DNS and DHCP

Stein Family Home Network 2017 Edition

The main goals for DNS and DHCP are:

- Allow configuration of dynamic and static IP assignments
- Work across multiple VLANs and subnets
- Handle IPv6 lookups and assignments
- Provide internal subdomains hosted locally and not exposed to the Internet
- Update internal forward and reverse DNS with dynamically-assigned DHCP client information
- Outsource top-level DNS to a third party with dynamic DNS support
- Have the ability do all this with redundant servers and automatic failover
- Simplicity and cost

The technologies used to provide this solution are:

- Two Ubuntu server instances running in virtual machines on separate physical servers
- dnsmasq for DNS and DHCP services
- drdb for replicated storage
- heartbeat for managing services, virtual IPs and DRBD
- ddclient for updating dynamic DNS

Design Discussion

A major concern facing this design is that DNS and DHCP services are very important and need to be highly available. Secondary factors are low cost and simplicity. Both the ISC tools and Pacemaker were considered, but it was decided that they increased complexity without providing enough of a gain in functionality.

On a high level the design consists of two Linux virtual machines hosted on two different host servers. They both run dnsmasq, but with only one server being active at a time. DNS lookup services are directed to a virtual IP, (IPv4 and ipV6), that moves between the two servers. While it is possible to store DHCP leases in a database, a mirrored disk is used so that configuration files can also be shared. The heartbeat software manages which server holds the IP, is running the dnsmasq service and has control of the disk.

Example Implementation

The rest of this document details how this solution was implemented on the Stein Family Home Network. The instructions in this document are specific to this network and may not match yours. Therefore they are intended to just be an example of a configuration option and not as a tutorial. That said, file names are highlighted in **bold** and options that may need to be changed are in *italics*. When commands need to be run they will start with a $ character to denote a regular user. If commands are to be executed on a specific server the server name will appear before the $ character e.g. lsvm1$.

Environment

DHCP and DNS services are desired for two VLANs: one being an internal network (VLAN1 using the 192.168.1.0/24 subnet and the internal.thesteins.org subdomain) and the other being a guest network (VLAN3 using the 192.168.3.0/24 subnet and the guest.thesteins.org subdomain). The guest network is only allowed Internet access through the main edge router, but uses the Linux servers for DHCP assignments and DNS service. This is to allow a single place to see what has connected.

The two servers must be configured with static IPs in order for dnsmasq to work, so are given *192.168.1.6 / 192.168.3.6* (hostname *lsvm1*) and *192.168.1.7 / 192.168.3.7* (hostname *lsvm2*). The virtual IPs that clients will use are *192.168.1.4* and *192.168.3.4*. The floating IPv6 addresses use the IPv4 mapped format and are currently *2601:40f:4100:13c2:0:ffff:c0a8:0104* and *2601:40f:4100:13c2:0:ffff:c0a8:0304*. (Note that *2601:40f:4100:13c2* is assigned by my Internet provider).

For various reasons in this deployment we will use the *lsvm2* VM as the primary provider of these services with the *lsvm1* VM as the backup. The host servers both run unRAID as the operating system.

Build

Two virtual machines were created and loaded with a base install of Ubuntu 16.04.2 LTS as described in the document Ubuntu Base Install. (These have now been upgraded to 18.04.1 LTS.) On both instances a 250M virtual disk was attached. This will hold configuration information and will be replicated using DRBD. (Note that this is overkill for DNS and DHCP data, but this allows other items to be stored on it.)

Configure Ethernet Interfaces

The interfaces show up as *enp2s1* and *enp2s1.3*. The file **/etc/network/interfaces** is edited as root to look like the following on *lsvm1*:

```
source /etc/network/interfaces.d/*

auto lo
iface lo inet loopback

auto enp2s1
iface enp2s1 inet static
        address 192.168.1.6
        netmask 255.255.255.0
        gateway 192.168.1.1
        dns-nameservers 192.168.1.4
iface enp2s1 inet6 auto

auto enp2s1.3
iface enp2s1.3 inet static
        address 192.168.3.6
        netmask 255.255.255.0
        dns-nameservers 192.168.1.4
        vlan-raw-device enp2s1
```

```
iface enp2s1.3 inet6 auto
```

The file looks the same on *lsvm2* except the IPv4 address ends in "*.7*" instead of "*.6*" (two lines affected).

Partition Disks

The 250M virtual disk image should be accessible by the system but needs to have a partition created on it. In our case the disk device is ***/dev/vdb***. To do so enter the following making sure to use the correct disk device:

```
$ sudo fdisk /dev/vdb

        n (create new partition)
        p (primary)
        <ENTER> (partition number 1)
        <ENTER> (first sector)
        <ENTER> (last sector)
        t (change partition type)
        8e (Linux LVM)
        w (write and exit)
```

You should then have the device ***/dev/vdb1***.

Create Backing Block Device

Before we install DRBD we must determine if we want the ability to grow the size of the replicated disk space. If this flexibility is desired with minimal or no downtime then LVM or some other filesystem that allows for dynamic resizing should be used as the backing block device for DRBD. While it is unlikely that DNS and DHCP data will consume more than a few kilobytes, LVM was used as the backing store in case future replication needs arose.

Create a new physical volume, a new volume group and a new logical volume be entering the following, (note the slightly different arguments for *lsvm1* and *lsvm2*):

```
$ sudo pvcreate /dev/vdb1
lsvm1$ sudo vgcreate lsvm1-vg2 /dev/vdb1
lsvm1$ sudo lvcreate -l 100%FREE -n drbd0 lsvm1-vg2
lsvm2$ sudo vgcreate lsvm2-vg2 /dev/vdb1
lsvm2$ sudo lvcreate -l 100%FREE -n drbd0 lsvm2-vg2
```

Instructions for increasing the LVM space available on the logical volume and having DRBD make use of the new space may be find on the following web sites:
https://www.rootusers.com/how-to-increase-the-size-of-a-linux-lvm-by-expanding-the-virtual-machine-disk
https://docs.linbit.com/doc/users-guide-83/s-resizing/

For some reason during testing Ubuntu did not seem to recognize the new logical volumes after a reboot and show it as not being attached. This would prevent a machine from taking over services. A proper solution is still being investigated, but for now edit **/etc/lvm/lvm.conf** and change the line “use_lvmetad = 1” to “use_lvmetad = 0”. This was still the case after the (in-place) upgrade to 18.04.1 LTS.

Install DRBD

To install DRBD the following should be entered on both hosts:

```
$ sudo apt install drbd8-utils
```

Next, create the file ***/etc/drbd.d/drbd0.res*** on both hosts, (the name does not matter, but should end in .res), and populate with the following making sure that you use the correct values for the disk:

```
resource drbd0 {
	protocol C;

	startup {
		wfc-timeout 10;
		degr-wfc-timeout 10;
		outdated-wfc-timeout 10;
	}

	disk {
		on-io-error detach;
	}

handlers {
		local-io-error "/usr/lib/drbd/notify-io-error.sh root";
		out-of-sync "/usr/lib/drbd/notify-out-of-sync.sh root";
		pri-lost-after-sb " /usr/lib/drbd/notify-pri-lost-after-sb.sh root; /usr/lib/drbd/notify-
emergency-reboot.sh root; reboot -f";
		pri-on-incon-degr "/usr/lib/drbd/notify-pri-on-incon-degr.sh root; /usr/lib/drbd/notify-
emergency-reboot.sh root; reboot -f";
		split-brain "/usr/lib/drbd/notify-split-brain.sh root";
	}

	net {
		cram-hmac-alg sha256;
		shared-secret "SOME_PASSWORD_STRING";
		after-sb-0pri discard-least-changes;
		after-sb-1pri discard-secondary;
		after-sb-2pri call-pri-lost-after-sb;
	}

	on lsvm1 {
		device /dev/drbd0;
		disk /dev/mapper/lsvm1--vg2-drbd0;
```

```
                address 192.168.1.6:7788;
                meta-disk internal;
        }

        on lsvm2 {
                device /dev/drbd0;
                disk /dev/mapper/lsvm2--vg2-drbd0;
                address 192.168.1.7:7788;
                meta-disk internal;
        }
}
```

Since this file contains a password we will also set it to readable only by root:

```
$ sudo chmod 640 /etc/drbd.d/drbd0.res
```

This configuration is rather heavy-handed with short timeouts and aggressive handling of a split-brain scenario. We can do this because even if we lose some lease information it will not be a major issue. Given the rather static nature of the home network it is unlikely that duplicate IPs will be assigned. While it is possible to write the leases to a database, it was determined on this network that the cost of creating and maintaining a highly available database was not worth the benefits provided.

You may now initialize and start the service by entering the following on both hosts first making sure that the firewall is set to allow the traffic:

```
lsvm1$ sudo ufw allow from 192.168.1.7 to any port 7788 proto tcp
lsvm1$ sudo ufw allow from 192.168.1.7 to any port 7789 proto tcp
lsvm2$ sudo ufw allow from 192.168.1.6 to any port 7788 proto tcp
lsvm2$ sudo ufw allow from 192.168.1.6 to any port 7789 proto tcp
$ sudo drbdadm create-md drbd0
$ sudo systemctl start drbd.service
```

On the primary system enter the following:

```
lsvm2$ sudo drbdadm -- --overwrite-data-of-peer primary all
```

On the secondary system you may enter the following to monitor the synchronization process:

```
lsvm1$ watch -n1 cat /proc/drbd
```

Once the synchronization is complete you may create a file system by entering the below on your primary system.

```
lsvm2$ sudo mkfs.ext4 /dev/drbd0
```

We can now create a mount point on each host. This may be wherever you want. I like to use **/srv** as a mount point.

```
$ sudo mkdir /srv/drbd0
```

On the primary server we can now mount the new file system and create a location for the dnsmasq data files. In the future this mounting will be handled by heartbeat.

```
lsvm2$ sudo mount /dev/drbd0 /srv/drbd0
lsvm2$ sudo mkdir /srv/drbd0/dnsmasq
```

Install dnsmasq

Enter the following on each host to install the dnsmasq package and promptly stop it:

```
$ sudo apt install dnsmasq
$ sudo systemctl stop dnsmasq.service
```

Next, edit **/etc/default/dnsmasq** on both systems to set the configuration file to ***/srv/drbd0/dnsmasq/dnsmasq.conf*** and make sure dnsmasq will start when run by the init scripts by uncommenting / changing the following lines as necessary:

```
DNSMASQ_OPTS="--conf-file=/srv/drbd0/dnsmasq/dnsmasq.conf"
ENABLED=1
```

On the primary server copy **/etc/dnsmasq.conf** to ***/srv/drbd0/dnsmasq/dnsmasq.conf*** as root and edit. How you edit this file is highly dependent on your network. The following are examples of the non-default options used on my network:

```
domain-needed
bogus-priv
no-resolv

server=fe80::6eb0:ceff:fee1:7018
server=192.168.1.1

local=/internal.thesteins.org/
local=/guest.thesteins.org/

no-hosts
addn-hosts=/media/drbd0/dnsmasq/banner_add_hosts
expand-hosts

domain=internal.thesteins.org
domain=guest.thesteins.org,192.168.3.0/24

dhcp-range=set:enp2s1,192.168.1.21,192.168.1.234,12h
dhcp-range=set:enp2s1.3,192.168.3.21,192.168.3.234,12h

dhcp-range=set:enp2s1v6,::,constructor:enp2s1,ra-stateless,ra-names

# Static IPv4 IPs based on MAC address.
# Add as many or as few as desired. This is just an example.
dhcp-host=53:54:01:11:43:e7,192.168.1.235
```

```
# Static names based on MAC address.
# Add as many or as few as desired. This is just an example.
dhcp-host=06:34:24:47:9a:3d,AD1G2

# IPv4 DHCP options
dhcp-option=tag: enp2s1,option:router,192.168.1.1
dhcp-option=tag: enp2s1,option:dns-server,192.168.1.4
# Just send VLAN3 to the router for DHCP and DNS—guest access
dhcp-option=tag: enp2s1.3,option:router,192.168.3.1
dhcp-option=tag: enp2s1.3,option:dns-server,192.168.3.1

# IPv6 DHCP options
dhcp-option=option6:dns-server,[2601:40f:4100:13c2:0:ffff:c0a8:104]

dhcp-leasefile=/srv/drbd0/dnsmasq/dnsmasq.leases
dhcp-authoritative
no-negcache

# CNAMEs
# Add as many or as few as desired. This is just an example.
cname=smtp,lsvm4.internal.thesteins.org
cname=smtp.internal.thesteins.org,lsvm4.internal.thesteins.org
```

On the primary server you may create the file ***/srv/drbd0/dnsmasq/banner_add_hosts*** and populate with any devices using a static IP and not retrieving one from DHCP. For example:

```
192.168.1.1     rtbr1
2601:40f:4100:13c2:0:ffff:c0a8:104  rtbr1
192.168.1.4     linuxserver
192.168.1.6     lsvm1
2601:40f:4100:234c:215:5dff:fe01:207    lsvm1
192.168.1.7     lsvm2
2601:40f:4100:234c:5054:ff:fe63:1e59    lsvm2
192.168.3.1     rtbr1.guest.thesteins.org
192.168.3.4     linuxserver.guest.thesteins.org
192.168.3.6     lsvm1.guest.thesteins.org
192.168.3.7     lsvm2.guest.thesteins.org
```

The firewall should then be configured to allow DNS and DHCP traffic by entering the following on both hosts:

```
$ sudo ufw allow from any to any port 53 proto tcp
$ sudo ufw allow from any to any port 53 proto udp
$ sudo ufw allow from any to any port 67 proto udp
```

We will let heartbeat manage starting and stopping dnsmasq, so disable it loading on boot:

```
$ sudo systemctl disable dnsmasq
```

Finally on the primary system we will unmount the file system in preparation of heartbeat performing the mount:

```
lsvm2$ sudo umount /srv/drbd0
```

Install ddclient

My registrar is Namecheap and it supports dynamic DNS updates at least for IPv4. The ddclient software works with Namecheap to update the IP address as needed. On both hosts enter the following:

```
$ sudo apt install ddclient
```

The install process will prompt you for various bits of information, but does not contain the necessary options for Namecheap. Therefore you may enter what you want as we will be editing the configuration file directly after.

Once the install is finished, edit **/etc/ddclient.conf** as root. Both servers will contain the same data which should look like the following:

```
protocol=namecheap
use=web, web=dynamicdns.park-your-domain.com/getip
server=dynamicdns.park-your-domain.com
login=thesteins.org
password='YOUR_PASSWORD_SUPPLIED_BY_NAMECHEAP'
@,imap,smtp
```

I only have one IP address available, so just use @ as my dynamic DNS entry on Namecheap to refer to "thesteins.org". Other addresses like www.thesteins.org are just CNAME entries. In order to give imap.thesteins.org and smtp.thesteins.org different IPv6 entries than thesteins.org they need their own IPv4 records as well. (If they are a CNAME then the IPv6 address for “thesteins.org” will be returned even if a separate IPv6 record exists for e.g. imap.thesteins.org.)

To allow ddclient to run in the background edit the file /etc/default/ddclient on both hosts and change the following lines:

```
run_ipup=”false”
run_daemon=”true”
```

While it would not hurt anything to have ddclient running on two servers at once, I decided to have it only run on the primary system letting heartbeat control starting and stopping it. To disable ddclient from automatically starting on boot, enter the following and ignore any warnings it may give:

```
$ sudo systemctl disable ddclient.service
```

Install heartbeat

On both hosts enter the following to install the heartbeat package:

```
$ sudo apt install heartbeat
```

There are then three files that need to be created on each host two of which have slight differences. First, create **/etc/ha.d/authkeys** and make the same on both hosts. It should look something like the following, but with your own key value defined, (replace YOUR_KEY_HERE with your own value):

```
auth 3
3 md5 YOUR_KEY_HERE
```

Protect this file by entering the following:

```
$ sudo chmod 600 /etc/ha.d/authkeys
```

The next file to create as root is **/etc/ha.d/ha.cf**. In this example we will have *lsvm2* set as the primary meaning it will take over services if it is available. If you do not care which server provides the service then auto_failback may be set to off. The file should look like the following:

```
auto_failback on
autojoin none
crm off
deadtime 10000ms
initdead 20000ms
node lsvm1 lsvm2
ucast enp2s1 192.168.1.6
ucast enp2s1 192.168.1.7
warntime 5000ms
```

Reference documentation for this file may be found here:
http://linux-ha.org/wiki/Ha.cf

The final file to create as root on both hosts is **/etc/ha.d/haresources**. The file on both hosts should be the same and look like the following (note that the server name at the very start defines the primary server, so should be the same on both hosts):

```
lsvm2 \
drbddisk::drbd0 \
Filesystem::/dev/drbd0::/srv/drbd0::ext4::defaults \
IPaddr::192.168.1.4/24/enp2s1 \
IPaddr::192.168.3.4/24/enp2s1.3 \
IPv6addr:: 2601:40f:4100:13c2:0:ffff:c0a8:0104/64/enp2s1 \
IPv6addr:: 2601:40f:4100:13c2:0:ffff:c0a8:0304/64/enp2s1.3 \
dnsmasq \
ddclient
```

At this point your setup should be complete and you may start the heartbeat software by entering the following on both hosts:

```
$ sudo ufw allow from 192.168.1.6 to any port 694 proto udp
```

Finally enable and start the heartbeat service by entering the following on both hosts as well as turn off STONITH to avoid errors in the logs:

```
$ sudo systemctl enable heartbeat.service
$ sudo systemctl start heartbeat.service
$ sudo crm configure property stonith-enabled=false
$ sudo crm configure property no-quorum-policy=ignore
```

If all goes well you should see one system start up ddclient and dnsmasq, mount the ***/srv/drbd0*** disk, and obtain the *192.168.1.4* / *192.168.3.4* IPv4 addresses and the *2601:40f:4100:13c2:0:ffff:c0a8:0104 / 2601:40f:4100:13c2:0:ffff:c0a8:*0304 IPv6 addresses on virtual network interfaces. Testing may be done by doing things like rebooting the primary system and checking if the secondary takes over, (then reverts to the primary if you have auto_failback on), pulling network cables and rebooting the switch.

Historical Implementations

Until the early 2000's, DNS and DHCP duties were handled by a pair of Linux servers in a fail-over configuration using heartbeat. ISC tools were used. Also, the servers were registered name servers with a static addressable IP on the Internet, so answered queries for the domain. Two instances of bind were used with one answering external queries and another answering internal ones. The ISC tools allowed for replication internally. Configuration data that was not replicated was stored in a repository and pulled onto the machines on a regular basis. This solution while powerful was complex to maintain. At the time the network was supporting web sites, email, DNS hosting and more for multiple parties.

Around 2007 the network requirements were vastly simplified. At this point a hardware device was used as a router, DNS server and DHCP server. It did not always work well with internal lookups, so systems and devices could not always be accessed by name. Also, when moving to IPv6, there were some limitations in DNS. A later version of the router did support VLANs, but DNS/DHCP features were even more restrictive there.

In 2016, dnsmasq was first used to perform DNS and DHCP duties. This worked well, but when the system was rebooted there was a period of time where DNS and DHCP were unavailable. While outages were infrequent and short, Windows computers would cache "null" responses for a period of time. In order to provide a better user experience, redundant systems were installed using DRBD to share files such as leases and configurations. Heartbeat was used to manage migration of the services and a virtual IP between them.

In 2017 one of the servers hosting virtual machines moved from running Windows to running unRAID. Some tweaks were also made to the configuration of DRBD and Pacemaker based on lessons learned.

Future Opportunities

- IPv6 reverse DNS is not being populated properly for hosts getting IP addresses via DHCP.