An Introduction to DNS Terminology, Components, and Concepts

**Introduction**

DNS, or the Domain Name System, is often a very difficult part of learning how to configure websites and servers. Understanding how DNS works will help you diagnose problems with configuring access to your websites and will allow you to broaden your understanding of what's going on behind the scenes.

In this guide, we will discuss some fundamental DNS concepts that will help you hit the ground running with your DNS configuration. After tackling this guide, you should be ready to [set up your domain name with DigitalOcean](https://digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean) or [set up your very own DNS server](https://digitalocean.com/community/articles/how-to-install-the-bind-dns-server-on-centos-6).

Before we jump into setting up your own servers to resolve your domain or setting up our domains in the control panel, let's go over some basic concepts about how all of this actually works.

Domain Terminology

We should start by defining our terms. While some of these topics are familiar from other contexts, there are many terms used when talking about domain names and DNS that aren't used too often in other areas of computing.

Let's start easy:

**Domain Name System**

The domain name system, more commonly known as "DNS" is the networking system in place that allows us to resolve human-friendly names to unique addresses.

**Domain Name**

A domain name is the human-friendly name that we are used to associating with an internet resource. For instance, "google.com" is a domain name. Some people will say that the "google" portion is the domain, but we can generally refer to the combined form as the domain name.

The URL "google.com" is associated with the servers owned by Google Inc. The domain name system allows us to reach the Google servers when we type "google.com" into our browsers.

**IP Address**

An IP address is what we call a network addressable location. Each IP address must be unique within its network. When we are talking about websites, this network is the entire internet.

IPv4, the most common form of addresses, are written as four sets of numbers, each set having up to three digits, with each set separated by a dot. For example, "111.222.111.222" could be a valid IPv4 IP address. With DNS, we map a name to that address so that you do not have to remember a complicated set of numbers for each place you wish to visit on a network.

**Top-Level Domain**

A top-level domain, or TLD, is the most general part of the domain. The top-level domain is the furthest portion to the right (as separated by a dot). Common top-level domains are "com", "net", "org", "gov", "edu", and "io".

Top-level domains are at the top of the hierarchy in terms of domain names. Certain parties are given management control over top-level domains by ICANN (Internet Corporation for Assigned Names and Numbers). These parties can then distribute domain names under the TLD, usually through a domain registrar.

**Hosts**

Within a domain, the domain owner can define individual hosts, which refer to separate computers or services accessible through a domain. For instance, most domain owners make their web servers accessible through the bare domain (example.com) and also through the "host" definition "www" ([www.example.com](http://www.example.com/)).

You can have other host definitions under the general domain. You could have API access through an "api" host (api.example.com) or you could have ftp access by defining a host called "ftp" or "files" (ftp.example.com or files.example.com). The host names can be arbitrary as long as they are unique for the domain.

**SubDomain**

A subject related to hosts are subdomains.

DNS works in a hierarchy. TLDs can have many domains under them. For instance, the "com" TLD has both "google.com" and "ubuntu.com" underneath it. A "subdomain" refers to any domain that is part of a larger domain. In this case, "ubuntu.com" can be said to be a subdomain of "com". This is typically just called the domain or the "ubuntu" portion is called a SLD, which means second level domain.

Likewise, each domain can control "subdomains" that are located under it. This is usually what we mean by subdomains. For instance you could have a subdomain for the history department of your school at "[www.history.school.edu](http://www.history.school.edu/)". The "history" portion is a subdomain.

The difference between a host name and a subdomain is that a host defines a computer or resource, while a subdomain extends the parent domain. It is a method of subdividing the domain itself.

Whether talking about subdomains or hosts, you can begin to see that the left-most portions of a domain are the most specific. This is how DNS works: from most to least specific as you read from left-to-right.

**Fully Qualified Domain Name**

A fully qualified domain name, often called FQDN, is what we call an absolute domain name. Domains in the DNS system can be given relative to one another, and as such, can be somewhat ambiguous. A FQDN is an absolute name that specifies its location in relation to the absolute root of the domain name system.

This means that it specifies each parent domain including the TLD. A proper FQDN ends with a dot, indicating the root of the DNS hierarchy. An example of a FQDN is "mail.google.com.". Sometimes software that calls for FQDN does not require the ending dot, but the trailing dot is required to conform to ICANN standards.

**Name Server**

A name server is a computer designated to translate domain names into IP addresses. These servers do most of the work in the DNS system. Since the total number of domain translations is too much for any one server, each server may redirect request to other name servers or delegate responsibility for a subset of subdomains they are responsible for.

Name servers can be "authoritative", meaning that they give answers to queries about domains under their control. Otherwise, they may point to other servers, or serve cached copies of other name servers' data.

**Zone File**

A zone file is a simple text file that contains the mappings between domain names and IP addresses. This is how the DNS system finally finds out which IP address should be contacted when a user requests a certain domain name.

Zone files reside in name servers and generally define the resources available under a specific domain, or the place that one can go to get that information.

**Records**

Within a zone file, records are kept. In its simplest form, a record is basically a single mapping between a resource and a name. These can map a domain name to an IP address, define the name servers for the domain, define the mail servers for the domain, etc.

How DNS Works

Now that you are familiar with some of the terminology involved with DNS, how does the system actually work?

The system is very simple at a high-level overview, but is very complex as you look at the details. Overall though, it is a very reliable infrastructure that has been essential to the adoption of the internet as we know it today.

**Root Servers**

As we said above, DNS is, at its core, a hierarchical system. At the top of this system is what are known as "root servers". These servers are controlled by various organizations and are delegated authority by ICANN (Internet Corporation for Assigned Names and Numbers).

There are currently 13 root servers in operation. However, as there are an incredible number of names to resolve every minute, each of these servers is actually mirrored. The interesting thing about this set up is that each of the mirrors for a single root server share the same IP address. When requests are made for a certain root server, the request will be routed to the nearest mirror of that root server.

What do these root servers do? Root servers handle requests for information about Top-level domains. So if a request comes in for something a lower-level name server cannot resolve, a query is made to the root server for the domain.

The root servers won't actually know where the domain is hosted. They will, however, be able to direct the requester to the name servers that handle the specifically requested top-level domain.

So if a request for "[www.wikipedia.org](http://www.wikipedia.org/)" is made to the root server, the root server will tell not find the result in its records. It will check its zone files for a listing that matches "[www.wikipedia.org](http://www.wikipedia.org/)". It will not find one.

It will instead find a record for the "org" TLD and give the requesting entity the address of the name server responsible for "org" addresses.

**TLD Servers**

The requester then sends a new request to the IP address (given to it by the root server) that is responsible for the top-level domain of the request.

So, to continue our example, it would send a request to the name server responsible for knowing about "org" domains to see if it knows where "[www.wikipedia.org](http://www.wikipedia.org/)" is located.

Once again, the requester will look for "[www.wikipdia.org](http://www.wikipdia.org/)" in its zone files. It will not find this record in its files.

However, it will find a record listing the IP address of the name server responsible for "wikipedia.org". This is getting much closer to the answer we want.

**Domain-Level Name Servers**

At this point, the requester has the IP address of the name server that is responsible for knowing the actual IP address of the resource. It sends a new request to the name server asking, once again, if it can resolve "[www.wikipedia.org](http://www.wikipedia.org/)".

The name server checks its zone files and it finds that it has a zone file associated with "wikipedia.org". Inside of this file, there is a record for the "www" host. This record tells the IP address where this host is located. The name server returns the final answer to the requester.

**What is a Resolving Name Server?**

In the above scenario, we referred to a "requester". What is the requester in this situation?

In almost all cases, the requester will be what we call a "resolving name server" A resolving name server is one configured to ask other servers questions. It is basically an intermediary for a user which caches previous query results to improve speed and knows the addresses of the root servers to be able to "resolve" requests made for things it doesn't already know about.

Basically, a user will usually have a few resolving name servers configured on their computer system. The resolving name servers are usually provided by an ISP or other organizations. For instance [Google provides resolving DNS servers](https://developers.google.com/speed/public-dns/) that you can query. These can be either configured in your computer automatically or manually.

When you type a URL in the address bar of your browser, your computer first looks to see if it can find out locally where the resource is located. It checks the "hosts" file on the computer and a few other locations. It then sends the request to the resolving name server and waits back to receive the IP address of the resource.

The resolving name server then checks its cache for the answer. If it doesn't find it, it goes through the steps outlined above.

Resolving name servers basically compress the requesting process for the end user. The clients simply have to know to ask the resolving name servers where a resource is located and be confident that they will investigate and return the final answer.

Zone Files

We mentioned in the above process the idea of "zone files" and "records".

Zone files are the way that name servers store information about the domains they know about. Every domain that a name server knows about is stored in a zone file. Most requests coming to the average name server are not something that the server will have zone files for.

If it is configured to handle recursive queries, like a resolving name server, it will find out the answer and return it. Otherwise, it will tell the requesting party where to look next.

The more zone files that a name server has, the more requests it will be able to answer authoritatively.

A zone file describes a DNS "zone", which is basically a subset of the entire DNS naming system. It generally is used to configure just a single domain. It can contain a number of records which define where resources are for the domain in question.

The zone's $ORIGIN is a parameter equal to the zone's highest level of authority by default.

So if a zone file is used to configure the "example.com." domain, the $ORIGIN would be set to example.com..

This is either configured at the top of the zone file or it can be defined in the DNS server's configuration file that references the zone file. Either way, this parameter describes what the zone is going to be authoritative for.

Similarly, the $TTL configures the "time to live" of the information it provides. It is basically a timer. A caching name server can use previously queried results to answer questions until the TTL value runs out.

Record Types

Within the zone file, we can have many different record types. We will go over some of the more common (or mandatory types) here.

**SOA Records**

The Start of Authority, or SOA, record is a mandatory record in all zone files. It must be the first real record in a file (although $ORIGIN or $TTL specifications may appear above). It is also one of the most complex to understand.

The start of authority record looks something like this:

domain.com. IN SOA ns1.domain.com. admin.domain.com. (

12083 ; serial number

3h ; refresh interval

30m ; retry interval

3w ; expiry period

1h ; negative TTL

)

Let's explain what each part is for:

* **domain.com.**: This is the root of the zone. This specifies that the zone file is for the domain.com.domain. Often, you'll see this replaced with @, which is just a placeholder that substitutes the contents of the $ORIGIN variable we learned about above.
* **IN SOA**: The "IN" portion means internet (and will be present in many records). The SOA is the indicator that this is a Start of Authority record.
* **ns1.domain.com.**: This defines the primary master name server for this domain. Name servers can either be master or slaves, and if dynamic DNS is configured one server needs to be a "primary master", which goes here. If you haven't configured dynamic DNS, then this is just one of your master name servers.
* **admin.domain.com.**: This is the email address of the administrator for this zone. The "@" is replaced with a dot in the email address. If the name portion of the email address normally has a dot in it, this is replace with a "\" in this part ([your.name@domain.com](mailto:your.name@domain.com) becomes your\name.domain.com).
* **12083**: This is the serial number for the zone file. Every time you edit a zone file, you must increment this number for the zone file to propagate correctly. Slave servers will check if the master server's serial number for a zone is larger than the one they have on their system. If it is, it requests the new zone file, if not, it continues serving the original file.
* **3h**: This is the refresh interval for the zone. This is the amount of time that the slave will wait before polling the master for zone file changes.
* **30m**: This is the retry interval for this zone. If the slave cannot connect to the master when the refresh period is up, it will wait this amount of time and retry to poll the master.
* **3w**: This is the expiry period. If a slave name server has not been able to contact the master for this amount of time, it no longer returns responses as an authoritative source for this zone.
* **1h**: This is the amount of time that the name server will cache a name error if it cannot find the requested name in this file.

**A and AAAA Records**

Both of these records map a host to an IP address. The "A" record is used to map a host to an IPv4 IP address, while "AAAA" records are used to map a host to an IPv6 address.

The general format of these records is this:

host IN A IPv4\_address

host IN AAAA IPv6\_address

So since our SOA record called out a primary master server at "ns1.domain.com", we would have to map this to an address to an IP address since "ns1.domain.com" is within the "domain.com" zone that this file is defining.

The record could look something like this:

ns1 IN A 111.222.111.222

Notice that we don't have to give the full name. We can just give the host, without the FQDN and the DNS server will fill in the rest with the $ORIGIN value. However, we could just as easily use the entire FQDN if we feel like being semantic:

ns1.domain.com. IN A 111.222.111.222

In most cases, this is where you'll define your web server as "www":

www IN A 222.222.222.222

We should also tell where the base domain resolves to. We can do this like this:

domain.com. IN A 222.222.222.222

We could have used the "@" to refer to the base domain instead:

@ IN A 222.222.222.222

We also have the option of resolving anything that under this domain that is not defined explicitly to this server too. We can do this with the "\*" wild card:

\* IN A 222.222.222.222

All of these work just as well with AAAA records for IPv6 addresses.

**CNAME Records**

CNAME records define an alias for canonical name for your server (one defined by an A or AAAA record).

For instance, we could have an A name record defining the "server1" host and then use the "www" as an alias for this host:

server1 IN A 111.111.111.111

www IN CNAME server1

Be aware that these aliases come with some performance losses because they require an additional query to the server. Most of the time, the same result could be achieved by using additional A or AAAA records.

One case when a CNAME is recommended is to provide an alias for a resource outside of the current zone.

**MX Records**

MX records are used to define the mail exchanges that are used for the domain. This helps email messages arrive at your mail server correctly.

Unlike many other record types, mail records generally don't map a host to something, because they apply to the entire zone. As such, they usually look like this:

IN MX 10 mail.domain.com.

Note that there is no host name at the beginning.

Also note that there is an extra number in there. This is the preference number that helps computers decide which server to send mail to if there are multiple mail servers defined. Lower numbers have a higher priority.

The MX record should generally point to a host defined by an A or AAAA record, and not one defined by a CNAME.

So, let's say that we have two mail servers. There would have to be records that look something like this:

IN MX 10 mail1.domain.com.

IN MX 50 mail2.domain.com.

mail1 IN A 111.111.111.111

mail2 IN A 222.222.222.222

In this example, the "mail1" host is the preferred email exchange server.

We could also write that like this:

IN MX 10 mail1

IN MX 50 mail2

mail1 IN A 111.111.111.111

mail2 IN A 222.222.222.222

**NS Records**

This record type defines the name servers that are used for this zone.

You may be wondering, "if the zone file resides on the name server, why does it need to reference itself?". Part of what makes DNS so successful is its multiple levels of caching. One reason for defining name servers within the zone file is that the zone file may be actually being served from a cached copy on another name server. There are other reasons for needing the name servers defined on the name server itself, but we won't go into that here.

Like the MX records, these are zone-wide parameters, so they do not take hosts either. In general, they look like this:

IN NS ns1.domain.com.

IN NS ns2.domain.com.

You should have at least two name servers defined in each zone file in order to operate correctly if there is a problem with one server. Most DNS server software considers a zone file to be invalid if there is only a single name server.

As always, include the mapping for the hosts with A or AAAA records:

IN NS ns1.domain.com.

IN NS ns2.domain.com.

ns1 IN A 111.222.111.111

ns2 IN A 123.211.111.233

There are quite a few other record types you can use, but these are probably the most common types that you will come across.

**PTR Records**

The PTR records are used define a name associated with an IP address. PTR records are the inverse of an A or AAAA record. PTR records are unique in that they begin at the .arpa root and are delegated to the owners of the IP addresses. The Regional Internet Registries (RIRs) manage the IP address delegation to organization and service providers. The Regional Internet Registries include APNIC, ARIN, RIPE NCC, LACNIC, and AFRINIC.

Here is an example of a PTR record for 111.222.333.444 would look like:

444.333.222.111.in-addr.arpa. 33692 IN PTR host.example.com.

This example of a PTR record for an IPv6 address shows the *nibble* format of the reverse of Google's IPv6 DNS Server 2001:4860:4860::8888.

8.8.8.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.6.8.4.0.6.8.4.1.0.0.2.ip6.arpa. 86400IN PTR google-public-dns-a.google.com.

The command line tool dig with the -x flag can be used to look up the reverse DNS name of an IP address.

Here is an example of a dig command. The +short is appended to reduce the output to the reverse DNS name.

* dig -x 8.8.4.4 +short

The output for the dig command above will be the domain name in the PTR record for the IP address:

google-public-dns-b.google.com.

Servers on the Internet use PTR records to place domain names within log entries, make informed spam handling decisions, and display easy-to-read details about other devices.

Most commonly-used email servers will look up the PTR record of an IP address it receives email from. If the source IP address does not have a PTR record associated with it, the emails being sent may be treated as spam and rejected. It is not important that the FQDN in the PTR matches the domain name of the email being sent. What is important is that there is a valid PTR record with a corresponding and matching forward A record.

Normally network routers on the Internet are given PTR records that correspond with their physical location. For example you may see references to 'NYC' or 'CHI' for a router in New York City or Chicago. This is helpful when running a [traceroute or MTR](https://www.digitalocean.com/community/tutorials/how-to-use-traceroute-and-mtr-to-diagnose-network-issues) and reviewing the path Internet traffic is taking.

Most providers offering dedicated servers or VPS services will give customers the ability to set a PTR record for their IP address. **DigitalOcean will automatically assign the PTR record of any Droplet when the Droplet is named with a domain name.** The Droplet name is assigned during creation and can be edited later using the settings page of the Droplet control panel.

**Note:** It is important that the FQDN in the PTR record has a corresponding and matching forward A record. Example: 111.222.333.444 has a PTR of server.example.com and server.example.com is an A record that points to 111.222.333.444.

Conclusion

You should now have a pretty good grasp on how DNS works. While the general idea is relatively easy to grasp once you're familiar with the strategy, this is still something that can be difficult for inexperienced administrators to put into practice.

A Comparison of DNS Server Types: How To Choose the Right DNS Configuration

PostedJuly 1, 2014 168.8kviews [NETWORKING](https://www.digitalocean.com/community/tags/networking?type=tutorials) [DNS](https://www.digitalocean.com/community/tags/dns?type=tutorials)

**Introduction**

DNS, or the Domain Name System, is an integral part of how systems connect with each other to communicate on the internet. Without DNS, computers, and the people who use them, would be required to connect using only numerical addresses known as IP addresses.

Besides the obvious problem of having to remember a large number of complex numbers for simple tasks, communicating through IP addresses also causes some additional problems. Moving your website to a different hosting provider, or moving your servers to different locations would require you to inform every client of the new location.

DNS servers, the computers that together form the system that allow us to use names instead of addresses, can server many different functions, each of which can contribute to your ability to accessing servers by name.

In a [previous guide](https://www.digitalocean.com/community/tutorials/an-introduction-to-dns-terminology-components-and-concepts) we discussed some of the basic terminology and concepts of the domain name system. We will assume some familiarity with the concepts covered in that article. In this guide, we will talk about some of the different types of DNS server setups and what the advantages, use cases, and properties are of each.

The Path of a DNS Query

When a client program wants to access a server by its domain name, it must find out how to translate the domain name into an actual routable address that it can use to communicate. It needs to know this information in order to get or send information to the server.

Some applications, including most web browsers, maintain an internal cache of recent queries. This is the first place the application will check, if it has this capability, in order to find the IP address of the domain in question. If it does not find the answer to its question here, it then asks the **system resolver** to find out what the address of the domain name is.

A **resolver** in general is any component that acts as a client-side participant in a DNS query. The system resolver is the resolving library that your operating system uses to seek out the answer for DNS queries. In general, system resolvers are usually what we consider **stub resolvers** because they are not capable of much complexity beyond searching a few static files on the system (like the /etc/hosts file) and forwarding requests to another resolver.

So generally, a query goes from the client application to the system resolver, where it is then passed to a DNS server that it has the address for. This DNS server is called a **recursive DNS server**. A recursive server is a DNS server that is configured to query other DNS servers until it finds the answer to the question. It will either return the answer or an error message to the client (the system resolver in this case, which will, in turn, pass it to the client application).

Recursive servers generally maintain a cache as well. It will check this cache first to see if it already has the answer to the query. If it does not, it will see if it has the address to any of the servers that control the upper level domain components. So if the request is for www.example.com and it cannot find that host address in its cache, it will see if it has the address of the name servers for example.com and if necessary, com. It will then send a query to the name server of most specific domain component it can find in order to query for more information.

If it does not find the address to any of these domain components, it has to start from the very top of the hierarchy by querying the **root name servers**. The root servers know the addresses of all of the TLD (top level domain) name servers which control zones for .com, .net, .org, etc. It will ask the root servers if it knows the address of to www.example.com. The root server will refer the recursive server to the name servers for the .com TLD.

The recursive server then follows the trail of referrals to each successive name server that has been delegated responsibility for the domain components, until it can zero in on the specific name server that has the full answer. It puts this answer into its cache for later queries and then returns it to the client.

As you can see from this example, there are many different kinds of servers, and they each play a different role. Let's go over the specifics of the different types of DNS servers.

Functional Differences

Some of the differences between DNS servers are purely functional. Most servers that are involved with implementing DNS are specialized for certain functions. The type of DNS server you choose will largely depend on your needs and what type of problem you are hoping to solve.

**Authoritative-Only DNS Servers**

An authoritative-only DNS server is a server that only concerns itself with answering the queries for the zones that it is responsible for. Since it does not help resolve queries for outside zones, it is generally very fast and can handle many requests efficiently.

Authoritative-only servers have the following properties:

* **Very fast at responding to queries for zones it controls.** An authoritative-only server will have all of the information about the domain it is responsible for, or referral information for zones within the domain that have been delegated out to other name servers.
* **Will not respond to recursive queries.** The very definition of an authoritative-only server is one that does not handle recursive requests. This makes it a server only and never a client in the DNS system. Any request reaching an authoritative-only server will generally be coming from a resolver that has received a referral to it, meaning that the authoritative-only server will either have the full answer, or will be able to pass a new referral to the name server that it has delegated responsibility to.
* **Does not cache query results.** Since an authoritative-only server never queries other servers for information to resolve a request, it never has the opportunity to cache results. All of the information it knows is already in its system.

**Caching DNS Server**

A caching DNS server is a server that handles recursive requests from clients. Almost every DNS server that the operating system's stub resolver will contact will be a caching DNS server.

Caching servers have the advantage of answering recursive requests from clients. While authoritative-only servers may be ideal for serving specific zone information, caching DNS servers are more broadly useful from a client's perspective. They make the DNS system of the world accessible to rather dumb client interfaces.

To avoid having to take the performance hit of issuing multiple iterative request to other DNS servers every time it receives a recursive request, the server caches its results. This allows it to have access to a broad base of DNS information (the entire world's publicly accessible DNS) while handling recent requests very quickly.

A caching DNS server has the following properties:

* **Access to the entire range of public DNS data.** All zone data served by publicly accessible DNS servers hooked into the global delegation tree can be reached by a caching DNS server. It knows about the root DNS servers and can intelligently follow referrals as it receives data.
* **Ability to spoon-feed data to dumb clients.** Almost every modern operating system offloads DNS resolution to dedicated recursive servers through the use of stub resolvers. These resolving libraries simply issue a recursive request and expect to be handed back a complete answer. A caching DNS server has the exact capabilities to serve these clients. By accepting a recursive query, these servers promise to either return with an answer or a DNS error message.
* **Maintains a cache of recently requested data.** By caching the results as it collects them from other DNS servers for its client requests, a caching DNS server builds a cache for recent DNS data. Depending on how many clients use the server, how large the cache is, and how long the TTL data is on the DNS records themselves, this can drastically speed up DNS resolution in most cases.

**Forwarding DNS Server**

A alternative take on developing a cache for client machines is through the use of a forwarding DNS server. This approach adds an additional link in the chain of DNS resolution by implementing a forwarding server that simply passes all requests to another DNS server with recursive capabilities (such as a caching DNS server).

The advantage of this system is that it can give you the advantage of a locally accessible cache while not having to do the recursive work (which can result in additional network traffic and can take up substantial resources on high traffic servers). This can also lead to some interesting flexibility in splitting your private and public traffic by forwarding to different servers.

A forwarding DNS server has the following properties:

* **The ability to handle recursive requests without performing recursion itself.** The most fundamental property of a forwarding DNS server is that it passes requests on to another agent for resolution. The forwarding server can have minimal resources and still provide great value by leveraging its cache.
* **Provide a local cache at a closer network location.** Particularly if you do not feel up to building, maintaining, and securing a full-fledged recursive DNS solution, a forwarding server can use public recursive DNS servers. It can leverage these servers while moving the primary caching location very close to the client machines. This can decrease answer times.
* **Increases flexibility in defining local domain space.** By passing requests to different servers conditionally, a forwarding server can ensure that internal requests are served by private servers while external requests use public DNS.

**Combination Solutions**

While the above solutions are built with very specific purposes in mind, it is often desirable to set up your DNS server to combine the advantages of each.

A DNS server may be configured to act as a recursive, caching server for a select number of local clients, while answering only iterative, authoritative requests from other clients. This is a common configuration because it allows you to answer global requests for your domain, while also allowing your local clients to utilize the server for recursive resolution.

While certain DNS software is specially designed to fulfill one specific role, applications like Bind are incredibly flexible and can be used as hybrid solutions. While in some cases attempting to provide too many services in a single server can lead to performance degradation, in many cases, especially in the case of small infrastructure, it makes the most sense to maintain a single, all-in-one solution.

Relational Differences

While the most apparent differences between DNS server configurations are probably functional, the relational differences are also extremely important.

**Primary and Slave Servers**

Given the importance of DNS in making services and entire networks accessible, most DNS servers that are authoritative for a zone will have built-in redundancy. There are various terms for the relationships between these servers, but generally, a server can either be a **master** or a **slave** in its configuration.

Both master and slave servers are authoritative for the zones they handle. The master does not have any more power over the zones than the slave. The only differentiating factor between a master and a slave server is where they read their zone files from.

A master server reads its zone files from files on the system's disk. These are usually where the zone administrator adds, edits, or transfers the original zone files.

The slave server receives the zones that it is authoritative for through a zone transfer from one of the master servers for the zone. Once it has these zones, it places them in a cache. If it has to restart, it first checks its cache to see if the zones inside are up-to-date. If not, it requests the updated information from the master server.

Servers are not relegated to only be a master or a slave for all of the zones they handle. Master or slave status is assigned on a zone-by-zone basis, so a server can be a master for some zones and a slave for others.

DNS zones usually have at least two name servers. Any zone responsible for an internet routable zone *must* have at least two name servers. Often times, many more name servers are maintained in order to spread the load and increase redundancy.

**Public vs Private Servers**

Often, organizations use DNS both externally and internally. However the information that should be made available in both of these spheres is often drastically different.

An organization might maintain an externally available authoritative-only DNS server to handle public DNS queries for the domains and zones that it handles. For its internal users, the organization might use a separate DNS server that contains the authoritative information that the public DNS provides, as well as additional information about internal hosts and services. It might also provide additional features, such as recursion and caching for its internal clients.

While we mentioned the ability to have a single server handle all of these tasks in the "combination" server above, there are definite advantages to splitting the workload. In fact, maintaining completely separate servers (internal vs external) that have no knowledge of each other is often desirable. It is especially important, from a security standpoint, that the public server has no records of the private counterpart. This means not listing your private name servers with NS records in the public zone files.

There are some additional considerations to keep in mind. While it might be easier to have your public and private servers share zone data that they have in common in a traditional master-slave relationship, this can leak information about your private infrastructure into the wild.

Beyond just keeping your private servers out of the zone files themselves (essentially a publicly searchable entity), it is usually a good idea to also remove any reference to the private server in the public server's configuration files. This means removing transfer, notify, and masters configuration details so that a compromise of the public server does not mean that your internal name servers are suddenly exposed.

This means maintaining separate zone files for each, which can be extra work. However, this may be necessary for absolute separation and security.

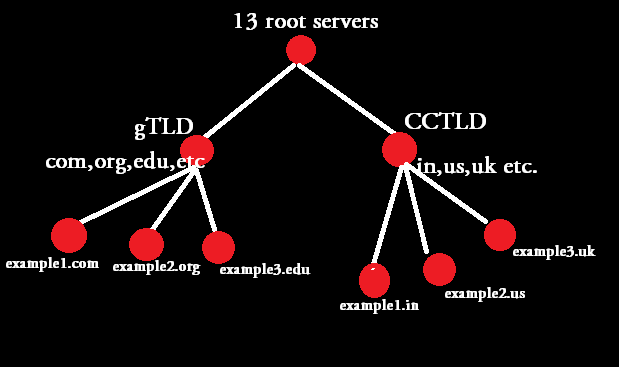
Conclusion

You are probably aware by this stage that there is quite a bit of flexibility in choosing your DNS configuration.

Your choices will largely depend on your organization's needs and whether your main priority is to provide faster DNS resolution for a selection of clients (caching or forwarding) or to serve your domains and zones to the internet at large (authoritative servers). Combination approaches are common and, in the end, both sides of the resolution process need to be accounted for.

In our next guides, we will demonstrate how to get started with some of these configurations. We will begin by teaching [how to set up a caching or forwarding server](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-caching-or-forwarding-dns-server-on-ubuntu-14-04). Later, we will cover how to serve your domains by [setting up a pair of authoritative-only DNS servers](https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-an-authoritative-only-dns-server-on-ubuntu-14-04).

DNS architecture works on an inverted tree structure. At the top of the inverted tree is the 13 DNS root servers, and then comes the TLD(Top Level Domain) servers, and beneath the TLD servers comes the authoritative DNS server for a particular domain(sometimes called as secondary domains.)



The above shown diagram depicts the classification of DNS inverted tree structure. You can clearly see that the root servers are at the top of the inverted tree, gTLD(Generic Top Level Domains) & CCTLD(Country Code Top Level Domains) comes below the root servers, and then comes the DNS servers for an example domain.

Consider the above shown tree structure as the flow chart of a DNS name resolution. So for example, if you want to find the IP address of mail.example.com two types of DNS queries can be made. One is called Iterative & the other is called recursive(which is the normal configuration of all local DNS servers).

# What are the different types of DNS queries?

DNS queries can be classified according the manner in which a complete request is processed. Generally queries can be classified as follows.

1. **recursive query**
2. **iterative query OR Nonrecursive query**
3. **Inverse queries**

## 

## What is a recursive query?

A recursive query is a kind of query, in which the DNS server, who received your query will do all the job of fetching the answer, and giving it back to you. During this process, the DNS server might also query other DNS server's in the internet on your behalf, for the answer.

Lets understand the entire process of recursive queries by the following steps.

Suppose you want to browse www.example.com, and your resolve.conf file has got the following entry.

[root@myvm ~]# cat /etc/resolv.conf

nameserver 172.16.200.30

nameserver 172.16.200.31

The above resolve conf entry means that,Your DNS servers are 172.16.200.30 & 31. Whatever application you use, the operating system will send DNS queries to those two DNS servers.

**STEP 1:** You enter www.example.com in the browser. So the operating system's resolver will send a DNS query for the A record to the DNS server 172.16.200.30 .

**STEP2:**The DNS server 172.16.200.30 on receiving the query, will look through its tables(cache) to find the IP address(A record) for the domain www.example.com. But it does not have the entry.

**STEP 3:** As the answer for the query is not available with the DNS server 172.16.200.30, this server sends a query to one of the DNS root server,for the answer. Now an important fact to note here is that root server's are always iterative servers.

**Related:**[DNS root servers and their Locations](http://www.slashroot.in/dns-root-servers-most-critical-infrastructure-internet)

**STEP 4:** The dns root server's will reply with a list of server's (referral) that are responsible for handling the **.COM** gTLD's.

**STEP 5:**Our DNS server 172.16.200.30 will select one of the .COM gTLD server from the list given by the root server, to query the answer for "www.example.com"

**STEP 6:**Similar to the root server's , the gTLD server's are also iterative in nature, so it replies back to our DNS server 172.16.200.30 with the list of IP addresses of the DNS server's responsible for the domain(authoritative name server for the domain) www.example.com.

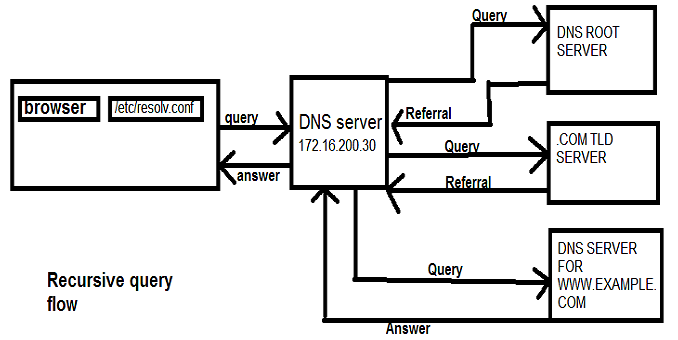
**Related:** [DNS Zone File And Its Contents](http://www.slashroot.in/what-dns-zone-file-complete-tutorial-zone-file-and-its-contents)

**STEP 7:** This time also our DNS server will select one of the IP from the given list of authoritative name servers, and queries the A record for www.example.com. The authoritative name server queried, will reply back with the A record as below.

**www.example.com = <XXX:XX:XX:XX> (Some IP address)**

**STEP 8**: Our DNS server 172.16.200.30 will reply us back with the ip domain pair(and any other resource if available). Now the browser will send request to the ip given, for the web page www.example.com.

Below shown diagram might make the concept clear.



As you can see from the above figure. Our DNS server(172.16.200.30) queries through other dns server's on behalf of us.

**Note:** The above explained scenario of recursive query happened, only because, our DNS server 172.16.200.30 was configured as a recursive name server. You can also disable this feature for your DNS server.

### How does the name server select one from the given list of servers to query?

In the above case, you might have seen that our DNS server 172.16.200.30, had to select one server, from the given list of servers to query, multiple times.

For example there are 13 root servers(Well when i say 13 root servers, 13 is the number of addresses that is universal. There are Hundreds of servers at different locations in the world. These 13 root server addresses are anycasted addresses.), which root server will be queried, for an answer?

**Related:**[What is IP Anycast, and how it works?](http://www.slashroot.in/what-anycast-and-how-it-works)

Almost all DNS server's uses an algorithm, to select one from the list, in order to distribute the load and response time.

The most Famous DNS server software BIND uses a technique called as rtt metric(Round Trip Time metric). Using this technique, the server tracks the RTT of each root server, and selects the one,with lower RTT.

## What is an iterative or Non-recursive query?

Before beginning the explanation for iterative query. An important thing to note is that, all DNS server's must support iterative(non-recursive)query.

In an iterative query, the name server, will not go and fetch the complete answer for your query, but will give back a referral to other DNS server's, which might have the answer. In our previous example our DNS server 172.16.200.30, went to fetch the answer on behalf of our resolver, and provided us with the final answer.

But if our DNS server 172.16.200.30 is not a recursive name server(which means its iterative), it will give us the answer if it has in its records. Otherwise will give us the referral to the root servers(it will not query the root server's and other servers by itself.).

Now its the job of our resolver to query the root server, .COM TLD servers, and authoritative name server's, for the answer.

Lets go through the steps involved.

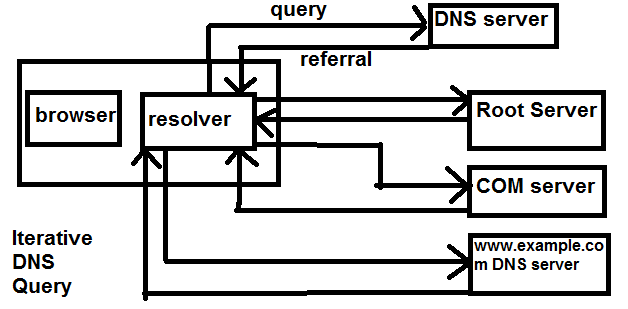
**STEP 1:**You enter www.example.com in the browser. So the operating system's resolver will send a DNS query for the A record to the DNS server 172.16.200.30 .

**STEP 2:** The DNS server 172.16.200.30 on receiving the query, will look through its tables(cache) to find the IP address(A record) for the domain www.example.com. But it does not have the entry.

**STEP 3:**Now instead of querying the root server's, our DNS server will reply us back with a referral to root servers. Now our operating system resolver, will query the root servers for the answer.

Now the rest of the steps are all the same. The only difference in iterative query is that

* if the DNS server does not have the answer, it will not query any other server for the answer, but rather it will reply with the referral to DNS root server's
* But if the DNS server has the answer, it will give back the answer(which is same in both iterative and recursive queries)
* in an iterative query, the job of finding the answer(from the given referral), lies to the local operating system resolver.



It can be clearly noted from the above figure, that in an iterative query, a DNS server queried will never go and fetch the answer for you(but will give you the answer if it already has the answer). But will give your resolver a referral to other DNS server's(root server in our case).

We will be discussing inverse queries in another post. Hope this post was helpful in understanding iterative(non-recursive) & recursive DNS queries.

# 4. DNS Configuration Types

Most DNS servers are schizophrenic - they may be masters (authoritative) for some [zones](http://www.zytrax.com/books/dns/apa/zones.html), slaves for others and provide caching or forwarding for all others. Many observers object to the concept of DNS **types** partly because of the schizophrenic behaviour of most DNS servers and partly to avoid confusion with the name.conf zone parameter 'type' which only allows master, slave, stub, forward, hint). Nevertheless, the following terms are commonly used to describe the primary function or requirement of DNS servers.

**Notes:**

1. Running any DNS server that supports recursive queries received from any or all users (an Open DNS) is an Extremely Bad Idea ™. While an Open DNS may look like a friendly and neighbourly thing to do such a server may be used in DDoS attacks and carries a significantly increased risk of cache poisoning. It is always possible to define the range of IP addresses that are allowed or permitted to use the recursive feature of any DNS server. Such a server is termed Closed. The various configurations have been modified to ensure that the DNS stays Closed to non-permitted users.
2. One of the basic rules of security is that only the minimum services necessary to meet the objectives should be deployed. This means that a secure DNS server should provide only a single function, for instance, authoritative only, or caching only, not both capabilities in the same server. This is a correct but idealistic position, generally possible only in larger organizations. In practice many of us run mixed mode DNS servers. While much can be done to mitigate any security implications it must always be accepted that, in mixed configurations, increased risk is the downside of flexibility.

## Contents

1. [4.1 Master (a.k.a. Primary) DNS Server](http://www.zytrax.com/books/dns/ch4/#master)
2. [4.2 Slave (a.k.a Secondary) DNS Server](http://www.zytrax.com/books/dns/ch4/#slave)
3. [4.2.1 But Slaves can also be Masters](http://www.zytrax.com/books/dns/ch4/#slave-master)
4. [4.3 Caching (a.k.a. hint) DNS Server](http://www.zytrax.com/books/dns/ch4/#caching)
5. [4.4 Forwarding (a.k.a Proxy, Client, Remote) DNS Server](http://www.zytrax.com/books/dns/ch4/#forwarding)
6. [4.5 Stealth (a.k.a. DMZ or Hidden Master) DNS Server](http://www.zytrax.com/books/dns/ch4/#stealth)
7. [4.6 Authoritative Only DNS Server](http://www.zytrax.com/books/dns/ch4/#authoritative)
8. [4.7 Split Horizon DNS Server](http://www.zytrax.com/books/dns/ch4/#split)

## 4.1 Master (Primary) Name Servers

A Master DNS defines one or more [zone files](http://www.zytrax.com/books/dns/apa/zones.html) for which this DNS is **Authoritative** ('type master'). The zone has been delegated (via an [NS Resource Record](http://www.zytrax.com/books/dns/ch8/ns.html)) to this DNS.

The term **master** was introduced with BIND 8.x and replaced the term 'primary'.

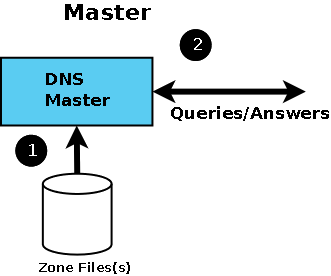


Diagram 1 DNS Master

Master status is defined in BIND by including 'type master' in the zone declaration section of the [named.conf file](http://www.zytrax.com/books/dns/apa/conf.html) as shown by the following fragment.

// example.com fragment from named.conf

// defines this server as a zone master

zone "example.com" in{

type master;

file "pri.example.com";

};

**Notes:**

1. The terms Primary and Secondary DNS entries in Windows TCP/IP network properties mean nothing, they may reflect the 'master' and 'slave' name-server or they may not - you decide this based on operational need, not BIND configuration.
2. It is important to understand that a zone 'master' is simply a server which gets its zone data from a local source as opposed to a 'slave' which gets its zone data from an external (networked) source (typically the 'master' but not always). This apparently trivial point means that you can have any number of 'master' servers for any zone if it makes operational sense. You have to ensure (by a manual or other process) that the zone files are synchronised but apart from this there is nothing to prevent it.
3. Just to confuse things still further you may run across the term 'Primary Master' this has a special meaning in the context of [dynamic DNS updates](http://www.zytrax.com/books/dns/ch2/index.html#dyn-update) and is defined to be the name server that appears in the [SOA RR record](http://www.zytrax.com/books/dns/ch8/soa.html).

When a master DNS receives [Queries](http://www.zytrax.com/books/dns/apa/query.html) for a zone for which it is authoritative then it will respond as 'Authoritative' (AA bit is set in a query response).

If a DNS server receives a query for a zone for which it is neither a Master nor a [Slave](http://www.zytrax.com/books/dns/ch4/#slave) then it will act as configured (in BIND this behaviour is defined in the[named.conf file)](http://www.zytrax.com/books/dns/apa/conf.html):

1. If [caching behaviour](http://www.zytrax.com/books/dns/ch4/#cache) is permitted and recursive queries are allowed the server will completely answer the request or return an error.
2. If [caching behaviour](http://www.zytrax.com/books/dns/ch4/#cache) is permitted and Iterative (non-recursive) queries are allowed the server can respond with the complete answer (if it is already in the cache because of another request), a referral or return an error.
3. If caching behaviour is NOT permitted (an ['Authoritative Only'](http://www.zytrax.com/books/dns/ch4/#authoritative) DNS server) the server will return a referral or an error.

A master DNS server can NOTIFY zone changes to defined (typically slave) servers - this is the default behaviour. NOTIFY messages ensure zone changes are rapidly propagated to the slaves (interrupt driven) rather than rely on the slave server periodically polling for changes. The BIND default is to notify the servers defined in [NS records](http://www.zytrax.com/books/dns/ch8/ns.html) for the zone - except itself, obviously.

A zone master can be 'hidden' (only one or more of the slaves know of its existence). There is no requirement in such a configuration for the master server to appear in an NS RR for the domain. The only requirement is that two (or more) name servers support the zone. Both servers could be any combination of master-slave, slave-slave or even master-master.

If you are running [Stealth Servers](http://www.zytrax.com/books/dns/ch4/#stealth) and wish them to be notified you will have to add an [also-notify parameter](http://www.zytrax.com/books/dns/ch7/xfer.html#also-notify) as shown in the BIND [named.conf](http://www.zytrax.com/books/dns/apa/conf.html)file fragment below:

// example.com fragment from named.conf

// defines this server as a zone master

// 192.168.0.2 is a stealth server NOT listed in a NS record

zone "example.com" in{

type master;

also-notify {192.168.0.2;};

file "pri/pri.example.com";

};

You can turn off all NOTIFY operations by specifying ['notify no'](http://www.zytrax.com/books/dns/ch7/xfer.html#notify) in the zone declaration.

Example configuration files for a master DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#master).

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.2 Slave Name Servers

A Slave DNS gets its zone data using a zone transfer operation (typically from a zone master) and it will respond as authoritative for those zones for which it is defined to be a 'slave' and for which it has a currently valid zone configuration. It is impossible to determine from a query result that it came from a zone master or slave.

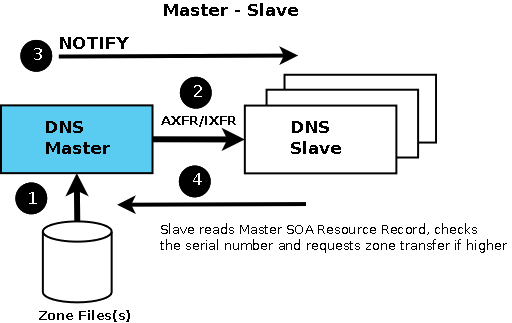


Diagram 2 DNS Slave Server

The term 'slave' was introduced with BIND 8.x and replaced the term 'secondary'.

There can be any number of slave DNS's for any given zone.

Slave status is defined in BIND by including 'type slave' in the zone declaration section of the [named.conf file](http://www.zytrax.com/books/dns/apa/conf.html) as shown by the following fragment.

// example.com fragment from named.conf

// defines this server as a zone slave

zone "example.com" in{

type slave;

file "sec/sec.example.com";

masters {192.168.23.17;};

};

**Notes:**

1. The master DNS for each zone is defined in the 'masters' statement of the zone clause and allows slaves to refresh their zone record when the 'expiry' parameter of the [SOA Record](http://www.zytrax.com/books/dns/ch8/soa.html) is reached. If a slave cannot reach the master DNS when the 'expiry' time has been reached it will stop responding to requests for the zone. It will not use time-expired data.
2. The file parameter is optional and allows the slave to write the transferred zone to disc and hence if BIND is restarted before the 'expiry' time the Slave server will use the saved data. In large DNS systems this can save a considerable amount of network traffic.

Assuming NOTIFY is allowed in the master DNS for the zone (the default behaviour) then zone changes are propagated to all the servers defined with [NS Records](http://www.zytrax.com/books/dns/ch8/ns.html) in the zone file. Other acceptable NOTIFY sources can be defined using the [also-notify](http://www.zytrax.com/books/dns/ch7/xfer.html#also-notify) parameter in named.conf.

Example configuration files for a slave DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#slave).

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.2.1 But Slaves can also be Masters

Oh, stop this pain. This section can get a bit confusing. Read it only when accompanied by your favorite keep-me-awake-cos-I-can't-take-anymore-of-this-stuff beverage.

The definition of a slave server is simply that it gets its zone data via zone transfer, whereas a master gets its zone data from a local file system. The source of the zone transfer could just as easily be another slave as a master.

So what sane human would want to do that?

1. Assume you want to hide your master servers in, say, a [stealth](http://www.zytrax.com/books/dns/ch4/#stealth) configuration then at least one slave server will sit on the public side of a firewall, or similar configuration, providing perimeter defence. To provide resilience you would need two or more such public slaves. The second slave can be updated from the same master as the first or it could be updated from the slave server - we'll call it the 'boss' slave to avoid getting into tortuous terminology (is it a master-slave or a slave-master?). To configure this miracle the second slave server would define the 'boss' slave's IP in its [masters](http://www.zytrax.com/books/dns/ch7/zone.html#masters) statement. When the 'boss' slave has sucessfully transfered a zone file (from the master) it will send out NOTIFY messages (the default) unless configured not to do so. This type of configuration will marginally increase latency for updating the zone on the second slave - but that may be more than offset by increased stealth.
2. In a DNSSEC environment the master will likely have all kinds of whizzo dodads concerned with keeping keys secure. Whereas DNSSEC slaves simply send the data in the zone file in response to queries and have no requirements for secure key maintenance. Hidden master configurations will become increasingly the norm in this environment.

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.3 Caching Name Servers (Resolver)

A DNS Caching Server (frrequently called a Resolver) obtains information from another server (a Zone Master) in response to a host query and then saves (caches) the data locally. On a second or subsequent request for the same data the Caching Server (Resolver) will respond with its locally stored data (the cache) until the [time-to-live (TTL)](http://www.zytrax.com/books/dns/apa/ttl.html) value of the response expires, at which time the server will refresh the data from the zone master.

If the caching server (resolver) obtains its data directly from a zone master it will respond as 'authoritative', if the data is supplied from its cache the response is 'non-authoritative'.

The default BIND behaviour is to cache and this is associated with the [recursion](http://www.zytrax.com/books/dns/ch7/queries.html#recursion) parameter (the default is 'recursion yes'). There are many configuration examples which show caching behaviour being defined using a *type hint* statement in a zone declaration. These configurations confuse two distinct but related functions. If a server is going to provide caching services then it must support [recursive queries](http://www.zytrax.com/books/dns/ch2/index.html#recursive) and recursive queries need access to the root servers which is provided via the 'type hint' statement.

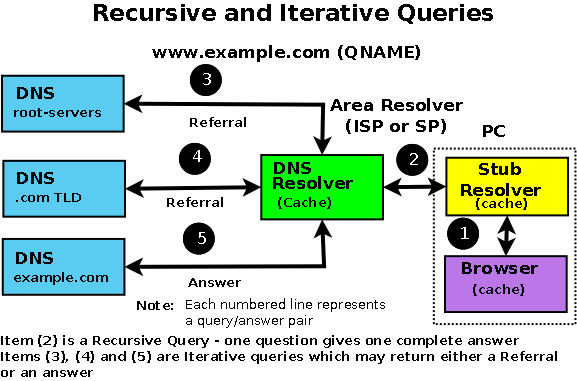


Diagram 3 DNS Resolver (Recursive Server)

A caching server will typically have a [named.conf file](http://www.zytrax.com/books/dns/apa/conf.html) which includes the following fragment:

// options section fragment of named.conf

// recursion yes is the default and may be omitted

options {

directory "/var/named";

version "not currently available";

recursion yes;

};

// zone section

....

// the DOT indicates the root domain = all domains

zone "." IN {

type hint;

file "root.servers";

};

**Notes:**

1. BIND defaults to [recursive queries](http://www.zytrax.com/books/dns/ch2/index.html#recursive) which by definition provides caching behaviour. The named.conf [recursion](http://www.zytrax.com/books/dns/ch7/queries.html#recursion) parameter controls this behaviour.
2. The zone '.' is shorthand for the root domain which translates to 'any domain not defined as either a master or slave in this named.conf file'.
3. cache data is discarded when BIND is restarted.

The most common DNS server caching configurations are:

* A DNS server acting as master or slave for one or more zones (domains) and as cache server for all other requests. A general purpose DNS server.
* A caching only local server - typically used to minimise external access or to compensate for slow external links. This is sometimes called a Proxy server though we prefer to associate the term with a [Forwarding server](http://www.zytrax.com/books/dns/ch4/#forwarding)

To cache or not is a crucial question in the world of DNS. BIND is regarded as the reference implementation of the DNS specification. As such it provides excellent - if complex to configure - functionality. The down side of generality is suboptimal performance on any single function - in particular caching involves a non-trivial performance overhead.

For general usage the breadth of BIND functionality typically offsets any performance concerns. However if the DNS is being 'hit' thousands of times per second performance is a major factor. There are now [a number of alternate Open Source DNS servers](http://www.zytrax.com/books/dns/apc/) some of which stress performance. These servers typically do NOT provide caching services (they are said to be ['Authoritative only' servers)](http://www.zytrax.com/books/dns/ch4/#authoritative).

Example configuration files for a caching DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#caching).

**Note:** The response to a query is Authoritative under three conditions:

1. The response is received from a Zone master.
2. The response is received from a Zone slave with non time-expired zone data.
3. The response is received by a caching server directly from either a Zone master or slave. If the response is supplied from the cache it is not authoritative.

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.4 Forwarding (a.k.a Proxy) Name Servers

A forwarding (a.k.a. Proxy, Client, Remote) server is one which simply forwards requests to another DNS and caches the results. On its face this looks like a pretty pointless exercise. It is, however, a frequently undervalued and extremely useful configuration in a number of situations:

1. Where access to the external network is slow or expensive:
   1. Local DNS caching - results are cached in the forwarding server so that frequently requested domains will provide fast results from the cache.
   2. The Remote (forwarded to) DNS server provides recursive query support resulting in a single query across the network (from the forwarding DNS to the 'forwared to' DNS) thus reducing traffic congestion (on busy networks), traffic volume (on expensive networks) and increasing performance (on slow networks).
2. Forwarding servers also can be used to ease the burden of local administration by providing a single point at which changes to remote name servers may be managed, rather than having to update all hosts. Thus, all hosts in a particular network section or area can be configured to point to a fixed forwarding DNS which can be configured to stream DNS traffic as desired and changed over time with minimal effort.
3. Sanitizing traffic. Especially in larger private networks it may be sensible to stream DNS traffic for local domain access by forwarding to the local DNS servers while forwarding external DNS requests to a **dirty** or hardened caching DNS (or resolver).
4. Forwarding can also be used as part of a [Split Server](http://www.zytrax.com/books/dns/ch4/#stealth) configuration for perimeter defence.

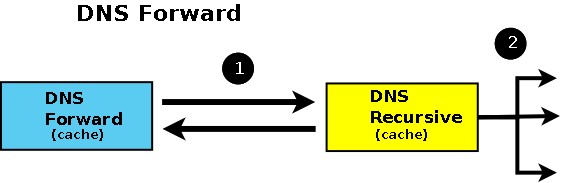


Diagram 4 - DNS Forvarding Server

BIND allows configuration of forwarding using the [forward](http://www.zytrax.com/books/dns/ch7/queries.html#forward) and [forwarders](http://www.zytrax.com/books/dns/ch7/queries.html#forwarders) parameters either at a 'global' level (in an options section) or on a per-zone basis in a zone section of the [named.conf file](http://www.zytrax.com/books/dns/apa/conf.html). Both configurations are shown in the examples below:

#### Global Forwarding - All Requests

// options section fragment of named.conf

// forwarders can have multiple choices

options {

directory "/var/named";

version "not currently available";

forwarders {10.0.0.1; 10.0.0.2;};

forward only;

};

// zone file sections

....

#### Per Domain Forwarding

// zone section fragment of named.conf

zone "example.com" IN {

type forward;

forwarders {10.0.0.1; 10.0.0.2;};

};

Where dial-up links are used with DNS forwarding servers BIND's general purpose nature and strict standards adherence may not make it an optimal solution. A number of the [Alternate DNS solutions](http://www.zytrax.com/books/dns/apc/) specifically target support for such links. BIND provides two parameters dialup and heartbeat-interval (neither of which is currently supported by BIND 9) as well as a number of others which can be used to minimise connection time.

Example configuration files for a forwarding DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#forwarding).

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.5 Stealth (a.k.a. DMZ or Hidden Master) Name Server

A stealth server is defined as being a name server which does not appear in any **publicly visible** [NS Records](http://www.zytrax.com/books/dns/ch8/ns.html) for the domain. The stealth server can be roughly defined as having the following characteristics:

1. The organisation needs a public DNS to enable access to its public services e.g. web, mail ftp etc..
2. The organisation does not want the world to see any of its internal hosts either by interrogation (query or zone transfer) or should the DNS service be compromised.

A Stealth configuration is shown in Figure 4-5.

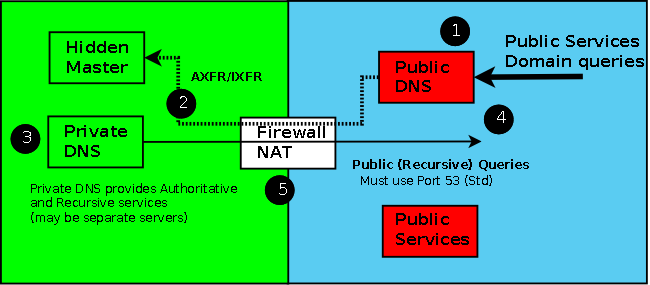


Figure 4-5 Stealth Server Topology

The external server(s) is(are) configured to provide [Authoritative Only](http://www.zytrax.com/books/dns/ch4/#authoritative) responses and no caching (no recursive queries accepted). The zone file for this server would be unique and would contain ONLY those systems or services that are publicly visible e.g. SOA, NS records for the public (not stealth) name servers, MX record(s) for mail servers and www and ftp service A records. Zone transfers can be allowed between the public servers as required but they MUST NOT transfer or accept transfers from the Stealth server. While this may seem to create more work, the concern is that should the host running the external service be compromised then inspection of the named.conf or zone files must provide no more information than is already publically visible. If 'master', 'allow-notify','allow-transfer' options are present in named.conf (each of which will contain a private IP) then the attacker has gained more knowledge about the organisation - they have penetrated the 'veil of privacy'.

There are a number of articles which suggest that the [view statement](http://www.zytrax.com/books/dns/ch7/view.html) may be used to provide similar functionality using a single server but this does not address the problem of the DNS host system being compromised and by simple inspection of the named.conf file additional data about the organisation could be discovered. In our opinion 'view' does not provide adequate security in a 'Split DNS' solution.

A minimal public zone file is shown below:

; public zone master file

; provides minimal public visibility of external services

example.com. IN SOA ns.example.com. root.example.com. (

2003080800 ; se = serial number

3h ; ref = refresh

15m ; ret = update retry

3w ; ex = expiry

3h ; min = minimum

)

IN NS ns1.example.com.

IN NS ns2.example.com.

IN MX 10 mail.example.com.

ns1 IN A 192.168.254.1

ns2 IN A 192.168.254.2

mail IN A 192.168.254.3

www IN A 192.168.254.4

ftp IN A 192.168.254.5

The internal server (the Stealth Server) can be configured to make visible internal and external services, provide recursive queries and all manner of other services. This server would use a private zone master file which could look like this:

; private zone master file used by stealth server(s)

; provides public and private services and hosts

example.com. IN SOA ns.example.com. root.example.com. (

2003080800 ; se = serial number

3h ; ref = refresh

15m ; ret = update retry

3w ; ex = expiry

3h ; min = minimum

)

IN NS ns1.example.com.

IN NS ns2.example.com.

IN MX 10 mail.example.com.

; public hosts

ns1 IN A 192.168.254.1

ns2 IN A 192.168.254.2

mail IN A 192.168.254.3

www IN A 192.168.254.4

ftp IN A 192.168.254.5

; private hosts

joe IN A 192.168.254.6

bill IN A 192.168.254.7

fred IN A 192.168.254.8

....

accounting IN A 192.168.254.28

payroll IN A 192.168.254.29

Using BIND 9's [view](http://www.zytrax.com/books/dns/ch7/view.html) statement can provide different services to internal and external requests can reduce further the Stealth server's visibility e.g. forwarding all DNS internal requests to the external server.

Example configuration files for a stealth DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#stealth).

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.6 Authoritative Only Server

The term **Authoritative Only** is normally used to describe two concepts:

1. The server will deliver Authoritative Responses - it is a zone master or slave for one or more domains.
2. The server will NOT cache.

There are two configurations in which Authoritative Only servers are typically used:

1. As the public or external server in a [Stealth (a.k.a. DMZ or Hidden Master) DNS](http://www.zytrax.com/books/dns/ch4/#stealth) used to provide perimeter security.
2. High Performance DNS servers. In this context general purpose DNS servers such as BIND may not provide an ideal solution and there are a number of [Open Source Alternatives](http://www.zytrax.com/books/dns/apc/) some of which specialise in high performance Authoritative only solutions.

You cannot completely turn off caching in BIND but you can control it and provide the functionality described above by simply turning off recursion in the 'option' section of named.conf as shown in the example below.

// options section fragment of named.conf

// recursion no = limits caching

options {

directory "/var/named";

version "not currently available";

recursion no;

};

// zone file sections

....

BIND provides three more parameters to control caching, [max-cache-size](http://www.zytrax.com/books/dns/ch7/queries.html#max-cache-size) and [max-cache-ttl](http://www.zytrax.com/books/dns/ch7/queries.html#max-cache-ttl) neither of which will have much effect on performance in this particular case and [allow-recursion](http://www.zytrax.com/books/dns/ch7/queries.html#allow-recursion) which uses a list of hosts that are permitted to use recursion (all others are not).

Example configuration files for a authoritative-only DNS [are provided](http://www.zytrax.com/books/dns/ch6/index.html#authoritative).

[up icon](http://www.zytrax.com/books/dns/ch4/#contents)

## 4.7 Split Horizon DNS Server

This section was introduced at the suggestion of Maren Leizaola - many thanks for both taking the time to write and for providing interesting usage examples.

The term **Split Horizon** is normally used to describe a DNS server that will give different responses (IP addresses) based on the source address, or some other characteristic, of the query. While it has similar configuration properties to the [Stealth Server](http://www.zytrax.com/books/dns/ch4/#stealth) it can also be used in a varity of unique situations such as:

1. **Geographic Mapping:** Assume that, for example, a web service is replicated in a number of locations (for either performance or access latency reasons) then a specific IP address may be returned based on the source address of the query to ensure the shortest possible path from the user to the service. For those familiar with **anycast** you could consider this as a poor man's anycast service.
2. **Naming Consistency:** Assume that you have, say, a corporate in-house LDAP service and that you want to keep certain highly secure data on one server only accessible to certain individuals or organizational sections, which have unique or identifiable IP addresses or address ranges, but for reasons of consistency (scripts, configuration files etc) you want both the secure and insecure LDAP services to be named, say, ldap.example.com.
3. **Load Balancing:** Assume that an analysis of incoming service users shows that their source-ip addresses can be separated into contiguous ranges: 50% from a to b, 50% from b to c. In this case rather than simply provide multiple A/AAAA RRs (where load balancing is essentially random) it may be more effective to use a split-horizon strategy.

Other possibilities may strike imaginative readers. The unifying element is that some characteristic of the incoming query will cause the DNS to generate a query-dependent result.

BIND's [view](http://www.zytrax.com/books/dns/ch7/view.html) clause provides a method that can be used to build such configurations and example files [are provided](http://www.zytrax.com/books/dns/ch6/index.html#split-view) .

The best way to understand the difference between DNS servers is to think of them in three groups which correspond to three basic aspects of the domain naming system:

1) Authoritative Nameservers  
2) Resolvers  
3) Root Servers

### Authoritative Nameservers

Authoritative Nameservers handle name-to-number translation for a specific set of domain names. If you are an easyDNS customer and use our DNS, then our name servers are the authoritative name servers for your domain. In other cases your web hosting provider may run your name servers for you.

### Resolvers

Resolvers come at this from the other direction: they handle name-to-number translation for every domain name you would ever want to know about. As you are surfing the web or sending email, it’s your resolvers that are going out to various authoritative name servers and getting the IP addresses of whatever hostnames you are after. Resolvers are transparent to most internet users. They are set automatically by your operating system, often via DHCP from your connectivity provider.  Most of the time, your ISP is operating resolvers that their customers end up using.

In recent years third-party resolver services have also come into existence. The most popular ones being [**OpenDNS**](http://opendns.com/) and [**Google’s Public DNS service**](http://code.google.com/speed/public-dns/docs/using.html). easyDNS also offers [**DNSResolvers.com**](http://dnsresolvers.com/).

### Root Servers

Root Servers are the final piece. How does a resolver know \*which\* authoritative name server to send a query to? By asking the root servers. All the root servers do is maintain a list of what authoritative name servers are responsible for each domain name and top-level-domain.

### Why You Should Know This

Once you understand the role of authoritative, resolver and root servers you can begin to debug certain issues by sending the appropriate queries to the relevant servers.

For example, if you cannot reach a given webserver, your browser is throwing a “hostname not found” error you would like to know why.

If the domain’s authoritative DNS servers are not answering for it, it’s a problem with the domain itself. It could be specific to the domain, or affect all domains served by those nameservers.

If you can’t get anywhere or most places are throwing the same error, then it is your local resolvers that are failing or having issues.

If the domain is not present in the relevant root servers, that domain name may have expired.

There is also a reason why we posted this today, as our next mailing will have important security alert about Resolvers that everybody should be aware of.