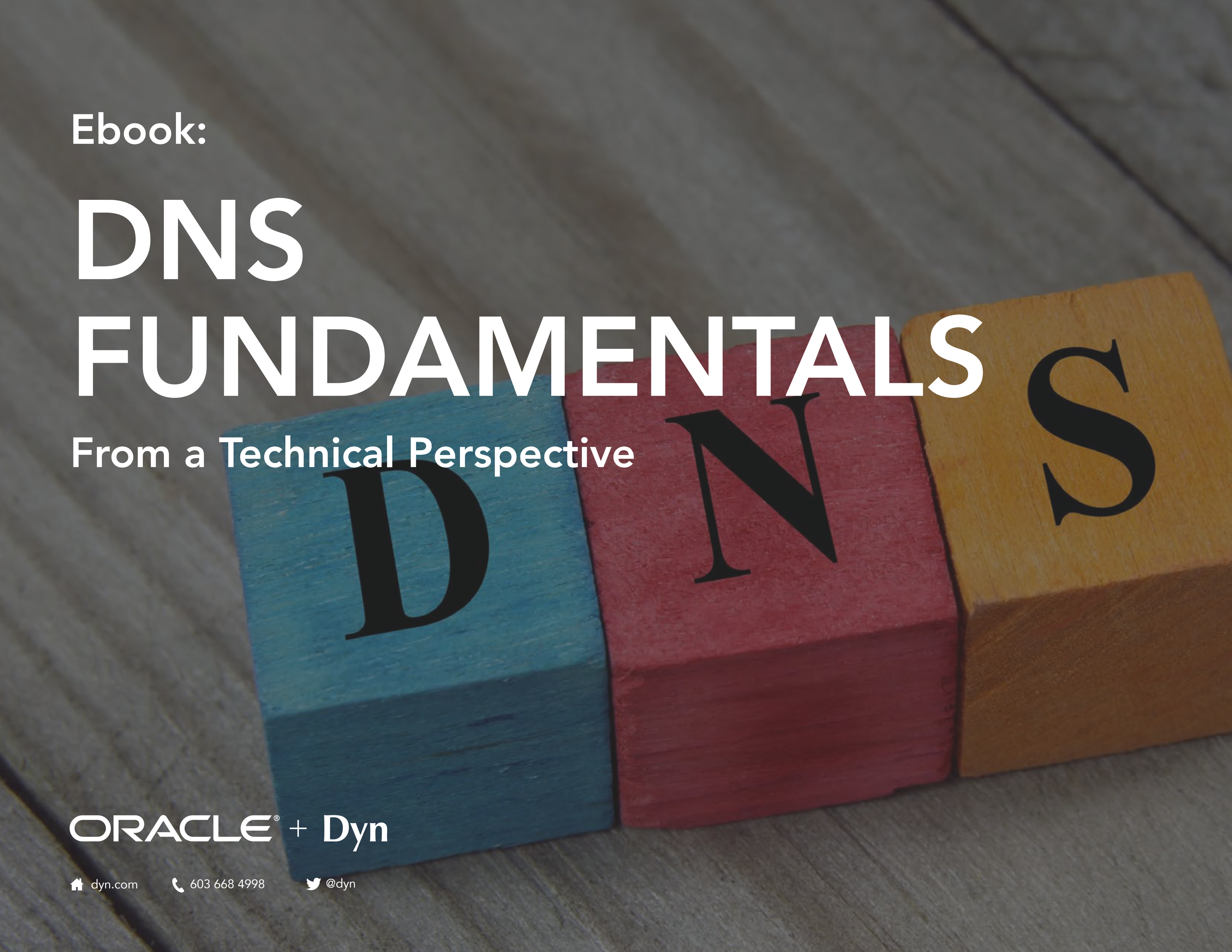


Ebook:

DNS FUNDAMENTALS

From a Technical Perspective

ORACLE® + Dyn

dyn.com ☎ 603 668 4998 @dyn

DNS Fundamentals

From a Technical Perspective

Introduction:

Understanding fundamental Domain Name System (DNS) concepts is a critical part of your understanding of how the internet works. DNS is the mechanism that helps find the network endpoint that you're trying to reach. This paper will cover basic **DNS terminology**, in general terms, with nothing vendor-specific or proprietary. There are many opinions on this subject, so this paper will lay out multiple points of view, wherever possible. The goal is to provide a good representation of all these terms and how they function in the real world.

Global DNS

Let's begin by defining the actors on the stage. The DNS can really be broken down from a request perspective: There are authoritative servers, and there are things that talk to those authoritative servers. Often, the things that talk to the authoritative servers are doing it on behalf of someone else, which are "stubs." As we'll discuss later, there are a number of things you can be authoritative for.

This can't possibly be repeated enough: **users** are our eyeballs on the edge of the internet accessing internet resources. Whenever we hit a URL like http://www.dyn.com/blog, our devices parse it out into multiple parts.

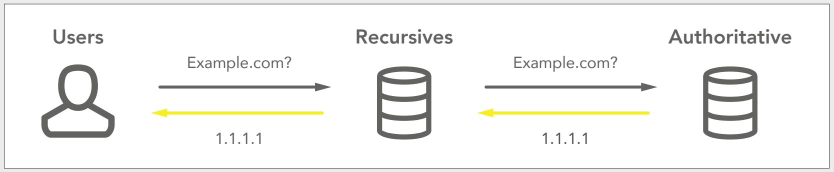
Your browser first **uses DNS to look up** the target hostname's IP address (or the URL may have an IP-literal hostname that directly specifies an IP address, skipping DNS altogether). The URL has some distinct parts. The beginning is the scheme. In general terms, it says which protocol is in use (in this case, http).

After the two slashes comes some network location information, which is almost always an Internet **host** expressed as a **domain name** that can be looked up in the DNS. In our example, this is **www.dyn.com**. There are other kinds of information that can be here, too, but a domain name is most common. After another slash comes the **path** (in this case, blog) which says where the desired resource is found on the host. To get to the blog, you must know how to find the host on the internet. This is where the DNS comes in, using a process called **name resolution**.

How a Request Is Resolved

Most of the time, when you resolve a name on the Internet, you are working on an endpoint—your phone, tablet, laptop, or some other device. On that device you usually find a minimal resolver called a **stub resolver** (often, just "stub"). It's called a stub because it can't do very much. It knows it needs an answer to a question about a domain name. So, it asks something else: it sends its resolution requests to another service on the network, called a **recursive resolver**.

For most of the world, the default designated **recursive resolver** (often just "recursive") is a server provided by the local internet service provider (**ISP**) or the business maintaining the network. The job of the recursive is to make requests to the larger DNS ecosystem on behalf of the user, allowing for economies of scale and freeing up resources for the user. Ultimately, the recursive will ask a series of DNS servers that are authoritative for components of the DNS in order to process the request and hand back to the user the IP of the DNS name that was requested. It looks broadly like this:



When the recursive is interacting with the DNS, it is navigating the largest distributed database in the world. The DNS is formatted in a large tree structure. Each node has a label, which is zero to 63 octets in length.

Like any tree structure it has a beginning known as root. If we go back to our www.dyn.com example, the actual real host has a usually unrepresented "." at the end - www.dyn.com. - which represents the root zone.

The recursive will check its cache left to right (do I have www.dyn.com? Dyn.com? Com?). However, if it doesn't have any of the answers in cache, the recursive goes to the root to get things started. The idea of the **root** is to provide an origin to the query, providing the nameservers for all the top-level domains (**TLD**) such as com, net, fr, edu, and others. The root then delegates the authority for the namespace of that name to the authoritative DNS of an organization designated to run that TLD independently. This process is intuitively called a **delegation**.

TLDs come in a couple varieties. First, they can be run for a country (like .fr for France and .sg for Singapore), and these are called **ccTLDs**. They can also be generic like .com or .net. These are called **gTLDs**. Lastly, there is a variation of gTLD, which is wholly owned and operated by an organization as though it were a normal domain like .nike. These are called colloquially ".brand" TLDs, even though they are just commercially special TLDs.

Sometimes, TLDs will divide up their **namespace** a little further before allowing names to be individually registered, such as breaking out edu.sg, co.uk, or gov.il. These are called Second Level Domains, or **SLDs**. It's important to remember that there is no real technical difference between any "level" of domain name—they all work the same way. But there are administrative and policy differences at the root and in the TLDs and SLDs, which is why they have special names.

Registering a Domain and Creating a Zone

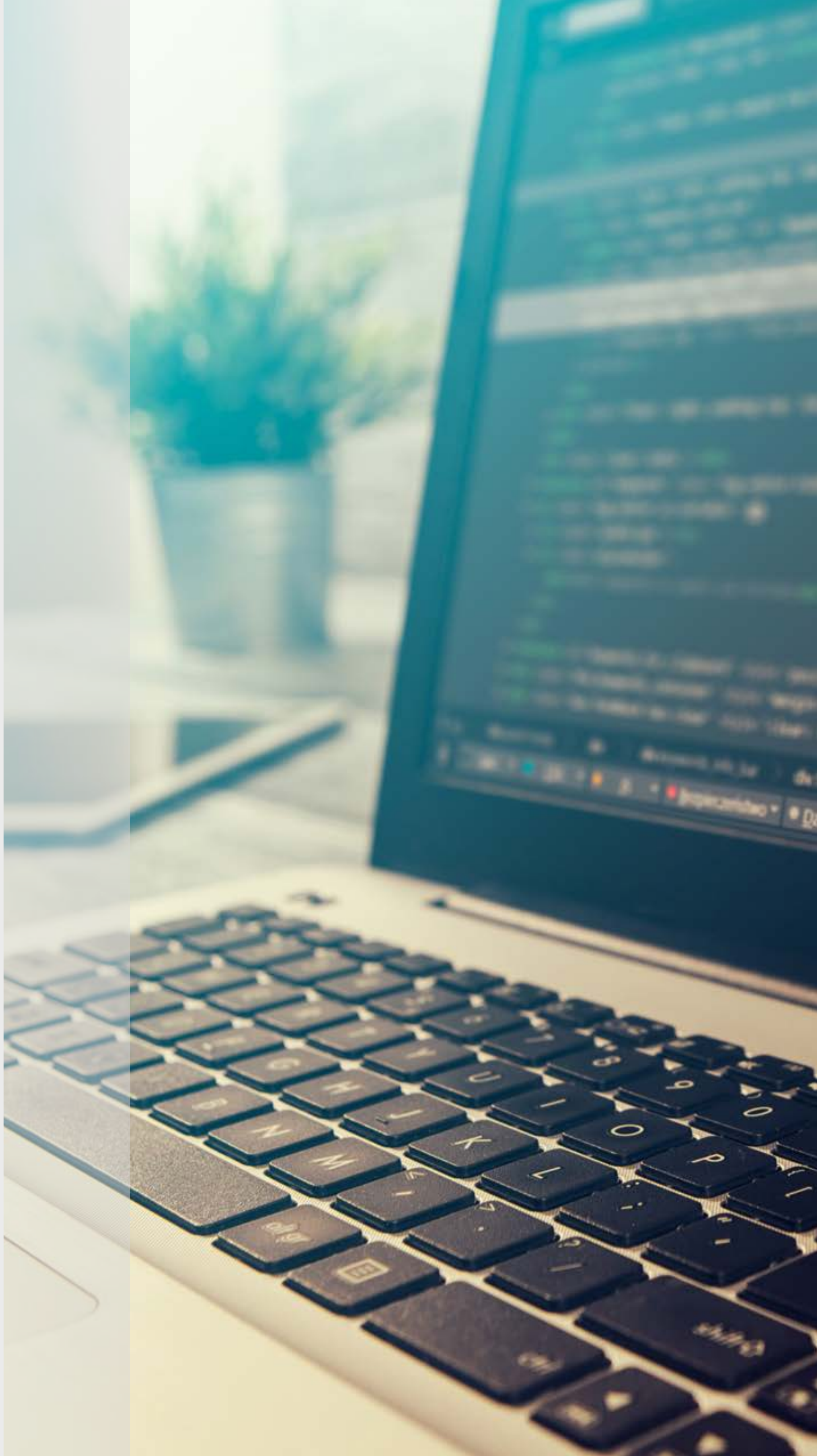
Most TLDs are commercially operated, and lease off portions of their namespace for private operation. The group that maintains a TLD is known as a **Registry**, while the authorized resellers of those names are **Registrars**. Therefore, when you go to the Dyn website and register wacky-awesome-cats.info, you are interacting with a registrar for the .info registry.

When you register a domain, a few things happen. You register the domain, which designates the DNS namespace of that **domain**—and all its children—to you. This may initiate the process to create a DNS **zone**, a unit of DNS administration. The registration of the name, if it's permitted, creates an allocation of that name space. But there's no requirement to add nameservers. And if you don't, you have a name without a delegation. If you add at least two nameservers—and assuming everything else is ok—you have a delegation (that is, a pointer in the parent zone to your nameservers).

If those nameservers are not responding authoritatively, then the delegation is **lame**. If they respond authoritatively and have an appropriate start of authority (SOA) record, then you have a proper delegation and a new zone.

The zone is fully established when a SOA record is created at the location the nameservers designated in the registration. This registered domain can spawn multiple zones within the total namespace, but there can be only one zone at the delegated name.

That is, if .com delegates example.com, there can only be one example.com zone. example.com could further delegate, to a.example.com and b.example.com, by adding nameserver entries at those names. But that's not at the registered domain—but at a different domain (a.example.com, for instance). And the operator might not even be different for these different zones—you can delegate something to yourself (that is, to another nameserver that you control).



When you register that domain, one of the things the registrar will ask is which DNS servers will act as the authority for your domain. This acts to delegate the authority of your domain to your own **authoritative** DNS servers, exactly as the root delegated the TLDs above. Do you sense a pattern? It's "**turtles on turtles**" all the way down. But, it can get a little confusing, because all of these things are called domain names and are delegated, so it is not always clear whether two domain names are under the control of the same operator.

Let's review before we get into the terms used to manage a zone at a DNS provider on a day-to-day basis. We registered a domain, which created a zone, via a registrar, acting for a registry of a TLD, which is a division of the DNS namespace below the root, through a process of delegations—all in order to provide answers to users through their recursives to our domain authoritative DNS.

What's in a Zone?

So far, everything has been largely behind the scenes and broadly consistent across the industry. It's really when things start being presented to users themselves that terminology starts to get squirrelly. The terminology used by implementers and developers of DNS protocols, and by operators of DNS systems, has changed in the decades since the DNS was first defined.

For the next set of terms, we should build a zone with just a few of the normal records we use on a day-to-day basis. Records are formally called **Resource Records** (RR), which are those individual entries in the zone.

If you're just starting to become familiar with DNS, you may notice that different types of RRs have different kinds of data: we have IPs (**A** for IPv4 and **AAAA** for IPv6) for things like webservers for the blog and APIs, mail servers; Mail Exchanger (**MX**) record for inbound mail; and a Service Record (**SRV**) for Voice over IP (VoIP).

The terms for various portions of the individual record are not too controversial. Each string between the dividing dots (www, example, com) are known as **labels**. The labels together form a DNS name that specifies the exact location (www.example.com.) in the DNS tree where a particular record can be found. This is often called the "name" of the record or a "host" in various DNS portals, but it is more properly called a domain name.

The next value is the Time to Live (**TTL**), which is the maximum value the record should be held in cache before a new query occurs. The record **type** lets us specify what kind of data goes in the record: some take IP addresses, some take domain names, one can take arbitrary text (**TXT**), and so on.

Not represented in a standard BIND zone file is the record **class**, which is nearly always IN for internet. There is also an option for CH for Chaos network, but is only really used by us DNS personnel to query DNS server information. Lastly, we have the meat of the record known variously as **RDATA** for Resource Data, **Value**, or sometimes, target. A group of all the records of a particular type with the same name and class exist as an **RRset**.

Some of the specific terms used to describe a scope for any location within the DNS tree are contentiously debated, both across organizations and within them. If the description below doesn't match what you might be used to. we will try to provide some context for the discrepancy.

The first is the location at the very top of that zone - the apex, represented by the @ symbol in our zone file, with the SOA record designating the **Start of Authority (SOA)** for the new zone with information on how to handle that zone in certain scenarios. The **apex** of a zone is the name of the location ("node") where the SOA record is located. There is always at least one nameserver record at this location, too. If the zone name was example.com, then the apex of that zone is also example.com, which may hold important records like TXT or MX for email functionality.

While this is sometimes called "the root of the zone," this should be discouraged, if possible. The root is a specific location at the root of all DNS names. If you think about it, it's as if you referred to the home directory on your computer as "root." People wouldn't know if you meant that location or actual root. That term is taken, and we have one for this concept, the apex.

Everywhere there is an SOA record, that is also an **apex** in the DNS distributed database. Domains can have "subdomains": that is just the relationship between two names. For instance, the example.com domain can have a subdomain, a.example.com. Also, example.com is a subdomain of com, and they're all subdomains of the root zone.

Every domain name might have several resource records RRsets: SOA if the name is an apex, NS, A, AAAA, MX, TXT, or other types. And the example.com zone might also have subdomains in it, because not every subdomain is a delegation. So, in the example.com zone file, you might find entries for a.long.domain.name.example.com.

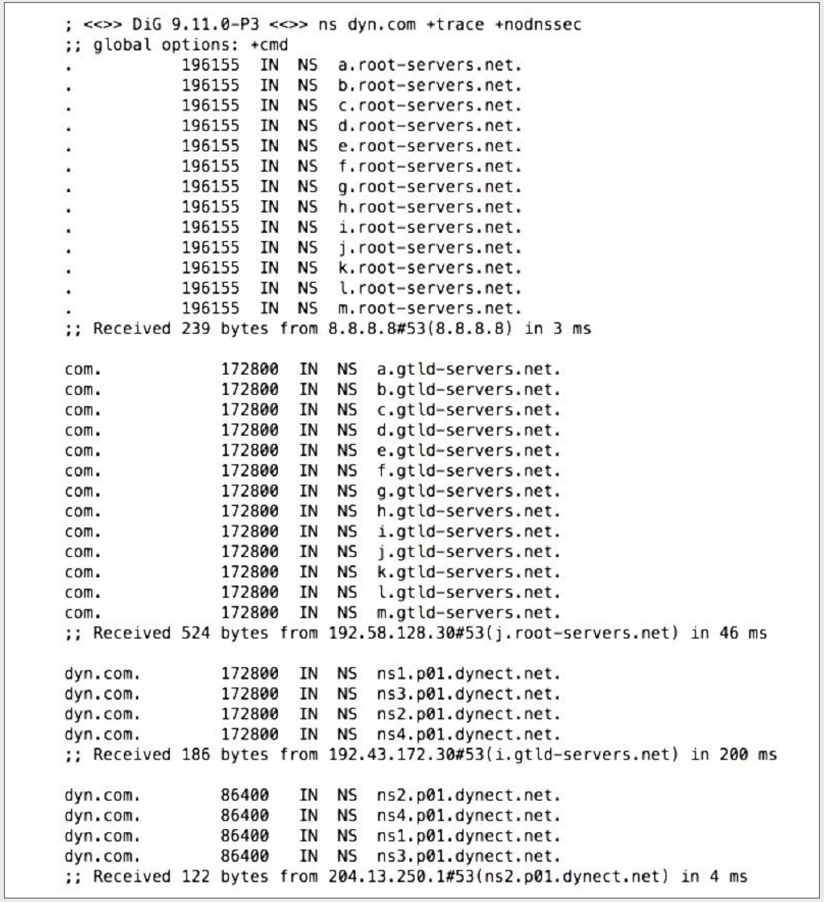
Therefore, there might be an entry for www.example.com, for instance, that is not an nameserver (NS) record. If so, then that's not a delegation, and it's in the example.com zone even though it is a subdomain.

The next attempt uses Fully Qualified Domain Name (**FQDN**) to refer to the entire string of labels with all parts. This is reasonably good, but there is an issue with the full qualification. A true FQDN must include all labels right down to the trailing root (www.example.com.). But on a day-to-day basis, most of us will present a domain name without the trailing dot (www.example.com). Does a FQDN include situations when the trailing dot is excluded? We find ourselves in a situation where many outside the DNS sphere haven't heard the term, but DNS experts are stuck wondering if it really is or isn't qualified.

The other major term is **hostname** or **host name**; both mean the same thing. As RFC 7719 points out, the DNS was born out of the use of host tables, so it is likely this term has existed since nearly the beginning and is in common use. There is a flaw, however. Technically speaking, not all FQDNs can be hostnames. Host names are only allowed to contain letters a-z (or A-Z), digits 0-9, and the hyphen. Domain names, on the other hand, can have special characters such as the underscores for _udp and _sip for our SRV record. There is a set of rules—discussed in detail in RFC 7719—for what may be a hostname.

DNS Delegation

So far we have covered the major components of a single zone and zone file, but the DNS is actually a **series of delegations**: the root . zone to the .com zone to the .example.com zone. How do those zones link together? This is done by that process mentioned earlier, delegation, in which one zone points the authority to the next in the chain. The process for dyn.com looks like this using my own computer through Google DNS:



```
; <<>> DiG 9.11.0-P3 <<>> ns dyn.com +trace +nodnssec
;; global options: +cmd
.           196155  IN  NS  a.root-servers.net.
.           196155  IN  NS  b.root-servers.net.
.           196155  IN  NS  c.root-servers.net.
.           196155  IN  NS  d.root-servers.net.
.           196155  IN  NS  e.root-servers.net.
.           196155  IN  NS  f.root-servers.net.
.           196155  IN  NS  g.root-servers.net.
.           196155  IN  NS  h.root-servers.net.
.           196155  IN  NS  i.root-servers.net.
.           196155  IN  NS  j.root-servers.net.
.           196155  IN  NS  k.root-servers.net.
.           196155  IN  NS  l.root-servers.net.
.           196155  IN  NS  m.root-servers.net.
;; Received 239 bytes from 8.8.8.8#53(8.8.8.8) in 3 ms

com.            172800  IN  NS  a.gtld-servers.net.
com.            172800  IN  NS  b.gtld-servers.net.
com.            172800  IN  NS  c.gtld-servers.net.
com.            172800  IN  NS  d.gtld-servers.net.
com.            172800  IN  NS  e.gtld-servers.net.
com.            172800  IN  NS  f.gtld-servers.net.
com.            172800  IN  NS  g.gtld-servers.net.
com.            172800  IN  NS  h.gtld-servers.net.
com.            172800  IN  NS  i.gtld-servers.net.
com.            172800  IN  NS  j.gtld-servers.net.
com.            172800  IN  NS  k.gtld-servers.net.
com.            172800  IN  NS  l.gtld-servers.net.
com.            172800  IN  NS  m.gtld-servers.net.
;; Received 524 bytes from 192.58.128.30#53(j.root-servers.net) in 46 ms

dyn.com.        172800  IN  NS  ns1.p01.dynect.net.
dyn.com.        172800  IN  NS  ns3.p01.dynect.net.
dyn.com.        172800  IN  NS  ns2.p01.dynect.net.
dyn.com.        172800  IN  NS  ns4.p01.dynect.net.
;; Received 186 bytes from 192.43.172.30#53(i.gtld-servers.net) in 200 ms

dyn.com.        86400   IN  NS  ns2.p01.dynect.net.
dyn.com.        86400   IN  NS  ns4.p01.dynect.net.
dyn.com.        86400   IN  NS  ns1.p01.dynect.net.
dyn.com.        86400   IN  NS  ns3.p01.dynect.net.
;; Received 122 bytes from 204.13.250.1#53(ns2.p01.dynect.net) in 4 ms
```

Google initially knows the names of the root nameservers because they are hard-coded into the hints file. Otherwise, how do you know where to start? The root zone looks at the request for dyn.com. and notices that it is in the com namespace. There is a label for com in the root zone, with 13 nameservers as NS records. The nameserver records found in the zone performing the delegation (root in this case) are known as the **parent** nameservers of the delegation. The inclusion of these nameservers at this spot indicates the answer to this query is not on the current nameserver or zone, and the resolver should try the ones provided.

This produces a **zone cut** to a new zone within the new **delegated zone**. At the location of those 13 nameservers, there is a zone file for the domain of com, with a Start of Authority (SOA) record so indicating. Along with the SOA, there are 13 nameservers in the apex of the com zone signaling that you are in the right place. These are known as the **child** nameservers of the delegation. The recursive follows this process, again and again, until it gets to the authoritative for the DNS name in question and “voila!” gets the answer.

For this example, the domain name is delegated to a nameserver that is a different domain entirely, but sometimes domain operators will choose to have the domain delegated to a nameserver within the zone itself. This is known as being **in bailiwick** and would look like example.com being delegated to a nameserver ns1.example.com. How did we get the IP of the original nameserver to ask the question in the first place!?

We have created a version of the **bootstrap paradox**. How do we get around it? Nameservers are able to pass on information in a DNS request such as the **authority section** to provide information on which nameserver is currently responding, as well as an additional section to provide more information on the answer.

In the case of nameservers, the **additional section** contains the IP addresses of the nameservers, to be used for the initial lookup—breaking the paradox. These are **glue records**, and they must be in the parent zone file.

Let's look at an example using ns1.example.com as a name server for example.com then the com zone has an NS record "example.com NS ns1.example.com," and a glue record "ns1.example.com A 192.0.2.1"

It's true that, in most top-level domains (TLDs), just like everything else that goes into the registry's zone, the glue records must be added by your registrar.

It is interesting to note that some recursives will prefer the parent NS records for nameserver selection, others will prefer to query the child nameservers for the child NS record, and still others will use the authority section within a DNS response handed out by those child nameservers.

There could be differences in TTLs between the parent and child NS records, and even the number and content of the records themselves if you misconfigured them, or have a **lame delegation** in which one of the nameservers in delegation doesn't respond to queries.

It is, therefore, highly advisable that your parent and child nameservers match on both sides of the delegation, with all nameservers correctly responding. Of course, sometimes they can be different, in order to allow you to change nameservers. But, as a general rule, they should be the same.

If you look at the example above, you will see the last two sections are almost identical, with a small but noticeable difference. "Matching RRsets" means: the algorithm in DNS doesn't compare TTLs when looking for a match. It just looks at name, class, and type.

This parent NS TTL set by the parent (including the TLD nameservers) within the parent zone, and there is nothing a child domain operator can do about it.

DNS Administration

We have a zone on a single provider with our answers. What happens if that provider has a problem? A very common topic is that of multiple provider DNS configurations. While the specifics have been covered in depth in a different piece, the terms related to that process belong here.

In order to have multiple providers, the zone files between them must be in sync, as any nameserver in the delegation might receive the traffic. Historically, this has been performed by a process of a **Master DNS** server which handles zone file management updating a **Secondary DNS** server. Today this is performed by letting the secondary know there is an update via a NOTIFY which activates the secondary to compare the serial value in the SOA record which designates the version of the zone.

If the master serial is higher, it is time to update. The secondary may initiate an **IXFR** to get the changes that have occurred between those two serials, or will perform an **AXFR** for the whole zone in cases where the number of changes is too large or this is the first time loading the zone, or if it does not support IXFR. Because this update process is a sensitive one, a **Transaction Signature (TSIG)** can be used to ensure the master DNS server really is who it says it is by way of a symmetric key verification.

Nameserver Selection and Announcement

From the perspective of the recursive, all of this is hidden. The only thing the recursive will see is a list of nameservers handed out by the parent delegation. Whatever relationship those nameservers have is completely lost. If that's not how the resolver chooses which nameserver to send traffic to, how does traffic get there?

To start the process off, if the resolver is just getting the delegation for the first time it will merely pick one randomly and send the first query off. After that, it might continue using that same name server. To improve performance, however, it may then mark down the **Round Trip Time (RTT)** – the time between when the query was sent until the reply was received.

It may then send the next query to one of the other unknown nameservers and repeat the process until all the known nameservers have a round trip time associated with them.

Many resolvers will then use these times to send more traffic to the fastest nameserver. This is called **nameserver affinity**, or a preference for some NS over the others. This accelerates responses for users and reduces some process time for the resolver. In order to adapt to changes, resolvers will periodically test the nameservers that responded more slowly to see if an improvement has occurred.

That is how the resolver chose which nameserver to go to, but how did the resolver get to the DNS server itself? While this lies in network engineering, there are two terms which are important to DNS itself. One way of operating a network is to have a one-to-one relationship in the number of servers to nameservers, and announce these like any other address. This is what is called a **unicast** announcement strategy.

This can present problems however. As we mentioned above, the resolvers are going to try every NS—including those which might be on the other side of the world. Furthermore, an attacker could deliberately target a nameserver because the IP maps to a single location. To combat this, network engineers use a technique called **anycast**, in which the routes to a single IP for a nameserver are announced from many locations on a regional or global network leading to greater performance and reliability. For modern DNS providers, this has largely become a standard practice.

With an anycast addressing scheme, multiple endpoints can share the same IP address. With dynamic routing, traffic is then steered to the network endpoint that is closest. RFC 4786, Section 2, discusses **anycast** in some detail for those who have a desire to do a deeper dive in this area.

DNSSEC

As more actors began to use the internet, malicious parties began to take advantage of its open nature. To ensure the fastest possible response, DNS uses **UDP** packets, rather than establishing a **TCP** connection. While there is a gain in speed, it is also easier to spoof the packet address. Additionally, resolvers historically used the first response to come in. These features combined to establish an environment in which the DNS lacked security. If you are going to yourbank.com, you really want it to be the right location, right?

DNS Security Extensions (DNSSEC) provide a framework to establish the integrity of answers received from the DNS. It does this in a way compatible with caches, but offering a **Chain of Trust** within the DNS tree, with each parent providing a signed hash of a key to verify the delegated child's key. The chain of trust starts at the very top with Root, then goes to the TLDs, and so on. Within a zone, the private **Zone Signing Key (ZSK)** signs each RRset for that zone, and publicly stores a Resource **Record Digital Signature (RRSIG)** record with each. This is combined with the public portion of the ZSK, held as a DNSKEY record which when all together allows a resolver to verify the record itself.

To verify the ZSK is valid, it itself is signed with a **Key Signing Key (KSK)** which creates a DNSKEY RRSIG. Of course, it needs a public DNSKEY again. But how do you verify the KSK in the first place? To prevent this from going on forever, DNSSEC uses that chain of trust mentioned earlier.

Rather than put the public KSK on the actual zone alongside the ZSK, it is hashed into a **Delegation Signer (DS)** record and provided to the parent of that domain. So the TLD hands out the DS records which you can use to verify the KSK which you then can use to verify the ZSK which can verify the Record. Of course, the TLD must be verified, so it has a series of RRsets, ZSKs and KSKs up to Root, which itself is signed. Because you are now at the root of the tree, there is nowhere else to go.

Each resolver around the world is configured to point to an initial trust anchor of the Root public KSK. The private Root KSK is used to sign the Root RRSIG of the DNSKEY RRset. Because this signing process is so critical to the security of the DNS, no single individual, organization, or nation was trusted to perform the signing.

Instead the signing actually occurs in person, in an **elaborate key signing ceremony**. Interestingly, the KSK itself has stayed the same since it was first used in 2010. The KSK was supposed to be rolled over in October 2017, but has been postponed because there were indications that a "significant number" of resolvers are not ready. So stay tuned for 2018 updates.

But wait! There's more! Another way in which malicious actors may spoof a zone is to pretend to have access to a host that doesn't exist within a zone. In this scenario, the response will be the absence of a record, not an explicit declaration that the record doesn't exist. This has been documented as a method for spoofing.

To combat this, **NSEC** records which point to the next valid host were created, thereby proving the lack of existence of anything in between. The problem there is that it becomes possible to walk the zone and know everything which does exist. While DNS is public information, and private information shouldn't be published in the first place, this worried people enough to create **NSEC3** records which change this into a hash. There is even a proposal for **NSEC5** which likely works by wizardry and unicorn dust alone. As we've seen, DNSSEC isn't easy.

Wrapping It Up

This paper's goal was to provide the information needed to help you better understand how all these pieces fit together. If you have questions as you find yourself staring down a long block of DNS, **contact Dyn**. We will be happy to help.

If you are interested in reading more, **here is a list** of all the DNS Requests for Comment (RFC) at the **Internet Systems Consortium (ISC)** who write BIND, the most widely used DNS software on the internet. For DNS terminology specifics, **RFC 7719** has the definitive list. And if you're interested in learning more about why it's time to rethink DNS, visit **dyn.com/dns**.

Rethink DNS.

Oracle Dyn is global business unit (GBU) focused on critical cloud infrastructure. Dyn is a pioneer in DNS and a leader in cloud-based infrastructure that connects users with digital content and experiences across a global internet. Dyn's solution is powered by a global network that drives 40 billion traffic optimization decisions daily for more than 3,500 enterprise customers, including preeminent digital brands such as Netflix, Twitter, LinkedIn and CNBC. Adding Dyn's best-in-class DNS and email services extend the Oracle cloud computing platform and provides enterprise customers with a one-stop shop for infrastructure as a service (IaaS) and platform as a service (PaaS).

ORACLE® + Dyn

dyn.com ☎ 603 668 4998 @dyn