Hostnames were not enough.

Originally, everything just had a hostname, and you either memorized the IP address or added it to /etc/hosts.

Managing hosts files was a problem, resulting in at least two systems to solve discoverability: NIS and DNS.

NIS was invented by Sun and assumed that every NIS client was able to store and query a map of the entire local network.

DNS worked for non-local hosts, and did not require publishing a network map to clients.

Resolver libraries were needed.

NIS worked well for local networks and DNS worked well for offsite networks.

To query both systems usefully, various folks wrote resolver libraries.

Linux servers typically use simple resolvers; all lookups are unevenly distributed across a set of DNS servers.

macOS and Windows offer more nuanced resolvers; e.g. VPN domains are resolved using the VPN.

Linux name resolution is relatively simple.

Most applications offload name lookups to libresolv, which uses /etc/resolv.conf and /etc/nsswitch.conf.

/etc/nsswitch.conf by default says “look up the name in /etc/hosts first and DNS second”. We do not alter this file.

/etc/resolv.conf specifies one or more DNS servers, a default domain name for lookups, and zero or more options.

libresolv often prefers the first nameserver over others and is not particularly resilient to outages.

HA name resolution is somewhat simple.

Daemons have been developed to provide a reliable local DNS instance that is then listed first in /etc/resolv.conf.

They forward requests to the datacenter DNS servers, cache replies, and offer TrafficScript-like abilities to modify queries.

We typically use 'dnsmasq' in a simple forwarding configuration with limited caching and no rewriting.

'dnsmasq' is smart enough to try multiple resolvers, to balance queries correctly across them, and is very stable.

Names: "I do not think it means what you think it means".

Each of these are different: Hostnames, domain names, fully-qualified domain names, subdomains, domains, top-level domains.

To use one example: genericadm, genericadm.private, genericadm.private.phx1.mozilla.com, phx1.mozilla.com, mozilla.com, com

Hostnames have 0 dots. Domain names have 1+ dots. FQDNs are domain names that end a domain issued by a TLD.

These distinctions are extremely difficult to keep track of and can lead to significant confusion in everyday operations work.

Name resolution is hard.

/etc/resolv.conf option "ndots:1" (the default) appends the default domain if the name being resolved has less than 1 dot.

Given "domain mozilla.com", resolving the name 'genericadm' would result in a lookup for 'genericadm.mozilla.com'.

Given "ndots:4", resolving the name 'genericadm.private.scl3' would result in a lookup for 'genericadm.private.scl3.mozilla.com'.

We choose not to alter the default from "ndots:1" and instead require FQDNs in all use cases, sidestepping all of this.

DNS lookups are performed right-to-left until referrals stop.

To resolve ‘genericadm.private.scl3.mozilla.com.’, start with the FQDN and reverse it: .com.mozilla.scl3.private.genericadm

The first dot indicates an empty string — “”. — which indicates that the DNS resolver must start with the root nameservers.

The resolver transmit the FQDN being looked up to the root, which responds with the “com” nameservers.

The resolver repeats its query to the “com” nameservers, and then to the “mozilla.com”, etc, until a server replies with a result.

DNS can provide much more than IP addresses.

DNS is a key-value store, using a two-column primary key of FQDN and Record Type.

Record types are defined as integers by the specification: 1, 12, 15, 257

Many record types have human-recognizable labels: A, PTR, MX, CAA

Only a few record types are typically supported: A, AAAA, CNAME, PTR, MX, SOA, TXT

Why BIND zone files rarely (or never) contain FQDNs.

You can use the special FQDN ‘@‘ to say “the domain name BIND is configured to associate with this data file”.

You can use a blank FQDN to say “the same FQDN as the previous line”.

If you don’t terminate the FQDN with a dot, BIND assumes you want to append that ‘@‘ domain name to your record.

This is why most zone files start with @ IN SOA, and then say IN MX, IN A, IN TXT, and then “genericadm IN CNAME”.

CNAME, DNAME records are the equivalent of symlinks in DNS.

These records allow us to indicate that a given FQDN is ‘symlinked’ to another FQDN.

The target FQDN may be outside of your domain name, and the target’s permission is not required.

For legacy reasons the DNS specification prohibits the use of CNAME records on domain names, e.g. mozilla.com

AWS and others offer custom DNS servers that implement top-level aliases using custom records such as DNAME.

CAA records are used to control SSL issuance for domains.

As of this year, SSL issuers require us to provide a valid response for CAA requests for domains.

That response must be NOERROR. A misconfigured domain may return SERVFAIL or time out without replying.

If zero records are returned in the response, then the issuer assumes they are permitted to proceed.

If one or more records are returned, then the issuer will refuse to proceed unless they are listed amongst those records.

Why Inventory and Akamai have trouble supporting CAA.

CAA (257) is an uncommon record type that was invented relatively recently and is not widely understood.

Inventory does not offer any UX support for records of this type, as it was not considered necessary back then.

Akamai does not accept this record type over the AXFR transfer method we chose for uploading zone files to them.

This type of problem occurs frequently when trying to use uncommon DNS record types: CAA, DANE, SRV, etc.

SPF, DKIM, DMARC records control email relaying/accepting behaviors for domains.

Each record is a TXT record associated with a domain name. For example: metrics.mozilla.com, _dmarc.mozilla.com

When email servers process messages, they validate properties of the RFC822 headers against these DNS records.

DKIM and DMARC verify a crypto-signature in the headers against a public key listed in DNS under a special hostname.

SPF verifies the originating IP address against a policy listed in DNS in a special domain-level TXT record.

DNSSEC records permit nameservers to validate our authority to define DNS records.

DNSSEC uses DNSKEY, DS, NSEC, RRSIG records to prove via crypto-signatures that we control a domain's records.

They require cooperation between DNS providers, top-level domains (e.g. "com"), and the root nameservers.

Implementation of DNSSEC is very difficult and requires careful planning (especially for key rotation) and validation.

A detailed explanation of how DNSSEC works is beyond the scope of this training, but is readily available online.