Best Practices for Disk Based Backup

Using Bacula

This document is intended to provide insight into the considerations and processes required to implement a backup strategy using disks with Bacula.

Bacula Systems White Paper

Version 2.3, November 25, 2014

www.baculasystems.com

Contents

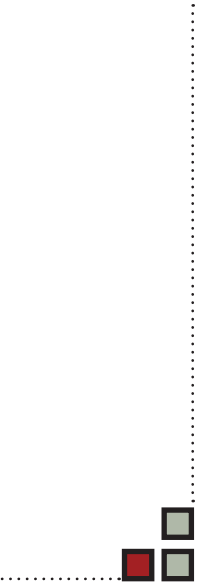
Overview

This white paper presents different ways to use the community version Bacula to backup to disk and will give information on how to configure Bacula properly in each case. As you will see, Bacula has several different ways to do disk backup, some of them are easy to implement but not heavily tested, others are more complex (requires scripting), but are more widely used.

This document has been adapted from the Bacula Enterprise document and is made available for your personal use in the hopes that it will allow community users to make better use of Bacula.

Scope

This paper will present solutions for Bacula 7.0 and later, which are not applicable to prior versions.



Volume Management

When using disk based backup, you should consider using the following options in your configuration:

1.1 Limit Volume Usage

By default, Bacula doesn't have a limit on the amount of data it will write to a Volume, so if you do not explicitly specify some limitation, Bacula will continue to write to your first volume until your disk drive fills, which is not very desirable, as it effectively prevents **Bacula** from re-using volumes close to the time the data on them is out of the configured retention times.

Fortunately, Bacula has a number of options to resolve this potential problem. For example you may set one of the following:

- Volume Use Duration
- Maximum Volume Jobs
- Maximum Volume Bytes
- ...

Setting `Volume Use Duration` to 12 or 23 hours, and `Maximum Volume Bytes` to between 10 and 100GB is probably a reasonable choice for most sites, as in the example below:

```
Pool {
  Name = Default
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes                    # Prune expired volumes
  Volume Retention = 365d            # one year
  Label Format = Vol                 # Allow to auto label
  Action On Purge = Truncate         # Allow to truncate volume
  Volume Use Duration = 14h          # Force volume switch
  Maximum Volume Bytes = 50GB        # Limit volume size
  Maximum Volumes = 50               # allow 2.5TB for this Pool
}
```

1.2 Pruning Volumes

Since automatic catalog pruning can be time consuming, for maximum performance, you may want to avoid automatic pruning of old jobs at the end of each backup Job. To do that, set the directive `AutoPrune = No` in all Client resources, and Bacula will not automatically try to prune jobs. However, unless you want your catalog to grow very large, you will need to schedule a RunScript to explicitly handle the pruning. described in the following section.

1.2.1 New Prune "Expired" Volume Command

It is now possible to prune all volumes (from a pool, or globally) that are "expired". This option can be scheduled after or before the backup of the Catalog and can be combined with the Truncate On Purge option.

This new Expired option to the Prune commands can be scheduled in a RunScript to explicitly handle the pruning. The following two variations are accepted:

```
* prune expired volumes
* prune expired volumes pool=FullPool
```

To schedule this option automatically, it can be added as a Runscript to the BackupCatalog job definition.

```
Job {
  Name = CatalogBackup
  ...
  RunScript {
    Console = "prune expired volume yes"
    RunsWhen = Before
  }
}
```

1.2.2 Manual Pruning

In older Bacula versions prior to the Expired option on the Prune command, manual pruning was done via a Perl script. This method is now deprecated, but shown here for historical reasons:

```
Job {
   Name = "PRUNING"
   Type = Admin
   JobDefs = JobDefault

   RunBeforeJob = "manual_prune.pl --doprune --expired"
}
```

Note, the file manual_prune.pl is available on the bacula.org web site with this white paper.

1.3 Enable Attribute Spooling

By default, when using disk backup, Attribute Spooling is disabled to save space on the Storage Daemon side. This is important to turn this feature ON for all your jobs.

```
Job {
  ...
  Spool Attribute = Yes
}
```

1.4 Truncate Volume on Purge

By default, when Bacula purges a volume, in addition to removing the old Job and File catalog records, it just changes the volume status in the catalog to Purged. It doesn't physically touch your Volume or its data. Thus you can still recover data

after the retention period has expired, but the Volume will continue to occupy disk space. This is generally what one wants providing one has sufficient disks space. For tape based backup, this default behavior is also very useful because you may want to avoid extra mount operations to just truncate volumes.

However, by truncating the Volume after it is purged, you can recover the disk space allocated. There are two steps: 1. You will need to add the directive `Action On Purge = Truncate` to all your Pool resources; 2. You will need to schedule a console RunScript to execute the truncate (special option on the purge command) during a period when Bacula is not backing up files. (Note: if you add this directive, you may need to run an `update` command to apply Pool's configuration changes to the Volumes already created in the catalog).
Below, we show an example of executing the truncate in the Job that runs the nightly catalog backup.

```
Job {
 Name = CatalogBackup
 ...
 RunScript {
   RunsWhen=After
   RunsOnClient=No
   Console = "purge volume action=truncate allpools storage=File"
 }
}
```

1.5 Creating New Volumes

In disk based backup configurations, you may want to let Bacula create new volumes when needed. For that, you can add the `Label Format` option to all your Pool resources, and configure your Storage daemon (SD) devices to label new volumes automatically (`Label Media = Yes`). When using these directives, you may also want to limit the number of Volumes that can be created with the `Maximum Volumes` Pool option (not mandatory if you schedule a pruning job).

1.6 Volume Fragmentation

By default, Bacula is able to share a device between multiple concurrent jobs by allowing them to write in parallel. This causes the Volume to have interleaved Job blocks as shown in the picture on the left below. The drawback is that the restore process could be a bit slower, because the Storage Daemon will have to read more chunks of data and must seek between each block. However, this basic configuration should be suitable for 5 to 10 concurrent Jobs per Volume unless you have particularly strict restore service level agreements.
Using Data Spooling allows writing large blocks of data (shown in the picture on the right above) and is particularly advised for tape based backup. It will increase tape drive throughput by avoiding shoe-shining (tape stop and start), and it makes restores faster. When using disk as a storage target, the extra cost of using data spooling makes little sense and is not easy to setup properly. Using a RAM staging area with a spool size of 20 100MB could be a solution for this problem, but it doesn't scale very well, so for disk storage, we recommend not to turn on Data Spooling (off by default).

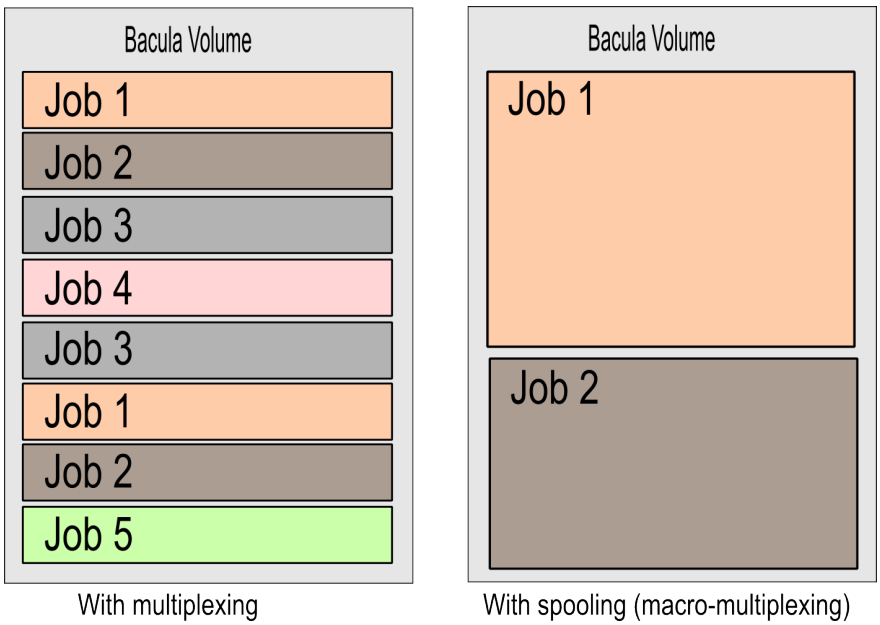


Figure 1.1: Impact of volume fragmentation

1.7 Writing to Multiple Disks

Bacula can, of course, use multiple disks, but in general, each disk must be a separate Device specification in the Storage daemon's configuration file, and you must then select which clients to backup to each disk. You will also want to give each Device specification a different Media Type so that during a restore, Bacula will be able to find the appropriate drive.

The situation is a bit more complicated if you want to treat two different physical disk drives (or partitions) logically as a single drive, which Bacula does not directly support. However, it is possible to back up your data to multiple disks as if they were a single drive by symbolic linking the Volumes from the first disk to the second disk.

For example, assume that you have two disks named `/disk1` and `/disk2`. If you then create a standard Storage daemon Device resource for backing up to the first disk, it will look like the following:

```
Device {
  Name = client1
  Media Type = MyFile
  Archive Device = /disk1
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}
```

Since there is no way to get the above Device resource to reference both `/disk1` and `/disk2` we do it by pre-creating Volumes on `/disk2` with the following:

```
$ ln -s /disk2/Disk2-vol001 /disk1/Disk2-vol001
$ ln -s /disk2/Disk2-vol002 /disk1/Disk2-vol002
$ ln -s /disk2/Disk2-vol003 /disk1/Disk2-vol003
...
```

At this point, you can label the Volumes as Volume `Disk2-vol001`, `Disk2-vol002`, ... and Bacula will use them as if they were on /disk1 but actually write the data to `/disk2`. The only minor inconvenience with this method is that you must explicitly name the disks and cannot use automatic labeling unless you arrange to have the labels exactly match the links you have created.

An important thing to know is that you can only have a single Volume mounted at one time on a disk Device resource defined in the Storage daemon's configuration file. You can have multiple concurrent jobs that simultaneously write to the Volume that is being used, but if you want to have multiple concurrent jobs that are writing to separate disks drives (or partitions), you will need to define separate Device resources for each one, exactly as you would do for two different tape drives. There is one fundamental difference, however. The Volumes that you create on the two drives cannot be easily exchanged as they can for a tape drive, because they are physically resident (already mounted in a sense) on the particular drive. As a consequence, you will want to give them different Media Types so that Bacula can distinguish what Device resource to use during a restore. An example would be the following:

```
Device {
  Name = Disk1
  Media Type = File1
  Archive Device = /disk1
  LabelMedia = yes
  Random Access = Yes
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
}
Device {
  Name = Disk2
  Media Type = File2
  Archive Device = /disk2
  LabelMedia = yes
  Random Access = Yes
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
}
```

With the above device definitions, you can run two concurrent jobs each writing at the same time, one to `/disk1` and the other to `/disk2`. The fact that you have given them different Media Types will allow Bacula to quickly choose the correct Storage resource in the Director when doing a restore.

2

Writing Concurrently to Multiple Devices

We have seen that disk backup can use multiplexing with multiple concurrent jobs without problems, we will now study how to configure Bacula to spread the work load over multiple devices automatically.

You will need to define X Device resources in your Storage Daemon configuration file, where $X = \textit{Storage-MaximumConcurrentJobs}/5$. Of course, you can configure your system to have $X = \textit{Storage-MaximumConcurrentJobs}$, but you need to know that if your volume won't suffer from fragmentation, your filesystem can't always do miracles, and the underlying block structure will be fragmented.

New filesystems like ext4, zfs or btrfs should handle this situation quite well, but finding the right configuration will need some testing from your side.

2.1 Grouping Storage Devices

This section will illustrate how to group multiple devices of a Storage daemon in a single Storage resource of the Director's configuration file. This method permits to simplify your configuration by using the same Storage resource for all your jobs.

In the Storage daemon configuration file.

```
Device {
  Name = FileStorage1
  mediatype = FileMedia
  Archive Device = /disk
  LabelMedia = yes           # lets Bacula label unlabeled media
  Random Access = Yes
  AutomaticMount = yes       # when device opened, read it
  RemovableMedia = no
  AlwaysOpen = no
}
Device {
  Name = FileStorage2
  mediatype = FileMedia
  Archive Device = /disk
  LabelMedia = yes           # lets Bacula label unlabeled media
  Random Access = Yes
  AutomaticMount = yes       # when device opened, read it
  RemovableMedia = no
  AlwaysOpen = no
}
```

In the Director configuration file, you must define a Storage resource that lists all devices.

```
Storage {
  Name = StorageGroup
  Address = bacula-sd
  Password = "xxx"
  Device = FileStorage1
```

```
  Device = FileStorage2
  Device = FileStorage3
# ...
  Media Type = FileMedia
  Maximum Concurrent Jobs = 100
}
```

Important Note

Some operations like relabel, truncate on purge, mount, unmount won't work very well in this configuration. Bacula will always try to do the operation on the first device of the group. For mount/unmount, it should not be a problem because disk based backup doesn't use them. For relabel or truncate on purge action, you should be able to schedule them outside your backup window.

Using a Virtual autochanger permits you to specify the drive index in most commands, so when the first drive is busy, the operator can easily continue to work.

2.2 Using a Virtual Autochanger

2.2.1 Device Definition

```
Device {
  Name = vDrive-1
  Device Type = File
  Media Type = MyDisk
  Archive Device = /disk
  AutomaticMount = yes           # when device opened, read it
  AlwaysOpen = yes
  RemovableMedia = yes

  Autochanger = yes
  Drive Index = 0
  Maximum Concurrent Jobs = 5    # Allow 5 jobs in //
  Volume Poll Interval = 15
  Label Media = yes
}

Device {
  Name = vDrive-2
...
  Drive Index = 1
  Maximum Concurrent Jobs = 100    # This last backup device should take all
}                                  # backup job

Device {
  Name = vDrive-3                  # This last device will be used for Restore
...
  Drive Index = 2
}
```

Each Device should point to the same directory, have the same MediaType, and must have a different `Drive Index`. That means also that if two Devices point to different directories, they should have different MediaType.

Setting `Maximum Concurrent Jobs` will limit the number of jobs that will be able to use this Device at the same time. By default, when a Device is full, Bacula will use automatically the next Device available.

To be able to restore a file concurrently with backups, the last Backup device should have a large "Maximum Concurrent Jobs" setting, and the last device will be used automatically for Restore.

2.2.2 Virtual Autochanger Definition

Once your devices are defined, you need to add in the storage daemon configuration file a Virtual Autochanger to group them together.

```
Autochanger {
  Name = Virtual
  Changer Device = /dev/null
  Changer Command = /dev/null
  Device = vDrive-1, vDrive-2, vDrive-3
}
```

When using Bacula 5.2 and earlier version, the `Changer Command` should be configured as:

```
 Changer Command = ""        # For Bacula 5.2
```

2.2.3 Director Configuration

To finish the configuration, we just need to add a Storage definition that points to the Virtual autochanger in the Director configuration file.

```
Storage {
  Name = FileStorage
  Address = bacula-sd
  Password = "xxxx"    # password for Storage daemon
  Device = Virtual     # must be same as Device in Storage daemon
  Media Type = MyDisk  # must be same as MediaType in Storage daemon
  Maximum Concurrent Jobs = 100
  Autochanger = yes    # Note: this is important for selecting drives
}
```

2.3 Using disk-changer Autochanger

For testing purposes, Bacula allows you to use the disk-changer script to simulate a real autochanger. Bacula will transfer, load or unload media, list the content of the virtual changer like in a real one. This solution could be useful when using some external storage like Amazon S3. This changer script was written principally for testing. One disadvantage of using it is that the filename of the Volumes (slot1-slotxx) is not the same as the Volume name you assign, and this can be confusing to humans. As a consequence, we do not particularly recommend using this script, though many people do use it in production.

3

Distribute Jobs over Devices

We will see in this section different techniques to distribute jobs over multiple devices. (Grouped or using a Virtual autochanger)

3.1 Using Pools

A simple way to use multiple devices at the same time is to define different pools and distribute your jobs among them. Those Pools can have the same attributes (like retention), their main purpose will be to let the Storage Daemon write concurrently to multiple devices. Depending on how many jobs you run at the same time (Max Concurrent Jobs), this technique could be a bit complex to setup.

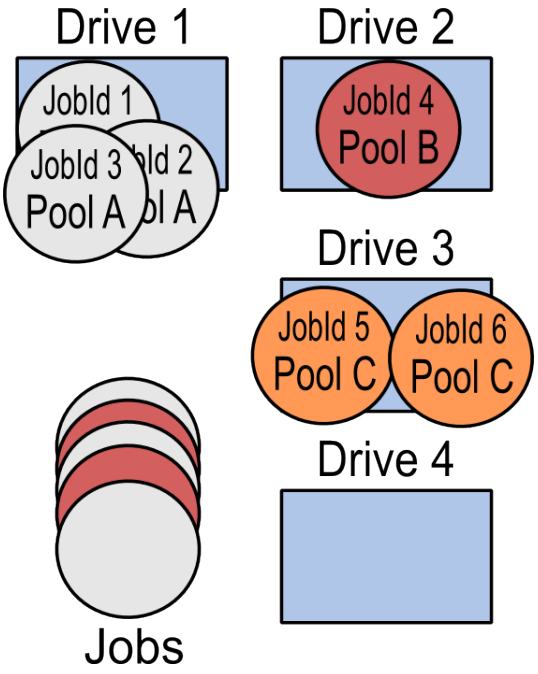


Figure 3.1: Distribute jobs using Pools

3.2 Using "Maximum Concurrent Jobs" on Devices

This option permits you to limit the number of jobs that will use a given Device, when allocating a device to a Job, the Storage Daemon will ensure that a device doesn't run more than `Maximum Concurrent Jobs` at the same time. When this limit is reached, the Storage Daemon will use the next available Device.

```
Device {
  Name = FileStorage2
  mediatype = FileMedia
  Archive Device = /disk
  LabelMedia = yes           # lets Bacula label unlabeled media
  Random Access = Yes
  AutomaticMount = yes       # when device opened, read it
  RemovableMedia = no
  AlwaysOpen = no

  Maximum Concurrent Jobs = 3
}
```

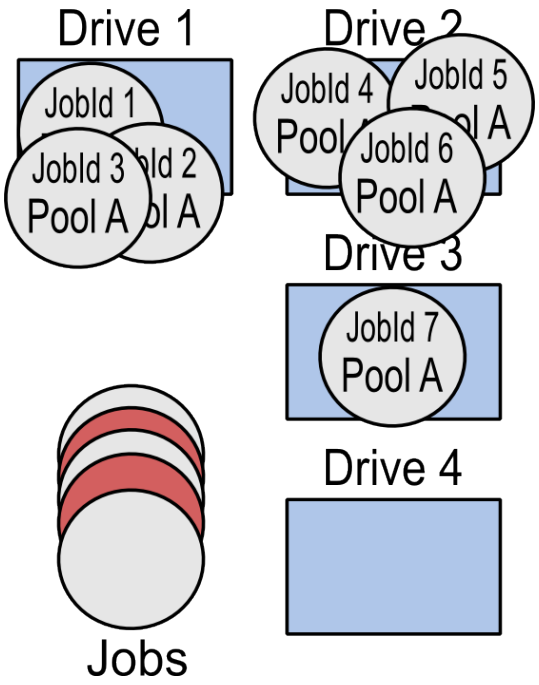


Figure 3.2: Distribute jobs using Maximum Concurrent Jobs on Device

This method has no major drawbacks and is simple to use.

3.3 Using "Prefer Mounted Volume"

The Job parameter `Prefer Mounted Volume = No` in the Director's configuration file permits automatic distribution of your jobs over all your devices. A new Job will always prefer to use a new volume on a free device. When all devices are busy, new jobs will use the least busy compatible device of the group.

```
Job {
  Name = "NightlySave"
  Type = Backup
  Client = bacula-fd
  FileSet = "Full Set"
```

```
  Storage = StorageGroup
  Pool = Default
  Prefer Mounted Volume = No
}
```

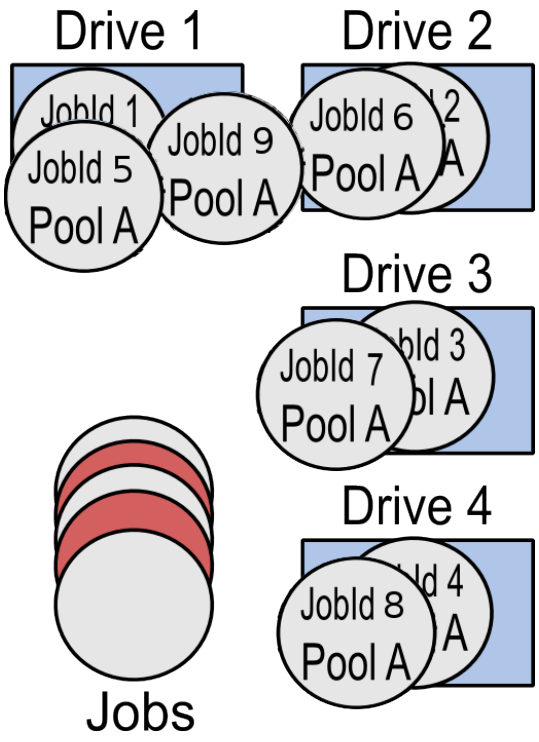


Figure 3.3: Distribute jobs using "Prefer Mounted Volume" on Device

As this option will try to spread Jobs over all devices, if you are using multiple Pools in your backup strategy, all devices can be busy writing to Pool A, and other Jobs that want to use something else will wait for a free slot. A workaround to this is to define a number of jobs equivalent to your maximum concurrent jobs.

Bacula Systems doesn't recommend to use this method, the `Prefer Mounted Volume` option may not work correctly in all situations. Using `Maximum Concurrent Jobs` option is a better choice to restrict the concurrency on a given Device.

3.4 Using a Dedicated Storage Device for each Job

An other very simple solution is to allow each job to use its own Pool and its own Device. This is a bit hard to manage by hand, but if you are generating your configuration by script, it's rather easy to do.

Please note that the above solution will work quite well for small to medium size organizations, but if you have thousands of Jobs to run, you will be pushing Bacula past its design limits since the overhead could drastically increase as the Pools and the number of Devices increase. If you have a very large number of jobs per night

(more than 500), you probably should read our White Paper entitled Disk Backup Design, which can describes some more advanced backup techniques.

3.5 Allowing Restore and Backups to Run Concurrently

In order to be able to restore a file at any time, you may need to define a special restore Device that will be used only for restore purpose. This is to ensure that even during a heavy backup load, there will be at least one Device that is always available for restores.

```
Device {
  Name = Restore
  Device Type = File
  Media Type = MyDisk
  Archive Device = /disk
  AutomaticMount = yes           # when device opened, read it
  AlwaysOpen = yes
  RemovableMedia = yes
}
```

On the Director side, you will define the Storage resource as:

```
Storage {
  Name = RestoreStorage
  Address = bacula-sd
  Password = "xxxx"    # password for Storage daemon
  Device = Restore     # must be same as Device in Storage daemon
  Media Type = MyDisk  # must be same as MediaType in Storage daemon
}
```

During the restore process, Bacula will choose the original Storage resource that was used to backup your files to restore your files, and if all devices are busy, you will be able to manually select the RestoreStorage and thus complete the restore.

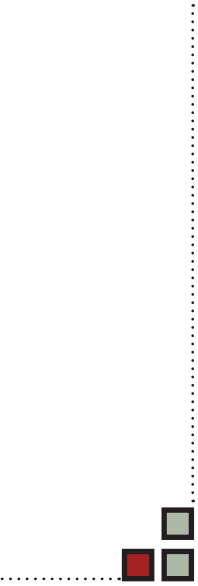
If you are using Virtual autochanger or group of devices, you can add this Device to the Autochanger device group.

4

Conclusions

To summarize, Bacula Systems advises you to :

- Properly configure your Pools with limits on the usage duration and the maximum size of your volumes
- To configure the truncate operation for purged volumes
- To turn off AutoPrune, and prune your volumes regularly
- To label your volumes automatically with AutoLabel
- To configure multiple devices in the Storage daemon and group them in a Virtual Autochanger resource in the Director's configuration file
- To configure extra Device dedicated for Restore in the Virtual Autochanger.
- Use different MediaType for each physical location
- To limit the Maximum Concurrent Jobs for each device
- To test your disk system to find the best global Maximum Concurrent Jobs for the Storage daemon
- Turn on attribute spooling on all your jobs (Spool Attribute=Yes)
- To use xfs with barrier=0 over traditional filesystems such as ext3
- To use new filesystems like ext4, zfs or btrfs, xfs, specially if your storage array is larger than 8TB
- If possible, to mirror your volumes to another place over the network, or to copy important volumes to tape



Revision History

Version	Date	Owner	Changes
1.0	9 November 2010	Eric	Initial creation
1.7	11 Sept 2013	Eric	Updates
1.8	12 Sept 2013	Kern	Pretty up
1.9	01 Jul 2014	Philippe	Pretty up
2.0	18 Sep 2014	Kern	Adapt to community version
2.1	18 Sep 2014	Kern	Tweak
2.2	25 Nov 2014	Kern	Tweak