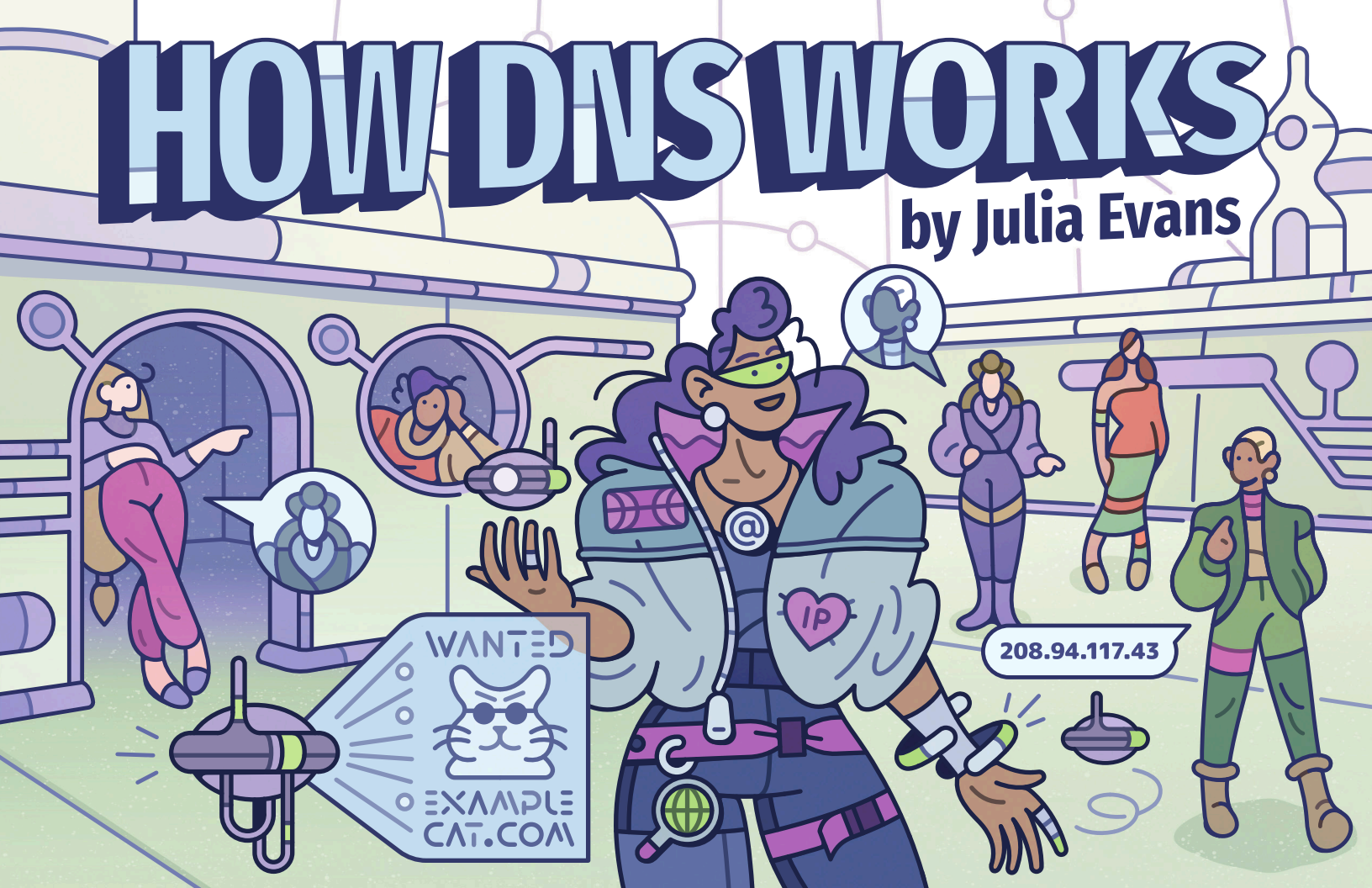
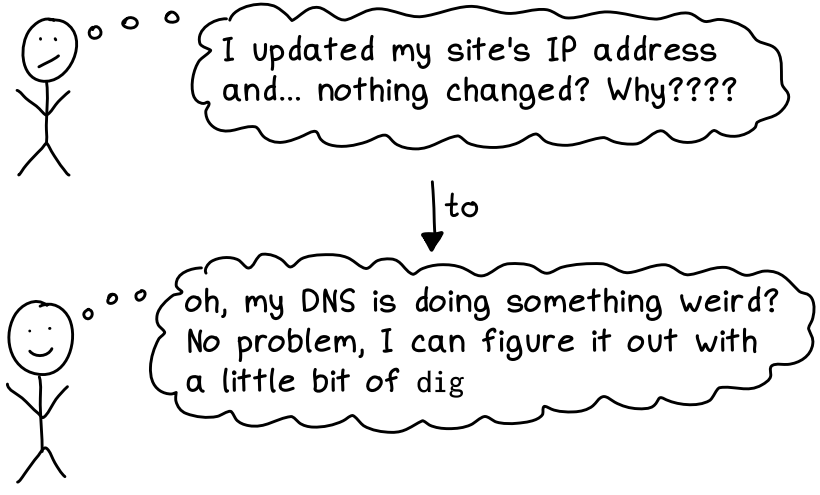

HOW DNS WORKS
by Julia Evans
WANTED
EXAMPLE CAT.COM
@
IP
208.94.117.43

about this zine

Hello! This zine is about DNS: the ☆Domain Name System☆

My goal is to help you get from:



Let's go learn how DNS works!

table of contents

overview

authoritative nameservers

DNS query

response

resolvers

DNS query

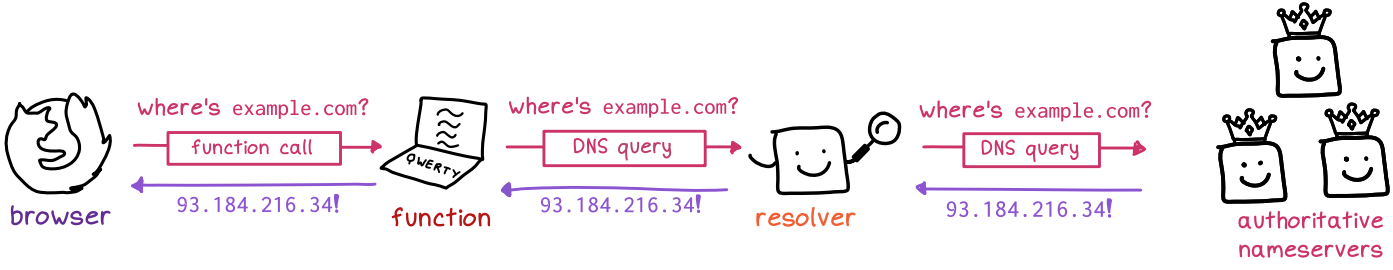
response

on your computer

DNS record types

cast of characters

Let's meet the cast and see how they communicate with each other!



Your browser uses DNS to look up IP addresses every time it visits a domain, like `example.com`.

The browser has a DNS cache.

Your operating system provides a function to do DNS lookups. On Linux and Mac it's `getaddrinfo`.

Your operating system also might have a DNS cache.

The function sends requests to a server called a resolver which knows how to find the authoritative nameservers.

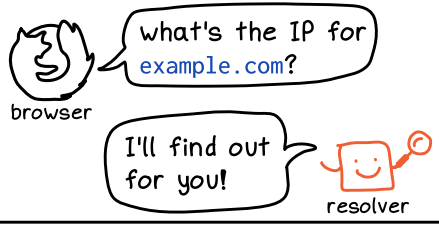
The resolver has a DNS cache.

The authoritative nameservers are the servers where the DNS records are actually stored. They're wearing crowns because they're In Charge.

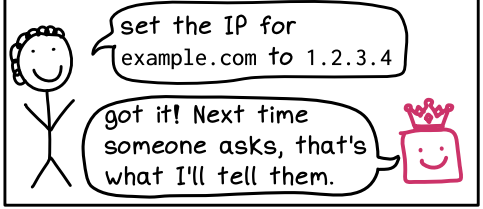
resolvers vs authoritative nameservers

One reason DNS is confusing is that the DNS server you query (a resolver) is different from the DNS server where the records are stored (a network of authoritative nameservers)

anytime your browser makes a DNS query, it's asking a resolver



anytime you update a domain's DNS records, you're updating an authoritative nameserver



how a resolver handles queries

1. check its cache, or (if that fails)
2. find the right authoritative nameserver and ask it

how an authoritative nameserver handles queries

1. check its database for a match
2. that's it, there's no step 2. It's the authority!

the terminology is really confusing

Other names for resolvers:

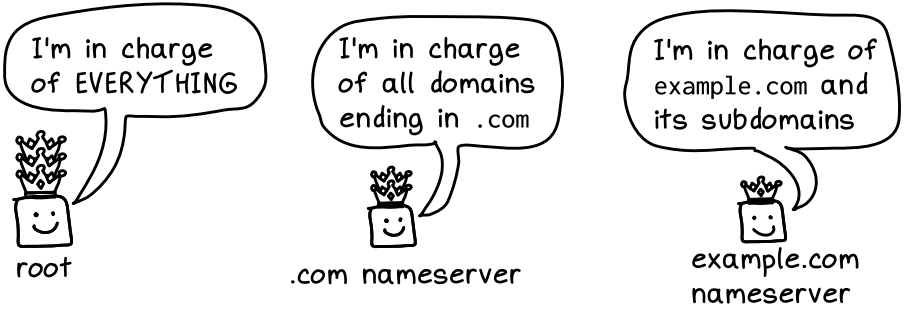
recursive resolver	recursive nameserver
DNS recursor	DNS resolution service
public DNS server	caching-only nameserver

Types of authoritative nameservers:

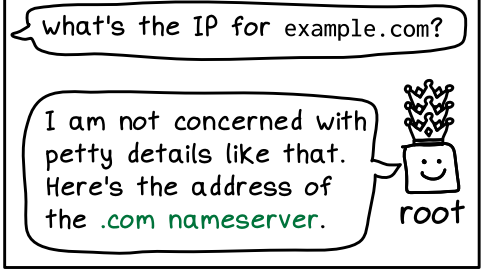
- root nameserver
- TLD nameserver (like .com or .ca)

the DNS hierarchy

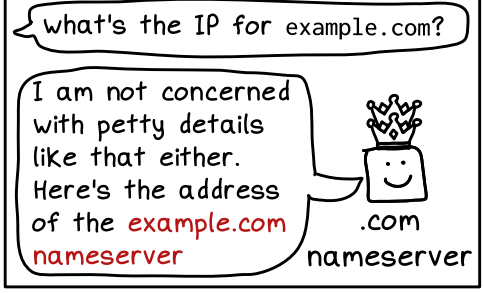
there are 3 main levels of authoritative DNS servers



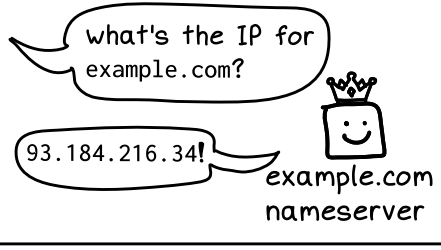
the root nameserver delegates



the .com nameserver also delegates

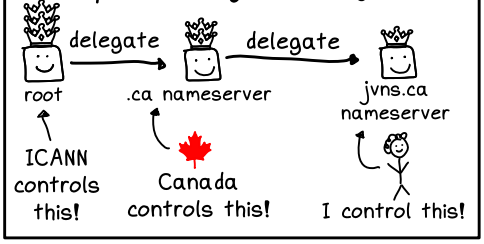


the example.com nameserver actually answers your questions

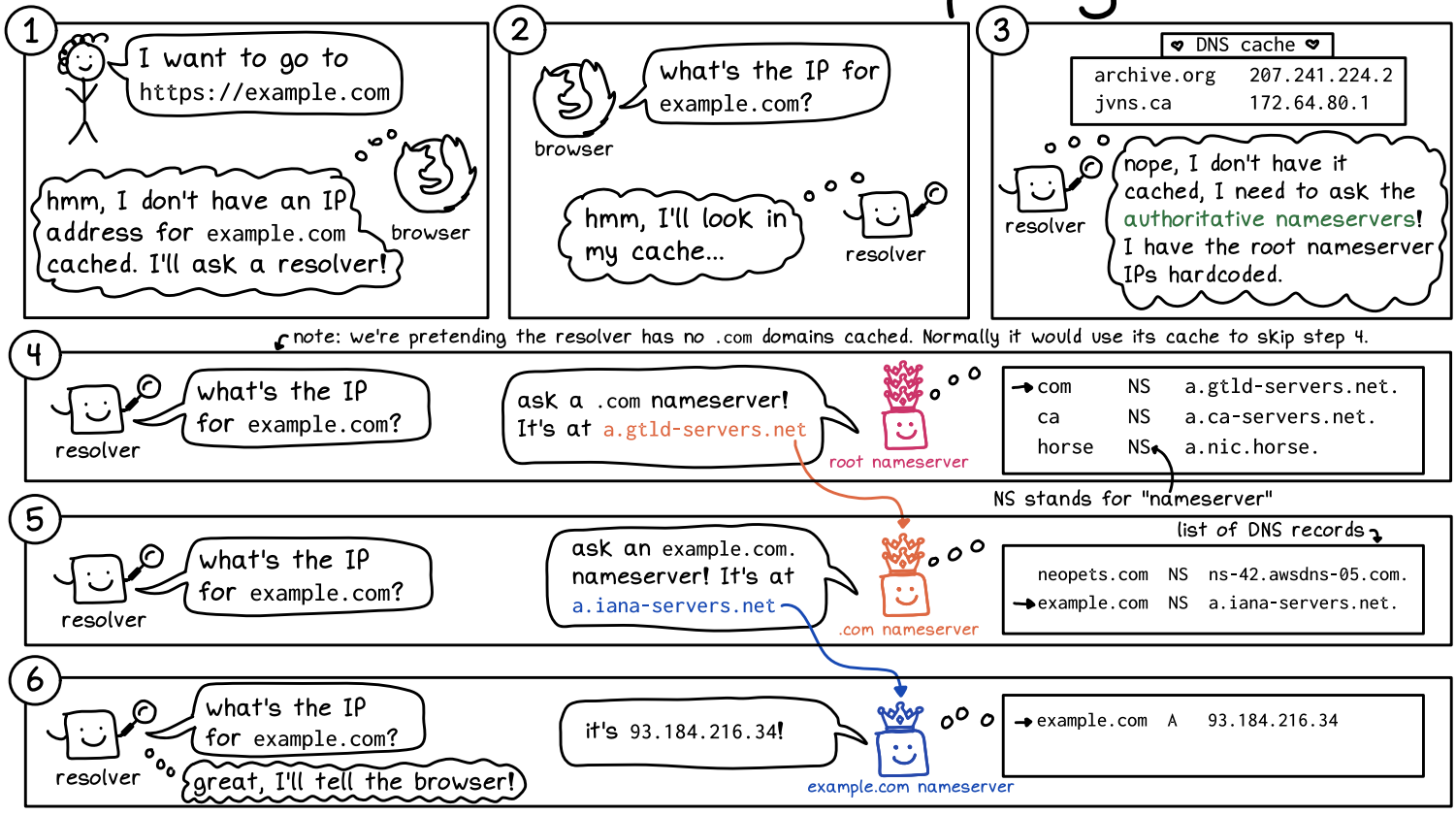


this design lets DNS be decentralized

example: for my domain jvns.ca



life of a DNS query



DNS records

When you make DNS changes for your domain, you're editing a DNS record

Here's what the same record looks like with dig ← we'll explain dig on page 18

```
$ dig +noall +answer paw.examplecat.com
paw.examplecat.com.    60    IN    A    1.2.3.4
```

DNS records have 5 parts

- name (eg tail.examplecat.com)
- type (eg CNAME)
- value (eg tail.jvns.ca)
- TTL (eg 60)
- class (eg IN)

different record types have different kinds of values: A records have an IP address, and CNAME records have a domain name.

name

paw.examplecat.com

When you create a record, you'll usually write just the subdomain (like paw).

When you query for a record, you'll get the whole domain name (like paw.examplecat.com).

TTL

60

"time to live". How long to cache the record for, in seconds.

record type

A

"A" stands for "IPv4 Address".

class

IN

"IN" stands for "INternet". You can ignore it, it's always the same.

value

1.2.3.4

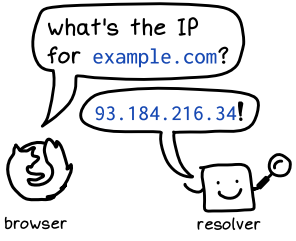
the IP address we asked for!

everything inside a DNS packet

I literally mean everything, I copied this verbatim from a real DNS request using Wireshark.

(DNS packets are binary but we're showing a human-readable representation here)

Let's look at the actual data being sent during a DNS query:



request

```
Query ID: 0x05a8
Flags:    0x1000
Questions:            1
Answer records:       0
Authority records:    0
Additional records:   0
Question:
  Name:   example.com
  Type:   A
  Class:  IN
```

- Query ID: randomly generated
- Flags: these flags just mean "this is a request"
- Type: A is for IPv4 Address. other types: MX, CNAME, AAAA, etc
- Class: IN stands for "INternet"

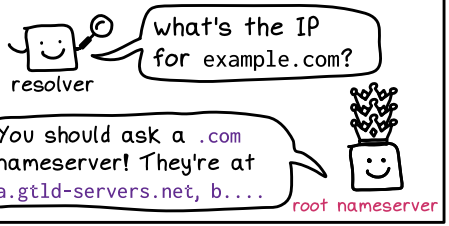
response

```
Query ID: 0x05a8
Flags:    0x8580
Questions:            1
Answer records:       1
Authority records:    0
Additional records:   0
Question:
  Name:   example.com
  Type:   A
  Class:  IN
Answer records:
  Name:     example.com
  Type:     A
  Class:    IN
  TTL:      86400
  Content:  93.184.216.34
Authority records:
  (empty)
Additional records:
  (empty)
```

- Query ID: matches request ID
- Flags: the response code is encoded in the last 4 bits of these flags. The 3 main response codes are:
  - NOERROR (success!)
  - NXDOMAIN (doesn't exist!)
  - SERVFAIL (error!)
- Question: copied from request
- Answer records: domain names aren't case sensitive
- Content: the IP we asked for
- Authority records / Additional records: page 12 ("NS records") talks more about these 2 sections

the root nameservers

every DNS resolver starts with a root nameserver



root nameserver IP addresses almost never change

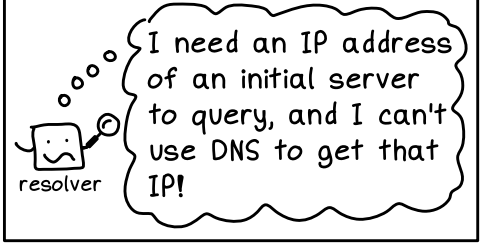
a.root-servers.net's IP (198.41.0.4) hasn't changed since 1993. ← DECADES ago!

there are thousands of physical root nameservers, but only 13 IP addresses

Each IP refers to multiple physical servers, you'll get the one closest to you. ← this is called "anycast"

There's a map at https://root-servers.org

if they didn't exist, resolvers wouldn't know where to start



every resolver has the root IPs hardcoded in its source code

example: https://wzrd.page/bind

Here they are! →

You can query one like this:

```
dig @198.41.0.4 example.com
```

All the IPs will give you the exact same results, there are just lots of them for redundancy.

a.root-servers.net	198.41.0.4
b.root-servers.net	199.9.14.201
c.root-servers.net	192.33.4.12
d.root-servers.net	199.7.91.13
e.root-servers.net	192.203.230.10
f.root-servers.net	192.5.5.241
g.root-servers.net	192.112.36.4
h.root-servers.net	198.97.190.53
i.root-servers.net	192.36.148.17
j.root-servers.net	192.58.128.30
k.root-servers.net	193.0.14.129
l.root-servers.net	199.7.83.42
m.root-servers.net	202.12.27.33

your domain's authoritative nameservers

11

when you register a domain, your registrar runs your authoritative nameservers by default

You can change your nameservers in your registrar's control panel.

LOTS of services can be your authoritative nameserver

how to find your domain's nameservers

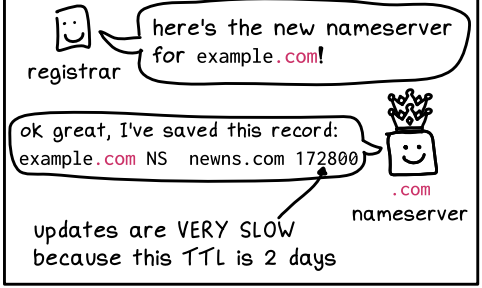
```
$ dig +short NS neopets.com
ns-42.awsdns-05.com.
ns-1191.awsdns-20.org.
```

neopets.com is using AWS's nameservers right now

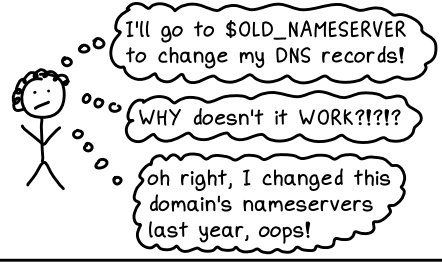
how to change your nameservers

1. Copy your DNS records to the new nameservers
   (use `dig` to check that it worked)
2. On your registrar's website, update your nameservers
3. Wait 48 hours
4. Delete the old DNS records
   (to save your future self confusion)

why changing your nameservers is slow

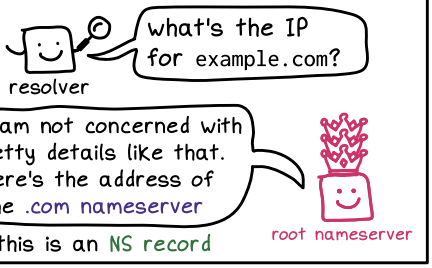


what can go wrong if you don't delete the old records



NS records

What's actually happening when the root nameserver redirects to the .com nameserver, on page 6?



The root nameserver can return two kinds of DNS records:

NS records: (in the Authority section)

```
com.   172800 NS a.gtld-servers.net
com.   172800 NS b.gtld-servers.net
```

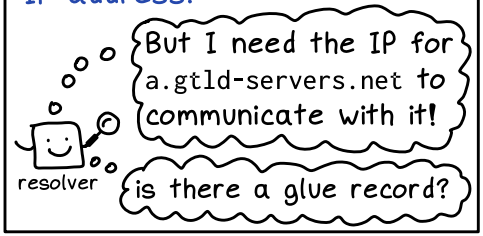
name, TTL, type, value

glue records: (in the Additional section)

```
a.gtld-servers.net 86400 A 192.5.6.30
b.gtld-servers.net 86400 A 192.33.14.30
```

name, TTL, type, value

The NS record gives you the domain name of the server to talk to next, but not its IP address.



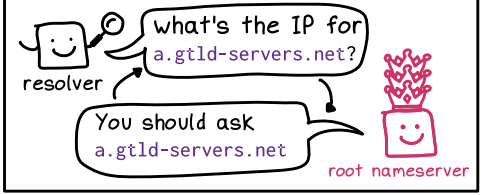
2 ways the resolver gets the IP address

① If it sees a glue record for a.gtld-servers.net, the resolver will use that IP

② otherwise, it'll start a whole separate DNS lookup for a.gtld-servers.net

glue records help resolvers avoid infinite loops

without a glue record for a.gtld-servers.net: disaster!



terminology note

NS records are DNS records with type "NS".

Also, an "A record" means "record with type A", "MX record" means "record with type MX", etc.

(confusingly, this is not true for glue records, glue records have type A or AAAA. It's weird, I know.)

subdomains

to make a subdomain, you just have to set a DNS record!

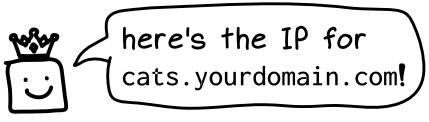
To set up `cats.yourdomain.com`, create a DNS record like this in your authoritative nameservers:

```
cats.yourdomain.com   A   1.2.3.4
```

name ↑ record type ↑ value ↑

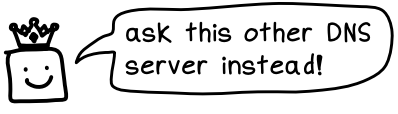
there are 2 ways a nameserver can handle subdomains

① Store their DNS records itself



② Redirect to another authoritative nameserver

(this happens if you set an `NS` record for the subdomain, it's called "delegation")



you can create multiple levels of subdomains

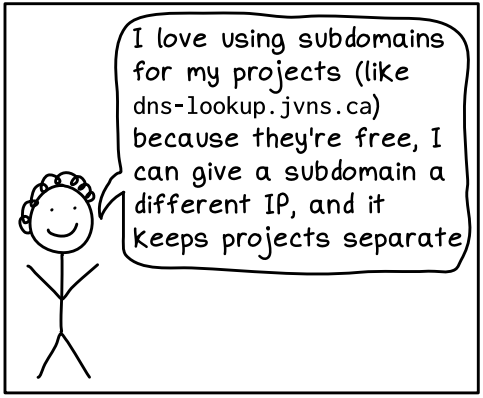
For example, you can make:

`a.b.c.d.e.f.g.example.com`

up to 127 levels is allowed!

www is a common subdomain

Usually `www.yourdomain.com` and `yourdomain.com` point to the exact same IP address. If you wanted to confuse people, you could make them totally different websites!



why DNS updates are slow: caching

You might have heard that DNS updates need time to "propagate".

What's actually happening is that there are old cached records which need to expire.

DNS records are cached in many places

- browser caches
- DNS resolver caches
- operating system caches

google.com: my DNS records are cached on billions of devices!

let's see what happens when you update an IP

bananas.com	A	300	1.2.3.4
		60	5.6.7.8

beware: even if you change the TTL to 60s, you still have to wait 300 seconds for the old record to expire

30 seconds later...

(you go to bananas.com in your browser)

browser: hey what's the IP for bananas.com?

resolver: let's check my cache for bananas.com... found it!!

resolver: it's 1.2.3.4!

400 seconds later...

(you refresh the page again)

browser: hey what's the IP for bananas.com?

resolver: The TTL (300s) is up, better ask for a new IP...

resolver: it's 5.6.7.8!

12 hours later...

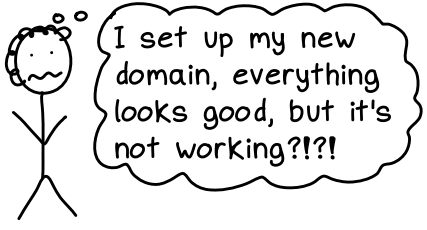
(you check 1.2.3.4's logs to make sure all the traffic has moved over)

that's weird, the old server is still getting a few requests...

the culprit: a rogue DNS resolver: I don't care about your TTL! I just cache everything for 24 hours!

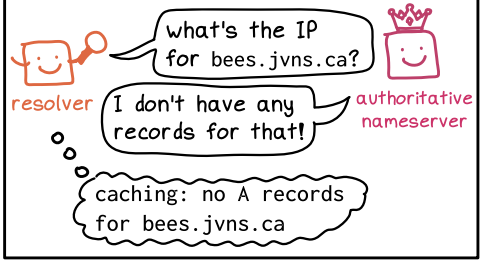
negative caching

Here's a problem I've had many times



I finally learned last year that my problem was "negative caching"

resolvers cache negative results



the TTL for caching negative results comes from the SOA record

```
example.com. 3600 IN SOA ns.icann.org. noc.dns.icann.org. 2021120741 7200 3600 1209600 3600
```

it's the smaller of these 2 numbers (in this case 3600 seconds)

what you need to know about SOA records

1. they control the negative caching TTL
2. you can't change them (unless you run your own authoritative nameserver)
3. how to find yours: `dig SOA yourdomain.com`

how to avoid this problem

Just make sure not to visit your domain before creating its DNS record! That's it!

(if you really want more details, see RFC 2308)

resolvers can lie

When a resolver gets a DNS query, it has 2 options:

I could tell you what the authoritative nameservers said... or I could LIE!

reason to lie: block ads / malware

what's the IP for doubleclick.net?

ad domain, definitely exists

that domain doesn't exist

resolver

PiHole blocks ads this way.

reason to lie: (rude!) to show you ads

what's the IP for zzz.jvns.ca?

doesn't exist

here's an IP that will show you ads!

resolver

This is called "DNS hijacking".

reason to "lie": internal domain names

what's the IP for corp.examplecat.com?

doesn't exist on the public internet

here's an internal IP address!

corporate resolver

airport DNS resolvers sometimes lie

what's the IP for google.com?

you didn't log in yet so I will lie! here is our login page's IP!

airport resolver

how does your computer know which resolver to use?

When you connect to a network, the router tells your computer which search domain and resolver to use (using DHCP).

router

resolver: 192.168.1.1
search domain: lan

a tiny DNS resolver* (in Python)


```python
def resolve(domain):
    # Start at a root nameserver
    nameserver = "198.41.0.4"
    # A "real" resolver would check its cache here
    while True:
        reply = query(domain, nameserver)
        ip = get_answer(reply)
        if ip:
            # Best case: we get an answer to our query and we're done
            return ip
        nameserver_ip = get_glue(reply)
        if nameserver_ip:
            # Second best: we get the *IP address* of the nameserver to ask next
            nameserver = nameserver_ip
        else:
            # Otherwise: we get the *domain name* of the nameserver to ask next
            nameserver_domain = get_nameserver(reply)
            nameserver = resolve(nameserver_domain)
```

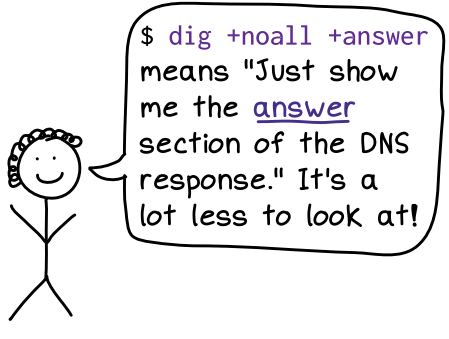
On page 5 (life of a DNS query), we saw how resolvers work. This code does the same thing, but it actually works.

* Actual DNS resolvers are more complicated than this, but this is the core algorithm.

let's meet dig

dig is my favourite tool for investigating DNS issues

I find its default output unnecessarily confusing, but it's the only standard tool I know that will give you all the details.



tiny guide to dig's full output

```
$ dig example.com
; <<>> DiG 9.16.24 <<>> +all example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27580
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;example.com.      IN    A

;; ANSWER SECTION:
example.com.   86400   IN    A    93.184.216.34
;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Jan 26 11:32:03 EST 2022
;; MSG SIZE  rcvd: 56
```

response code

the answer to our DNS query

The "." at the end means that example.com isn't a subdomain of some other domain (like it's not example.com.degrassi.ca). This might seem obvious, but DNS tools like to be unambiguous.

```
$ dig +noall +answer example.com
example.com.   86400   IN    A    93.184.216.34
```

name · TTL · class · record type · content

just the answer! so much less overwhelming!

dig command line arguments

the basics: dig @SERVER TYPE DOMAIN

both optional

Examples:

```
dig example.com

dig @8.8.8.8 example.com

dig TXT example.com

dig @8.8.8.8 NS example.com
```

default type: A

default server: from /etc/resolv.conf (on Linux)

tip: put `+noall +answer` in your `~/.digrc`

This makes your output more readable by default, and you can always go back to the full output with `dig +all`.

dig +noall

Hide all output.

Useless by itself, but `dig +noall +authority` will just show you the "Authority" section of the response.

dig +short DOMAIN

Only show the record content.

```
$ dig +short example.com
93.184.216.34
```

dig +trace DOMAIN

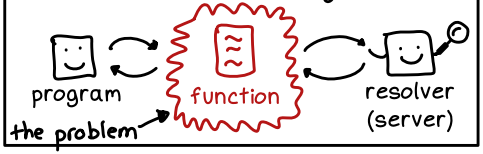
Traces how the domain gets resolved, starting at the root nameservers.

This avoids all the caches, which is useful to make sure you set your record correctly.

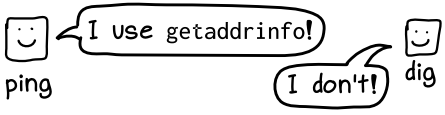
getaddrinfo

One weird thing about DNS is that different programs on a single computer can get different results for the same domain name.

Let's talk about why!



reason 1: many (but not all!!) programs use the function getaddrinfo for DNS lookups...



So if you see an error message like "getaddrinfo: nodename or servname not provided...", that's a DNS error.

... and not using getaddrinfo might give a different result

- the program might not use /etc/hosts (dig doesn't)
- the program might use a different DNS resolver (some browsers do this)

reason 2: there are many different versions of getaddrinfo...

- the one in glibc
- the one in musl libc
- the one in Mac OS

And of course, they all behave slightly differently :)

you can have multiple getaddrinfos on your computer at the same time

For example on a Mac, there's your system getaddrinfo, but you might also be running a container that's using musl.

glibc and musl getaddrinfo are configured with /etc/resolv.conf

IP of resolver to use

```
# Generated by NetworkManager
nameserver 192.168.1.1
nameserver fd13:d987:748a::1
```

On a Mac, /etc/resolv.conf exists, but it's not used by the system getaddrinfo.

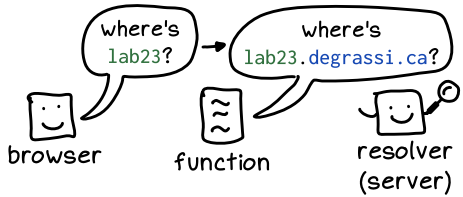
search domains

In an internal network (like in a company or school), sometimes you can connect to a machine by just typing its name, like this:

```
$ ping labcomputer-23
```

Let's talk about how that works!

many DNS lookup functions support "local" domain names



(the function appends a base domain degrassi.ca to the end)

the base domain is called a "search domain"

On Linux, search domains are configured in /etc/resolv.conf

Example:

```
search degrassi.ca
```

this tells getaddrinfo to turn lab23 into lab23.degrassi.ca

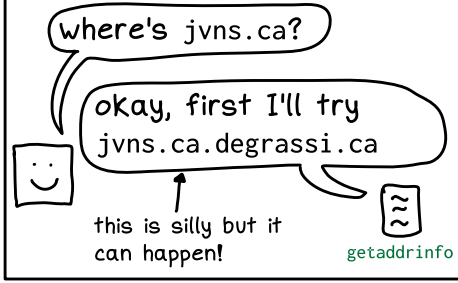
getaddrinfo doesn't always use search domains

It uses an option called ndots to decide.

```
search degrassi.ca
options ndots:5
```

this means "only use search domains if the domain name contains less than 5 dots"

search domains can make DNS queries slower

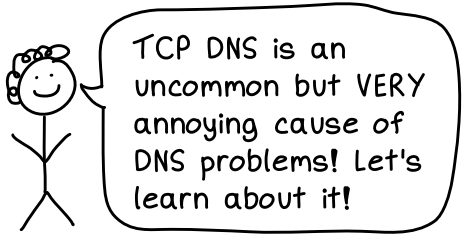


avoid search domains by putting a "." at the end

Use http://jvns.ca. instead of http://jvns.ca

TCP DNS

If you manage servers, sometimes DNS just breaks for no obvious reason

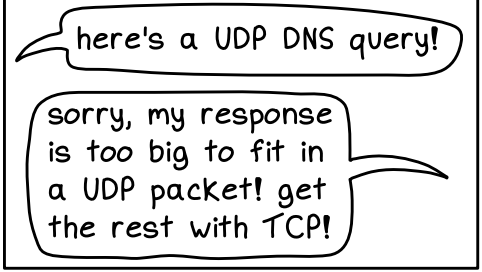


DNS queries can use either UDP or TCP

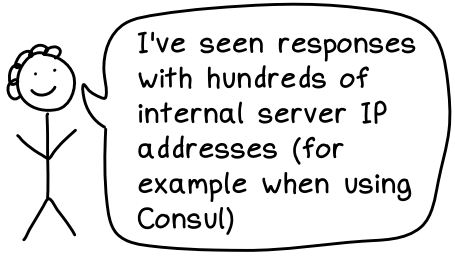
A UDP DNS response has to be less than 4096 bytes. UDP is the default.

TCP can send an unlimited amount of data. It's only used when UDP wouldn't work.

large DNS responses automatically use TCP



what's in a giant DNS response?



how not supporting TCP DNS can ruin your day

1. your server is happily making UDP DNS queries
2. one day, the responses get bigger and switch to TCP
3. oh no! the queries fail!

2 reasons TCP DNS might not work

1. some DNS libraries (like `musl`'s `getaddrinfo`) don't support TCP. This is why DNS sometimes breaks in Alpine Linux.
2. it could be blocked by your firewall. You should open both UDP port 53 and TCP port 53.

A & AAAA records

there are two kinds of IP addresses: IPv4 and IPv6

Every website needs an IPv4 address.

IPv6 addresses are optional.

A stands for IPv4 Address

Example: 93.184.216.34

AAAA stands for IPv6 AAAAddress

Example: (joke, but kinda true)

2606:2800:220:1:248:1893:25c8:1946

it's called AAAA (4 As) because IPv6 addresses have 4x as many bytes

in theory, the Internet is moving from IPv4 to IPv6

This is because there are only 4 billion IPv4 addresses (the internet has grown a LOT since the 1980s when IPv4 was designed!)

happy eyeballs*

If your domain has both an A and an AAAA record, clients will use an algorithm called "happy eyeballs" to decide whether IPv4 or IPv6 will be faster.

* yes that is the real name

using IPv6 isn't always easy

→ not all web hosts give you an IPv6 address

→ lots of ISPs don't support IPv6 (mine doesn't!)

IP addresses have owners

You can find any IP's owner by looking up its ASN ("Autonomous System Number").

(except local IPs like 192.168.x.x, 127.x.x.x, 10.x.x.x, 172.16.x.x)

CNAME records

there are 2 ways to set up DNS for a website

① set an A record with an IP

```
www.cats.com    A    1.2.3.4
```

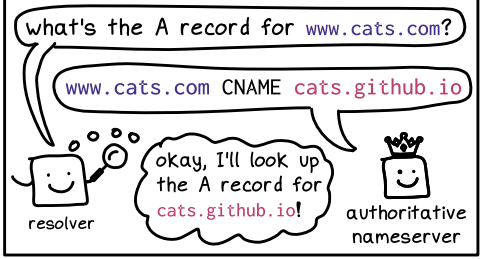
② set a CNAME record with a domain name

```
www.cats.com CNAME cats.github.io
```

CNAME records redirect every DNS record, not just the IP

I like to use them whenever possible so that if my web host's IP changes, I don't need to change anything!

what actually happens during a CNAME redirect



rules for when you can use CNAME records

① you can only set CNAME records on subdomains (like www.example.com), not root domains (like example.com)

② if you have a CNAME record for a subdomain, that subdomain can't have any other records

(technically you can ignore these rules, but it can cause problems, the RFCs say you shouldn't, and many DNS providers enforce these rules)

some DNS providers have workarounds to support CNAME for root domains

Look up "CNAME flattening" or "ANAME" to learn more.

MX records

there are two important problems in email

From: kermit@frog.com
To: julia@example.com

1. Make sure the message gets to the right recipient.

   This is what MX records are for.

2. Make sure the sender didn't lie about their From: address.

   This is what SPF, DKIM, and DMARC records are for.

   SPF/DKIM/DMARC are very complicated but we'll give a tiny incomplete summary.

MX records tell you the mail server for a domain

```
$ dig +short MX gmail.com
5 gmail-smtp-in.l.google.com.
```

priority — server's domain name

copy and paste your MX records

tiny guide to SPF/DKIM/DMARC records

SPF: list of allowed sender IP addresses

Example: `v=spf1 ip4:2.3.4.5 -all`

DKIM: sender's public key

Example: `v=DKIM1; k=rsa; p=MIGFMA0GCSqGSI.....`

DMARC: what to do about SPF/DKIM failures

Example: `v=DMARC1; p=reject; rua=mailto:dmarc@example.com`

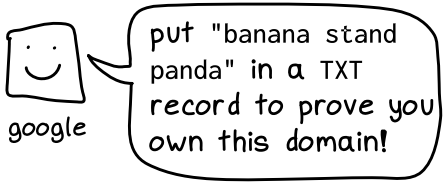
TXT records & more

TXT records can contain literally anything

```
examplecat.com TXT   "hello! I'm
                      an example
                      cat!"
```

(though they're usually ASCII)

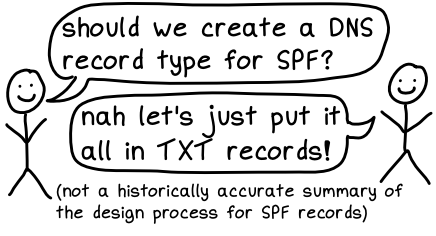
they're often used to verify that you own your domain



reasons to verify your domain

- to issue SSL certificates with Let's Encrypt
- to use Single Sign On (SSO) for a service
- to get access to Google/Facebook's data about your domain (eg search data)

they're also used for email security (SPF/DKIM/DMARC)



TXT records can contain many strings

Each string is at most 256 characters, and clients will concatenate them together.

You'll see this in DKIM records, because they're usually more than 256 characters.

some other record types

CAA: restrict who can issue certificates for your domain

PTR: reverse DNS -- map IP addresses to domain names (look these up with `dig -x`)

SRV: holds both an IP address and a port number

thanks for reading

As with everything, I think the best way to learn more about DNS is to experiment and break things. So this zine comes with a playground!

https://messwithdns.net

More DNS tools I made while writing this zine:

https://dns-lookup.jvns.ca	(to make DNS queries)
https://github.com/jvns/tiny-resolver	(to see how resolvers work)

credits

Cover art: Vladimir Kašiković
Copy editing: Gersande La Flèche
Technical review: Ray Bejjani
Editing: Dolly Lanuza, Kamal Marhubi
Pairing: Marie Claire LeBlanc Flanagan
Any remaining errors: mine

♡ this?
more at
★ wizardzines.com ★