

SUBSCRIBE

SIGN IN

HOME AAAA ::1 —

# Understanding DNS—anatomy of a BIND zone file

Need a clear, thorough guide to understanding how DNS records work? We got you.

JIM SALTER - 8/24/2020, 7:30 AM

Santo Heston



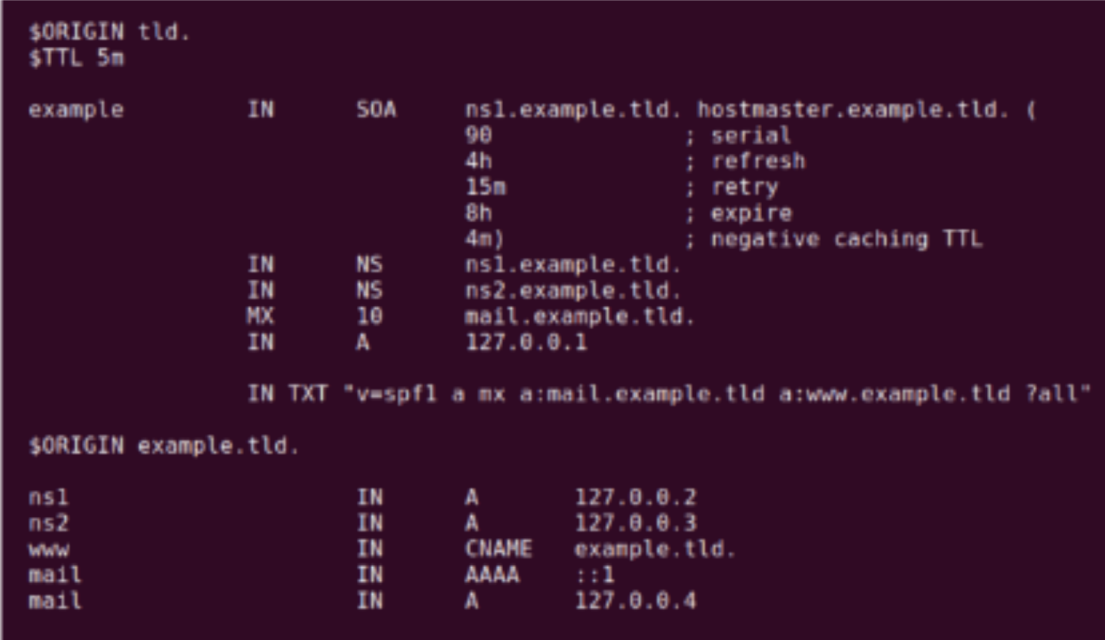
**Enlarge** / What does this stream of binary digits have to do with DNS? Nothing, really—but good luck finding a pretty pic somewhere that does!

If you want to be a sysadmin or network administrator of any kind, there's a fundamental

technology you really need to understand—DNS, the Domain Name System. There was a time when a sysadmin with no aspirations to managing Internet-accessible services might have gotten by without understanding DNS, but that time is long, long gone.

You can't learn everything there is to know about DNS in a single article. But that's not what we're looking to do today; instead, we want to give you a clear, concise guide to the structure and meaning of the most important part of the Domain Name System: a zone file, as seen in BIND, the Berkeley Internet Name Daemon.

## Sample zone file



```

$ORIGIN tld.
$TTL 5m

example      IN      SOA      ns1.example.tld. hostmaster.example.tld. (
                                90           ; serial
                                4h           ; refresh
                                15m          ; retry
                                8h           ; expire
                                4m)          ; negative caching TTL

              IN      NS       ns1.example.tld.
              IN      NS       ns2.example.tld.
              MX      10       mail.example.tld.
              IN      A        127.0.0.1

              IN      TXT      "v=spf1 a mx a:mail.example.tld a:www.example.tld ?all"

$ORIGIN example.tld.

ns1           IN      A        127.0.0.2
ns2           IN      A        127.0.0.3
www           IN      CNAME     example.tld.
mail          IN      AAAA      ::1
mail          IN      A        127.0.0.4

```

**Enlarge** / This sample zone file doesn't have every possible record type in it—but it's a good start.

### Origin and TTL

Above, we have a small but complete example of a typical zone file—in fact, it's an anonymized version of a production zone file on a domain I manage. Let's go through it line by line.

```

$ORIGIN tld.
$TTL 5m

```

Whenever you see an `$ORIGIN` line in a zone file, this is a shortcut that lets BIND know that any unterminated hostname references following that line should be presumed to end in the argument following `$ORIGIN`. In this case, that's `.tld`—the fictional Top Level Domain for `example.tld`.

The next line, `$TTL 5m`, declares that following lines will have a Time To Live of five minutes. This relatively short value means that remote DNS resolvers should only cache records retrieved from this zone for five minutes before requesting them again. If you're relatively certain that your DNS for a given domain won't change very often, you might consider increasing that value in order to reduce the number of times remote hosts must query your nameserver—but keep in mind that a longer TTL also means longer periods of downtime, when you must make a change to your DNS (or make a change that accidentally breaks it).

Both `$ORIGIN` and `$TTL` can be defined multiple times in the same zone—each time you redefine them, you change their value for any lines beneath the new values in the same zone file.

## SOA record

```
example      IN      SOA      ns1.example.tld. hostmaster.example.tld. (
                                90          ; serial
                                4h          ; refresh
                                15m         ; retry
                                8h          ; expire
                                4m)         ; negative caching TTL
```

The first actual record in our sample zone file—or in any normal zone file—is the SOA record, which tells us the Start Of Authority for the domain. It's also easily the most confusing record type in the entire DNS system.

For any record listing, including this SOA record, the first argument is the hostname the record applies to—in this case, `example`. Remember how we set `$ORIGIN tld` on the first line of the zone file? That means that this unterminated hostname `example` expands to `example.tld`—so, we're defining the SOA for the FQDN (fully qualified domain name) `example.tld`.

We're referring to this hostname `example` as "unterminated" because it doesn't end in a dot. If we wanted to bypass the `$ORIGIN` setting and refer to a FQDN directly, we'd terminate it with a final dot—eg, `example.tld.` would be the FQDN here, *with* the trailing dot.

The next argument we see is `IN`, short for "Internet." This is the record class. There are other DNS record classes, but you can easily go your entire career without seeing one of them (such as `CH`, for Chaos) in production. The record class is optional; if omitted, BIND will assume that the record being specified is of class `IN`. We recommend *not* omitting it, however, lest something change and all your zone files suddenly be broken after a BIND update!

The next two arguments are FQDNs—at least, they look like it. The first FQDN really is an FQDN, and it should be the FQDN of the primary name server for the domain itself—in this case, `ns1.example.tld`. Note that you *can* use unterminated hostnames here—for example, we could have just used `ns1.example` for this argument, which would have expanded to

`ns1.example.tld.` thanks to our `$ORIGIN .tld` line—but it's probably best to be explicit here.

The second FQDN, `hostmaster.example.tld.`, isn't actually an FQDN at all. Instead, it's a perverse way of rewriting an email address. `@` is a reserved character in zone files, and the original BIND uses the first section of this "FQDN" as the user portion of an email address—so, this would translate to the address `hostmaster@example.tld`. It's *incredibly* common to see this screwed up in real-life zone files—thankfully, it doesn't much matter. We're not aware of literally anyone who actually uses this feature of a DNS zone to contact anyone.

Moving on, we have `serial`, `refresh`, `retry`, `expire`, and `negative TTL` for the zone inside parentheses. Note that the comments you see here labeling them are not required—and in real life, you'll rarely see them. We strongly prefer to put these comments in production zone files in order to make it easier to read them, but BIND itself doesn't care!

- **serial** —this is a simple serial number for the zone file, which must be incremented each time the contents of the zone are changed. If you don't update the zone file serial, your changes to the zone will not be picked up by DNS resolvers that have previously cached records from your zone! This used to be a YYMMDDnn format in days gone by—but that format is no longer required, or in some cases even supported. Just start your zones with serial `1`, increment to `2` the next time you make a change to the zone, and so forth.
- **refresh** —after this period of time, secondary nameservers should query the primary nameserver for this SOA record, to detect changes in serial number. If the serial number has incremented, any cached records must be invalidated and fetched again from the primary nameserver.
- **retry** —if the primary nameserver doesn't respond to an SOA request, a secondary nameserver should wait this long before attempting to query the primary nameserver again.
- **expire** —if the primary nameserver has failed to respond to a secondary nameserver's SOA request for this period of time, the secondary nameserver should stop offering DNS resolution for the domain entirely.
- **negative caching TTL** —this controls how long a "negative" response—eg, "I don't have the record you're asking for"—should be cached.

One of the most common areas for confusion in the SOA record is what effect the `refresh`, `retry`, and `expire` arguments have. These arguments don't affect DNS resolvers at all—only secondary authoritative nameservers for the domain. If you don't have one or more secondary nameservers for your domain, which use BIND replication to retrieve updates from the primary, these arguments won't have any effect at all.

One final note: older versions of BIND required all of these times to be in seconds... even when the actual time interval was in days, or weeks. BIND9—released almost 20 years ago, in October 2000—supports human-readable time suffixes such as "m" for minutes, "h" for

hours, and "d" for days. *Please* use these human readable suffixes when writing zone files; nobody should have to break out a calculator to figure out that 86,400 seconds is one day!

## NS records

```
IN      NS      ns1.example.tld.  
IN      NS      ns2.example.tld.
```

In these two records, we define the hostnames, which are authoritative nameservers for our zone. Once again, we've used dot-terminated FQDNs for these records. Once again, we *could* have used unterminated hostnames— `ns1.example` and `ns2.example`—and relied on our `$ORIGIN .tld` to expand them. Doing so would make the zone more confusing and difficult to read, though.

Note that the NS record specifies *hostnames*, not IP addresses. This is a common source of confusion for DNS newbies, and it's important to get it right. You *cannot* specify a bare IP address as the nameserver for a domain; you absolutely must specify a hostname here.

Finally, note that we haven't specified the domain name itself on either line—this is because we've inherited it from the SOA record above. We started that line with `example`, which expands to `example.tld`. Since we haven't specified another hostname, these new NS records also apply to that hostname by default.

In the real world, you may also see zone files with `$ORIGIN example.tld.`, and beginning the SOA and possibly other lines with the special reserved character `@`. When you see `@` as a hostname in a zone file, that just means you're using the bare `$ORIGIN` without any further qualifiers.

## MX record(s)

```
IN      MX  10    mail.example.tld.
```

In this simple domain, we have a single mailserver, and it's `mail.example.tld`. The MX record just tells anyone who wants to send email to any address at `example.tld` to make their SMTP connection to the hostname specified in this record.

The preceding argument— `10` in this case—is the numeric priority of the mailserver in this specific record. Lower numbers mean higher priority. When multiple SMTP servers are available for a domain, you'll see multiple MX records as well, each with a different priority. In theory, higher priority mailservers should always be tried first, and lower priority mailservers only tried if the higher priority server fails.

Well-behaved SMTP servers do follow this protocol—but spammers have a tendency to deliberately target the lower-priority mailservers first, operating on the theory that high-priority servers might be anti-spam gateways, and the lowest priority servers might be the bare, unfiltered end server. Spammers suck.

## A record(s)

```
IN      A      127.0.0.1
```

A records are the part of a zone file that actually do what most people think of DNS as doing—they translate a hostname to a bare IPv4 address. In this case, this is a sample file only—and our A record for `example.tld` merely resolves to `localhost`, on the same principle that phone numbers in movies always start with the exchange 555. In real life, of course, you'd put in the IP address of the server you expected to answer when you ping `example.tld`, point a Web browser to `https://example.tld/`, and so forth.

In this simple zone file, we only have a single A record for `example.tld`. In real life, you might encounter several—there could be multiple gateway servers capable of answering Web requests for `https://example.tld/`; and if so, each would get their own A record on their own line.

## TXT record(s)

```
IN      TXT     "v=spf1 a mx a:mail.example.tld a:www.example.tld ?all"
```

This TXT, or text record, is still in the head section of our zone file, under the hostname `example.tld`. So its scope is the entire `example.tld` domain. You can put just about anything in a TXT record; this specific one is an SPF record, formatted to give mailservers information about what machines are authorized to emit mail on behalf of `example.tld`.

In this case, we're saying that we're using the SPF1 version of formatting. We then inform anyone querying this record that any valid A record for `example.tld` is authorized to send mail on its behalf, as is any valid MX for the domain, and finally that the IP addresses associated with the A records for `mail.example.tld` and `www.example.tld` are authorized to send mail. Finally, `?all` says that if any other machine says it wants to send mail from some address at `example.tld`, it should be allowed... but it should be examined more closely than specifically authorized hosts are.

## Hostnames, subdomains, and CNAMEs beneath example.tld

```
$ORIGIN example.tld.
```

```
ns1     IN      A      127.0.0.2
ns2     IN      A      127.0.0.3
www     IN      CNAME  example.tld.
mail    IN      AAAA   ::1
mail    IN      A      127.0.0.4
```

Now that we've defined everything we need to for the domain, we can start adding records for any hostnames and subdomains *beneath* `example.tld` itself. The first thing we do here is change our `$ORIGIN` to `example.tld`. Again, notice that final terminating dot—if you forget

it, things are going to get really strange and you'll tear your hair out wondering why none of your records resolve properly!

We see `A` records here for `ns1`, `ns2`, and `mail`. These `A` records work the same way that the `A` record for the domain itself did—we are telling BIND what IP address to resolve requests for that hostname to.

We also have an `AAAA` record for `mail.example.tld`.—`AAAA` records are just like `A` records, but they're for resolving IPv6 rather than IPv4. Once again, we've chosen in our example to use a localhost address. You'll need to be familiar with `AAAA` records if you expect to set up your own mailserver—Google stopped being willing to talk to mailservers without fully working IPv6 DNS a few years ago!

The last record type we see here is `CNAME`, short for Canonical Name. This is an alias—it allows you to tell BIND to always resolve requests for the `CNAMEd` host using the `A` or `AAAA` record specified in the target argument. In this case, `www IN CNAME example.tld.` means that the IP address for `example.tld` itself should also be handed out if somebody asks for `www.example.tld`.

`CNAME` records are handy, but they're a bit funky. It's worth remembering that each level of `CNAME` necessitates another DNS lookup—in this case, a remote machine that asked to resolve `www.example.tld` would be told "please look up `example.tld`. in order to find your answer," and then would need to issue a *separate* DNS request for the `A` record associated with `example.tld`. If you have `CNAMEs` pointing to `CNAMEs` pointing to `CNAMEs`, you'll introduce unnecessary latency into requests for your resources, and your domain will appear "slow" and "laggy" to your users!

There are further limitations in `CNAME` records. Remember how we told you that `MX` and `NS` records must point to hostnames, not to raw IP addresses? More specifically, they must point directly to `A` records, not to `CNAME` records. If you try to set `MX mail.example.tld.` followed by `mail.example.tld. CNAME example.tld.`, your zone file will be broken, and `MX` lookup attempts will return errors.

## Tools of the trade

If you're managing your own DNS, you'll need to be proficient in using command line tools to query your DNS server directly and see how it responds to requests—it's difficult to be certain whether the problem is DNS or something else just by putting `https://example.tld/` in a browser and seeing what happens.

### dig

```
you@box:~$ dig @127.0.0.1 NS example.tld
; <<>> DiG 9.16.1-Ubuntu <<>> NS example.tld
;; global options: +cmd
```

```
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 51417
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;example.tld.                IN      NS

;; ANSWER SECTION:
example.tld.                300     IN      NS      ns1.example.tld.
example.tld.                300     IN      NS      ns2.example.tld.

;; Query time: 40 msec
;; SERVER: 127.0.0.1(127.0.0.1)
;; WHEN: Sat Aug 22 00:54:26 EDT 2020
;; MSG SIZE rcvd: 74
```

If you have access to Linux, Mac, or Windows Subsystem for Linux, by far the best command line tool is `dig`. Using `dig` is as simple as specifying a server to query, the record type you want to look for, and the FQDN it should be associated with.

In the example above, we asked the DNS server at `127.0.0.1` to show us all `NS` records associated with `example.tld`. In addition to the answers we wanted, we got a ton of diagnostic information—the DNS server we queried did not return an `ERROR` when queried, it says it is authoritative for the domain in question, and so forth.

You can also supply a `+short` argument if you want `dig` to just shut up and give you the answer you're looking for without all the verbose diagnostics:

```
you@box:~$ dig +short @127.0.0.1 NS example.tld
ns1.example.tld.
ns2.example.tld.
```

Be aware, though, that if there aren't any answers available for a `+short` query—for example, if you typo the domain name—you won't get any response at all, even if the DNS server queried returned an error.

```
you@box:~$ dig +short @127.0.0.1 NS example.tmd
you@box:~$
```

If you want to find out *why* you didn't get an answer, you'll need to lose the `+short` argument to find out.

## nslookup



If you don't have access to `dig`, you can generally get by with `nslookup`. Most commonly, this is a semi-cursed workaround for users sitting at a Windows box without access to Windows Subsystem for Linux, cygwin, or some other way to gain access to more advanced tools than the Windows CLI provides.

`nslookup` is usually invoked without arguments and queried in interactive mode. Here's a sample session:

```
C:\> nslookup
> server 127.0.0.1
Default server: 127.0.0.1
Address: 127.0.0.1#53
> example.tld
Server: 127.0.0.1
Address: 127.0.0.1#53
```

```
Non-authoritative answer:
Name: example.tld
Address: 127.0.0.1
```

By setting `server 127.0.0.1`, we specified to `nslookup` to use that machine as the DNS server to query. You don't have to specify this; if you don't, `nslookup` will use whatever the default DNS resolver on your machine would.

After optionally setting the `server`, you can just type a bare hostname into `nslookup`'s interactive prompt, and it will return any `A` or `AAAA` records it can find for that hostname.

If you want to query the remote server for a different type of record, you'll need to use a `set` type command.

```
> set type=ns
> example.tld
Server: 127.0.0.1
Address: 127.0.0.1#53
```

```
Non-authoritative answer:
example.tld nameserver = ns1.example.tld.
example.tld nameserver = ns2.example.tld.
```

Authoritative answers can be found from:

```
> set type=mx
> example.tld
Server: 127.0.0.1
Address: 127.0.0.1#53
```

```
Non-authoritative answer:
```

```
example.tld mail exchanger = 10 mail.example.tld.
```

```
Authoritative answers can be found from:  
example.tld nameserver = ns2.example.tld.  
example.tld nameserver = ns1.example.tld.  
mail.example.tld internet address = 127.0.0.4  
> exit
```

In the above examples, we used `set type=ns` and `set type=mx` to query the remote DNS server for NS and MX records for `example.tld`. It works, and you get your answers... but the syntax is fiddly, there's less diagnostic information available, it's vastly less scriptable, and if you're anything like us, you'll likely curse the antiquated thing once or twice before you're done.

The proper way to get out of `nslookup`'s interactive mode is the command `exit`. Hopefully, you never need to look up information about a top-level domain also named `exit`—or if you do, you'll have a better tool available than `nslookup` when you do.

## Conclusions

Hopefully, you picked up something valuable today about how DNS works and how its information is stored. Although BIND is not the only DNS server platform out there—in particular, Windows admins will need to work with Active Directory DNS—the lessons learned here apply near-equally to all platforms and applications.

Although the storage format may change somewhat from server to server—such as an Active Directory domain controller literally storing zones inside Active Directory itself, rather than a plain text file—the record types are universal, and the formatting at least near-universal.

If you're a budding sysadmin or enthusiast who's interested in running your own DNS server, I highly recommend doing it—and using the original platform when you do; BIND on either Linux or BSD. The system load of running a nameserver is nearly nonexistent at any scale short of truly massive; a \$5 Digital Ocean or Linode box can handle the job just fine.

In addition to the sheer joy of learning how to manage these things, you may also find you value the ability to set your TTLs absurdly short—most managed DNS servers won't allow a TTL of less than 30m, and most will attempt to default you to TTLs of up to a week. This is fine and dandy for a DNS zone, which is already properly set up and doesn't need changing... but if your IP address changes and your DNS needs to change along with it, a five-minute TTL is a very, very fine thing to have.

---

**JIM SALTER**

Jim is an author, [podcaster](#), mercenary sysadmin, [coder](#), and father of three—not necessarily in that order.

**EMAIL** [jim.salter@arstechnica.com](mailto:jim.salter@arstechnica.com) // **TWITTER** [@jrssnet](#)