The Phone Book

Understanding DNS in 20 minutes

Jamie Barton

Contents

Preface

This book is short on purpose.

The blame for that lies with Karl Seguin. *The Little Go Book* kicked off my whole habit of learning something and writing the cheat sheet as I went. It turns out the fastest way to find out whether you understand a thing is to try explaining it in as few words as possible.

That habit matters more than ever when you join a new company. Onboarding somewhere with a whole platform of new products means learning the core concepts fast, and DNS sits underneath nearly all of them.

Here's my confession, though: I'm a domain hoarder. I've been comfortably getting myself around DNS for two decades (pointing records, moving nameservers, waiting out TTLs), but I'd never actually sat down and documented what I know, or checked whether what I "know" is even true. I didn't know DNSSEC existed until I joined bunny.net. It also never clicked that the AAAA record is called that because an IPv6 address is four times the length of an IPv4 one. Four A's, four times the bits. Twenty years of adding those records, and I learnt why last month.

Turns out you can teach an old dog new tricks. This book is the cheat sheet I wrote while learning them.

You can read it in about twenty minutes. By the end, you'll understand how a domain name turns into an IP address, why changes "take

time to propagate" (spoiler: they don't, really), and how to debug the whole thing with one command.

You don't need to be a DNS expert. But if you've ever added a record at a registrar and waited for it to show up, you know enough to start.

Chapter 1

The Problem

Computers talk to each other using IP addresses, like `79.127.237.104`. Humans are terrible at remembering those.

So we invented names. `bunny.net` is easier to remember than any string of numbers, and the name can stay the same even when the numbers behind it change.

DNS (the Domain Name System) is the thing that translates one into the other. That's the whole job.

Everything else in this book is just details about *how* it does that job at internet scale without falling over.

Chapter 2

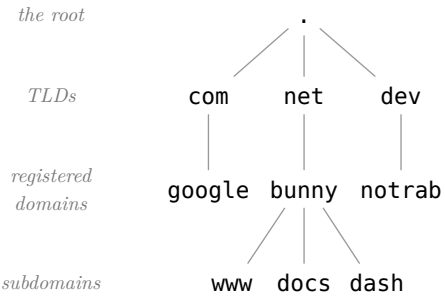
A Name, Read Backwards

Here's a trick that makes DNS click: **read domain names right to left.**

Take `www.bunny.net.` (yes, there's a real trailing dot, more on that in a second):

```
.        →  the root
net      →  a top-level domain (TLD)
bunny    →  a domain registered under net
www      →  a subdomain of bunny.net
```

DNS is a tree. The root sits at the top, TLDs like `com`, `net` and `dev` hang off it, registered domains hang off those, and you can nest subdomains as deep as you like.



The trailing dot represents the root of the tree. You almost never type it, but it's implicitly there. `www.bunny.net` really means `www.bunny.net.`, a full path from leaf to root.

Each level of the tree can be run by a completely different organisation. The root zone is overseen by ICANN, `net` is run by Verisign, `bunny.net` is run by whoever registered it. Nobody owns the whole tree. This is why DNS scales: responsibility is *delegated* downwards.

Chapter 3

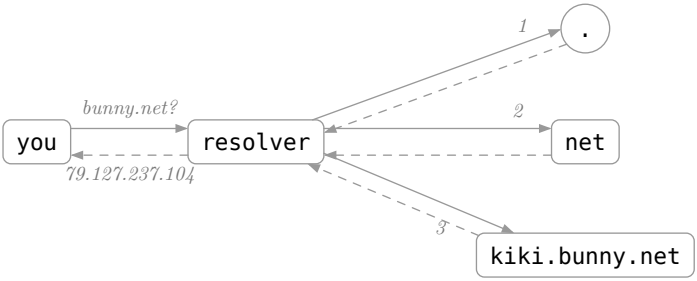
A Lookup, Step by Step

When you type `bunny.net` into a browser, here's what actually happens.

Your computer asks a **resolver** (sometimes called a *resolving name server*), usually one run by your ISP, or a public one like `8.8.8.8` or `9.9.9.9`. The resolver's job is to find the answer, no matter how many questions it takes.

If the resolver doesn't already know the answer, it walks the tree from the top:

1. **Ask a root server:** "Where's `bunny.net`?"
   Root says: "No idea, but here are the servers for `net`."
2. **Ask a `net` server:** "Where's `bunny.net`?"
   It says: "No idea, but here are the nameservers for `bunny.net`."
3. **Ask `bunny.net`'s nameserver:** "Where's `bunny.net`?"
   It says: "`79.127.237.104`. Final answer."



Three questions, each one getting you a step closer. The root and TLD servers never know the final answer, they just know who to ask next. Only the last server, the one that's **authoritative** for the domain, gives you the actual record.

There are names for these two kinds of questions, and you'll see them in docs everywhere. Your question to the resolver is a **recursive query**: "get me the complete answer, however many steps it takes." The resolver's questions up the tree are **iterative queries**: each server answers as best it can, or points to someone closer. You ask once and the resolver does the legwork.

Then the resolver hands the answer to your computer, your browser connects to the IP and you've already forgotten DNS was involved.

The whole dance typically takes tens of milliseconds. And usually it's even faster than that, because of the subject of the next chapter.

Chapter 4

Caching (or: Why DNS Is Fast)

If every lookup walked the whole tree, the root servers would melt. So DNS caches *everywhere*: in your browser, your operating system, your router, and especially in resolvers.

Every DNS record carries a **TTL** (time to live), measured in seconds. It means: "you may remember this answer for this long."

```
bunny.net.  300  IN  A  79.127.237.104
              └── cache me for 5 minutes
```

A resolver that just answered this question for someone else will answer it for you instantly, straight from cache, without asking anyone.

TTLs are a trade-off you control:

- **Long TTL** (hours): fewer lookups, faster answers, but changes take longer to be noticed.
- **Short TTL** (seconds to minutes): changes spread quickly, but more traffic hits your nameservers.

A common trick: run a long TTL normally, then drop it to 60 seconds *before* a planned migration. Once caches have picked up the short TTL, your actual change will be noticed everywhere within a minute.

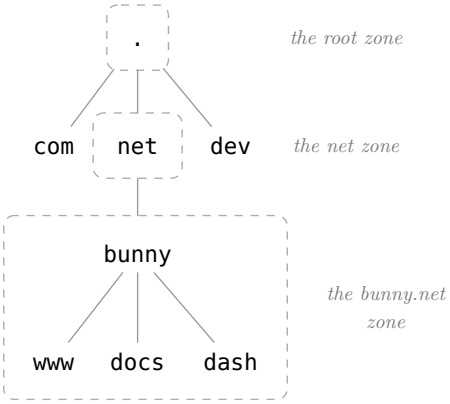
This is also the truth behind “DNS propagation”. Nothing is being pushed anywhere. Your change is live on your nameserver the instant you save it. The “delay” is just old answers sitting in caches around the world, waiting for their TTLs to expire.

Chapter 5

Records

First, a word you'll see on every DNS dashboard: **zone**.

A zone is the slice of the tree that one party actually manages. Its records live together in one place, under one authority. Chapter 2 said nobody owns the whole tree. Zones are how it's carved up. The root is a zone. `net` is a zone. When you register `bunny.net`, you get a zone of your own.



The subtle bit: a *domain* is a name, a *zone* is a unit of management, and the two don't have to line up. `docs.bunny.net` is part of the `bunny.net` domain, but you could delegate it, with NS records, into a separate zone run by someone else entirely. That's the same delegation trick from Chapter 3, just happening inside your own domain. Most of

the time, though, one domain = one zone, and "add a zone" in a DNS dashboard means "start managing records for this domain".

And a zone is basically a small database of records. Each record has a name, a type, a TTL and a value. You only need to know a handful of types.

A: maps a name to an IPv4 address.

```
bunny.net.  IN  A  79.127.237.104
```

AAAA: same thing, but IPv6. (Four A's because IPv6 addresses are four times longer. Really.)

```
bunny.net.  IN  AAAA  2400:52e0:1e03::1205:1
```

CNAME: an alias. "This name is really that other name."

```
dash.bunny.net.  IN  CNAME  panel-production.b-cdn.net.
```

One famous rule: a name with a CNAME can't have any other records. That's why you traditionally can't put a CNAME on the apex (`bunny.net` itself, which needs SOA and NS records), and why DNS providers invented workarounds with names like ANAME, ALIAS or CNAME flattening.

MX: where email for this domain should be delivered. The number is a priority, lower wins.

```
bunny.net.  IN  MX  1 aspmx.l.google.com.
bunny.net.  IN  MX  5 alt1.aspmx.l.google.com.
```

TXT: arbitrary text. In practice, that means proving domain ownership and email authentication (SPF, DKIM, DMARC live here).

```
bunny.net.  IN  TXT  "v=spf1 mx ~all"
```

NS: names the servers that are authoritative for a zone. These records *are* the delegation mechanism from Chapter 3.

```
bunny.net.  IN  NS  coco.bunny.net.
bunny.net.  IN  NS  kiki.bunny.net.
```

SOA: start of authority. Zone metadata: a serial number, refresh timers, the admin contact. Every zone has exactly one. You'll rarely touch it, but it's how secondary servers know when a zone has changed.

Two more you'll bump into eventually: **PTR** does the reverse trick, mapping an IP address back to a name (mail servers check these a lot), and **SRV** is like MX but for other services, telling clients which host and port to find them on.

That's nine types. There are over forty more, and you can ignore nearly all of them. Two worth recognising when they cross your path: **CAA** lists which certificate authorities are allowed to issue certificates for your domain, and **HTTPS** is a newer type that carries connection hints and finally gives the apex CNAME problem a proper fix. The DNSSEC ones turn up in Chapter 9. Beyond that, these nine cover 95% of real life.

Chapter 6

Who Does What

Four roles keep getting confused, so let's take them one at a time:

The registry runs a TLD. Verisign runs `com`. They maintain the master list of who owns what under that TLD.

The registrar sells you the domain. Namecheap, IWantMyName, GoDaddy. They talk to the registry on your behalf.

The DNS host runs the authoritative nameservers that answer queries for your zone. Often bundled with your registrar, but it doesn't have to be. You point NS records wherever you like.

The resolver does lookups on behalf of users. Your ISP runs one, and so do Google (`8.8.8.8`) and Quad9 (`9.9.9.9`).

The most common real-world confusion is people editing records at their registrar while their NS records point at a different DNS host entirely. The records they're editing are never consulted. If a change seems to have no effect at all, not even after TTLs expire, check where the NS records actually point first.

Chapter 7

dig

One tool debugs nearly everything: `dig`.

Look up a record:

```
$ dig bunny.net A +short
79.127.237.104
```

Ask a *specific* server, bypassing all caches. This is how you check what your authoritative nameserver really says, right now:

```
$ dig @kiki.bunny.net bunny.net A
```

See a domain's nameservers:

```
$ dig bunny.net NS +short
```

Watch the entire resolution walk from the root, step by step, exactly like Chapter 3:

```
$ dig bunny.net +trace
```

A debugging recipe that solves most "my DNS change isn't working" problems:

1. `dig @your-nameserver name type`: is the record correct at the source?
2. `dig name type`: what does your resolver's cache say?
3. If (1) is right and (2) is stale: it's just TTL. Wait.
4. If (1) is wrong: you edited the wrong place. Reread Chapter 6.

Chapter 8

The Sharp Edges

A few things that bite everyone eventually.

Negative caching. Resolvers cache "that record doesn't exist" answers too. If you query a name *before* creating the record, the NXDOMAIN gets cached, and your freshly created record appears broken. Create first, query second.

The apex CNAME trap. Covered in Chapter 5, causes real outages. If your provider offers CNAME flattening or ALIAS records, use them at the apex.

Trailing dots in zone files. In raw zone files, `mail.bunny.net.` (with dot) is absolute. `mail` (without) gets the zone name appended. Forget the dot on an absolute name and you've created `mail.bunny.net.bunny.net`. Every DNS engineer has done this once.

UDP, mostly. DNS runs primarily over UDP port 53 for speed, falling back to TCP for large answers and zone transfers. Firewalls that allow UDP/53 but block TCP/53 cause weird, intermittent failures.

DNS is public. Anyone can query your records. Never put secrets in TXT records, and assume your subdomains can be enumerated.

Classic DNS trusts everyone. Answers arrive unauthenticated and unencrypted. DoH (DNS over HTTPS) and DoT (DNS over TLS) encrypt the path between you and your resolver so it can't be snooped,

and DNSSEC signs the answers themselves so they can't be forged. That one gets its own chapter, next.

Chapter 9

DNSSEC (or: How Do You Know the Answer Is Real?)

An uncomfortable truth about everything in this book so far. DNS was designed in the 1980s, when the internet was small and everyone mostly trusted each other. There's no built-in way to verify that a DNS answer actually came from the right server. Your resolver asks a question, something answers, and it believes whatever comes back.

An attacker who can slip a fake answer into that conversation can send you to their server instead of the real one, and thanks to caching, one poisoned answer can sit in a resolver's cache misdirecting *everyone* who asks. This attack has a name: **cache poisoning**.

DNSSEC fixes this with digital signatures. The idea in one sentence: every answer from a signed zone comes with a signature, and your resolver checks that signature before believing anything.

You don't need the cryptographic details, just the shape of it:

- The zone owner signs their records. Each answer now travels with a signature (stored in a record type called **RRSIG**).
- The zone publishes its public key (in a **DNSKEY** record) so resolvers can check those signatures.

- So how do you know the *key* is real? The parent zone vouches for it. `com` holds a fingerprint of `example.com`'s key (a **DS** record), the root holds a fingerprint of `com`'s key, and so on up the tree.

This is called the **chain of trust**, and it ends at the root zone, which is signed in a small piece of internet theatre called the Root Signing Ceremony, a public, audited event where trusted people from around the world gather to sign the root keys. The whole internet's DNS trust bottoms out in a room with witnesses.

DNSSEC validation walks the same root → TLD → domain path as the lookup in Chapter 3. Delegation hands down the *answers*, the chain of trust hands down the *proof*.

Two things worth knowing so you're not surprised later:

DNSSEC doesn't encrypt anything. Queries and answers are still visible to anyone watching the wire. DNSSEC proves answers are *authentic*, not *private*. Privacy is DoH and DoT's job (Chapter 8). Different problems, different tools.

It's opt-in, per zone. If a domain hasn't enabled it, there's simply nothing to verify. In practice, turning it on usually means one switch at your DNS host and pasting a DS record at your registrar. The signing, key management and re-signing happen behind the scenes.

That's DNSSEC: signatures on answers, keys vouched for by parents, all the way to the root.

Chapter 10

The Whole Book in One Page

- DNS turns names into IP addresses. It's a distributed, delegated tree, read right to left, rooted at a trailing dot.
- Resolvers walk the tree (root, TLD, authoritative), caching every answer. You ask recursively, the resolver asks iteratively.
- TTLs control cache lifetime. "Propagation" is just caches expiring. Lower TTLs *before* migrations.
- Nine record types, grouped into zones, cover almost everything: A, AAAA, CNAME, MX, TXT, NS, SOA, PTR, SRV.
- Registry $\neq$ registrar $\neq$ DNS host $\neq$ resolver. Most mystery bugs are edits made at the wrong one.
- `dig` answers every question. `+short` for answers, `@server` to skip caches, `+trace` to watch the walk.
- Create before you query, mind the apex, mind the trailing dot.
- Plain DNS believes anything. DNSSEC signs answers and chains the trust up to the root. DoH/DoT keep them private on the wire.

That's DNS. It's a phone book with really good caching.

Now go run `dig +trace` on your favourite domain and watch the tree talk.

Bonus: What If the Phone Book Could Think?

Everything in this book has assumed one thing: a DNS server stores records, and when a query arrives, it looks the answer up and sends it back.

```
bunny.net?  →  look up the record  →  79.127.237.104
```

That's how DNS has worked for decades. The records might change over time, but by the time the question arrives, the answer is already sitting there waiting. Most of the internet still works exactly this way.

You'll sometimes hear the term *Dynamic DNS* (DDNS), but that's something older and smaller, records that update themselves. Your ISP changes your home IP address overnight, a little client notices and rewrites your A record. The lookup itself hasn't changed at all. DNS still returns a record from a database. The database just keeps itself up to date.

Some DNS providers have started asking a different question: what if the answer didn't have to exist until someone asked for it?

```
bunny.net?  →  run your code  →  whatever it decides
```

Instead of reading a record from a database, the authoritative nameserver runs code you wrote. bunny.net calls this **Scriptable DNS**. Full disclosure, I work there, but other providers have their own take on the same idea. Nothing about the protocol changes. Resolvers ask the same questions, clients receive ordinary DNS responses. The only difference is how the answer was made.

Why write code to return something you could have stored? Because sometimes the right answer depends on who's asking. Here's a script that sends Australian visitors to a nearby server and everyone else to Europe:

```
export default function handleQuery(query) {
  if (query.request.geoLocation.country == "AU") {
    return new ARecord("203.0.113.7", 300);
  }
  return new ARecord("79.127.237.104", 300);
}
```

The same trick covers a lot of ground. A customer on a beta programme is sent to the new deployment. A failed origin is quietly swapped for a healthy one. A single script serves thousands of customer subdomains without storing a record for each. The answer becomes something you compute instead of something you store.

The clever bit is that none of this changes DNS itself. That `300` is an ordinary TTL, so resolvers cache the answer exactly as they always have. Return five minutes if the answer is stable, or thirty seconds if it isn't. To everyone asking, it looks like ordinary DNS. The intelligence lives entirely inside the authoritative nameserver.

For most of its history, DNS has been a simple system, a distributed phone book with really good caching. Scriptable DNS doesn't replace that model, it builds on it. The phone book still answers your questions. It just gets to think for a moment before it replies.

Afterword

DNS is one piece of a much larger system.

This book answered one question: how does a name become an IP address? The questions next door are just as interesting. How does someone come to own a domain in the first place? How does your browser know it's talking to the real server? How does a request actually find its way across the internet?

I've already been living with those questions, in a DNS CLI I've been building that handles certificates, and in years of wrestling with Let's Encrypt. I just haven't put pen to paper yet. If I do, those books will follow the same rule as this one and try to explain one idea in as few words as possible, without assuming you already know the jargon.

Thanks for reading.