Linux migration tips and tricks

In this article we give some advice on how to make your Linux server migration safer and discuss migrating on a per-package basis.

Other scenarios

In the **previous article** in this series we discussed using rsync to do a live migration of your system from one Linux server to another. We looked at preparing the destination environment to be similar to the source server, then set up an rsync exclude file, then performed the sync.

It's not always practical to perform a live sync the way we outlined it, or you may want to migrate just a couple applications instead of the entire system. Let's look at your options in those cases.

Remember that this process requires that rsync be installed on both the origin and destination servers. The package name is usually "rsync". Use your package manager to install it if you get a "not found" response from running:

```
which rsync
```

Syncing with the server inactive

A live sync has the advantage of minimizing downtime during the migration but it can require multiple runs of rsync to complete if you have a lot of files that change frequently. After a pass or two with rsync on a live server it can be practical to perform the final sync while your origin server is not running at all.

It's also possible that you may be unable to boot the source system for reasons unrelated to the installed software. In that case you'd also need to be able to access your files while the server is dormant.

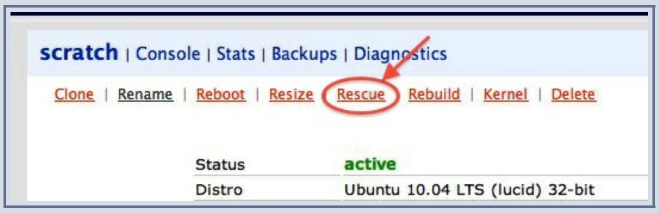
For real (non-virtual) servers you'd copy files while keeping the server down by booting the machine from a rescue disk (usually a live CD distribution), mounting the filesystem, then performing the final sync from there. Fortunately there are ways to simulate that for virtual servers.

An approach many virtual server providers take is to provide an option to boot your server using a temporary server instance. It acts as a virtual rescue disk, allowing you to mount your server's filesystem while the system isn't running. On Slicehost and Rackspace Cloud it's called "rescue mode".

Rescue mode

For a server hosted at Slicehost you can put the VPS into rescue mode via the SliceManager.

When you're in the SliceManager navigate to the screen for the slice you want to put into rescue mode. Look at the top of the slice's controls to find the "Rescue" link.


scratch | Console | Stats | Backups | Diagnostics
Clone | Rename | Reboot | Resize | Rescue | Rebuild | Kernel | Delete

Status	active
Distro	Ubuntu 10.04 LTS (lucid) 32-bit

You'll be asked if you really want to put the server into rescue mode. If you say yes you'll be given a temporary root password you can use to connect to the server once it's done switching into rescue mode. The SSH key for the host will have changed, so you'll probably need to delete the server's key from your "~/.ssh/known_hosts" file or equivalent.

Once you're logged into the server in rescue mode you should be able to mount your server's filesystem and then proceed with the sync. For example:

```
mkdir /mnt/origin
mount /dev/sda1 /mnt/origin
```

Note that rescue mode has a time limit of two hours, after which your server will reboot in normal mode. If your final sync takes longer than that you may need to put the slice back into rescue mode then run the rsync command again to pick up where things left off. If you still run into trouble with the time limit **talk to our support staff** and they can assist you.

For more information about using rescue mode you can visit **this article on the subject**.

Rsyncing from a mount point

After booting your server into a rescue mode and mounting your server's filesystem to a mount point like "/mnt/origin" you will need to adjust your rsync command accordingly.

First make sure you create an exclude file and make any changes necessary for your environment as explained in the "Rsync excludes" section of **this article on live server migration**. Create the list without mentioning the mount point stuff, just list them as they would appear in your regular file system.

An example exclude file would look just like one used for a live migration:

```
/boot
/proc
/sys
/tmp
/dev
/var/lock
/etc/fstab
/etc/mdadm.conf
/etc/mtab
/etc/resolv.conf
/etc/conf.d/net
/etc/network/interfaces
/etc/networks
/etc/sysconfig/network*
/etc/sysconfig/hwconf
/etc/sysconfig/ip6tables-config
/etc/sysconfig/kernel
/etc/hostname
/etc/HOSTNAME
/etc/hosts
/etc/modprobe*
/etc/modules
/etc/udev
/net
/lib/modules
/etc/rc.conf
```

Next we'll set up our rsync command so it takes the mount points of the file systems on both servers into account. The rsync command with the origin server in a rescue mode would be:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after --exclude-from="/mnt/origin/home/demo/exclude.txt" /mnt/origin/ root@1.2.3.4:/
```

Note the trailing "/" on the origin directory. Including the slash at the end makes sure rsync treats the origin and destination directories as the same relative locations, so don't leave that part out. Otherwise you might end up with your files getting copied into a new subdirectory on the destination instead of sending the files to their proper locations.

As a bonus, with that trailing slash on the directories rsync will treat the exclude file list as relative to the source directory. That's why we don't need to change the exclude file to account for the mount point.

If you're being extra careful and have both the origin and the destination in a rescue mode you would change the destination directory too. With the destination server mounted at "/mnt/destination" the rsync command would look like:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after --exclude-from="/mnt/origin/home/demo/exclude.txt" /mnt/origin/ root@1.2.3.4:/mnt/destination/
```

Once the final sync is done you can boot up the destination server and run your tests.

Per-package approaches

It may be impractical to migrate your entire server, or you may only have a couple services you need to bring over. In that case you can migrate the system on a per-package basis. It might be a little more work than running a full sync but it could be faster overall.

In general this approach would require installing the requisite package on the destination server then copying its configuration and data files from the origin server to the appropriate place on the destination. Once you're done start or restart the service on the destination and test to make sure everything is in its place.

You may need to tweak any aspects of the system you'd changed on the origin server once you've completed the copies. If you created a logrotate config (or changed it) for the service you'll need to copy that over. If you had a cron job set up for the service that would also need to be migrated.

A couple examples follow to illustrate the approach.

Web servers

If you're migrating a web server you'll need to make sure you bring over your configuration files (including virtual host definitions) as well as the files used by your website.

If you've been keeping your web files in a user's home directory make sure you have that user created on the destination server. If the user name is "demo" and the web files are all in the directory "public_html" you can run an rsync command similar to the following:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after ~demo/public_html root@1.2.3.4:/~demo/public_html
```

We left out the "exclude-from" flag because we wouldn't usually need to exclude any files from this sync.

The configuration directory for your web server may vary by distribution, particularly for apache. Ubuntu and Debian use "/etc/apache2", CentOS and other Red Hat-based distributions use "/etc/httpd", and so on. So first, find your config directory.

Once you have the configuration directory identified run an rsync command similar to the above but copying the configuration directory instead. If you're running nginx this might look like:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after /etc/nginx root@1.2.3.4:/etc/nginx
```

If you're using PHP you may also need to bring over any changes you made to your php.ini.

After that restart your web server and run it through some tests.

Databases

A similar approach works with a database service. Install the database on the destination server, making sure you get as close to the version running on the origin as you can.

Bring the database service down and identify where its configuration and data files are kept. For mysql the configuration files are usually in /etc/mysql and the databases themselves are in /var/lib/mysql.

It's easier to do this with two rsync commands, one for the config and one for the databases. For a mysql installation the commands might look like this:

```
sudo rsync -e 'ssh -p 30000' -azPx --delete-after /etc/mysql root@1.2.3.4:/etc/mysql
sudo rsync -e 'ssh -p 30000' -azPx --delete-after /var/lib/mysql root@1.2.3.4:/var/lib/mysql
```

Next check to make sure there aren't other changes you made on the origin server related to the database (like cron jobs or logrotate configuration). Then start or restart the database service on the destination and get to testing.

Summary

Now you should have your system migrated to another server and are enjoying the results. Congratulations! But do remember to test thoroughly before going into production with the new server. Surprises are bad (on a server, anyway).

-- Jered

Want to comment?

Name:

Email Address: (not made public)

Website: (optional)

Comment: (use plain text or Markdown syntax)

Post comment

Tags: admin apache api arch at awstats backup capistrano centos cloud courier
cron dapper debian dig django dns dstat ebook email fail2ban failover fedora feisty

cron dapper debian dig django dns dstat ebook email fail2ban failover fedora feisty
forensics ftp gentoo git gutsy ha hardy heartbeat ibex intrepid iotop iptables jaunty karmic
kernel lenny logrotate logs lucid mail maverick migration mod_rails mongrel
monitoring munin mysql nagios network nginx ntp open relay passenger
permissions php postfix postgresql pv-grub rails rescue rescue mode resize rhel
rootkit rsync security setup sftp shorewall slice slice administration slice manager
slicemanager spam spamhaus spf ssh ssl subversion tar tcpdump telnet thin tomcat ubuntu
untar upgrade vhosts windows winscp