This is a Tutorial for building a ‘fairly simple’ backup system for those of us that host ERPNext on 3rd party VPS cloud servers. It is called the Poor Man’s Backup System because it uses a few scripts and ‘cron’ jobs to to automate a backup system that keeps 1 (or more) ‘hot spare’ server(s) updated with backups of your main production ERPNext server on your schedule.

If you are reading this then you have probably already been researching some of the ‘backup services’ available on the market, or read up on the complex ‘fail-over’ server configurations and found them both too complex to understand and more expensive that your budget allows. I know I was there a year ago and had to find something workable and cheap as fast as I could. I spent several weeks thinking about it and learning all I could about backing up ERPNext. At that time I put together some easy to understand scripts and got them to run in a semi-automated way so I could sleep better at night.

The first version of this [Tutorial] only managed making the backup of the database and moving the files around the internet for me. It was missing the important ‘external files’ required to support the database. Since those files were not changing or updating very often I was manually moving those files every Sundy during weekly maintenance. When the number of servers grew, this process became an event that consumed much of my day of rest. This new version includes handling those external files and is a complete data backup system now. It still does not restore the files for you, but it sets everything up so that you only have to runt the restore commands to be up and running on a hot spar server in a few minutes.

If you spot something I left out please chime in and I will update the text of the post. Likewise if you see an error in syntax, also let me know and I will humbly fix it. However, if after reading all of this you know of a better solution, then please start a new thread with your simpler and better solution explained in step by step detail, then place your link to it here so all of the new users may benefit from it. Anything less is just you wanting to hear yourself talk and not really say much.

\*\* Fair Warning \*\* I am not a developer or even a fair programmer of high level languages, I am just a systems implementer with just enough knowledge of linux to keep my head above water. So I hope I can get the point across for most newcomers to ERPNext.

Poor Man’s Backup System Details (v2):

This plan requires you to have 2 KVM type VPS servers on different physical hardware. These can be in the same building, different buildings or even in different cities as long as they are different physical hardware. If you are using OpenVM style servers, this solution may not work (ERPNext may not work either).

One server is the primary live production server and the other is an identical standby production server. If you are just setting up ERPNext for the first time, it is best to set up both servers at the same time. You can then install everything on both servers at the same time. If you already have a live production server running (even if it is only a day or two old), it is probably best to use the image backup service provided by your cloud VPS service provider to make the second server.

I use the same VPS service provider for many of my ERPNext server clients. The advantages in this for me is the ability to have the service provider clone my servers and spin them up on new hardware. You would have to check with your provider to see if these services are available to you.

The Poor Man’s Backup System essentially runs database backups and collects the supporting external files on a schedule that you choose then sends a copy of the complete backup to the standby server so that it can be restored on very short notice in the event of a catastrophic failure of the primary live production server. You can set the interval of the “backup and send” routine to best fit your comfort level.

8 Steps to building the Poor Man’s Backup System:

1.) Setup 2 identical VPS servers on different hardware and different ip addresses

2.) Create a simple directory structure on primary server for the created backup files

3.) Add one script file on the primary server to run the backups, collect the external support files, combine everything into single portable files, manage the backup files, and send the files to other servers for safe keeping

4.) Add one crontab entry on the primary server to run the custom script file

5.) Create simple directory structure on standby server to manage inbound backup files

6.) Add one script file on the standby server to manage the inbound backup files and make them ready for quick restore

7.) Install the “incron” utility on the standby server

8.) Add one incrontab entry on the standby server to run the backup file management script

So, let’s get started!! If you followed the instructions up to this point you already have step one finished ( you have 2 identical ERPNext servers running) and are ready to begin adding the proper elements to the servers. So Step #1 is finished.

From this point forward there will be many references to the “default user account” on the servers. Since they are both identical the account should also be identical on them. The default user account is the linux user account that has the ERPNext installation within it. If you used the defaults in the easy install script that account would be /home/frappe/ however I recommend that you add the --user switch to the easy install script at install time and make the default account your cloud servers linux user login account. In this document it will always be called “def\_user”

Step #2:

On the primary server perform the following commands from the default user account.

mkdir /home/def\_user/bin

mkdir /home/def\_user/backup

mkdir /home/def\_user/backup/current

mkdir /home/def\_user/backup/last

mkdir /home/def\_user/backup/bk\_wip

You now have a simple directory structure for holding your backup files on the primary server and easily managing them.

[NOTE: If you are reading this after having already implemented the first version of the PMBS (PoorMan’s Backup System) then you will notice the directories that hold the data are named slightly different. That is a deliberate attempt to allow you to build and test this newer version without tearing down what you already have running. The old one continues to run until you are ready to delete it in favor of the new version ☺]

The /home/def\_user/bin directory is where the executable scripts you generate in the next step will be stored. In order for them to be functional, you have to tell the system that the executable scripts in this directory can be run without specifying the directory path to the script.

To do this run the following command:

export PATH=$PATH:/home/def\_user/bin

Now it will not matter where you are in the directory structure, if you type the name of the script file it will run as it it were a regular linux command. :grin:

Step #3:

Use the nano editor and create the following file named ‘bkup\_cycle’ in the new /bin directory

#!/bin/bash

#

# This script performs one backup cycle.

rm -f -r /home/def\_user/backup/bk\_wip/\*

mv /home/def\_user/backup/current/\*.tar.gz /home/ def\_user /backup/last/

find /home/def\_user/backup/last -maxdepth 1 -type f -name "\*.tar.gz" -print0 | xargs -r0 ls -t | tail -n +16 | tr '\n' '\0' | xargs -r0 rm

mysqldump -u root –pMySQLpassword 1bd3e0294da19198 | gzip > /home/def\_user/backup/bk\_wip/db\_bkup.sql.gz

tar -czf /home/def\_user/backup/bk\_wip/pri-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/private/files .

tar -czf /home/def\_user/backup/bk\_wip/pub-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/public/files .

tar -czf /home/def\_user/backup/current/"$(date '+%m%d-%H%M').tar.gz" -C /home/def\_user/backup/bk\_wip .

scp /home/def\_user/backups/current/\*.tar.gz def\_user@192.168.222.111:/home/def\_user/drop/

# This should complete the backup cycle

# End

You are not out of the woods yet… You still have to make this new script file executable. To do this run the following command:

chmod +x ~/bin/bkup\_cycle

Now we can move onto the explanations of the lines in the script file:

rm -f -r /home/def\_user/backup/bk\_wip/\*

This first line clears out the wip (work in progress) directory that the script uses to collect the files together before turning them into a single compressed file.

mv /home/def\_user/backup/current/\*.tar.gz /home/ def\_user /backup/last/

This command moves the previous backup file out of the /current directory so it is clear for the creation of the next backup file. Previous files are stored in /backup/last/

find /home/def\_user/backup/last -maxdepth 1 -type f -name "\*.tar.gz" -print0 | xargs -r0 ls -t | tail -n +16 | tr '\n' '\0' | xargs -r0 rm

This command was pretty hard to develop. It uses the linux ‘find’ command and digs into the /backup/last/ directory to remove all but the most recent 15 files. My system makes 15 backup files per day and I keep one day’s worth on hand on the primary server. You can adjust this to whatever number of files you are comfortable having on hand. Just edit the number 16 in “tail –n +16” to be the number of files you want to keep plus 1.

mysqldump -u root –pMySQLpassword 1bd3e0294da19198 | gzip > /home/def\_user/backup/bk\_wip/db\_bkup.sql.gz

This is the command that runs the database backup. You will need to edit in your mariadb password and your specific database name so this command will work for you. The mariadb password begins with the character immediately following the “-p” so in my case it is MySQLpassword. The next random string of characters is actually the database name. You can find yours by reading the “site\_config.json” file for your site. It is very easy to identify there.

tar -czf /home/def\_user/backup/bk\_wip/pri-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/private/files .

tar -czf /home/def\_user/backup/bk\_wip/pub-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/public/files .

These next two commands gather the files from your sites /private/files and /public/files directories and compresses them into two files ending with “.tar.gz” and places them in the /bk\_wip directory.

tar -czf /home/def\_user/backup/current/"$(date '+%m%d-%H%M').tar.gz" -C /home/def\_user/backup/bk\_wip .

This last ‘tar’ command collects all 3 backup files in the /bk\_wip directory and creates a single compressed file for easy transport and places that complete file into the /current directory.

scp /home/def\_user/backups/current/\*.tar.gz def\_user@192.168.222.111:/home/def\_user/drop/

This last command is the linux Secure Copy command and allows you to securely copy files from on server to another across the internet. Just replace the dummy ip address in this command with the ip address of your standby server.

If you wanted to keep multiple hosts as on standby in different cities, you just add another ‘scp’ command to the end of the script with the ip address if the other server. I try to keep one primary production server and 2 standby servers on opposite sides of the country just in case of some major disaster that interferes with one of the sites.

NOTE: In order for the ‘scp’ command to run properly from a script, you will have to go through the process of setting up the security keys on the main server and the standby server to allow them to communicate securely. Please read THIS POST or THIS OTHER INFORMATION for details about how to do this. It is really simple but not part of this tutorial so use these links for the instructions.

Step #4

From the default linux user account, use the following command to open the nano editor into the crontab list:

crontab -e

Add the following line to the bottom of the list:

58 5-18,23 \* \* 1-6 /home/def\_user/bin/bkup\_cycle

Adjust the numbers to best fit your desired frequency of running backups. I do it every hour at 2 minutes before the hour for the busiest 14 hours of the day and then one last time at 2 minutes before midnight. The script runs every day except Sunday. If you are not familiar with crontab syntax, please do a google search for an explanation of each setting and build your string to match your desired number of backup cycles per day.

This completes the setup steps for the primary server. The next group of steps will take place on the standby server so ssh into that server to begin.

[NOTE: If you are a current user of the first version of the PMBS, then before you continue forward with this new version, you need to make sure all of the changes you have made up to this point on the primary production server are working to your satisfaction. All of the instructions, up to this point, for the production server in this version were created to allow you to run both the old and the new versions at the same time so you could verify the operation of the creating of the backup files. ONLY if you are certain that the changes to the primary production server are working should you continue here. From this point forward you must ‘replace’ everything from the previous PMBS on the standby server beginning immediately after this notice. This means that you should disable the version one script in crontab on the production server and delete all the directories under /home/def\_user/backups]

Step #5

On the standby server perform the following commands from the default user account:

(if they already exist from a previous version of PMBS then delete their contents)

mkdir /home/def\_user/bin

mkdir /home/def\_user/drop

mkdir /home/def\_user/current

mkdir /home/def\_user/local\_ready

mkdir /home/def\_user/local\_restore

This creates the simple directory structure for managing the database backup files generated on the primary server and sent over to this server. You may not wish to keep as elaborate a set of data files as I keep, so adjust this to your own liking.

/bin – is where the script that incrontab runs will be stored.

/drop - is where the inbound fresh database backup file is deposited from the primary server

/current – is where I keep the same 15 files that I keep on the primary server

/local\_ready – is where I put a copy of the most recent inbound file (easier to find this way)

/local\_restore – is a location to stage the unzipped database file ready for the restore command

Again, the /home/def\_user/bin directory is where the executable scripts you generate in the next step will be stored. In order for them to be functional, you have to tell the system that the executable scripts in this directory can be run without specifying the directory path to the script.

To do this run the following command:

export PATH=$PATH:/home/def\_user/bin

Step #6

Use the nano editor and create the following file named ‘cycle\_backups’ in /def\_user/bin/

#!/bin/sh

#

# When incron detects a new file in the /drop directory it triggers this script.

rm /home/def\_user/local\_ready/\*

rm /home/def\_user/local\_restore/\*

cp -f /home/def\_user/drop/\*.tar.gz /home/def\_user/local\_ready/

mv /home/def\_user/drop/\*.tar.gz /home/def\_user/current/

rm /home/def\_user/drop/\*

find /home/def\_user/current -maxdepth 1 -type f -name "\*.tar.gz" -print0 | xargs -r0 ls -t | tail -n +16 | tr '\n' '\0' | xargs -r0 rm

tar -xzf /home/def\_user/local\_ready/\*.tar.gz -C /home/def\_user/local\_restore/

gunzip /home/def\_user/local\_restore/db\_bkup.sql.gz

# This should complete the movement of files

Also again, you must set the script file to be executable with the following command:

chmod +x ~/home/def\_user/bin/cycle\_backups

Now you are ready to move on…

Commands in the script explained:

rm /home/def\_user/local\_ready/\*

rm /home/def\_user/local\_restore/\*

These two remove commands clean out the files from the previous drop cycle to make ready for the new files just arriving.

cp -f /home/def\_user/drop/\*.tar.gz /home/def\_user/local\_ready/

This command takes a copy of the new file and loads it into the /local\_ready directory

mv /home/def\_user/drop/\*.tar.gz /home/def\_user/current/

rm /home/def\_user/drop/\*

After the copy function, these 2 commands move the file from the /drop directory to the /current directory for storage. This should leave the /drop directory empty. But just in case there is a stray file the rm command deletes everything that might be left over.

find /home/def\_user/current -maxdepth 1 -type f -name "\*.tar.gz" -print0 | xargs -r0 ls -t | tail -n +16 | tr '\n' '\0' | xargs -r0 rm

Again, this command keeps the /current directory purged down to your chosen number of files just like on the primary server.

tar -xzf /home/def\_user/local\_ready/\*.tar.gz -C /home/def\_user/local\_restore/

This tar command decompresses the 3 main backup files into the /local\_restore directory

gunzip /home/def\_user/local\_restore/db\_bkup.sql.gz

The final gunzip command decompresses the sql database backup file so it is ready in the event you need to execute the restore command to get the server running quickly.

If you were to issue the bench restore command now, this server would be ready for use with the most recent backup. The only other thing that has to happen is to have the external files un-tarred (unzipped) into the proper directories and everything would be complete. If you are not as familiar with regular linux maintenance and want to automate that process as well, then you can create this additional script to do that for you.

#!/bin/sh

#

rm /home/def\_user/frappe-next/sites/site1.local/private/files/\*

rm /home/def\_user/frappe-next/sites/site1.local/prublic/files/\*

tar -xzf /home/def\_user/local\_restore/pri-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/private/files/

tar -xzf /home/def\_user/local\_restore/pub-files.tar.gz -C /home/def\_user/frappe-bench/sites/site1.local/public/files

I named this script “external\_files” and placed it in the /bin directory so I can type it like a command.

Also again, you must set the script file to be executable with the following command:

chmod +x ~/home/def\_user/bin/cycle\_backups

Now you are ready to move on…

Step #7

The ‘incron’ utility allows you to constantly monitor a directory and then take action when something changes or is added to the directory. In our case we will be monitoring the /drop directory for new inbound database backup files form the primary server and immediately using a script (from Step #6) to handle all of the files management

From the default linux user account use the following command to install ‘incron’ utility:

sudo apt-get install incron

This will install the incron utility and allow us to use incrontab to run you script later. However, we are not done with ‘incron’ yet. The thing that makes the incron utility even better than the regular ‘cron’ utility is that it requires someone with ‘sudo’ access to tell the utility exactly who is authorized to use incrontab. So, to complete the installation and setup for incron, you need to run the following command:

sudo nano /etc/incron.allow

When the editor opens up, be sure to enter the default linux user account that you have been using everywhere else ( in our example that is def\_user ). Then close the editor saving the updated file. If you also wanted the root user to be able to use incrontab then you would add a second line to the file for root.

Step #8

Use the following command to open the incrontab editor:

incrontab –e

Add to following line in the editor and save it:

/home/def\_user/drop/ IN\_CLOSE\_WRITE /home/def\_user/bin/cycle\_backups

Everything is now in place to keep the same set of database backup files on both servers and we are even automatically setting up the most recent files on the standby server for you to easily be able to restore the right file on your first try. When the primary server suffers some major failure, you are already stressed enough and choosing the right file to restore from a list of available files may be a source of further problems if you accidently pick the wrong file. This is why the /local\_ready and /local\_restore is in place.

You can create other scripts for yourself if you want to further simplify the process. One example was the short one I added to un-tar the support files to their correct locations. However you want to do it, you need to have an unzipped database backup file so you can run the bench command to restore to the standby server.

BTW… here is the best command to use for running the restore (from ~/frappe-bench):

sudo bench --force --site Your.Site.Name restore ~/local\_restore/YourUnzippedFile.sql

Using the above command reduces the chances of getting errors and forces the restore even if the target server is not the same as the source server.

Note: In case you are not familiar with how to create executable script files, the following command should be used after you create the file in order for it to be executable.

chmod +755 ~/bin/script\_file\_name

For some this might also work:

chmod +x ~/bin/script\_file\_name

Conclusion:

I know this is maybe not the best way to have a backup system, but it IS a method that even most novices can understand and actually maintain without a great deal of linux or developer knowledge. In my case I set my backups to be every hour during the busy part of the workday. This means that the worst case failure for me would be that the users have to repeat up to an hour of their past work in order to catch up after they login to the standby server.

I also have intentionally NOT made everything too automated. I want to make sure I am asking the users questions before I restore something to the standby system. I want to make sure the file I use is the best file for their current needs.

Recently, I had a primary server failure due to a DNS and DoS attack. The most recent backup on the standby server was only 9 minutes old. It took me a total of 23 minutes to question the users, determine the right path forward, restore the database to the standby server, and get them back to running on the standby server. Total down time was 32 minutes and they only had to repeat 9 minutes of their past work to get caught up. But that could have been as much as one hour due to how I have my Poor Man’s Backup System configured. You need to determine your own comfort level and adjust everything to fit your needs. However, I would not recommend running this process any more frequent than every 15 minutes. Running backups does put a strain on the system when it is heavily used. You will have to balance that yourself.

And finally, I have been asked why I bother to keep so many backup files on hand when you only need the most recent one for a disaster recovery. So there are 2 reasons for this. The first is just in case the disaster interrupts the scheduled backup process, you at least have the one immediately before that to use instead (you know, kind of a Murhpy’s Law protection). The second reason is a bit more insidious when it comes to the cause of the problem.

Over a decade ago, I happened to be a regular user on a system for another company. A disgruntled employee sabotaged the system before she was fired by deleting some important directories and redirecting some regular tasks to perform incorrectly so they would cause further record damage for a while without detection. It was just over 3 business days later before we figured out what had been going wrong with our system. We fortunately had a backup from 5 days prior that we were able to restore to the system. This meant we had 5 days of work to be repeated in order to catch up. However, we DID recover, and that was an important lesson learned. That is the second reason I keep so many backups on hand. The space the backups use is NOTHING compared to the possible tragic loss of business if those files didn’t exist at all.

Again, you will have to determine your comfort level for this.

Bonus Information

In order to keep even more restore opportunities available to my system, I have added an additional set of files for those “just-in-case” times. I keep 6 days of the most recent midnight backups rotating constantly on the standby server. To do this I added a /recent directory and created a regular crontab entry (not incron) on the server that looks like this:

15 0 \* \* 0,2-6 /home/def\_user/bin/cycle\_files

And I have another script file called “cycle\_files” that takes care of keeping those 6 days of files.

#!/bin/bash

#

# This script will cycle the most recent file in the /current directory to the /recent directory

# and then purge /recent down to the last 6 files. This script is intended to be run by a crontab

# entry at about 12:15am to load the last backup from the previous day into /recent thereby

# creating a 6 day record of the midnight backups.

# First copy most recent file from /current to /recent

cp $(ls -t /home/def\_user/current/\* | head -1) /home/def\_user/recent

find /home/def\_user/recent -maxdepth 1 -type f -name "\*.sql.gz" -print0 | xargs -r0 ls -t | tail -n +7 | tr '\n' '\0' | xargs $

# This should complete the rotation of the midnight backup files

This is just a starting point for you to design you own Poor Man’s Backup System. Good Luck!

BKM