**DNS**

**Basic Terminology**

**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. 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.

**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.

**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).

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** - 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". 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.".

**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**

**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" 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". 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" is located.

Once again, the requester will look for "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".

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 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**

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 ; exiry 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 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.

**Load Balancing**

Load Balancing refers to efficiently distributing incoming network traffic across a group of back-end servers, also known as a “server farm” or “server pool”.

Modern high-traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

* Distributes client requests or network load efficiently across multiple servers
* Ensures high availability and reliability by sending requests only to servers that are online
* Provides the flexibility to add or subtract servers as demand dictates

**Load Balancing Algorithms**

Different load balancing algorithms provide different benefits; the choice depends on your needs:

* Round-robin –Requests are distributed across the group of servers sequentially.
* Least connections – A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.
* IP hash – The IP address of the client is used to determine which server receives the request.

**Session Persistence**

Information about a user’s session is often stored locally in the browser. For example, in a shopping cart application the items in a user’s cart might be stored at the browser level until the user is ready to purchase them. Changing which server receives requests from that client in the middle of the shopping session can cause performance issues or outright transaction failure. In such cases, it is essential that all requests from a client are sent to the same server for the duration of the session. This is known as *session persistence*.

The best load balancers can handle session persistence as needed. Another use case for session persistence is when an upstream server stores information requested by a user in its cache to boost performance. Switching servers would cause that information to be fetched for the second time, creating performance inefficiencies.

**Dynamic Configuration of Server Groups**

Many fast-changing applications require new servers to be added or taken down on a constant basis. This is common in environments such as the Amazon Elastic Compute Cloud (EC2), which enables users to pay only for the computing capacity they actually use, while at the same time ensuring that capacity scales up in response traffic spikes. In such environments it greatly helps if the load balancer can dynamically add or remove servers from the group without interrupting existing connections.

**Hardware vs. Software Load Balancing**

Load balancers typically come in two flavors: hardware-based and software-based. Vendors of hardware-based solutions load proprietary software onto the machine they provide, which often uses specialized processors. To cope with increasing traffic at your website, you have to buy more or bigger machines from the vendor. Software solutions generally run on commodity hardware, making them less expensive and more flexible. You can install the software on the hardware of your choice or in cloud environments like AWS EC2.

<http://www.slideshare.net/AmazonWebServices/availability-scalability-with-elastic-load-balancing-route-53-cpn204-aws-reinvent-2013?qid=2d3e1261-4b90-41cf-b3cb-a8eeb804e11e&v=default&b=&from_search=2/>

**Proxy Server**

A proxy server, also known as a "proxy" or "application-level gateway", is a computer that acts as a gateway between a local network (e.g., all the computers at one company or in one building) and a larger-scale network such as the Internet. Proxy servers provide increased performance and security. In some cases, they monitor employees' use of outside resources.

A proxy server works by intercepting connections between sender and receiver. All incoming data enters through one port and is forwarded to the rest of the network via another port. By blocking direct access between two networks, proxy servers make it much more difficult for hackers to get internal addresses and details of a private network.

Some proxy servers are a group of applications or servers that block common Internet services. For example, an HTTP proxy intercepts web access, and an SMTP proxy intercepts email. A proxy server uses a network addressing scheme to present one organization-wide IP address to the Internet. The server funnels all user requests to the Internet and returns responses to the appropriate users. In addition to restricting access from outside, this mechanism can prevent inside users from reaching specific Internet resources (e.g., certain websites). A proxy server can also be one of the components of a firewall.

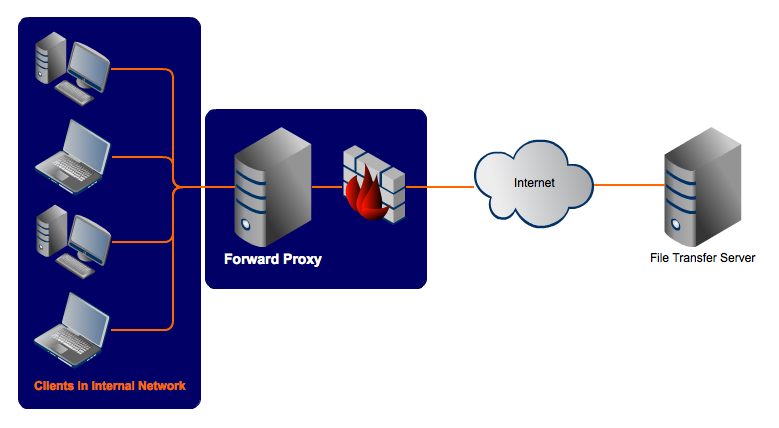
Proxies may also cache web pages. Each time an internal user requests a URL from outside, a temporary copy is stored locally. The next time an internal user requests the same URL, the proxy can serve the local copy instead of retrieving the original across the network, improving performance.

**Note:** Do not confuse a proxy server with a NAT (Network Address Translation) device. A proxy server connects to, responds to, and receives traffic from the Internet, acting on behalf of the client computer, while a NAT device transparently changes the origination address of traffic coming through it before passing it to the Internet.

## The Forward Proxy

When people talk about a proxy server (often simply known as a "proxy"), more often than not they are referring to a forward proxy. Let me explain what this particular server does.

A forward proxy provides proxy services to a client or a group of clients. Oftentimes, these clients belong to a common internal network like the one shown below.



When one of these clients makes a connection attempt to that file transfer server on the Internet, its requests have to pass through the forward proxy first.

Depending on the forward proxy's settings, a request can be allowed or denied. If allowed, then the request is forwarded to the firewall and then to the file transfer server. From the point of view of the file transfer server, it is the proxy server that issued the request, not the client. So when the server responds, it addresses its response to the proxy.

But then when the forward proxy receives the response, it recognizes it as a response to the request that went through earlier. And so it in turn sends that response to the client that made the request.

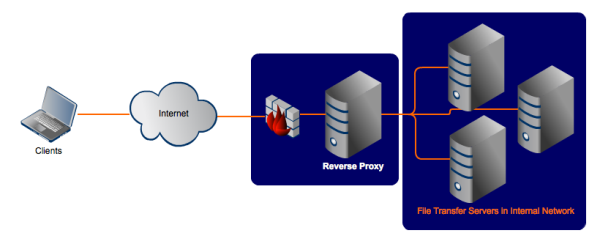
Because proxy servers can keep track of requests, responses, their sources and their destinations, different clients can send out various requests to different servers through the forward proxy and the proxy will intermediate for all of them. Again, some requests will be allowed, while some will be denied.

As you can see, the proxy can serve as a single point of access and control, making it easier for you to enforce security policies. A forward proxy is typically used in tandem with a firewall to enhance an internal network's security by controlling traffic originating from clients in the internal network that are directed at hosts on the Internet. Thus, from a security standpoint, a forward proxy is primarily aimed at enforcing security on client computers in your internal network.

But then client computers aren't always the only ones you find in your internal network. Sometimes, you also have servers. And when those servers have to provide services to external clients (e.g. field staff who need to access files from your FTP server), a more appropriate solution would be a reverse proxy.

## The Reverse Proxy

As its name implies, a reverse proxy does the exact opposite of what a forward proxy does. While a forward proxy proxies in behalf of clients (or requesting hosts), a reverse proxy proxies in behalf of servers. A reverse proxy accepts requests from external clients on behalf of servers stationed behind it just like what the figure below illustrates.



To the client in our example, it is the reverse proxy that is providing file transfer services. The client is oblivious to the file transfer servers behind the proxy, which are actually providing those services. In effect, whereas a forward proxy hides the identities of clients, a reverse proxy hides the identities of servers.

An Internet-based attacker would therefore find it considerably more difficult to acquire data found in those file transfer servers than if he wouldn't have had to deal with a reverse proxy. No wonder reverse proxy servers like

Just like forward proxy servers, reverse proxies also provide a single point of access and control. You typically set it up to work alongside one or two firewalls to control traffic and requests directed to your internal servers.

In most cases, reverse proxy servers also act as load balancers for the servers behind it. Load balancers play a crucial role in providing high availability to network services that receive large volumes of requests. When a reverse proxy performs load balancing, it distributes incoming requests to a cluster of servers, all providing the same kind of service. So, for instance, a reverse proxy load balancing FTP services will have a cluster of FTP servers behind it.

Both types of proxy servers relay requests and responses between source and destination machines. But in the case of reverse proxy servers, client requests that go through them normally originate from the Internet, while, in the case of forward proxies, client requests normally come from the internal network behind them.

**Miscellaneous**

**Install & Configure DNS Server**

**1.Network Information**

We’ll use “mydns.net” domain as an example for this DNS installation. “mail”, “web”, “ns” are the hosts that resides within this domain.

It is possible to configure a single system to act as a caching name server, primary/master and secondary/slave. We will configure this DNS as a Primay/Master as well as Caching DNS server.

We’ll be installing DNS server on “10.16.0.24”.

**2. Install Bind**

Install the bind9 package using the appropriate package management utilities for your Linux distributions.

On Debian/Ubuntu flavors, do the following:

$ sudo apt-get install bind9

On Redhat/CentOS/Fedora flavors, do the following:

# yum install bind9

All the DNS configurations are stored under /etc/bind directory. The primary configuration is /etc/bind/named.conf which will include other needed files. The file named /etc/bind/db.root describes the root nameservers in the world.

**3. Configure Cache NameServer**

The job of a DNS caching server is to query other DNS servers and cache the response. Next time when the same query is given, it will provide the response from the cache. The cache will be updated periodically.

Please note that even though you can configure bind to work as a Primary and as a Caching server, it is not advised to do so for security reasons. Having a separate caching server is advisable.

All we have to do to configure a Cache NameServer is to add your ISP (Internet Service Provider)’s DNS server or any OpenDNS server to the file /etc/bind/named.conf.options.

For Example, we will use google’s public DNS servers, 8.8.8.8 and 8.8.4.4.

Uncomment and edit the following line as shown below in /etc/bind/named.conf.options file.

forwarders {

8.8.8.8;

8.8.4.4;

};

After the above change, restart the DNS server.

$ sudo service bind9 restart

**4. Test the Cache NameServer**

You can use the dig command to test DNS services.

$ dig ubuntu.com

;; Query time: 1323 msec

Now when the second time you execute the dig, there should be an improvement in the Query time. As you see below, it took only 3 msec the second time, as it is getting the info from our caching DNS server.

$ dig ubuntu.com

;; Query time: 3 msec

**5. Configure Primary/Master Nameserver**

Next, we will configure bind9 to be the Primary/Master for the domain/zone “mydns.net”.

As a first step in configuring our Primary/Master Nameserver, we should add Forward and Reverse resolution to bind9.

To add a DNS Forward and Reverse resolution to bind9, edit /etc/bind9/named.conf.local.

zone "mydns.net" {

type master;

file "/etc/bind/db.mydns.net";

};

zone "0.16.10.in-addr.arpa" {

type master;

notify no;

file "/etc/bind/db.10";

};

Now the file /etc/bind/db.mydns.net will have the details for resolving hostname to IP address for this domain/zone, and the file /etc/bind/db.10 will have the details for resolving IP address to hostname.

**6. Build the Forward Resolution for Primary/Master NameServer**

Now we will add the details which is necessary for forward resolution into /etc/bind/db.mydns.net.

First, copy /etc/bind/db.local to /etc/bind/db.mydns.net

$ sudo cp /etc/bind/db.local /etc/bind/db.mydns.net

Next, edit the /etc/bind/db.mydns.net and replace the following.

1. In the line which has SOA: localhost. – This is the FQDN of the server in charge for this domain. I’ve installed bind9 in 10.16.0.24, whose hostname is “ns”. So replace the “localhost.” with “ns.mydns.net.”. Make sure it end’s with a dot(.).
2. In the line which has SOA: root.localhost. – This is the E-Mail address of the person who is responsible for this server. Use dot(.) instead of @.
3. In the line which has NS: localhost. – This is defining the Name server for the domain (NS). We have to change this to the fully qualified domain name of the name server. Change it to “ns.mydns.net.”. Make sure you have a “.” at the end.

Next, define the A record and MX record for the domain. A record is the one which maps hostname to IP address, and MX record will tell the mailserver to use for this domain.

Once the changes are done, the /etc/bind/db.mydns.net file will look like the following:

$TTL 604800

@ IN SOA ns.mydns.net. user.localhost. (

1024 ; Serial

604800 ; Refresh

86400 ; Retry

2419200 ; Expire

604800 ) ; Negative Cache TTL

;

@ IN NS ns.mydns.net.

mydns.net. IN MX 10 mail.mydns.net.

ns IN A 10.16.0.24

web IN A 10.16.0.28

mail IN A 10.16.0.30

**6. Build the Reverse Resolution for Primary/Master NameServer**

We will add the details which are necessary for reverse resolution to the file /etc/bind/db.10. Copy the file /etc/bind/db.127 to /etc/bind/db.10

$ sudo cp /etc/bind/db.127 /etc/bind/db.10

Next, edit the /etc/bind/db.10 file, and basically changing the same options as /etc/bind/db.mydns.net

$TTL 604800

@ IN SOA ns.mydns.net. root.localhost. (

20 ; Serial

604800 ; Refresh

86400 ; Retry

2419200 ; Expire

604800 ) ; Negative Cache TTL

;

@ IN NS ns.

Next, for each A record in /etc/bind/db.mydns.net, add a PTR record.

$TTL 604800

@ IN SOA ns.mydns.net. root.mydns.net. (

20 ; Serial

604800 ; Refresh

86400 ; Retry

2419200 ; Expire

604800 ) ; Negative Cache TTL

;

@ IN NS ns.

24 IN PTR ns.mydns.net.

30 IN PTR mail.mydns.net.

28 IN PTR web.mydns.net.

Whenever you are modifying the file db.mydns.net and db.10, you need to increment the “Serial” number as well. Typically admin uses DDMMYYSS for serial numbers and when they modify, the change the serial number appropriately.

Finally, restart the bind9 service:

$ sudo service bind9 restart

**7. Test the DNS server**

Now we have configured the DNS server for our domain. We will test our DNS server by pinging mail.mydns.net from web.mydns.net.

If the ping is success, then we have configured the DNS successfully.

You can also use nslookup and dig to test DNS servers.

On server, add the following to /etc/resolv.conf

nameserver 10.16.0.24

Now ping, mail.mydns.net, which should resolve the address appropriately from the DNS server that we just configured.

$ ping mail.mydns.net

PING mail.mydns.net (10.16.0.30) 56(84) bytes of data.

64 bytes from mail.mydns.net (10.42.0.30): icmp\_req=1 ttl=64 time=0.482 ms

64 bytes from mail.mydns.net (10.42.0.30): icmp\_req=2 ttl=64 time=0.532 ms