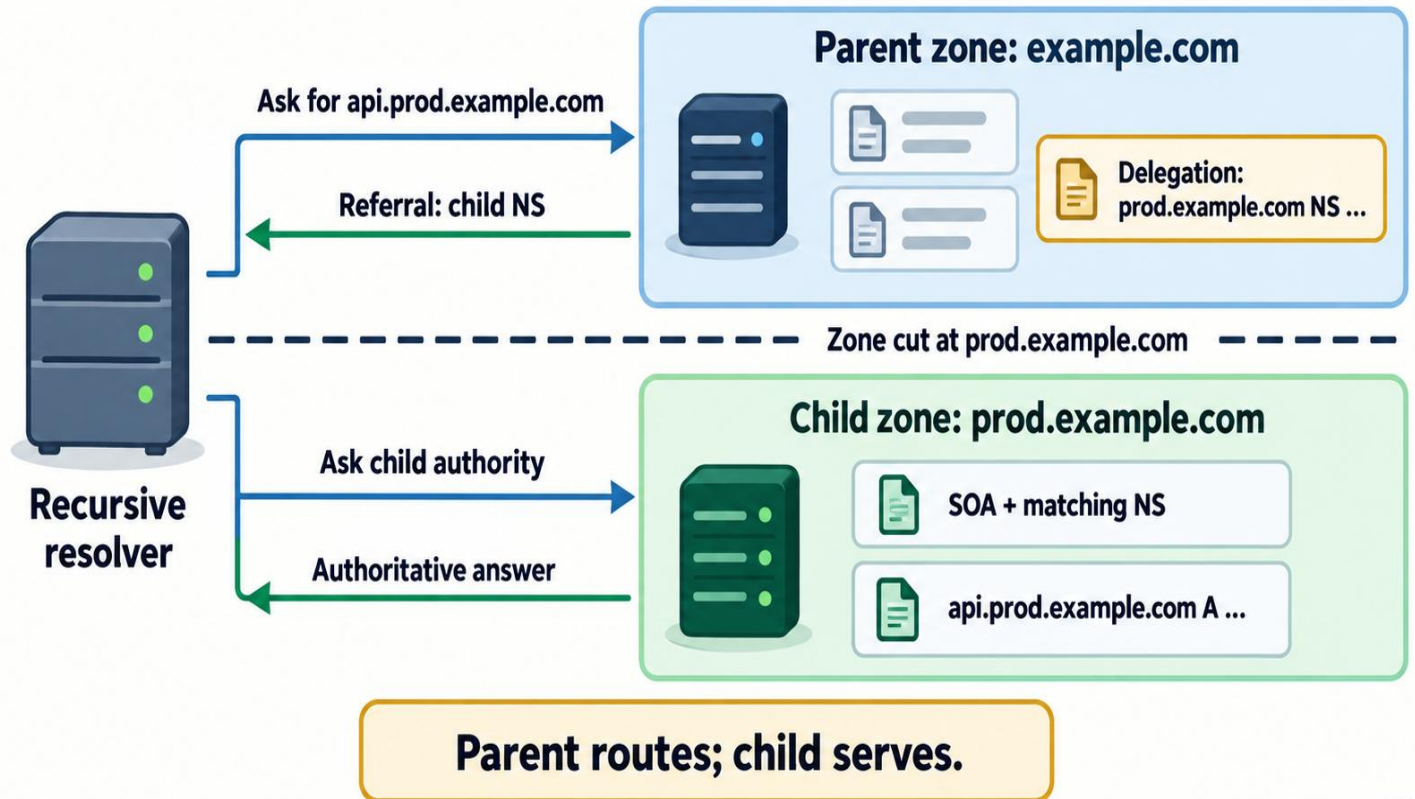
This resembles a company directory entry that says, “For the Production division, consult this separate directory.” The parent does not need to know every person in the division. It needs to provide a reliable path to the division’s directory.

A zone cut creates two related but distinct responsibilities:

- the **parent zone** must publish a valid delegation; and
- the **child zone** must exist and answer authoritatively on the delegated servers.

A failure on either side can make the child namespace unreachable.

2-6 Zone Cut Between Parent and Child Authority



012

Figure 2-4: Zone cut between parent and child authority

Absolute and relative names

An **absolute DNS name** specifies the complete name to the root, usually shown with a trailing dot in DNS configuration:

```
api.prod.example.com.
```

A **relative DNS name** is interpreted in relation to another name, often the current zone or a configured search domain:

```
api
```

Inside a zone file whose origin is `prod.example.com.`, the relative owner name `api` can represent

```
api.prod.example.com..
```

Relative names reduce repetition, but they can also create accidental names when used without understanding the current context. Consider this zone-file value:

```
backend.example.net
```

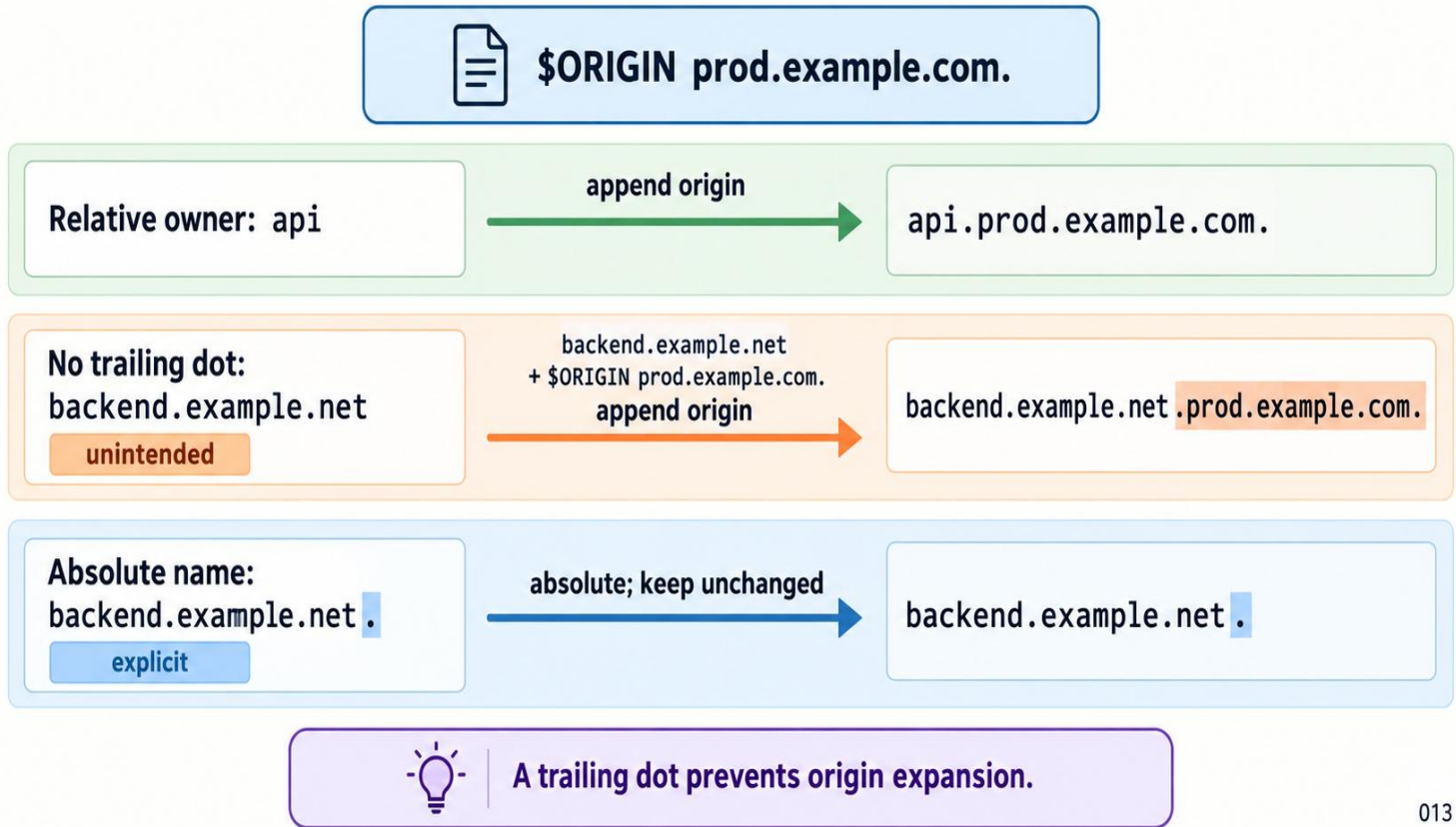
If the software treats it as relative within `prod.example.com.`, the resulting name could be:

```
backend.example.net.prod.example.com.
```

Adding the final dot makes the intended absolute name explicit:

```
backend.example.net.
```

2-7 Absolute and Relative Name Expansion



013

Figure 2-5: Absolute and relative owner-name expansion

In DNS configuration, a trailing dot is a precision tool. It prevents a complete name from being extended by an origin or search suffix.

Breaking down `api.prod.example.com.`

Read the name from right to left:

1. The final `.` identifies the root.
2. `com` identifies the top-level domain.
3. `example` identifies the registered domain label under `.com`.
4. `prod` identifies a subdomain under `example.com`.
5. `api` identifies a name under `prod.example.com`.

Possible zone boundaries include `example.com` and `prod.example.com`, but the name itself does not prove that either boundary exists. You must inspect delegation and authoritative data to know the actual zone structure.

For example, all of these designs are possible:

- one `example.com` zone contains the `api.prod.example.com` record;
- `prod.example.com` is a delegated child zone;
- `api.prod.example.com` is itself a delegated zone, although that would be less common; or
- an internal DNS view contains a different zone structure from the public namespace.

The last design is mentioned only to show that a name alone does not reveal its operational environment. Private and split-view designs are developed later in the full guide.

Hostnames are a practical subset of DNS names

A **hostname** is a DNS name used to identify a host or network interface under conventions accepted by host software. Not every valid DNS owner name is intended to be a hostname.

For example, some service-discovery records use owner names containing underscores:

```
_ldap._tcp.example.com.
```

That name is valid for the relevant DNS record, but it is not a conventional host address name. The target named by the record would normally be a hostname such as `ldap01.example.com.`

For beginning systems work, a safe convention for host labels is to use lowercase letters, digits, and hyphens, while avoiding a hyphen at the beginning or end. DNS comparisons are case-insensitive, so `API.EXAMPLE.COM` and `api.example.com` identify the same DNS name, but lowercase is easier to read and compare.

Naming conventions that reduce operational risk

A good name should remain useful even when the implementation changes. Prefer names that describe function or role rather than temporary infrastructure details.

For example:

- `api.prod.example.com` describes service and environment;
- `payments-db.prod.example.com` describes a role;
- `web-01.example.com` may be appropriate for a specific host; and
- `ec2-instance-3-replacement.example.com` exposes temporary implementation detail and ages poorly.

Useful naming practices include:

- make environment boundaries explicit when confusion would be dangerous, such as `dev`, `test`, and `prod`;
- keep service names stable while replacing the systems behind them;
- avoid using the same short label for unrelated purposes in nearby namespaces;
- document which team owns each zone or delegated subdomain; and
- use FQDNs in automation when search-suffix behavior would be unsafe.

Do not make names unnecessarily long. The purpose is clarity, not encoding every property of the system into DNS.

Existing zone or delegated subdomain?

Suppose an application team needs names under `dev.example.com`. Two common designs are possible.

Add records to the existing parent zone

The central DNS team keeps `dev.example.com` within the `example.com` zone and adds records for the application team.

This is suitable when:

- the number of records is small;
- changes are infrequent;
- central review is desirable; and
- the application team does not need independent automation or policy.

Delegate `dev.example.com` as a child zone

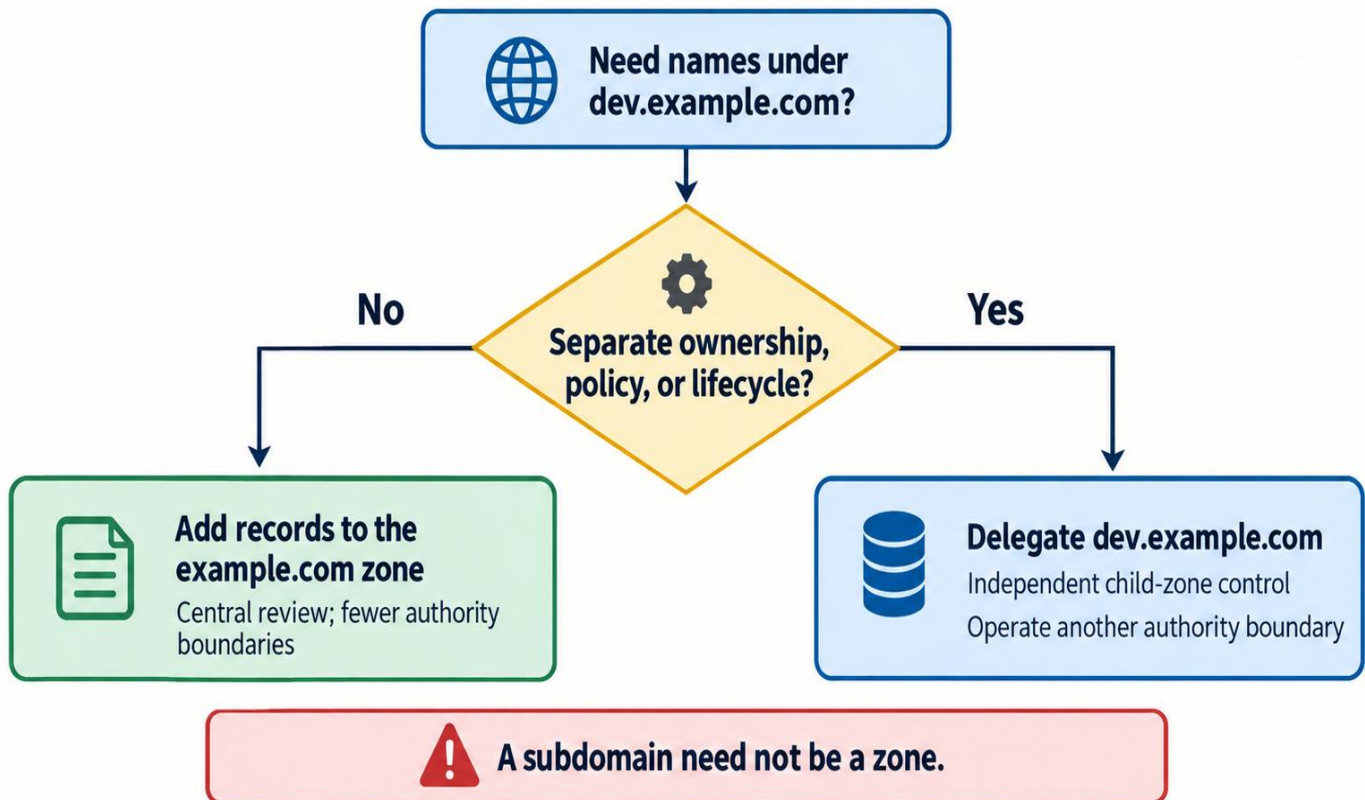
The central team delegates the subdomain, and the application or platform team operates the child zone.

This is suitable when:

- the child team needs independent change control;
- records change frequently;
- separate access control is required; or
- the child namespace has its own lifecycle and operational ownership.

Delegation adds flexibility, but also creates another boundary that must be monitored. **Use a separate zone when it provides a real administrative benefit, not merely because a subdomain exists.**

2-11 Use the Parent Zone or Delegate a Child Zone



014

Figure 2-6: Use the parent zone or delegate a child zone

Registration and DNS hosting are different services

A **domain registrar** is an organization through which a registrant manages registration for a domain and supplies delegation information to the relevant registry.

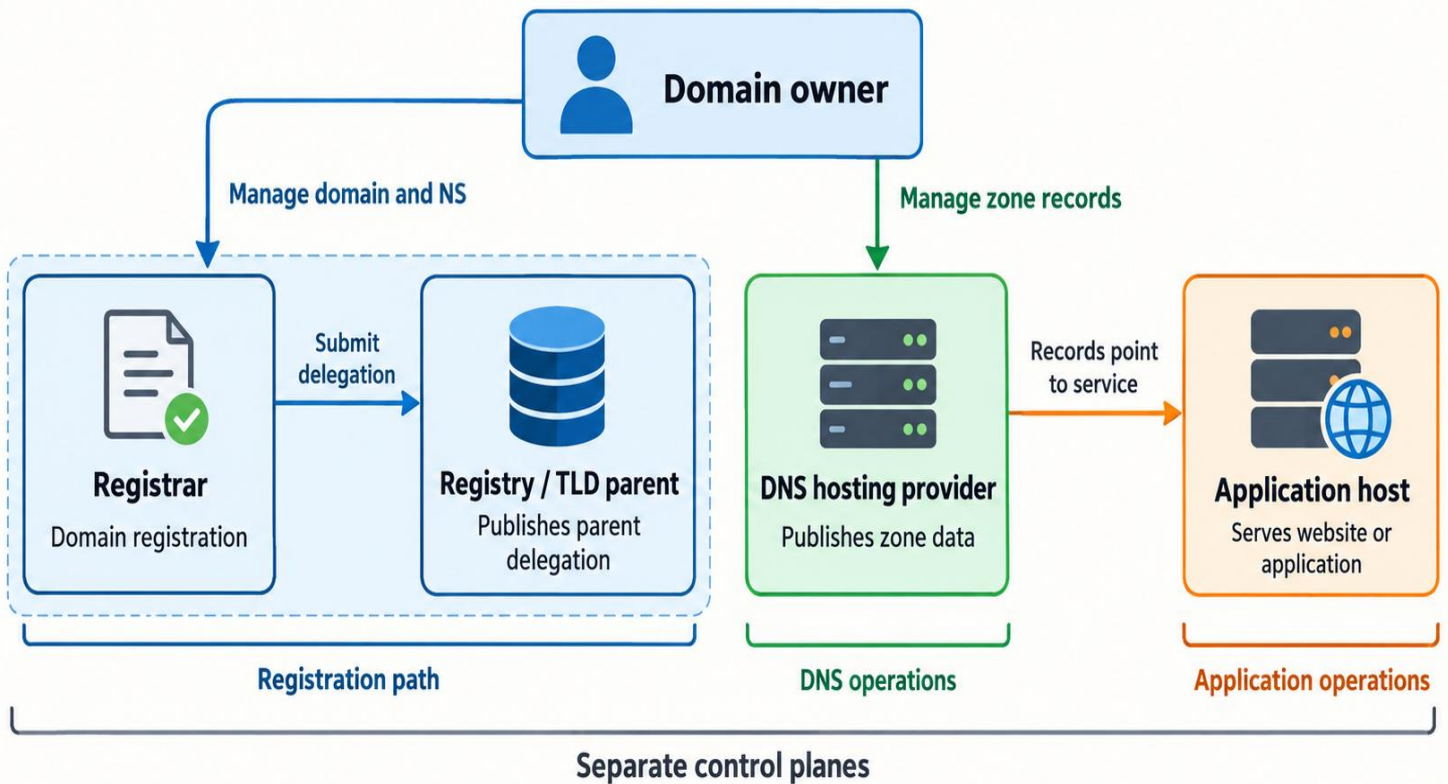
A **DNS hosting provider** operates authoritative name servers and stores the zone's DNS data.

The same company may provide both services, but the roles are distinct. A domain can be registered with one organization while its authoritative DNS is hosted by another. Changing web hosting also does not necessarily require changing either the registrar or DNS hosting provider.

That separation helps identify the correct control plane during a change:

- modify ordinary records at the DNS hosting provider;
- modify the domain's delegated name-server set through the registrar; and
- manage application content at the application or web hosting platform.

2-12 Registration, DNS Hosting, and Application Hosting



015

Figure 2-7: Registrar, registry, DNS host, and application host responsibilities

The model to retain

Names describe positions in a hierarchy. Domains describe branches. Zones describe administrative data boundaries. Delegations connect parent and child zones.

Keeping those four ideas separate prevents many high-impact configuration errors.

Chapter Summary

- DNS names form a hierarchy that is read from the root toward increasingly specific labels.
- A domain is a branch of the namespace; a zone is the portion served by one administrative authority.
- A subdomain becomes a separate zone only when a delegation creates a zone cut.

Review Questions

1. What does the trailing dot in a fully qualified domain name represent?
2. Why are a domain and a zone not necessarily the same set of names?
3. What is the zone apex?
4. When is delegation more appropriate than adding records to an existing zone?