Systems Administration

Building a server for Maintainability and Disaster Recovery

Building a server for Maintainability and Disaster RecoveryI like to build my servers a little differently than "the standard" found in most Linux Technical discussions. This article tells what I do, and why I do it.

I have three main goals when building a server.

1. Stability
2. Upgradeability
3. Ease of disaster recovery

To this end, I propose the following solutions

- Use software RAID
- Use LVM
- Move some "standard" directories around for greater ease of backups
- Use Xen where feasible

Note that using Xen, LVM and software RAID **decreases** the available resources of your machine. Each of these use CPU, memory, disk access speed and, in the case of Xen, decrease the throughput on your network. Thus, you must overbuild your machine to compensate for these problems.

Stability

Stability has one definition; Debian Linux "stable". This is arguably the most stable platform currently available. There is a reason why so many other distributions are based on this, the standard for stable, bullet proof platforms. Debian is **always** behind the times, but that is because it is tested, tested, tested. If you want the latest software, or the latest device drivers, go look someplace else. But, if you want a server that will work for years at a time, choose Debian. I think my personal record is five years on one server, and that was taken down because I was very afraid the hardware would die.

The other factor in stability is redundant storage. For this, I like to use RAID on any server I build. Recently, I've decided to accept the overhead of software RAID to allow the greater flexibility offered (see next section). While this means the processor does more work, and memory is used, for the most part these components can be initially designed to compensate. I tend to use RAID1 for less critical areas (system hard drives, temporary storage), but RAID6 for areas that will be accessed more often and require larger stores.

Upgradeability

Systems Administration

Operating System

Debian is also the only distro I know of that puts so much effort into seamless upgrades bewteen releases. The upgrade for Sarge to Etch included almost every package in the whole repository. And yet, I was able to do this with by editing /etc/apt/sources.list, and issuing the simple command apt-get update ; apt-get dist-upgrade. The upgrade from etch to lenny was almost as simple. **Every** other distro I have tried requires, basically, a complete re-install to upgrade.

Hardware

Many parts of the hardware are fixed, and can not be changed without problems. Upgrading RAM is easy, but changing CPU or motherboard can be difficult without careful research. However, storage is something that we can attack easily, if we accept the additional resources required to build using software RAID and LVM.

Software RAID

Software RAID has a good, if underdocumented, upgrade path. If your initial storage requirements were too low, you can replace one disk at a time with larger capacity drives, until your entire RAID set is larger. During the upgrade period, you are still limited to the original size, but software RAID allows you to dynamically resize the set once the new disks are fully in place. Consider the following scenario, with four 400G drives comprising a RAID-6 setup, giving us a little less than 800G of usable space. We will upgrade to terabyte drives, which will increase our set to slighly less than 2 Terabytes.

1. for each physical drive in RAID set
    1. Remove drive from RAID set via MD controls
    2. Shut down and replace 400G drive with Terabyte Drive
    3. Add new drive to RAID set via MD controls
    4. Wait for rebuild to complete. This can, literally, take days
    5. repeat
2. Using MD control set, extend RAID set to full capacity of drives

LVM

Logical Volume Manager allows you to allocate space to virtual partitions on a device. When placed on top of a RAID set, it allows extreme flexibility combined with the reliability of RAID. An LVM partion is no longer a fixed size, requiring a complete rebuild of the partition table, instead allowing partitions to be taken offline and resized rapidly, then placed back into the use it was intended for. One or two file systems even allow this to be done without dismounting the partition first.

It is important to understand that, while the sizes can be increased easily, it is more difficult to "resize down," ie make them smaller. LVM will do the downsizing very

easily, but few file systems will decrease their size with no problems. Thus, make the partitions as small as you can, and only increase when needed.

Ease of Disaster Recovery

Disaster recover relies on good backups. However, backing up an entire working server is difficult. The files used by PostgreSQL, MySQL and SVN can not be simply "copied" unless great pain is taken to ensure they are not being changed during the copy process. Mail Queues and log files are, again, subject to writing during the backup process (or worse, truncation).

The question then becomes, what would be the minimum set of data required to rebuild a complete replacement. We are assuming you have only this data, and an replacement computer. We can not even assume the replacement machine is the same type as the original.

The items I have come up with are:

- installed packages
- configuration for installed packages
- user files
- database
- svn/cvs
- samba file shares
- web sites
    - web pages (ie, DocumentRoot)
    - cgi scripts (ie, cgi-bin)
    - (optional) web logs -- important to some, not to all

Now, what are the minimum directories that need to be backed up to do this. It gets complicated, since some information is nested deep within areas we don't care about (apache log files are in /var/log/apache, and we generally can live without /var/log), or in a format that can not easily be backed up (databases and svn repositories).

Basically, under the old system, we need the following directories:

- /home (user files)
- /var/log/apache(2) (web logs)
- /var/www (DocumentRoot)
- /usr/lib/cgi-bin
- /var/lib/svn
- /var/lib/mysql
- /var/lib/postgresql
- /etc
- /root (many sysadmins place important info here)
- whereever apt puts its stuff

The schema I use makes it possible to only back up three directories; /home, /etc and /root. I do this by modifying the configurations listed in the subsection, or executing the scripts therein to make hot backups.

How we make the backup is up to the individual. I like rsync, and put the backup on a remote server, as physically separated from the source as possible.

Apache

I create a directory, /home/www, and under this create a separate directory tree for each web site (I often have multiple web sites on a server). Under each web site directory, I create the following: web cgi-bin logs webmin

I then create an apache macro that allows me to rapidly create a new web site, as follows:

```
  <Macro website $siteName>
     # use the following if you are doing virtual hosting off one IP
     <VirtualHost *>
        ServerName   $siteName
        ServerAdmin webmaster@localhost
        AddType application/x-httpd-php .html
        DocumentRoot /home/www/$siteName/web/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /home/www/$siteName/web/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>
        ScriptAlias /cgi-bin/ /home/www/$siteName/cgi-bin/
        <Directory "/home/www/$siteName/cgi-bin">
                AllowOverride None
                Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>
        # Include the webalizer directory. Note, you must configure we
balizer to get
        # its logs from /home/www/$siteName/logs/access.log, and put t
he output
        # into /home/www/$siteName/webalizer/
        Alias /webalizer/ /home/www/$siteName/webalizer/
        ErrorLog /home/www/$siteName/logs/error.log
        # Possible values include: debug, info, notice, warn, error, c
```

```
rit,
        # alert, emerg.
        LogLevel warn
        CustomLog /home/www/$siteName/logs/access.log combined
        ServerSignature On
     </VirtualHost>
  </Macro>
```

This allows me to create a "standard" web site with the following single line

```
Use website www.example.com
```

Which will assumes the directory /home/www/www.example.com/ exists and has the correct subdirectories.

Note: Many uses for historical data stored in the logs exist, and as such, I am including them in the data to be backed up. However, they can get quite large on an active site.

Databases

Since databases should not be copied hot, I have a script that runs each night, creating a database backup and copying the files into /home/dbbackup (set it to ownership root:root, permissions 700). It will create a file for every MySQL database, and a single file for all Postgresql databases.

```
  #! /bin/bash
  MYSQLDATAPATH=/var/lib/mysql/
  DBSAVEPATH=/home/dbbackup/
  POSTGRES_BACKUP_NAME=posgres.dmp
  # create the backup directory if it doesn't exist
  if [ ! -e $DBSAVEPATH  ]
  then
     mkdir -p $DBSAVEPATH
     chown postgres:root $DBSAVEPATH
  fi
  # pg_dumpall is a PostgreSQL utility
  if which pg_dumpall > /dev/null # don't do this if pg_dumpall not in
stalled
  then
     echo Backing up Postgres to $DBSAVEPATH
     # get all postgres databases, just do a db dump
```

```
     su postgres -c 'pg_dumpall -d -c ' > $DBSAVEPATH$POSTGRES_BACKUP_
NAME
     if [ -e $DBSAVEPATH$POSTGRES_BACKUP_NAME.gz  ]
     then
        rm -f $DBSAVEPATH$POSTGRES_BACKUP_NAME.gz
     fi
     gzip -9q $DBSAVEPATH$POSTGRES_BACKUP_NAME
  fi
  # back up all MySQL databases
  if [ -e $MYSQLDATAPATH  ]
  then
     echo Backing up MySQL tables to $DBSAVEPATH
     for db in `mysqlshow | grep -v ^+ | grep -v Databases`
     do
        if [ "$db" != "|"  ]
        then
           if [ -d $MYSQLDATAPATH$db  ]
           then
              echo -n ' '$db' '
              mysqldump -c --add-drop-table $db > $DBSAVEPATH$db.dmp
              if [ -e $DBSAVEPATH$db.dmp.gz  ]
              then
                 rm -f $DBSAVEPATH$db.dmp.gz
              fi
              gzip -9q $DBSAVEPATH$db.dmp
           fi
        fi
     done
  fi
  echo
```

This will create a dump file of each MySQL database (and compress it), and a dump of all Postgres databases as one file in DBSAVEPATH. Note: if you have plenty of disk space, and you want to optimized an rsync backup, remove the compression routines. While the compression routines save tons of space (90% in many cases), they entire file must be copied for one minor change if compressed. Using rsync on an uncompressed file, however, only copies the changes.

SVN

Again, SVN should not be copied hot, so I create a directory /home/svnbackup, then execute the following script.

```
  sourceDir=/var/lib/svn/
  targetDir=/home/svnbackup
```

```
  echo Backing up SVN Repository at $sourceDir to $targetDir
  for repository in `ls $sourceDir` ; do
     echo -n ' '$repository' '
     [ -d $targetDir/$repository  ] || mkdir $targetDir/$repository ;
     svn-backup-dumps -qbO $sourceDir/$repository  $targetDir/$reposit
ory > /dev/null ;
  done
  echo
```

Samba

Samba configuration is set to point to subdirectories under /home/samba, thus all users store their network share files under this. **Note**: Newer versions of Samba store their password files in a separate location, thus depriving you of the ability to write generic routines to back these up. Either use the older version of smbpasswd (which is similar to /etc/passwd, and stored in /etc/smb), or plan to do a manual backup of these files.

installed packages

I use dpkg to get a list of all standard, installed packages, and store this list in /home/osbackup. Even though this changes rarely, it is a low resource process, and I never can remember to do it everytime I update a server. So, I just tell it to back up nightly.

```
dpkg --get-selections \* > /home/packages/selections.list
```

Note: the * basically tells dpkg to get everything. This includes purges, which is handy.

One backup script to rule them all

Actually, one option solves a lot of the backup problems, and that is to create a single backup script that dumps the databases and svn, creates the package selection dump file, and then performs a backup. This can be as simple as concating all of the above scripts together, then putting a simple rsync command at the bottom, or it can be more formalized. Daily Data has rsbackup, which is designed to allow multiple clients to back up to a single server, and has the facilities for doing pre-processing such as database dumps, svn dumps, and the like. This is a series of Perl scripts in either source (no installer) or a Debian package for the client. The

server portion is just a bunch of scripts. The (very poor) documentation is located atthe RSBackup documentation here.

Putting it all together

You will note that, in the above, a lot of stuff is going in /home. Thus, I always create a /home/users, and modify /etc/adduser to place users in a directory under this. That decreases the complexity of the /home directory.

With the setup above, we can now partition our drives differently. A 4G / partition is sufficient. We actually have extra room if you follow these guidelines, but with the size of hard drives now, I don't see any reason to make root larger than that. /root can generally be within this space, as can everything else except /home (which can grow very large).

I partition my servers as follows:

- / = 4G
- /tmp = 100M (exception is mail servers, where this must be a lot larger)
- /var/tmp = 100M
- /home = however large you need it

/ needs to be set as normal, but /tmp, /var/tmp and /home do not need the wasted 5% space normally set aside for root access in case a partition fills up. Thus, you can format these three partitions as:

```
mke2fs -j -m 0 -L Volume Name path_to_partition
```

the -m 0 says "don't reserve blocks for root in this partition" (you can also modify an existing partition with tune2fs). Voila, you have just gained 5% of your disk space back, and have not decreased your ability to recover from a partition filling up.

Why are /tmp and /var/tmp set as separate partitions? The main reason is security. You should mount these directories as noexec and nosuid. /tmp and /var/tmp can be written to by anyone, and one of the ways of hacking a system is to write executeable scripts into these directories. Under many circumstances, you can do this with /home also.

Problem Solving

Some simple problem/solution scenarious may help you see if some or all of the procedures addressed here are useful for you.

One of my partitions is filling up

Problem: You have a currently mounted partition (not root, see below) that is filling up, and you need to fix this.

Solution: There are two cases here. One, simply dismount the partition, grow the partition size, then remount it. For ext3 file systems, the procedure is described in many places on the Internet, including a local "how to."

One of my partitions is filling up, and I can not unmount it

Problem: A mounted partition is filling up, but you can not unmount the partition to work on it. Maybe the partition is /, and you can not shut down the partition.

Solution: Create a new partition to be mounted in a high usage are of the partition that is filling up. For exampe, assume /var/lib/mysql is on the same partition as /, and the root partition is fill. You investigate and find multi-gigabyte databases.

Use LVM to create a new partition, appropriately sized for the MySQL data store. Format the new partition, and mount it someplace temporarily (say, /mnt). Now, shut down mysql, then move all files in /var/lib/mysql to the new partition. Unmount the partition from /mnt, and remount it at /var/lib/mysql. Start MySQL, and you have now freed up whatever space it was taking on the root partition:

```
# create a 20G partition named mysql_data in the volume group virtuals
lvcreate -L 20G -n mysql_data virtuals
# format it as ext3, no reserved blocks, labeling it mysql_data
mke2fs -jm 0 -L mysql_data /dev/virtuals/mysql_data
# mount the new partition someplace temporary
mount /dev/virtuals/mysql_data /mnt
# shut down mysql
/etc/init.d/mysql stop
# move all files from the mysql data directory to the new partition
mv /var/lib/mysql/* /mnt/
# unmount the new partition
umount /mnt
# you should change the permissions in /etc/fstab. This is wrong. But,
 it is a quick and dirty way to allow mysql access to its data files
chmod 777 /var/lib/mysql
# mount the new partition at the normal mysql data store
mount /dev/virtuals/mysql_data /var/lib/mysql
# start mysql
/etc/init.d/mysql start
```

Note: You must make an entry in /etc/fstab, or the next time you reboot the

computer your new partition will not be mounted.

I do not want users to be able to execute programs in their home directories, but I need web sites to have this ability

Problem: You need different mount options in different parts of your sub-directory.

Solution: Many necessary mount options can work only on a mounted partition (thus, the name mount option). quota, setuid, noexec, are all important security parameters to lock down your server, but they are not applicable in many cases. The only option here is to mount separate partitions where you need to change things. In this case, you would create a partition for /home and set it noexec, then a partition for /home/www and mount it exec.

My server suffered a fatal crash, and I need to build a new one to replace it fast

Problem: Your server is gone. Maybe the hard disks self-destructed. Maybe your building burned down. Or, someone broke into your office and stole your server (all three of these have happened to my clients). Whatever it is, all you have is a backup.

Solution: Actually, it is fairly easy to be the hero here. Get a machine. Don't worry about what it is, just get a machine. Borrow, beg, buy some kind of off the shelf machine, but just get a computer with enough space for your information. Send someone down to get a copy of your backup from the backup server (it is off site, isn't it?). I am assuming that you are using the same processor, or pretty close (ie, an AMD64 was the old one, and an AMD64 is the new one, or something like that). There are ways around it otherwise, but it is longer and more difficult.

1. Replicate your server setup.
    1. Do a basic Debian install. No extras, just a basic Debian install.
    2. rename /etc/apt/sources.list to something, then copy /etc/apt/sources.list from the backup
    3. apt-get update ; apt-get upgrade # this makes sure you are looking at the same store
    4. dpkg --set-selections < backup_location/selections.list
    5. apt-get -u dselect-upgrade
2. At this point, your computer is a duplicate of the original (well, if all your software was installed via apt, and you are good about updates).
3. Load your users
    1. This is a little more manual. Open /etc/passwd, and backup /etc/passwd. Look for users from the old system (hint, I set all my users to start at 1000), and copy/paste them to /etc/passwd. Now, do the same for shadow, group and gshadow (hint, I never set user groups; everyone is in users). Your users can not log in yet, as their home directories are not there, but it helps in the next step.

4. Load your data
    1. copy /home from your backup. Be sure and use -a, so your ownerships and permissions stay the same. cp -av backup/home/* /home
5. Configure your system
    1. You can do this by simply overwriting /etc with your backup copy, but that scares me. I simply copy the things I normally change, ie apache2, php, samba, postgres, postfix and mysql.
6. Recover mysql
    1. go into the database backup directory. For each mysql file, do mysql -u root -p < database.dmp where database.dmp is the dump file you created. Note: I'm not sure how the user permissions work on this. They are stored in the database named mysql, in the table Users. Do a test run first.
7. Recover PostgreSQL datavbases
    1. As superuser, psql < backup/postgres.dump. **This has not been tested. Verify**
8. Recover SVN
    1. RTFM. I've not done this, but I know it is possible. You must restore into the same directory if you want web_svn to work with the origional configuration.
    2. Restart your computer (easiest). Remember all those messages that go by that you ignore until the command prompt? This is the time to watch them. Look for anything you don't understand. When you can log in, do a dmesg | less and look for any errors.

The last time I did this, it took about 27 hours (the dataset was 925G, and that took a long, long time). Client called me on Saturday, at about 6pm, saying their server had been stolen. I had a replacement server in place at 9pm the next day. I was a hero, all because, luckily, the client had some good backups.

Unique solution ID: #1005
Author: Rod
Last update: 2012-01-30 05:20