DNS using BIND 9

COSC301 Laboratory Manual

Today we will learn the basics of setting up and maintaining a Domain Name Server using the popular BIND 9 package. You should work on `server1` unless specified otherwise.

DNS can be fraught with difficulty for the unsuspecting newbie. To help keep this lab manageable, it will be a very basic introduction to setting up a DNS server, for a small LAN for IPv4 and IPv6. You will be given a template to work from.

Because DNS is such a crucial part of a network, and pretty much every service makes use of it, you will use the knowledge gained in this lab throughout the rest of this paper. You will be required to make changes to your DNS server in some of the labs that follow.

> **Important**
>
> Please ensure that you read the entire lab before coming to class, otherwise it's very likely you will not complete it during the lab.
>
> Make sure to install the resolvconf package on server1 using **sudo apt install resolvconf**. This lab should be carried out on server1.

1. Using Dig

dig is the Domain Information Groper, and is the successor to a tool called **nslookup**. There is also a simplified version of **dig** called **host**, but we won't cover that. So without further ado, here are some ways you can use **dig**. You should be able to try all these on your server or client.

You will get more information by not using the `+short` option. The answer will be found in the Answer section. You'll also get some supplementary information, as well as result codes. Using the `+short` option is a good way of getting DNS information in shell scripts.

- What is the IP address of www.isc.org? This is querying for an A record. The domain is that of the Internet Software Consortium, the organisation behind software such as BIND.

```
$ dig isc.org
; <<>> DiG 9.18.1-1ubuntu1.2-Ubuntu <<>> isc.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47497
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 85aee58a08539cf4010000006423915c54ef93c26b80fdbe (good)
;; QUESTION SECTION:
;isc.org.            IN    A

;; ANSWER SECTION:
isc.org.        300    IN    A    149.20.1.66

;; Query time: 192 msec
;; SERVER: 127.0.0.1#53(127.0.0.1) (UDP)
;; WHEN: Wed Mar 29 01:16:12 UTC 2023
```

```
;; MSG SIZE  rcvd: 80
```

- The same query, but this time in short format.

```
$ dig +short isc.org
149.20.1.66   Don't panic if yours is different
```

- Same as above, `-t A` is the default behaviour. `-t A` asks for resource records of type `A`, which maps a hostname to an IPv4 address.

```
$ dig +short -t A isc.org
149.20.1.66
```

- You can use **dig** without specifying a fully qualified hostname. Note that **dig** will not use the default search path (in `/etc/resolv.conf`) by default; we can change this behaviour and use the search path using `+search`. Make sure you have added `search otago.ac.nz` in `/etc/resolv.conf`.

```
$ dig vertex

; <<>> DiG 9.18.1-1ubuntu1.2-Ubuntu <<>> vertex
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 38429           FAILED!
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 84117ec8e94478df0100000064239e7880e36d6bc5bce14e (good)
;; QUESTION SECTION:
;vertex.         IN      A           Because it wasn't searching in our domain.
…
$ dig +search +short vertex
10.81.166.21
```

- What's the (canonical) hostname for 149.20.1.66?

```
$ dig +short -x 149.20.1.66
victor.isc.org.
```

- Note that the above inverse query is equivalent to this:

```
$ dig +short -t PTR 66.1.20.149.in-addr.arpa
victor.isc.org.
```

Exercise

For the following exercise you will use **dig** to explore your local network enviroment a little. Make sure they work as expected.

- Who are the name servers for `otago.ac.nz`?

```
$ dig +short -t NS otago.ac.nz
```

- Which of the nameservers (each domain should have at least two) is the master server? You can find this out by looking at the `SOA` for the domain.

```
$ dig -t SOA otago.ac.nz
```

- You can query a particular name server using the `@` character before the nameserver's DNS name or IP address. Select one of the nameservers you discovered in the previous step.

```
$ dig @nameserver +short www.otago.ac.nz
```

- Who are the mail exchangers for `otago.ac.nz`?

```
$ dig +short -t MX otago.ac.nz
```

> **Note**
>
> Like web servers, which are also high-traffic, email services can be designed in various ways for high availability of for load balancing. So if you get a single result, it may be that there are multiple servers behind one address and a device called a *load balancer* is acting as the entry point to a cluster of email servers.

- While it is possible to use **dig** to perform a *zone transfer* --- to get a list of all the data in a particular zone (domain) --- it is considered rude or hostile.

> **Warning**
>
> **Do not** request this from a machine you do not administer, as it will be considered rude or hostile. You will have a chance to do the following later in the lab to your own network.

```
$ dig @127.0.0.1 -t AXFR domain
```

- Once you have your DNS server up and running, you can try to find out which version of BIND is running on the server. Servers are often configured to give a different result instead, to make it harder for an attacker to know what software version you are using. Don't use `+search`.

> **Note**
>
> You can do this in your virtual machine later.

```
$ dig @localhost -t TXT -c chaos +short version.bind
"9.7.0-P1"
```

The `chaos` "class" (like the "inet" class) refers to a historical networking system called Chaosnet that you don't need to know anything about. Just know that this it was around when the Internet was still referred to as the ARPAnet, and that the only reason we use it today is as a way to determine the version of the software running on a DNS server running BIND.

2. Basic Configuration

Unless otherwise specified, do all your work from this part onwards in your server.

First, you will need to install the following packages: **bind9**, **bind9-host** and **ipv6calc**:

```
# apt-get install bind9{,-host} ipv6calc
…
```

> **Note**
>
> If your shell (bash) sees a pattern like foo{bar,baz,bax}, it will expand to foobar foobaz foobax. Therefore, bind9{,-host} will expand to bind9 bind9-host.

You are about to edit some files which could easily lead to locking yourself out of **sudo**. You should have a root shell available (**sudo -s**) in another virtual console, or set a password for the root user (**sudo passwd root**) temporarily. You can lock the root account again later by using **sudo passwd -l root**.

The file `/etc/hosts` contains static mappings of hostnames to IP addresses. If you're going to set up a DNS server (and you are), then this file should contain a mapping between localhost and 127.0.0.1, and the name of the machine, similar to what is shown below; you probably won't need to modify anything. Note that here, we assume that you called your server `server1`; adjust to suit your environment.

> **Tip**
>
> DNS is case insensitive, and everything is generally input in lower-case.

```
127.0.0.1       localhost
127.0.0.1       server1
192.168.1.1     server1.localdomain
```

Before you set up a DNS server, you should configure the system resolver library with the address of your DNS server(s). You would typically put this into `/etc/resolv.conf` through proper configuration of **resolvconf**; however, this file is prone to being overwritten by tools like DHCP clients, such as **dhclient**, which we are using to configure our outside interface, so let's tell our DHCP client to put something different in there instead. Another alternative would be to disable **dhclient** from writing that file at all.[1]. Edit the file `/etc/dhcp/dhclient.conf`, and add these lines before the `request` stanza. This causes the DHCP *client* on Server1 to ignore some of the settings it was given, and replace (supersede) them with settings we manually specify.

> **Tip**
>
> You wouldn't normally expect to do this on a DNS server, as DNS servers wouldn't normally be on a machine configured by DHCP. But our server is a gateway machine, and is both client and server at the same time.

```
supersede host-name "server1";
supersede domain-name localdomain;
supersede domain-name-servers 127.0.0.1;
supersede domain-search "localdomain";
```

[1] We could, if we really wanted, even have the DHCP client run a script which updates the forwarders we will configure later.

Now disable *systemd-resolved*, which would interfere our changes of `/etc/resolv.conf` below. Note: do the same on Client1.

```
# systemctl disable systemd-resolved
# systemctl stop systemd-resolved
```

Next down/up the interface and check that your `resolv.conf` is as shown below:

```
# ifdown outside
…
# ifup outside
…
$ cat /etc/resolv.conf
nameserver 127.0.0.1
search localdomain
```

Okay, that looks like we would write it by hand.

Now we need to configure Client1 to use the DNS server on Server1. *For Client1*, it will use DHCP to configure its DNS subsystem correctly, but for now you can just manually tell Client1 to use 192.168.1.1 as the nameserver, since we don't have DHCP server set up yet. Remember that you can't modify `/etc/resolv.conf` directly. Instead you should change `/etc/resolvconf/resolv.conf.d/head` as you did before. Make sure `/etc/resolvconf/resolv.conf.d/head` on Client1 has only the following content.

```
search localdomain
nameserver 192.168.1.1
```

Then use the following command to make the change effective.

```
$ sudo resolvconf -u
```

In future, if we use DHCP, we will need to change `/etc/network/interfaces`, where the "static" method should be replaced by "dhcp". No matter which approach we use, the `/etc/resolv.conf` file should end up like something above.

The `search` keyword means that if we were to use an unqualified host-name, such as `server1`, then the local resolver would instead try to find `server1.localdomain` instead (assuming that `server1` was not found in `/etc/hosts`.

So, for example, if you *were* to type **ping server1** (not that you could, currently, as we haven't configured our DNS server yet), then the local resolver would try to find an address for `server1.localdomain`. Note that you can have multiple search domains, by separating them with spaces on the same line. The nameserver keyword specifies a DNS server to ask. If we have more than one nameserver line, then if the first fails, we ask the second, and so on. To specify multiple nameservers, enter each nameserver in its own entry, like below.

```
This is an example only, don't enter it today
search localdomain
nameserver 192.168.1.2
nameserver 192.168.2.4
```

To configure the order in which various name server databases are looked up, you need to configure two files, `/etc/nsswitch.conf` and `/etc/host.conf`, on both the server and the client. `host.conf` is the older version of `nsswitch.conf`, you should check both of these to keep both (very) old and new programs happy.

Generally you don't need to change them when setting up for DNS, but do have a look at them, especially `nsswitch.conf`. The line that says `hosts` is the relevant one for DNS, it specifies in what order to consult the file `/etc/hosts` and the DNS resolver (or server). We shall revisit the `nsswitch.conf` file later in the paper when we look at authentication.

Here is what is currently configured in `nsswitch.conf`:

```
…
hosts:          files dns
…
```

The full details are in nsswitch.conf(5). But let us go briefly over it. We shall assume that we are looking up 'foo.localdomain':

hosts
: This is the database that we wish to consult. This particular database determines mappings between hostnames and IP addresses.

  Other databases are 'passwd', for storing information about user accounts, and others exist too.

files
: Order matters here. *First*, the file `/etc/hosts` will be consulted to see if there is a 'foo.localdomain' can be found in there. If it is found, the lookup process ends and the result is returned.

mdns4_minimal
: Here's where things get a bit different. In environments where there is no DNS server locally available to answer questions about machines in the local network, such as at home or in on an ad-hoc wireless network, then we use something called Multicast DNS, or mDNS for short. Basically, we multicast a DNS query onto the local network, and if any machine sees that someone is asking for their own information, it will respond with the answer.

  Multicast DNS is defined for the domain "local.", which is why you should never call your home or private network domain "local.", as this will cause lookup problems.

  Note that a mDNS lookup is only initiated for requests about domains in "local.", which "foo.localdomain." is not, so no lookup is attempted: *no answer, positive or negative, is generated*.

[NOTFOUND=return]
: In the previous step, we *may* have executed a mDNS lookup. *If we did*, and if we didn't get a result (or in other words, "X.local." didn't exist), then we would return a lookup failure *and not continue*.

  This is why you should not call your home or private domain "local.", because by doing so Ubuntu (and others) will not proceed to lookup (standard unicast) DNS, even if the DNS server has the information.

dns
: If we get here, we do our regular DNS lookup to the nameservers listed in `/etc/resolv.conf`.

The current entry for *hosts* is fine. Keep it unchanged:

```
hosts: files dns
```

In our network, let's assume that Multicast DNS is not desired, because all hosts should be listed in DNS.

Finally, here is `host.conf`. Because this is only consulted by very old programs, you should never need to change it. **ping** is perhaps one program that might still use it, which may explain why it acts a bit different sometimes when diagnosing DNS problems on some systems.

```
order hosts, bind
multi on
```

3. The Master Bind Configuration File

Distributions based on Debian, such as Ubuntu, manage the layout of BIND's configuration files differently from the standard BIND software. This is to allow for greater modularity. The default configuration conforms well to best practices to DNS operation, such as being authoritative for the correct zones, including special cases such as broadcast and private addresses.

Firstly, have a good look at the files under `/etc/bind/`. Ensure you can understand the structure of the `named.conf*` files, starting with `named.conf`.

To aid your understanding, here is a functionally similar `named.conf`, to show you how it works. *It is not the same as what you see in your virtual machines, but we have added annotations to specify which files the various blocks should be added to.*

```
The following lines will go in named.conf.options

acl "clients" {
  192.168.1.0/24;
  fd6b:4104:35ce::/64;
  127.0.0.1;
  ::1;
};

options {
  directory "/var/cache/bind";
  allow-query { "clients"; };
  allow-transfer { 127.0.0.1; ::1; };
  allow-recursion { "clients"; };
  listen-on { any; };
  listen-on-v6 { any; };
  forwarders { 139.80.64.1; 139.80.64.3; };
  auth-nxdomain yes;
};

The following (and others) will already be in named.conf.default-zones
which is already included in named.conf.
There is no need to edit named.conf or named.conf.default-zones

zone "." {
  type hint;
  file "/etc/bind/db.root";
};

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

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

The rest goes into named.conf.local

zone "localdomain" {
  type master;
  notify no;
  file "/etc/bind/db.localdomain";
};

zone "1.168.192.in-addr.arpa" {
  type master;
  notify no;
  file "/etc/bind/db.192.168.1";
};

// convert using "ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::/64"
zone "0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa" {
  type master;
  notify no;
  file "/etc/bind/db.fd6b-4104-35ce-0000--64";
};
```

`acl "clients" { … };`
: This provides a way to reference a set of clients using IP addresses or DNS names. It is the list part of an access control list (ACL). We will use it to control access to various types of requests made to the server.

`options { … };`
: Here we set various global options for the rest of the config file.

  `directory` is where the zone files can be found if not given a fully-specified pathname. Because Debian puts the BIND configuration files and the master zones under `/etc/bind/`, but the `directory` still points to `/var/cache/bind`. This means that for the zones stored in `/etc/bind`, we must specify their location absolutely. Slave zones (which the server would update during runtime) get cached under `/var/cache/bind/`.

  `allow-query` specifies which clients may query the server. This can be an IP address or range, domain, or access control list (in double-quotes), as defined previously. It's important to limit your exposure, since DNS is a very important service, and if compromised, can make further compromise easier. This is why most networks use different DNS servers for the public zones than for the internal zones.

  `allow-transfer` says which clients may perform a zone transfer. A zone transfer is output which tells the receiver everything about a zone. You should only allow slave servers, and possibly localhost to receive a zone transfer. As we shall see later on, a zone-transfer is a great way to debug your zone information.

  Note that for all of these `allow-*` statements, you can specify the keywords `none` and `all`. Check the Bind9 Administrators Handbook for more information. This document is cited at the end of this labsheet.

  `allow-recursion` allows the DNS server to go and get the complete answer to the clients question, rather than just the next step. This makes more work for the server. Generally it's good to allow recursion to your known clients only, certainly never to the public internet. Recursion is fundamental principle in caching name servers.

`forwarders` is a very useful option for most smaller networks. It allows you to specify any upstream DNS servers, such as those of your ISP, which would probably be recursive and have a larger set of cached results, in order to improve performance, and keep unnecessary traffic off the root servers.

`listen-on` and `listen-on-v6` say where the server should accept packets from. We just say `any` here because later on we will further limit who can do what using ACLs.

Finally, `auth-nxdomain` enables the server to give authoritative answers in the case of a lookup failing because it doesn't exist (in DNS terms: no such domain). This allows these negative responses to be cached, which helps to reduce the amount of traffic going to the root nameservers, and reducing latency in the DNS system. It is off by default, but DNS experience has shown the industry that it is worth doing, and is now considered an operational requirement.

`zone "." { … };`

Here we have the entry for the root servers. This is of type `hint`, because we store these addresses in a file. The address list doesn't change very often, and was last updated on the 20th October, 2016. You should make it part of your periodic maintainance to check for updates, although it should be reasonably up-to-date anyway if you have installed Bind from a supported package. The file can be found at InterNIC [ftp://ftp.rs.internic.net/domain/named.root], though for the purposes of this lab, you needn't bother.

The `.` zone is called the root. All fully-qualified hostnames, such as `www.otago.ac.nz` have an implied dot at the end, as in `www.otago.ac.nz.`. Sometimes it is useful to specify the dot, especially when your domain only has one domain component, such as `localdomain.`.

Since we have configured our server to forward queries to other servers, we won't typically be using the root zone, unless our upstream server (which we send our forwarded queries to) fails to respond.

`zone "localhost" { … };`, **`zone "127.in-addr.arpa" { … };`**

Here are two very basic zones, for mapping between localhost and 127.0.0.1 (one for each direction). Not terribly interesting in themselves, they're just like the ones we'll see later. You will see other basic zones as well, in addition to those shown here; do not remove them.

Notice the unusual looking name for the second zone. This is how reverse lookups for IPv4 addresses are performed. The dotted decimal components are reversed[2] (the digits of each component are not, however) and "`.in-addr.arpa`" is appended. The following two commands are identical. You can try these on your Mac host, as you don't have DNS working yet.

```
$ dig -t PTR +short 49.32.80.139.in-addr.arpa
vertex.otago.ac.nz.
$ dig +short -x 139.80.32.49
vertex.otago.ac.nz.
```

And an IPv6 example is given below as well. Mind you that the reverse mapping may not work if the PTR record is not provided by the `www.kame.net` domain. Also note how horrible the reversed nibble is to work with; it is simple (very good), but explosively

[2]Making it into least-significant compoment first, just like a standard DNS name.

verbose, which is annoying and a source of many mistakes. We'll look at that issue later on.

```
$ dig -t AAAA +short www.kame.net
2001:200:0:8002:203:47ff:fea5:3085
$ ipv6calc --in ipv6 --out revnibbles.arpa 2001:200:0:8002:203:47ff:fea5:3085
5.8.0.3.5.a.e.f.f.f.7.4.3.0.2.0.2.0.0.8.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.
$ dig -t PTR +short \
> 5.8.0.3.5.a.e.f.f.f.7.4.3.0.2.0.2.0.0.8.0.0.0.0.0.0.2.0.1.0.0.2.ip6.arpa.
orange.kame.net.
$ dig +short -x 2001:200:0:8002:203:47ff:fea5:3085
orange.kame.net.
```

`zone "localdomain" { … };`
: The forward zone for "localdomain". Server1 has the authoritative data for this domain, so we specify the type as `master`. Since we don't have any slave servers, we don't need to `notify` them of any changes.

`zone "1.168.192.in-addr.arpa" { … };`, **`zone "0.0…d.f.ip6.arpa" { … };`**
: The reverse zones for "localdomain". We have two reverse zones, one for IPv4 and our IPv6 Unique-local addresses respectively (we never add link-local addresses into DNS).

  Just as with the forward zone, Server1 has the authoritative data for this domain, so we specify the type as `master`. Since we don't have any slave servers, we don't need to `notify` them of any changes.

Integrate the configuration above with your current configuration. You should only need to change `named.conf.options` and `named.conf.local`.

When you have entered the information, use the **named-checkconf** command to check the syntax of the file and make sure there are no errors. On a well-formed file, it should not print anything. Note that it will not check for the existence of files specified in the zone configuration statements.

```
$ named-checkconf
If it outputs nothing, it has nothing to complain about.
```

Exercise

As an exercise make sure there is no output from **named-checkconf**, and the contents of the three `named.conf*` files are as expected.

4. Forward Zones

Forward zones are for specifying the mappings from hostnames to IP addresses. we suggest you start with a template (such as the one below) and work from that. Put this into the `/etc/bind/db.localdomain` file. You will need to create it. Comments in zone files start with a `;` and continue to the end of the line.

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601         ; serial
        8H                 ; refresh
        2H                 ; retry
        4W                 ; expire
```

```
        1D                ; minimum
        )

        NS  ns1.localdomain.
        ; MX  0  mailhub1.localdomain.
        ; MX  5  mailhub2.localdomain.
        ; A  192.168.1.3 ; Would allow http://localdomain/

localhost     A        127.0.0.1
              AAAA     ::1

server1       A        192.168.1.1
              AAAA     fd6b:4104:35ce::1
server1.ipv4  A        192.168.1.1
server1.ipv6  AAAA     fd6b:4104:35ce::1
ns1           A        192.168.1.1        An alias
              AAAA     fd6b:4104:35ce::1

client1       A        192.168.1.11
              AAAA     fd6b:4104:35ce::look-this-up
```

Line 1

The TTL statement *must* be the first non-comment line in the file. This specifies the default time to live (3 days in this case) until cached entries should be invalidated. I strongly recommend that when you put your first Internet-serving DNS server on the internet, you use a short TTL until you are happy that all the data is correct, otherwise other servers will cache the data and won't see any changes until the TTL expires.

If you're anticipating that data might change soon (perhaps you know that next week you will be moving to a different IP address), decrease the TTL to a shorter and shorter value, so that on the day of the move no server will be caching an answer for longer than a few minutes.

Lines 3 to 9

The contents of line 3 must not be broken onto multiple lines, else things will break. `@` is shorthand for the origin (`.localdomain.` in this case). It's already declared in `named.conf.local`, so we needn't write it again[3]. `IN` says that this zone is an Internet addressing system. `SOA` (Start Of Authority) describes this zone: the server which is authoritative for it, who is responsible for it, and various timeouts.

`ns1.localdomain.` is the nameserver which is authoritative for this domain. Note: You need to include the dot at the end, otherwise, you'll end up with `ns1.localdomain.localdomain`. This is why it can be useful it allow zone transfers to localhost, as it's easiest to spot these sorts of errors by looking at the output of a zone transfer.

`hostmaster.localdomain` is actually the email address hostmaster@localdomain of the person responsible for the zone. It's customary to have an email address `hostmaster` for this purpose, which is an alias to a real person.[4]

The SOA resource record (RR) contains various timeouts, they are as follows:

Serial Number

Slave servers use this as a version number to find out if they need to update. You should update it whenever you make a change to the zone file. Generally you use a form `yyyymmdd` + today's version.

[3] Though you can by using the `$ORIGIN` directive.
[4] If the email user has a dot in it, you must escape the dot to make it `\.`.

Refresh
How often slave servers should check for an updated version of the zone file. The H suffix means hours, and you can also specify days (D), minutes (M) or seconds (no suffix) etc. The suffix is case-insensitive.

Retry
If a refresh failed, the retry interval says how long to wait before trying again.

Expire
After this interval, if the data hasn't been updated, remove it.

Negative or Minimum
Specifies how long a negative result should be cached for. A negative result is, for example, when the record being looked up does not exist. These values do just fine for normal environments. The maximum allowable value is 3 hours (3h).

Lines 11 to 14
Networks, as well as hosts, have resource records attached to them. We've already seen the SOA RR, here are a few more.

Important

Make certain to include some amount of white space at the start of these lines, otherwise **named** will think you're defining hosts. You'll end up trying (and failing) to define a host called NS with a RR called `ns1.localdomain.`, which is not a supported RR (well, of course not!).

NS
Gives a nameserver for this domain. There must be at least two NS records for each public domain. They could be master or slaves, the master is listed in the SOA record for the zone.

MX
Gives the various mail exchangers that accept mail for this domain. Each MX entry has a priority. Mail is send to the one with the highest priority (which, ironically, is the one with the lowest value), and if it fails, tries the others, in order of priority.

Since we haven't covered email servers at this time, these are just for example and are commented out.

Lines 16 to 27
Now we start to declare the hosts that are in our network. The form is: hostname, class, RR type, then RR value(s). The class is generally `IN` and can be omitted, as it is inherited from the SOA record. Note that the first two lines makes the hostname localhost.localdomain point to 127.0.0.1 and ::1, which isn't important for users, but can be of assistance when clients don't resolve localhost to themselves first (through `/etc/hosts` or similar).

Line 19 to 22
Here we've specified that server1 has an A record as well as an AAAA record, so if request for any records attached to server1, it will likely give you results for both IPv4 and IPv6.

In case you want to prefer IPv4 *or* IPv6 (for example, when rolling out IPv6 support for your own clients in your own domain), then we've also added records that return just

those results. This should only really be useful for troubleshooting, most of the time you should be using `server1.localdomain` instead of `server.ipv4.localdomain`.

You might wonder why we chose `foo`.ipv6 instead of ipv6.`foo`, or `foo_ipv6` or similar: one (minor) reason is that you can adjust who you roll-out IPv6 to by adjusting your clients DNS search path (perhaps via DHCP), but this is only a very minor difference really. RFC 4472 contains some guidance here. To make the difference clear, imagine you have `ipv6.localdomain` and `localdomain` in the search path of machines you wish to roll-out IPv6 to. Then, if a client is configured to access the unqualified hostname `foo`, it will first try `foo.ipv6.localdomain`, then `foo.localdomain`. In the same way, you could have other clients prefer IPv4. But this is only effective for clients in your own domain which are looking up domains that you control.

Line 23 to 24

DNS servers (nameservers) generally work in pairs. A public domain on the internet needs to have at least two nameservers, and are often refered to as ns1.`domain` and ns2.`domain` etc. In our case however, we only have one nameserver.

Here we are creating an alias called ns1.`domain`. There are two common ways of creating an alias; either using duplical address records or by using the CNAME RR, but CNAMEs are becoming less common, and it is generally preferred to use a duplicate address records. It also gives us better control over what IPv4 and IPv6 records get returned.

For example, `server1` currently has an A record for a particular address. A *duplicate address record* could be made for `www` that is also an A record *to the same address* (ie. it is the address that is duplicated, not the name).

Note

Another way (less preferred but still very common) of creating aliases is with the CNAME record.

If we were to see a line `ns1 CNAME server1` in the “localdomain” zone, that means that we are creating a label (record) called “ns1.localdomain”, which is an alias. The canonical (real) name of the machine the alias points to is “server1.localdomain”.

In this case we are using duplical address records: the only thing that makes it an alias (rather than a canonical name) is that in the reverse zone, 192.168.1.1 should only return a PTR record for `server1.localdomain.` (it’s *canonical* hostname).

It is common practice to call servers one name (its canonical name), then give them aliases that refer to their services, such as www or ftp. This makes it easier to reassign a particular service to another machine, because all the clients don’t need to be re-configured: a single point of change.

Once done, you can use the **named-checkzone** on the zone file to check for errors.

```
$ named-checkzone localdomain /etc/bind/db.localdomain
zone localdomain/IN: loaded serial serial-number
OK
```

Exercise

As an exercise make sure the above **named-checkzone** reports no error.

5. Reverse Zones

Reverse zones are for specifying the mappings from IP addresses to DNS names. It's often used on servers for logging purposes, and access control. If you notice that it takes about 6 seconds to use a service, it may be that your reverse zone is not well configured, and the lookup timed out. As with the forward zone, you should use a template, such as the following.

Put this into the `/etc/bind/db.192.168.1` file. You will need to create it. There's nothing terribly new at the start of the file, what's different is when it comes to the PTR (pointer) records.

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601 8H 2H 4W 1D)

       NS        ns1.localdomain.

The current $ORIGIN is 1.168.192.in-addr.arpa.
so this unqualifies entries like 192.168.2.1
as the first three bytes of the address have to be 192.168.1.
1      PTR       server1.localdomain.
11     PTR       client1.localdomain.
```

Exercise

As an exercise check the well-formedness of the file using **named-checkzone** as below.

```
$ named-checkzone 1.168.192.in-addr.arpa. /etc/bind/db.192.168.1
zone 1.168.192.in-addr.arpa/IN: loaded serial serial-number
OK
```

But wait, that's only the IPv4 reverse zone, we still have to create the IPv6 version. Create a new file `/etc/bind/db.fd6b-4104-35ce-0000--64` (there's nothing special about the name, its just a name mangling scheme we came up with to describe the zone).

```
$TTL 3D
@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2020030601 8H 2H 4W 1D)

     NS ns1.localdomain.

This $ORIGIN is as the default anyway, just to remind ourselves…
Remember to include the trailing .
We used the following command to get the right format
ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::/64
$ORIGIN 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa.

Here's another example of ipv6calc to the rescue:
ipv6calc --in ipv6 --out revnibbles.arpa fd6b:4104:35ce::1 | cut -d. -f-16
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0    PTR server1.localdomain.
Your Client1 will have a different address!
e.0.7.3.8.2.e.f.f.f.7.2.0.0.a.0    PTR client1.localdomain.
```

Exercise

Well, that was painful, and it's not immediately obvious what we have specified, which makes it harder to recognise and diagnose errors. We'll have a look at how we can

make this much easier a little later; for now, as an exercise let's get onto testing using **named-checkzone**, showing your progress.

```
$ named-checkzone 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa. \
> /etc/bind/db.fd6b-4104-35ce-0000--64
zone 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa/IN: loaded serial serial-number
OK
```

5.1. Self-assessment

1. Make sure you have done the exercises for: `named.conf*`, **named-checkconf**, `db.localdomain`, **named-checkzone** for that zone, `db.192.168.1` and its **named-checkzone**, and similar with the IPv6 reverse zone.

   If submitting remotely, also include the zone files themselves (remember, you can copy them in/out using the VirtualBox shared folder you set up).

6. Affecting the Changes

Restart Bind to affect the changes and load the new zones:

```
# /etc/init.d/named restart
```

Check the logs to familiarise yourself with what it says when it starts.

Ensure that the **named** process is running, and listening on UDP (and TCP) port 53 (the domain port), plus a few others. You should notice that there are entries for each interface.

```
# lsof -Pni
COMMAND … TYPE … NODE NAME
…
named   … IPv6 … TCP  *:53 (LISTEN)
named   … IPv4 … TCP  127.0.0.1:53 (LISTEN)
named   … IPv4 … TCP  192.168.1.1:53 (LISTEN)
named   … IPv4 … TCP  10.0.2.15:53 (LISTEN)
named   … IPv4 … TCP  127.0.0.1:953 (LISTEN)
named   … IPv6 … TCP  [::1]:953 (LISTEN)
named   … IPv6 … UDP  *:53
named   … IPv4 … UDP  127.0.0.1:53
named   … IPv4 … UDP  192.168.1.1:53
named   … IPv4 … UDP  10.0.2.15:53
```

Important

If a server is failing to start, check the logs!

You may be wondering what all these entries are for, and if so, then you're in the right mindset for an administrator. The TCP entries relating to port 53 is because if DNS requests don't fit into a UDP response (typically this is 512 bytes for UDP DNS, but can be increased) then it will try again using TCP; this can be a performance hit. Queries such as zone transfers would commonly need to use TCP. The entries relating to port 953 are related to the **rndc** tool, which allows control of the server during runtime (for example, to tell it to reload some zones), or to write out some statistics for performance analysis. The entries relating to UDP port 53 are the standard DNS traffic. Notice that we support both IPv4 and IPv6 traffic, and that both IPv4 and IPv6 queries can be made over each (indeed, currently, the majority of IPv6 queries go over IPv4).

6.1. Controlling the Server During Runtime

rndc is a tool that controls **named** during runtime. This means that when we make a configuration change, we can use **rndc** to tell **named** to reload the configuration. This is good, because when **named** is *restarted*, all the cached answers are forgotten.

When you installed the `bind9` package, a key was created for you in `/etc/rndc.key`. Both **named** and **rndc** read this file to retrieve the key they need to use in order to authenticate, so by default, there is no configuration needed to control **named** on the local machine[5].

If you change the contents of any of the zone files or `named.conf`, you should run the command below to tell the server to reload its configuration.

```
# rndc reload
```

It pays to check the logs just to make sure that it reloaded successfully, and that reloading didn't cause it to crash or to fail to load a particular zone.

7. Testing

The devil is in the details of most of the core things a Network Administrator does, so testing is important. It's all too easy to leave out a crucial dot or semi-colon, and have things either not start, or worse, start but not work correctly.

The easiest way to test what the server has read in from your zone files is to do a zone transfer on that zone. Remember, we have three zones to check, the forward zone and the two reverse zones for IPv4 and IPv6 respectively.

Exercise

As an exercise retrieve a zone transfer for each zone, and check it looks correct. Resolve (in both directions) names and numbers to make sure that resolution is working correctly.

```
$ dig @localhost -t AXFR localdomain
…
$ dig @localhost -t AXFR 1.168.192.in-addr.arpa
…
$ dig @localhost -t AXFR \
> 0.0.0.0.e.c.5.3.4.0.1.4.b.6.d.f.ip6.arpa.
…
```

Let's test some basic forward and reverse entries:

```
$ dig +short server1.localdomain
… should return an A record
$ dig +short -t ANY server1.localdomain
… should return an A and AAAA record
$ dig +short -t AAAA server1.localdomain
… should return a AAAA record
$ dig +short -t AAAA server1.ipv6.localdomain
… should return a AAAA record
$ dig +short server1.ipv4.localdomain
… should return an A record
```

[5] **rndc** can also be used to control other machines, but we won't go into that.

```
$ dig +short ns1.localdomain
… should return an A record (-t argument defaults to A)
$ dig +short -x 192.168.1.1
… should return server's canonical name
$ dig +short -x fd6b:4104:35ce::1
… should return server's canonical name
$ dig +short -x client1's fd6b::… address
…
```

It can be useful just to do a quick ping to check if you've mistyped an address (note though that this doesn't guarantee you've put in the correct address, just one that exists):

```
$ ping6 -c1 client1.localdomain
…
```

Now let's check that our forwarding is working by looking something that would not be satified either by our local zones or in our DNS server's cache:

```
$ host www.google.com
…
```

8. Self-assessment

Exercise

Make the following additions. For each of your changes, make sure they are correct by testing. This exercise will enhance your understanding of the DNS configuration.

1. Add in mappings for a (non-existent) host named goliah, which has an address 192.168.1.5 as well as fd6b:4104:35ce::dead:beef.

2. Create an alias (as duplicate A and AAAA records) for goliah called `www`. To test, use the following:

```
$ dig +short -t ANY www.localdomain
192.168.1.5
fd6b:4104:35ce::dead:beef
$ dig +short -x 192.168.1.5
goliah.localdomain.
$ dig +short -x fd6b:4104:35ce::dead:beef
goliah.localdomain.
```

3. Make it so if a user were to type http://localdomain/ into a web browser, it would end up going to goliah (who we are assuming is a web-server). You do not need to have a webserver installed or client installed. To test, use the following.

```
$ dig +short -t ANY localdomain
… You will get SOA and NS records as well
192.168.1.5
fd6b:4104:35ce::dead:beef
```

4. Assume you are providing a file mirroring service for many clients (eg. a public Ubuntu mirror). Assume further that you are providing mirrors for other distributions. What is the benefit of using the hostname `http://ubuntu.example.com/` instead of `http://www.example.com/ubuntu/`? Is there a difference? Why might one be more preferable above the other. Tip: consider what would need to happen if we wanted to change which server offered the Ubuntu repository.

9. [Optional] Fixing that Mess with IPv6

Running short on time?

The remainder of this lab is optional. You are not required to complete this for any assessment or assignment, but you will find that doing so can make later alterations a bit more pleasant.

We saw how horrible the IPv6 addresses were to deal with in the reverse zones. A tool such as **ipv6calc** helps a lot in some respects, but only when adding the new records; it's still quite difficult to see everything that is currently there. We can do better by generating the correct (verbose) format using a program with input from a zone file that goes through some filter to do the conversions. We have written just a program for you: **ipv6-dns-revnibbles**. This program works somewhat like the venerable **m4** macro processor, but is highly specialised (and therefore rather useless for other tasks). With this tool, your input starts looking like this:

```
$TTL 3D

@ IN SOA ns1.localdomain. hostmaster.localdomain. (
        2010042801 8H 2H 4W 1D)

       NS        ns1.localdomain.

%RN-PREFIX(fd6b:4104:35ce::/64)

%RN(::1)                  PTR   server1.localdomain.
%RN(::a00:27ff:fe28:370e) PTR   client1.localdomain.
```

You would store in a file such as `db.foo.rn` and then, using a simple Makefile, create `db.foo` from that.

If you haven't already, rename the reverse zone file you want to manage so it has a `.rn` extension.

```
# mv /etc/bind/db.fd6b-4104-35ce-0000--64{,.rn}
```

You'll need to build the software, which uses the Flex tool and the C compiler; you should already have the C compiler installed, but you will need to install the **flex** package:

```
# apt-get install flex
…
```

Now, to build the software. First copy the `ipv6-dns-revnibbles.tgz` file from the `Lab Resources/DNS` folder into your virtual machine's shared folder (you set this up during the System Installation lab, remember?). Now from inside your server, unpack it and build it:

```
$ mkdir -p ~/src/ipv6-dns-revnibbles
$ cd ~/src/ipv6-dns-revnibbles
$ tar -zxf /media/host/ipv6-dns-revnibbles.tgz
$ less db.foo.rn
$ make
# install --owner root --group root --mode 0755 \
> ipv6-dns-revnibbles /usr/local/bin/
$ make -f Makefile.etc-bind
Updating db.foo from db.foo.rn
…
# install --owner root --group root --mode 0755 \
```

```
> Makefile.etc-bind /etc/bind/Makefile
```

Now, using `db.foo.rn` as a guide, update your own IPv6 reverse zone and run **make** inside the `/etc/bind/` directory.

10. [Optional] Fun with Hexadecimal

All this verbosity of IPv6 addresses has had some people trying to spell words in hexadecimal, much as you do with personalised number plates. Here is a completely optional, entirely unassessed, although still rather fun activity of generating easy to remember address components using a dictionary and some scripting.

```
iconv -f utf8 \                         Dictionary is in UTF-8
      -t us-ascii//TRANSLIT \           We want ASCII, removing diacriticals
      /usr/share/dict/words \           This is the default dictionary
  | grep -i '^[a-flosg]\+$' \           We want hex letters, plus some others…
  | tr '[[:upper:]]' '[[:lower:]]' \    Make them all lowercase
  | tr 'losg' '1059' \                  ‘l’ can be replaced by ‘1’, ‘o’ with ‘0’…
  | egrep -v '([0-9].*){3,}' \          Avoid results with more than three digits
  | egrep -v '^[0-9]' \                 Avoid results that start with a digit…
  | egrep -v '[0-9]$' \                 …or end with a digit
  | egrep '^.{4,8}'                     Want results that are 4 to 8 characters.
```

Playing around with the generated works, you could come up with often humerous yet memorable names. For example, you might take `a1fa1fa` and `debac1e` and come up with an address such as `fd6b:4104:35ce::a1fa:1fa:deba:c1e` “Alfalfa Debacle”. Hmm, though perhaps it would be better to stick with blocks of 4 or 8 characters results, otherwise it gets potentially confusing where to put the `:`s, which are significant.

11. Last Words

Common DNS Operational and Configuration Errors
RFC1912 [ftp://ftp.rfc-editor.org/in-notes/rfc1912.txt]. Recommended reading, though quite old.

DNS-HOWTO
en.tldp.org [http://en.tldp.org/HOWTO/DNS-HOWTO.html]

BIND 9 Administrators Handbook
You’ll find an HTML version in the bind-doc package. After installation, you will find the documents in `/usr/share/doc/bind/html/`. A PDF version is also available from the Vendor's website.

Securing an Internet Name Server
PDF [http://www.cert.org/archive/pdf/dns.pdf] from the CERT Coordination Centre

RFC 4472 - Operational Considerations and Issues with IPv6 DNS
A useful introduction to the real-world deployment issues, observed behaviour and problems in the world of DNS.