CHAPTER 9

Configuring Network Infrastructure Services Using DNS, DHCP, and NTP

Linux servers are often used to configure services that help make networking happen. These services include DNS for name resolution, DHCP for IP address configuration, and NTP for time services. In this chapter, you'll read how to configure them. You'll also read how to enable some common Linux services using `xinetd`.

Configuring DNS

As you would expect, IP (Internet protocol) is used for all communications on the Internet. This protocol specifies unique IP addresses that computers use to talk to one another. To contact a computer, you just need to know its IP address. One of the most important reasons why the domain name system (DNS) was developed is because computers work better with numbers than humans do, and humans tend to prefer names. So, DNS translates IP addresses to DNS names (and back from DNS names to IP addresses). In this chapter, you'll learn how to configure DNS on Ubuntu Server.

Methods of Name Resolution

Before going into detail about configuring DNS servers, you first need to learn exactly what DNS is and how it works. In this section, you'll read about the differences between DNS and other methods of resolving names. You'll also find out how the DNS hierarchy is structured and what roles the different types of DNS servers play in this hierarchy.

DNS is not the only solution that you can use for name resolving. Let's have a quick look at two of the alternative methods: the `/etc/hosts` file and Sun's Network Information System (NIS).

Managing Host Name Information with the `/etc/hosts` File

Before centralized systems such as NIS and DNS were introduced, every host kept its own file that mapped IP addresses to names. In the days when the Internet was called (D)ARPANet and was still a very small network, this was a feasible solution, although the administrator had to make sure that these files were updated properly. Such a mechanism still exists, but in the form of the /etc/hosts file. In this file, you can keep a list of commonly used names and their IP addresses. Ubuntu Server creates this file by default to make sure that the localhost can be resolved. Listing 9-1 shows an example of the file. Note that you can still use this file as an addition to DNS. Depending on the settings in /etc/nsswitch.conf, its contents will be checked first before any DNS lookup.

Listing 9-1. *Displaying the Contents of* /etc/hosts

```
root@RNA:~# cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       RNA.lan RNA

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Using NIS to Manage Name Resolution

A more convenient method that you can use to keep mappings between host names and IP addresses is Sun's NIS, also known as the Yellow Pages. This system uses a database for important configuration files on a server, such as the /etc/hosts file, the /etc/passwd file, and /etc/shadow. As an administrator, you can determine for yourself what files to manage with NIS. These files are converted to NIS maps, which are the indexed files that comprise the NIS database. In NIS, one server is configured as the master server, which maintains the NIS database. All nodes are configured as NIS clients and send their name resolution requests to the NIS master. To provide redundancy, NIS can also use slave servers, which offer a read-only copy of the NIS master database. However, the master server is the single point of administration.

Although NIS was a good solution to manage relevant information within a network, it never became very popular as an Internet-level name service mainly because NIS does not provide a hierarchical solution, only flat databases. All these flat databases are managed by local administrators, and there's no relation among the databases that are used in different NIS domains.

The large amount of information on the Internet today makes it impossible to get quick results from a structure like NIS. For this reason, most organizations that still use NIS are phasing it out and configuring DNS to resolve host names to IP addresses and LDAP to manage user information (therefore, NIS is not covered in this book).

Managing Search Order with the /etc/nsswitch.conf File

Although DNS is the main system used for name resolution, it's not the only one. You can set it up in parallel with a NIS system and the /etc/hosts file. If you do this, the order in which the different systems are searched is important. The search order is determined by the `/etc/nsswitch.conf` file; see Listing 9-2 for an example.

Listing 9-2. *Contents of the* `/etc/nsswitch.conf` *File*

```
root@RNA:~# cat /etc/nsswitch.conf
# /etc/nsswitch.conf
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the 'glibc-doc-reference' and 'info' packages installed, try:
# 'info libc "Name Service Switch"' for information about this file.

passwd:         compat
group:          compat
shadow:         compat

hosts:          files dns
networks:       files

protocols:      db files
services:       db files
ethers:         db files
rpc:            db files

netgroup:       nis
```

For all the important information on your server, the `nsswitch.conf` file contains an indication of where it should be searched. In the case of hosts and network information, the example file is pretty clear: it first checks local configuration files and only after that does it check the DNS hierarchy. This means that you can use `/etc/hosts` to override information as defined in DNS.

Structure of the DNS Hierarchy

The most important advantage offered by DNS is that it's organized in a hierarchical manner. This makes the system very scalable because it can be extended by simply adding another branch to the tree-like hierarchy.

On top of the hierarchy are the root servers, which have one purpose only: to provide information about the top-level domains (TLDs). Some fixed domain names are used for top-level domains, including `.com`, `.org`, and `.info`. TLDs exist for all countries as well, such as `.nl`, `.uk`, `.fr`, and so on. Within these TLDs, persons and organizations can create their own domains, which can contain subdomains as well. For example, an organization could create a domain called `example.com` and, within the structure of `example.com`, it could create some subdomains as well, such as `east.example.com` and `west.example.com`.

The number of subdomains is virtually unlimited, although it becomes hard to work with more than four or five levels of domains. No one wants to type `www.servers.east.nl.sandervanvugt.com` all the time, do they? Figure 9-1 provides an example of the partial DNS hierarchy.

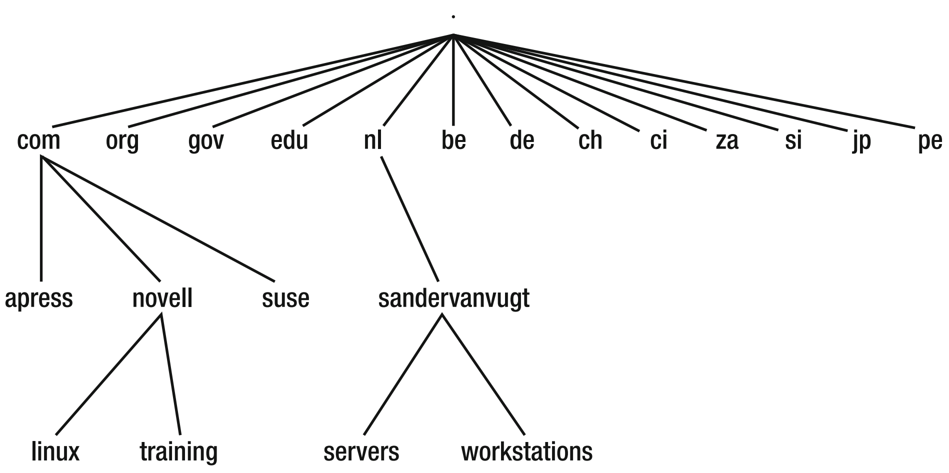

Figure 9-1. *Example of a part of the DNS hierarchy*

Master and Slave Servers

Within the DNS hierarchy, different servers are responsible for the data in specific domains (and sometimes subdomains as well). These are the so-called name servers and the part of the hierarchy that they are responsible for is the zone. A zone can include more than just one domain; for example, if one name server is responsible for everything in `sandervanvugt.nl`, including the subdomain's servers and workstations, the complete zone is `sandervanvugt.nl`. If, however, there's a subdomain called `sales.sandervanvugt.com` that has its own name server, the subdomain would be a zone by itself that is not part of `sandervanvugt.com`. Speaking in a very generic way, a zone is just a branch of the DNS hierarchy.

All zones should have at least two name servers. The first is the master name server, which is ultimately responsible for the data in a zone. For fault tolerance and to make the information more accessible, it's a good idea to use one or more slave servers as well. These slave servers will periodically get an update of all the data on the master server by means of a zone transfer; this is the process the master server uses to update the database on the slave server.

Note that DNS uses a single-master model: updates are performed on the master server and nowhere else, and the databases on the slave servers are read-only. You should also know that the name servers do not need to be in the zone that they are responsible for. For example, the name server of a given domain will often be hosted by the Internet provider that (of course) has its own domain. You can maintain your own DNS server, and it's useful to do so if your organization is larger than average, but you don't have to. You can also just purchase a domain and have your Internet server do the name server maintenance work.

Connecting the Name Servers in the Hierarchy

Because DNS uses a hierarchy, the servers in DNS need to know about each other, and this is a two-way process by its very nature. First, all servers in subordinate zones need to know where to find the root servers of the DNS hierarchy. Equally, the servers of the upper-level zones need to know how to find the servers of lower-level zones. You can very well create your own DNS domain called `mynicednsdomain.com` and run your DNS server in it, but it doesn't make sense if the DNS server that's responsible for the `.com` domain doesn't know about it. This is because a client trying to find your server will first ask the name server of the domain above your zone if it knows where to find authoritative information for your domain.

This is why DNS domain names need to be registered. Only then can the manager of the domain above yours configure your name server as the responsible name server for your domain. This is the delegation of authority.

It also helps to understand what happens when a user tries to resolve a DNS name that it doesn't know about already. The next procedure describes what happens:

1. To resolve DNS names, you need to configure the DNS resolver on the user's workstation or on the server that needs to be part of the DNS hierarchy. The DNS resolver is the part of the workstation where the user has configured how to find a DNS server. On a Linux system, this happens in the `/etc/resolv.conf` file.

2. Based on the information in the DNS resolver, the client contacts its preferred name server and asks that server to resolve the DNS name, no matter what server it is and where on Earth the server is running. So, if the client tries to resolve the name `www.sandervanvugt.nl`, it first asks its preferred name server. The advantage is that the client's name server can consult its cache to find out whether it has recently resolved that name for the client. If it knows the IP address of the requested server, the DNS name server returns that information to the client immediately.

3. If the name server of the client doesn't know the IP address of the requested server, it sees whether a forwarder is configured. (A *forwarder* is just a server that a name server contacts if it can't resolve a name by itself.)

4. If no forwarder is configured, the DNS name server contacts a name server of the root domain and asks that name server how to contact the name server of the top-level domain it needs. In the example in which you want to reach the host `www.sandervanvugt.nl`, this is the name server for the `.nl` domain.

5. Once the name server of the client finds the name server address of the top-level domain, it contacts it and asks for the IP address of the authoritative name server for the domain it's looking for. In our example, this would be the name server for `sandervanvugt.nl`.

6. Once the name server of the client finds out how to reach the authoritative name server for the domain the client asks for, it contacts that name server and asks to resolve its name. In return, the name server of the client receives the IP address it needs.

7. Ultimately, the IP address of the desired server is returned to the client, and contact can be established.

Resource Records

To answer all name resolution requests, your DNS server needs to maintain a database in which it maintains the *resource records*, which contain different types of data to find specific information for a domain. Table 9-1 presents some of the most important types of data that can be maintained in that database. Later in this chapter you'll learn how to add these resource records to the DNS database. Listing 9-3 shows an example of the DNS database in which different resource record types are used. You'll find an explanation of the resource record types in Table 9-1.

Listing 9-3. *Contents of the* `example.com` *Zone File*

```
RNA:/ # cat /etc/bind/db.example.com
$TTL 2D
@              IN SOA          SFO.example.com.      root.SFO.example.com. (
                               2006080700            ; serial
                               3H                    ; refresh
                               1H                    ; retry
                               1W                    ; expiry
                               1D )                  ; minimum

example.com.   IN MX           10 mail.example.com.
example.com.   IN NS           lax.example.com.
sfo            IN A            201.100.100.10
lax            IN A            201.100.100.40
web            IN CNAME        sfo.example.com.
```

Table 9-1. *Using the Important Resource Records*

Resource Record	Use
MX	This resource record finds the mail servers for your domain. In the first column, you'll find the name of the domain they are used for, and the fourth column reveals the primary mail server. The number 10 indicates the priority of this mail server. If more than one mail server is present in the domain, the mail server with the lowest priority number is used first. Following the priority number is the DNS name of the mail server.
NS	This resource record provides a list of name servers for this domain. Typically, you must have this resource record for all master and slave name servers of the domain. The first column reveals the name of the domain, and the fourth column provides the name of the server itself. Notice the dot at the end of the server name, which indicates it as an absolute name (a name that refers to the root of the DNS hierarchy directly).
A	The A resource record is used to define the relation between a host name and an IP address. The first column mentions the name of the host as it occurs in this domain, and the fourth column provides the IP address of this host.
CNAME	The CNAME ("common name") resource record is used to define an alias, which is just a nickname that is used for a host. A CNAME should always refer to the real name of the host. Aliases can be useful if one server hosts many DNS names. In that case, use an A resource record for "myserver" and create CNAMEs that refer to the A resource record for all services provided by your server. This way, if you have to change the IP address of your server, you'll change it only once.

Introducing Forward and Reverse DNS

Before I start talking about the actual configuration of DNS, you need to know about reverse DNS. Translating names into IP addresses is one task of the DNS server, and its other task is translating IP addresses to names. This translation from address to name is called *reverse DNS*, and it's necessary if you want to find the real name that is used by a given IP address. This feature is useful if you want names in your log files instead of IP addresses, but, if you want all IP addresses translated to names, you should realize that this comes at a cost in performance. For this reason, many services and commands allow you to specify whether to use reversed name resolution. To make name resolution for your domain possible, you should always configure it when setting up a DNS hierarchy.

To create a reverse DNS structure, you need to configure a zone in the `in-addr.arpa` domain, under which a structure is created that contains the inverse IP addresses for your network. If, for example, you're using the class C network 201.10.19.0/24, you should create a DNS domain with the name `19.10.201.in-addr.arpa`. Within this zone, you next have to create a pointer (PTR) resource record for all of the hosts that you want to include in the DNS hierarchy.

When working with reverse DNS, you should be aware of one important limitation: it doesn't know how to handle non-default subnet masks. In other words, it works only if you have the complete network, and it doesn't work if you've registered a couple of IP addresses only with your IP. If you have only one (or very few) IP addresses out of a complete range, you should ask your IP to set up reverse DNS for you.

Configuring DNS

When setting up DNS, you have to configure a series of configuration files, and in this section you'll learn how these relate to each other. At this point, make sure that the DNS server is installed by using `apt-get install bind9` as root.

/etc/bind/named.conf

The `/etc/bind/named.conf` file is the master configuration file for your DNS server. Listing 9-4 provides an example. The `named.conf` file is a master configuration file that contains all you need to get a working DNS set up. To set up your own additional zones, you have to use the `/etc/bind/named.conf.local` file.

Listing 9-4. *Default* `/etc/bind/named.conf` *File*

```
root@RNA:~# cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local
```

```
include "/etc/bind/named.conf.options";

// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
        type master;
        file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
        type master;
        file "/etc/bind/db.255";
};



include "/etc/bind/named.conf.local";
```

Several other files are called from the main configuration file (`/etc/bind/named.conf`). Before starting to configure your own DNS server, let's look at how these files relate to each other:

- `/etc/bind/named.conf.local`: This file contains the DNS zones that you set up on your server.
- `/etc/bind/named.conf.options`: In this file you'd put generic options that define the working of your DNS server.
- *The db files*: These are database files that store the information for specific zones. Every zone has its own db file, so there should be many of these db files on a completely configured DNS server. For example, the following code lines that come from `/etc/bind/named.conf` refer to the database for the localhost zone:

```
zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};
```

When setting up your own DNS server, it can be quite hard to configure the right files in the right way. So let's do a setup for the `example.com` zone.

■ **Tip** If you want to set up a working DNS environment, you should have your own DNS domain, which you can get from your IP. If you don't have your own DNS domain, you can use the `example.com` domain. This domain is not used for real on the Internet and can therefore be used by anyone who wants to set up a local-only DNS test environment.

---

1. Don't touch the `/etc/bind/named.conf` file. It contains default settings, and you never need to modify it on Ubuntu Server.
2. Open the `/etc/bind/named.conf.local` file with an editor and use the following code:

   ```
   zone "example.com" in {
           allow-transfer { any; };
           file "/etc/bind/db.example.com";
           type master;
   };
   ```

3. In this example configuration, the zone "`example.com`" statement is used as a definition of the zone that you want to use. After the definition of the zone itself and between brackets, specify the options for that zone. In this example, they are as follows:
   - `allow-transfer { any; };`: This option specifies what name servers are allowed to synchronize their databases with the information in this database.
   - `file "/etc/bind/db.example.com";`: This line indicates what file contains the specific configuration for this zone.
   - `type master;`: This option indicates the definition of the master name server for this zone.
4. You've now defined the file in which the DNS server can find the specific configuration for `example.com`. Next, you need to set up reversed name resolution as well. If `example.com` is using the IP network 201.100.100.0, you should open `/etc/bind/named.conf.local` once more and enter the following code:

   ```
   zone "100.100.201.in-addr.arpa" {
           type master;
           file "/etc/bind/db.100.100.201";
   };
   ```

5. Now that you've set up the basic structure for DNS, you need to create the database files in /etc/bind that contain the actual configuration of the DNS zones. I'll explain how to do this and all your available options in the next few sections.

Using named.conf Options

Before you create the database files that you refer to in the named.conf file and its related files, let's have a look at some of the options that you can use in the /etc/bind/named.conf file and its related /etc/bind/named.conf.local and /etc/bind/named.conf.options files.

Ubuntu Server uses the /etc/bind/named.conf.options file to include options in the DNS configuration. This file is included with the line include "/etc/bind/named.conf.options"; in /etc/bind/named.conf. Listing 9-5 shows the file as it is by default.

Listing 9-5. *The* /etc/bind/named.conf.options *File Contains Generic Options for Your* bind *Name Server*

```
root@RNA:~# cat /etc/bind/named.conf.options
options {
        directory "/var/cache/bind";

        // If there is a firewall between you and name servers you want
        // to talk to, you might need to uncomment the query-source
        // directive below.  Previous versions of BIND always asked
        // questions using port 53, but BIND 8.1 and later use an unprivileged
        // port by default.

        // query-source address * port 53;

        // If your ISP provided one or more IP addresses for stable
        // name servers, you probably want to use them as forwarders.
        // Uncomment the following block, and insert the addresses replacing
        // the all-0's placeholder.

        // forwarders {
        //      0.0.0.0;
        // };

        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };

        // By default, name servers should only perform recursive domain
        // lookups for their direct clients.  If recursion is left open
        // to the entire Internet, your name server could be used to
        // perform distributed denial-of-service attacks against other
        // innocent computers.  For more information on DDoS recursion:
        // http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0987
        allow-recursion { localnets; };

        // If you have DNS clients on other subnets outside of your
        // server's "localnets", you can explicitly add their networks
        // without opening up your server to the Internet at large:
        // allow-recursion { localnets; 192.168.0.0/24; };

        // If your name server is only listening on 127.0.0.1, consider:
        // allow-recursion { 127.0.0.1; };
};
```

As you see in the example file in Listing 9-5, you have quite a few options. In the example file, many options are disabled by default ,and others are just not available. Let's have a look at some of the more common options:

- `options { .... };`: Use this statement to indicate the start and the end of the section that contains the options. All generic options need to be in this section, so notice the structure used by this statement. It starts with a bracket, it ends with a bracket, and all specific options are defined between the brackets. When putting this in manually, do not forget the semicolon after the last bracket.
- `directory "/var/cache/bind";`: You can use this parameter to define the location in which all DNS configuration files are stored. If an incomplete file name is used anywhere in one of the DNS configuration files, the DNS name server looks for it in this directory. If, however, an absolute file name (a file with a complete directory reference) is used, it just follows the absolute file name. Also note the semicolon at the end of the line; this is an important syntax feature.
- `notify no;`: This option indicates that slave servers should not be notified of changes, which leaves it completely to the slave server to make sure that it is up to date. If you want an alert to be sent to a slave server when a change occurs, change this setting to `notify yes;`.
- `forwarders;`: By default, if your DNS server gets a request to resolve a name for which it is not responsible, it starts querying a root server of the DNS hierarchy to find the required information. You can change this behavior by using a forwarder, which is another DNS name server that typically has a large cache that it uses to resolve names very quickly. You could, for example, use your IP's DNS name server as a DNS forwarder.

Zone Definition in `/etc/bind/named.conf.local`

Among the most important DNS server options is the definition of zones. As you can see in the example in Listing 9-4, the first zone that is defined is the zone ".". This refers to the root of the DNS domain. The definition is required to hook up your DNS server to the rest of the DNS hierarchy. To do this, the zone definition in `/etc/bind/named.conf` indicates that a list of name servers for the root domain can be found in the `db.root` file. Listing 9-6 is a portion of the contents of that file.

Listing 9-6. *The* `db.root` *File Makes Sure That Your DNS Server Can Contact Other Servers in the DNS Hierarchy*

```
root@RNA:~# cat /etc/bind/db.root

; <<>> DiG 9.2.3 <<>> ns . @a.root-servers.net.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18944
;; flags: qr aa rd; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 13

;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       518400  IN      NS      A.ROOT-SERVERS.NET.
.                       518400  IN      NS      B.ROOT-SERVERS.NET.
...
.                       518400  IN      NS      L.ROOT-SERVERS.NET.
.                       518400  IN      NS      M.ROOT-SERVERS.NET.

;; ADDITIONAL SECTION:
A.ROOT-SERVERS.NET.     3600000 IN      A       198.41.0.4
B.ROOT-SERVERS.NET.     3600000 IN      A       192.228.79.201
...
L.ROOT-SERVERS.NET.     3600000 IN      A       198.32.64.12
M.ROOT-SERVERS.NET.     3600000 IN      A       202.12.27.33

;; Query time: 81 msec
;; SERVER: 198.41.0.4#53(a.root-servers.net.)
;; WHEN: Sun Feb  1 11:27:14 2004
;; MSG SIZE  rcvd: 436
```

The db Files

The zone files of your DNS server are stored in the `/etc/bind` directory, and the name of these files typically starts with "db" (although nothing says that you have to name them this way). The `named.conf` file specifies where to look for these database files. The next part you need to understand is how this file is structured to define your DNS zone. In Listing 9-7 you can see the example that I introduced in Listing 9-3.

Listing 9-7. *Contents of the* `example.com` *Zone File*

```
RNA:/ # cat /etc/bind/db.example.com
$TTL 2D
@              IN SOA          SFO.example.com.      root.SFO.example.com. (
                               2006080700            ; serial
                               3H                    ; refresh
```

```
                                1H                      ; retry
                                1W                      ; expiry
                                1D )                    ; minimum

example.com.    IN MX           10 mail.example.com.
example.com.    IN NS           lax.example.com.
sfo             IN A            201.100.100.10
lax             IN A            201.100.100.40
web             IN CNAME        sfo.example.com.
```

As you can see, the zone file starts with generic settings. First, the `TTL 2D` parameter specifies a validity of two days if your slave server cannot synchronize with the master. Next to be defined are the `SOA` settings for your server, which are the settings for the authoritative name server of this domain. Notice the mail address for the administrator of your DNS server: `root.SFO.example.com`. After that, you see the following synchronization settings:

- `serial`: This number should be changed every time you change the database on the master server. By changing it, a slave server that wants to synchronize with the master server can see that an update has occurred and start the zone transfer. Notice that the serial number typically consists of the current year, current month, and current day, followed by two digits that indicate the event number. For example, after the third change on December 12, 2007, the serial number would be 07121202.
- `refresh`: This indicates the interval used on a slave server between updates from the zone information at the master server.
- `retry`: If the update fails the first time the slave server tries to synchronize, this interval specifies how long it should wait before trying again.
- `expiry`: If a slave server fails to contact the master server for a longer period, this setting indicates how long before the information at the slave server expires. After expiration, the slave server no longer answers DNS queries.
- `minimum`: This is the length of time that a negative response is cached on this server.

Following the generic information, you can see the definition of the resource records. In the previous example, only the four most common resource records are used. (A more complete overview was provided in "Resource Records" earlier in this chapter.)

Configuring Reversed Lookup

Until this moment, we've looked at just normal name resolution in which a name is resolved into an IP address. As I've mentioned, on a DNS server, you need reversed name resolution as well. To configure reversed lookup, you first need to set up the `/etc/bind/named.conf.local` file with the information about the zone you want to configure it for. As discussed earlier, this part of the configuration should look like the following lines:

```
zone "100.100.201.in-addr.arpa" {
        type master;
        file "/etc/bind/db.100.100.201";
};
```

Next, you need to set up the zone file for reverse lookup as well. Listing 9-8 shows a typical reverse lookup file. As instructed in the `named.conf.local` file, this definition comes from the `/etc/bind/db.100.100.201` file.

Listing 9-8. *Example of a Reverse Lookup DNS Zone File*

```
$TTL 2D
@    N SOA   SFO.example.com.  root.SFO.example.com. (
              2006080700      ; serial
              3H              ; refresh
              1H              ; retry
              1W              ; expiry
              1D )            ; minimum

"                       IN NS              lax.example.com.
10                      IN PTR             sfo.example.com.
40                      IN PTR             lax.example.com.
```

You can see that Listing 9-8 uses one resource records type that's specific for a reverse DNS zone: the PTR record. As shown in Table 9-1, this record is used to connect a partial IP address (10 and 40 in Listing 9-8) to a complete DNS name.

Testing Your Name Server

After setting up the DNS name server, it's time to (re)start and test it. First, use the `/etc/init.d/bind9 restart` command (or start it if it wasn't started yet). Next, use `ps aux | grep named` to check whether the named process is really running. Then make sure that your local named process on your server is used for name resolving. Next, use the `ping` command to any host name to check if you can contact a server by its name. If this succeeds, your DNS server is working properly. If it fails, make sure that all your configuration files are set up properly.

If the `ping` command fails, you can use the `host` command for detailed testing of your DNS server. The general syntax of this command is `host computer nameserver`. For example, use `host myhost 193.79.237.39` to query the specific name server 193.79.237.39 about the records it has for host `myhost`. Next, the `host` command reveals the IP address that's related to that host (according to the name server). The opposite is possible as well; for example, the command `host 82.211.81.158` provides the name of the host you've queried. You can use the `host` command without referring to a specific DNS server, in which case the DNS servers as mentioned in `/etc/resolv.conf` are used. Listing 9-9 shows three examples of the `host` command in action.

Listing 9-9. *Using the* `host` *Command to Test a DNS Server*

```
root@RNA:~# host www.ubuntu.com 193.79.237.39
Using domain server:
Name: 193.79.237.39
Address: 193.79.237.39#53
Aliases:
```

```
www.ubuntu.com has address 82.211.81.158
root@RNA:~# host 82.211.81.158
158.81.211.82.in-addr.arpa domain name pointer arctowski.ubuntu.com.
root@RNA:~# host www.ubuntu.com
www.ubuntu.com has address 82.211.81.158
```

Configuring DHCP

Your network probably has a lot of computers that need an IP address and other IP-related information in their configuration. You can, of course, enter all this information by hand on each individual workstation, but it's much easier to automate this process with a dynamic host configuration protocol (DHCP) server. Let's see how to set this up on Ubuntu Server.

Understanding the DHCP Protocol

DHCP is a broadcast-based protocol. A client that's configured to obtain an IP address via DHCP sends a broadcast on startup to try to find one or more DHCP servers in the network. The client uses the DHCPDISCOVER packet to do this. If a DHCP server sees the DHCPDISCOVER packet coming by, it answers with a DHCPOFFER packet, in which it offers an IP address and related information.

If the client receives a DHCPOFFER from more than one DHCP server, it chooses only one. It's very difficult to determine beforehand what IP configuration information the client will work with, which is one of the reasons why you should take care that no more than one DHCP server is available per broadcast domain to offer a configuration to the DHCP clients.

To indicate that the client wants to use the IP address and related information offered by a DHCP server, it returns a DHCPREQUEST, thus asking to work with that information. The DHCP server then indicates that it's okay by returning a DHCPACK (acknowledgment) to the client. From this moment on, the client can use the IP address.

A lease time is associated with each offering from a DHCP server, and this lease time determines how long the client can use an IP address and associated information. Before the lease ends, the client has to send another DHCPREQUEST to renew its lease. In most cases, the server answers such a request by extending the lease period and sending the client a DHCPACK. If it's not possible to extend the lease for some reason, the client receives a DHCPNACK (negative acknowledgment). This indicates that the client cannot continue its use of the IP address and associated information. If this happens, the client has to start the process all over again, beginning with the DHCPDISCOVER packet.

When the client machine is shut down, it informs the server that it no longer needs the IP address by sending a DHCPRELEASE over the network. That IP address then becomes available for use by other clients.

One of the things that you should note in all this is that DHCP is a broadcast-based protocol, which means that if the DHCP server is on a different subnet than the DHCP client, the client cannot reach it directly. If this is the case, a DHCP relay agent is needed that forwards DHCP requests to a DHCP server. You'll learn how to configure all this in the "The DHCP Relay Agent" section.