Bacula Infrastructure Recovery

Bacula Systems Documentation

Contents

1 Disaster Recovery **2**
- 1.1 Disaster Recovery Preparation . . . 2
- 1.2 Storage Considerations . . . 3
- 1.3 Mirroring Data between Data Centers . . . 4
- 1.4 Disaster Recovery Plan . . . 4
- 1.5 Standard Recovery Solution . . . 8

2 High Availability **12**
- 2.1 Solution Comparison . . . 12
- 2.2 High Availability Clustering Solution . . . 13

3 Bootstrap File **16**
- 3.1 Bootstrap File Format . . . 16
- 3.2 Automatic Generation of Bootstrap Files . . . 21
- 3.3 Bootstrap for bscan . . . 21
- 3.4 Final Bootstrap Example . . . 22

Contents

The following chapter explains the measures to setup within your Bacula infrastructure, so that in case of disaster recovery or attacked, you could rebuild as quickly as possible a new Bacula infrastructure and start to restore your data. It presents how to prepare for and recover your Bacula installation in a disaster situation. It also presents different strategies to limit the downtime of your backup service after a major outage.

Obviously, the Security Chapter is key to consider in order to protect your data and the Bacula infrastructure.

1 Disaster Recovery

When disaster strikes, you must have a plan, and you must have prepared in advance otherwise the work of recovering your system and your files will be considerably greater. For example, if you have not previously saved the partitioning information for your hard disk, how can you properly rebuild it if the disk must be replaced?

1.1 Disaster Recovery Preparation

- Ensure that you always have a valid bootstrap file for your backup and your Catalog backup and that it is saved to an alternate machine. This will permit you to easily do a full restore of your system.
- Ensure that your Catalog is saved everyday. You can also add Bacula configuration files to the `BackupCatalog` default Fileset.
- If possible copy your Catalog nightly to an alternate machine. If you have a valid bootstrap file, this is not necessary, but can be very useful if you do not want to reload everything.

- Make a copy of your Bacula `.conf` files, particularly your `bacula-dir.conf`, and your `bacula-sd.conf` files, because if your server goes down, these files will be needed to get it back up and running, and they can be difficult to rebuild from memory. Using a source version control system such as **git** or **svn** with a remote repository on your configuration and scripts directories can help you to track configuration changes and do remote backup after each modifications.
- Perform tests using the Rescue USB or CDROM before you are forced to use it in an emergency situation.
- Ensure that you have captured the disk partitioning layout on all servers to be protected. The **Bacula Systems** USB key for Linux and for Windows automates this process.

1.2 Storage Considerations

If the building which houses your computers burns down or is otherwise destroyed, do you have off-site backup data? Having off-site storage is rather easy with tapes, you can store them in a vault or in a bank safety deposit box at regular interval. With big disk array it's not so easy, you should handle them with care and you can't move them easily. You can cross your backups between sites if you have more than one data-center, or use long distance replication over some Storage Cloud provider. Depending on the amount of data, the long distance replication could be rather expensive. **Incremental forever** techniques can help you to reduce network bandwidth, but you should also think about how fast *all* your data can be restored. Some Cloud Storage providers let you access to their data center in emergency case, and fill your hard drives directly to the source.

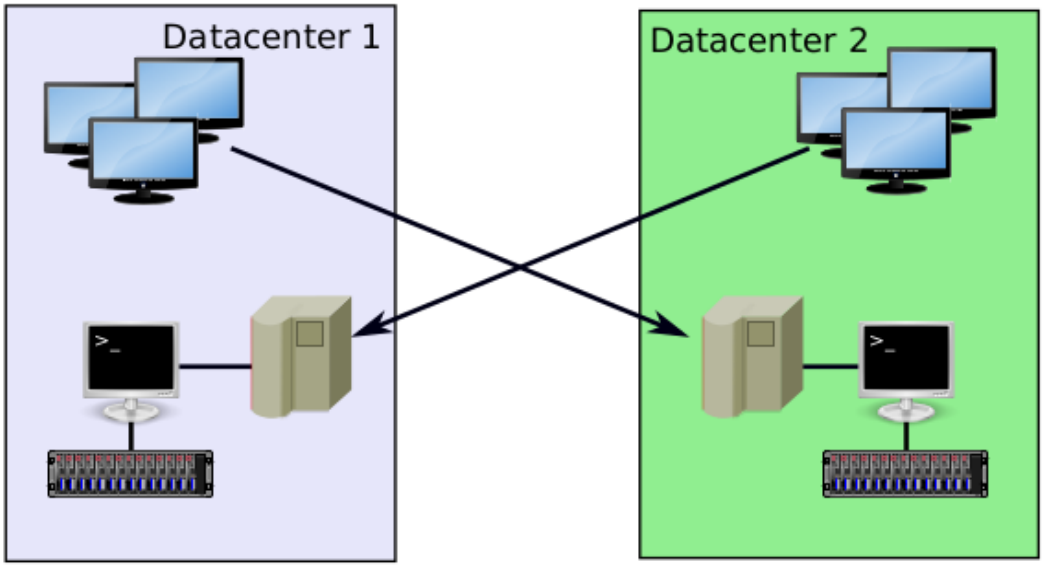

Fig. 1: Cross backup with multiple data center

Sometimes, to make profitable a big investment on expensive disk array infrastructure, users would like to store their Bacula volumes on the same hardware. Because everything is redundant, with multiple FC switches, multiple links, spare disks, etc., you may think that your data is safe, but if something goes wrong with your new hardware, Murphy's law may mean that you will lose your production data and your backups at the same time.

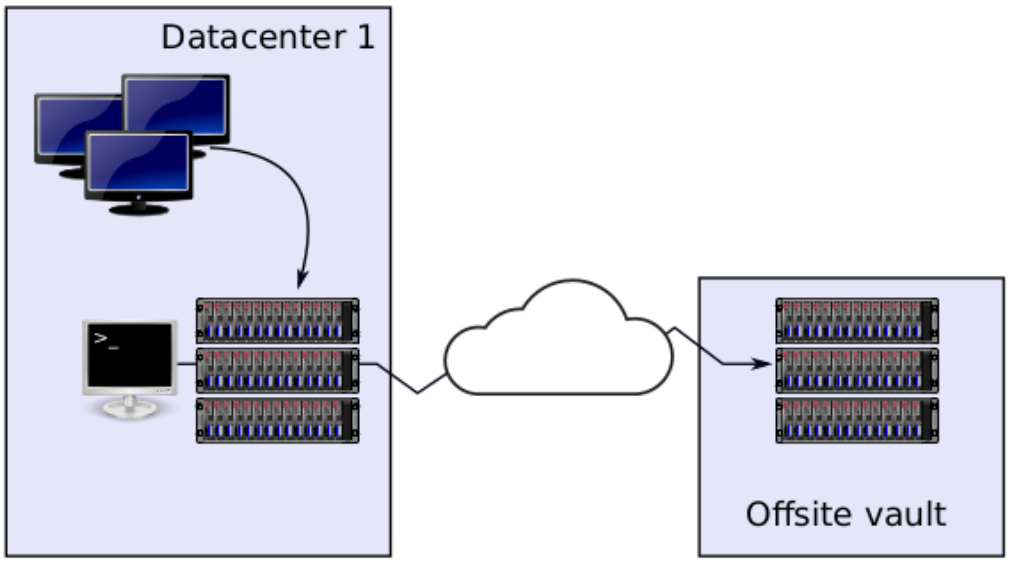

Fig. 2: Using Storage Cloud provider

1.3 Mirroring Data between Data Centers

Recovering the backup system *without* the backup system fully functional is a tricky job. Mirroring your Bacula backup servers between your data centers can avoid lot of stress in critical situations. As you will see, running the Director and the Catalog in *cluster* mode with configuration, binaries and database replication is a powerful way to minimize downtime of your Bacula backup service.

If you decide to mirror also your Bacula Storage Daemons, you should decide how your volumes will be available on both hosts. When dealing with large amount of data, mirroring it can double the cost of your solution. If you decide to make your storage area available for both nodes, it should be coherent with your disaster protection plan. For example, if the building hosting the storage device (tapes or disks) and the primary Storage Daemon burns down, your second Storage Daemon won't be very useful. In this case, you will be protected only for a Storage Daemon server hardware failure.

In this situation, you can decide to mirror only critical Pools.

1.4 Disaster Recovery Plan

Note: Having a full disaster recovery plan is something quite complicated. It must be tailored to exact requirements of each site, and thus can only be created by working closely with the decision makers and operational personnel of that data center. As a consequence, this paper will not provide a specific disaster recovery plan, but will present most of the elements involved in coming up with such a plan.

We specifically do not cover certain critical areas which are mostly "hardware" related such as power, A/C, flood prevention, multiple power sources, UPSes, telecommunications links, multiple telecommunications links, and preventing single point of failure.

Bacula Installation Disaster Recovery Plan

Design a backup strategy in case of disaster recovery (more information is available in the ItemsToImplementBeforeProduction chapter).

- Backup your Catalog and store the (bootstrap) file in a safe location.
- Backup your DIR and SD configuration files (if you are using BWeb Management Suite, please backup the folder).
- Backup the File Daemon private/public key pair and/or the master keys if Data Encryption is used for the Catalog backup.

As a best practice, document your backup design to provide a good overview and enhance the understanding of your backup infrastructure.

Test your tape library infrastructure using the btape utility.

Bacula Configuration and Backups Disaster Recovery Plan

Your Disaster Recovery plan should include Bacula configuration and backups. In order to do so, you can have off-site backups including your configuration.

The needs are simple, the backups (for example the last Full backups of your systems), the content of `/opt/bacula/ect` and the Catalog.

The backups can be copied or migrated using Copy/Migration Jobs from one site to a safe site using schedules.

If you are using tapes, you can remove a set of tapes after the Full backups and store them in an off-site vault.

The most important part of this plan is that the Catalog and the configuration files are kept together.

Example

This is an example which you need to adapt to your own system.

First, set up a job that will backup `/opt/bacula/etc` and your catalog dump.

We would advise to have all Bacula configuration (`/opt/bacula/etc` content) as well as the catalog backup in a dedicated pool to ease the process of recovery as the Job will not be mixed into numerous other Jobs. Catalog and configuration files are all you need for recovering your backup environment. We will split the backup in 2 parts to insure consistency of the Catalog data.

This way we set up a specific pool for our Bacula configuration and Catalog backup, 2 jobs per volume. Of course you can tweak this example to have more files backed up to build your own disaster recovery plan (eg. TLS keys etc.).

Here is an example of a Pool on hard drives backing up a disaster recovery Job

```
Pool {
  Name = "DisasterRecovery-pool"
  ActionOnPurge = Truncate
  AutoPrune = yes
  FileRetention = 7 days # Adjust to the schedule of the Disaster Recovery job
                         # here it is kept 7 days before recycling
  JobRetention = 7 days  # This is the minimum we want to keep our DR data␣
```

(continues on next page)

(continued from previous page)

```
↪safe
  LabelFormat = "Disaster-"
  PoolType = "Backup"
  Recycle = yes
  Storage = "OnDisk"
  MaximumVolumeJobs = 2  # Here we want to keep config files and dump on the␣
↪same
                         # volume. Adjust here if you don't want 2 jobs per␣
↪volume
  VolumeRetention = 8 days
  VolumeUseDuration = 20 hours # How long do you want  your volume available.␣
↪No more
                               # than one day (because you will run on DR␣
↪backup job
                               # per day, but not too short in case something␣
↪goes
                               # wrong during the backup.
                               # Adjust here depending on your policy
  Maximum Volumes = 10  # 7 generations kept, one written, some spare
}
```

Here is an example Job and Fileset for Disaster Recovery backups

```
Fileset {
   Name = "DisasterRecovery-catalog-fs" # Fileset for the Catalog
   Include {
      Options {
         Signature = SHA1
      }
      File = /opt/bacula/working/bacula.sql # where the Bacula catalog dump␣
↪goes
                                            # to be adjusted with your catalog
                                            # dump
   }
}

Fileset {
   Name = "DisasterRecovery-config-fs"  # Fileset for the Bacula Enterprise
                                        # configuration files
   Include {
      Options {
         Signature = SHA1
      }
      File = /opt/bacula/etc  # Director's config files
                              # you can add other files like keys,
                              # content of /etc, and other directories
                              # to make this Fileset more complete and
                              # adapted to your environment
   }
}

Job {
```

(continues on next page)

(continued from previous page)

```
   Name = "DisasterRecovery-catalog-job"
   Type = "Backup"
   Client = "baculaServer"   # change to the name of the fd running on the␣
↪Bacula DIR
   Fileset = "DisasterRecovery-catalog-fs"
   JobDefs = "Default-jd"
   Level = "Full"            # full backup is preferable
   Messages = "Standard"
   Pool = "DisasterRecovery-pool" # the pool we just defined to hold all the␣
↪DR config
   Priority = 20 # Adjust this priority  to be the highest of the schedule
   Runscript {
      Command = "/opt/bacula/scripts/make_catalog_backup bacula bacula"
      RunsOnClient = no
      RunsWhen = Before
   }
   Runscript {
      Command = "/opt/bacula/scripts/delete_catalog_backup"
      RunsOnClient = no
      RunsWhen = After
   }
   Schedule = "NightAfterJobs"
   Storage = "OnDisk"
   WriteBootstrap = "/opt/bacula/bsr/catalog-backup.bsr"
}


Job {
   Name = "DisasterRecovery-config-job"
   Type = "Backup"
   Client = "baculaServer"   # change to the name of the fd running on the␣
↪Bacula DIR
   Fileset = "DisasterRecovery-config-fs"
   JobDefs = "Default-jd"
   Level = "Full"  # full backup is preferable
   Messages = "Standard"
   Pool = "DisasterRecovery-pool" # the pool we just defined to hold your DR␣
↪config
   Priority = 15 # Adjust if necessary. Must be lower than the catalog's one
   Schedule = "NightAfterJobs"
   Storage = "OnDisk"
   WriteBootstrap = "/opt/bacula/bsr/config-backup.bsr"
}
```

Here we set different priority for the 2 jobs to ensure that the job that backs up the catalog runs after the backup of the configuration file.

This way, in case of Disaster, you reinstall the same version of Bacula, then you just need to grab the last volume from the DisasterRecovery-pool and use `bxtract` to extract all its content. The 2 Jobs will contain the catalog and the config.

You need to re-inject the catalog dump in your database, copy the config files to `/opt/bacula/etc/` and everything is back online. (Test with `/opt/bacula/bin/bacula-dir -t -u bacula -g bacula`

before launching the daemons.)

Important: Test your Disaster Recovery procedure and document it.

1.5 Standard Recovery Solution

As Bacula is able to backup and restore any Unix/Linux system files such as char and block devices, hardlinks, symlinks, etc., it's possible to use it directly to do system recovery.

System Recovery Preparation

The safest way to restore a system, is to backup everything. The following Fileset should work on any Linux systems.

```
Fileset {
  Name = "LINUX_SYSTEM"
  Include {
    Options { signature = MD5; onefs = no }
    File = /
  }
  Exclude {
    File = /tmp/
    File = /var/tmp/
    File = /proc
    File = /sys
    File = /opt/bacula/working
    File = /home                  # skip server data
  }
}
```

If you are using the Bare Metal Recovery kit, you will need to run the network and disk analysis tool as `ClientRunBeforeJob` Runscript.

You will probably want to exclude *non-system* data from this Fileset such as `/home` and use an other Fileset that will backup only these files with a more frequent schedule policy. If you have many identical servers (same OS, same version), you can consider to use Bacula Enterprise File Deduplication feature called **BaseJobs**.

```
Fileset {
  Name = "SERVER1_DATA"
  Include {
    Options { signature = MD5; onefs = no }
    File = /home
  }
}
```


General GNU/Linux System Recovery

You will take the following steps to get your system back up and running:

1. Boot with your Rescue USB or CDROM on your new or repaired system
2. Start the Network (local network)
3. Re-partition your hard disk(s) as it was before
4. Re-format your partitions
5. Install or start the Bacula File daemon (static version)
6. Perform a Bacula restore of all your files
7. Re-install your boot loader
8. Reboot

For details on recovery, see the DisasterRecovery chapter.

GNU/Linux Bare Metal Recovery

Bacula Systems provides Linux Bare Metal recovery kit that can help you to recover your servers very quickly. This tool is designed to collect and handle Network configuration and disk layout for your systems in an automatic way after each regular Full backup.

1. Boot with your Bare Metal Recovery USB or CDROM on your new or repaired system
2. Start the Network (local network). Do not proceed until your network is up.
3. Re-partition and format your hard disk(s) as it was before using Bare Metal Recovery tools
4. Perform a Bacula restore of all your files using Bare Metal Recovery tools
5. Re-install your boot loader
6. Reboot

For details on Bare Metal Recovery, see the Bare Metal Recovery for Linux chapter.

Windows Bare Metal Recovery

Bacula Systems provides Windows Bare Metal recovery kit that can help you to recover your servers very quickly. You can read more information about this tool in the Bare Metal Recovery for Windows chapter.

Bacula Configuration Recovery

When using Bacula Enterprise packages, everything needed to run and configure Bacula is located under `/opt/bacula`. Installing basic dependencies such as PostgreSQL or MySQL client library (using Bacula Systems RPMs), and restoring this directory on a new server means that you can start a new Bacula very quickly. This technique can be used to restore your Director as explained in the director_recovery section. The following Fileset can be used for this purpose:


```
Fileset {
  Name = "BACULA_DATA"
  Include {
    Options { signature = MD5 }
    File = /opt/bacula
    File = /etc
  }
  Exclude {
    File = /opt/bacula/working
  }
}
```

If you have special Linux configuration such as network parameters or system tuning, restoring `/etc` should cover almost everything.

Director Recovery

If your configuration and binaries are saved on an other backup server (such as a Storage Daemon, or the Catalog server) as advised, you should be able to recover your Director with the following actions:

- Adjust the new binary path in either the `bacula-dir.service` unit file or the `bacula-ctl-dir`` startup script if needed. Usually they are locate in the /opt/bacula/bin directory and the binary path doesn't require any change
- Comment all Job Schedule directives in `bacula-dir.conf`
- Affect the Director IP address to your temporary server
- Ensure that your temporary Director can connect to the Catalog (in `pg_hba.conf` or in mysql database).
- Start the temporary Director Daemon
- Start the Bare Metal or Standard Recovery procedure on your Director
- Stop the temporary Director and un-configure the Director IP address
- Reboot to the fresh Director
- Test your Director by doing one backup and restore of files per Storage Daemon

If your Director and your Catalog are on the same host, you need to restore the Catalog first or use a Bootstrap file as explained here.

Storage Daemon Recovery

Since this white paper is designed for multiple Storage Daemons environment, we advise you to cross Storage Daemon backups. In this case, this is very easy to restore the system using Bare Metal or General Recovery procedure and restore all needed configurations and binaries.

Once your server is up, you should test carefully your storage system as it was the first installation. These tests should include at least:

- Test your tape drive for compatibility with Bacula by using the test command in the btape program.
- Do one backup and restore of files.

Catalog Recovery

Using Warm Standby or Log Shipping

NTT has developed a shared-nothing replication system for PostgreSQL implemented with transaction log shipping. The goal is to minimize the system downtime and the impact for update performance. Failover can be done within 15 seconds and the overhead is at worst 7% on heavily updated workloads in the current implementation.

Fig. 3: PostgreSQL catalog hot-standby

Using this technology, you are able to always have a valid copy of your Catalog across the network. If something goes wrong with your Catalog server, you just have to activate the PostgreSQL master mode on the backup node, change the virtual IP address and restart the Director.

The detail of those procedures are available on https://www.postgresql.org/docs/current/high-availability.html

Using Catalog Backup

If you have backed up your database nightly (as you should) and you have made a bootstrap file, you can rapidly restore your database (or the ASCII SQL output). Make a copy of your current database, re-initialize it, then you should be able to run Bacula. If you now try to use the restore command, it will not work because the database will be empty. However, you can manually run a restore job and specify your bootstrap file. Once the database is restored, you can start the database import process. When restoring from the ASCII SQL file, depending of the Catalog size, it can take several hours to complete. Restoration can be done much faster if you use binary backups of the Catalog.

Note that it's also possible to recover your Catalog backup with the backup Job output, or by scanning volumes. All these procedures are completely described in The Restore Command chapter of the manual.

2 High Availability

Bacula has several different ways to perform *High Availability*, some of them are easy to implement but not fully automatic, others are more complex, but can reduce the downtime to a couple of seconds.

High Availability solutions are described for GNU/RHEL systems, database High Availability solutions are described for PostgreSQL.

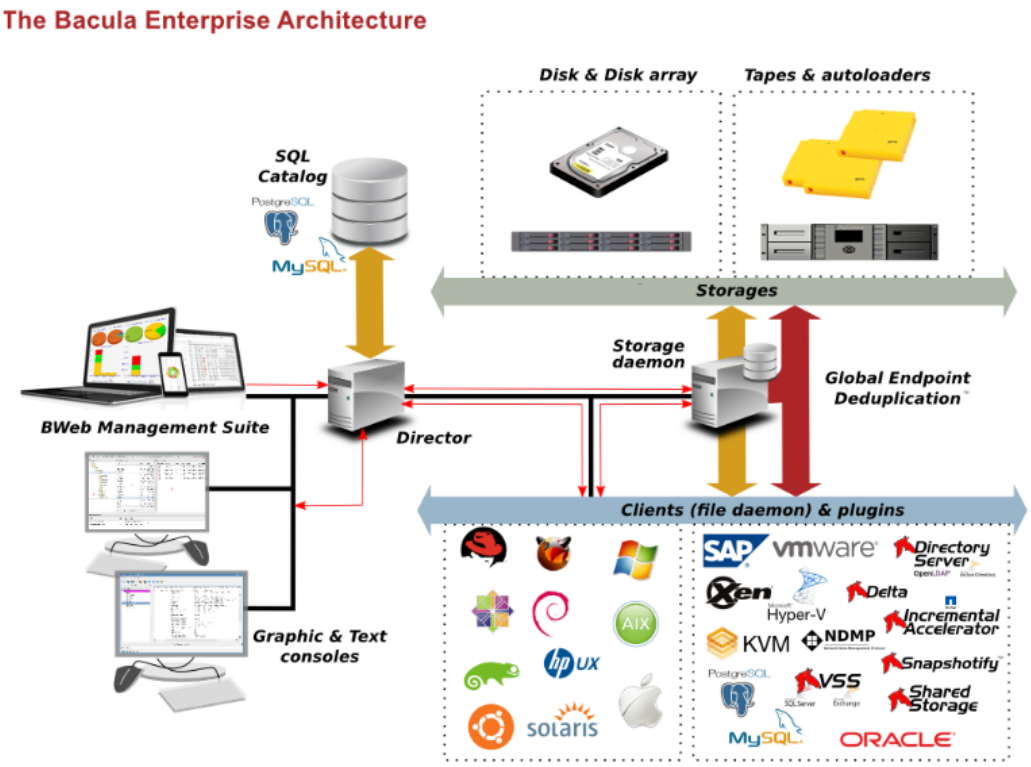

Fig. 4: Bacula Components

2.1 Solution Comparison

Max. down-time	Solution	Notes
< 5 mins	High Availability cluster and database block level replication	Need to be experienced with clustering technologies such as HACMP, HeartBeat- /Pacemaker, etc.
< 1-3 hours	Spare hardware and database replication	Need a clear procedure to restore Bacula and use PostgreSQL internal replication.
< 12 hours	Spare hardware and database restore	Need a clear procedure to restore Bacula and you can restore PostgreSQL catalog from the last backup.

If you loose your Catalog server, all records about jobs that ran after your previous Catalog backup will be lost. Keeping trace of emails and Bootstrap files is sufficient to restore files, but it's not very

convenient. To avoid this problem, you can use the PostgreSQL Continuous Archiving option and do binary Catalog backups instead of the default SQL dump procedure. See https://www.postgresql.org/docs/current/continuous-archiving.html for more information.

2.2 High Availability Clustering Solution

This solution provides a high-end solution for Bacula. If you are not experienced with these technologies, it can represent important training costs. Your needs should drive your decisions.

Using spare hardware, Bacula can be integrated with standard OpenSource Linux clustering solutions such as Heartbeat or Pacemaker from http://www.linux-ha.org

In the event of a failure, resource managers like **Pacemaker** or **Heartbeat** will automatically initiate recovery and make sure your application is available from one of the remaining machines in the cluster. Pacemaker is the new version of Heartbeat, it permits handling very complex cluster setups. With Bacula, this level of complexity is not needed so we advise you to run in a simple Primary/Slave situation, the rest of the document will refer to Heartbeat as the resource manager.

The data replication of the PostgreSQL server can be done with **DRBD** (Data Block Device Replication) tools from LINBIT. (http://linbit.com)

Proposed Architecture

A large site will need to run multiple Storage Daemons per Director (SD1 and SD2 in the schema), and you will probably need a dedicated PostgreSQL Catalog server per Director (SGBD on the schema).

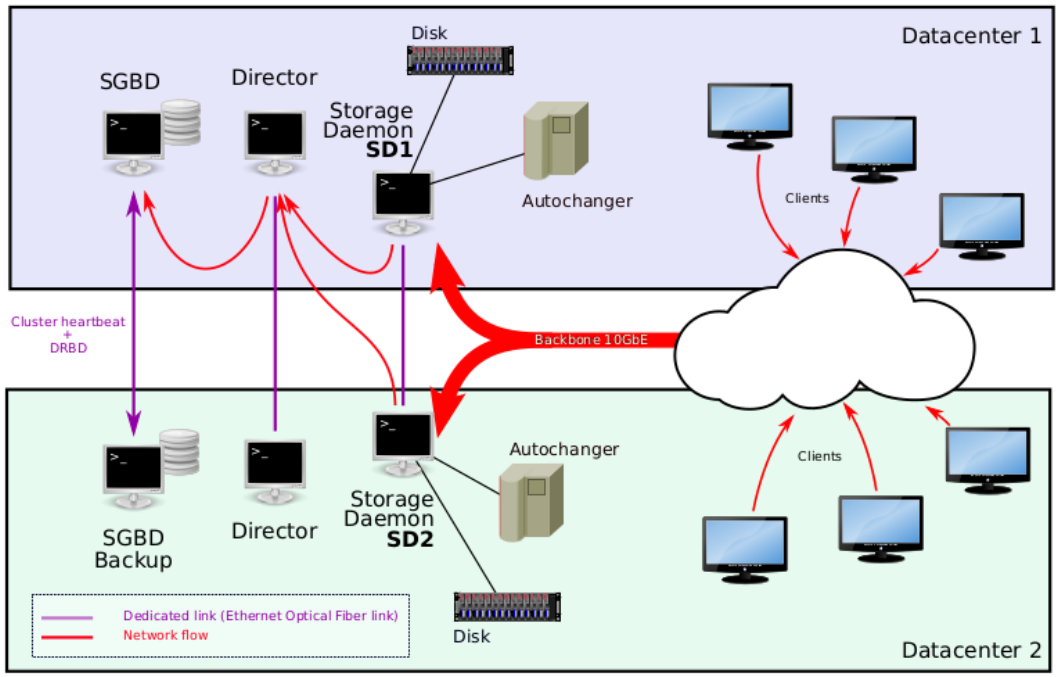

Fig. 5: Using Bacula in a multiple data center environment

All servers should have :

- RAID hardware with WriteBack capabilities

- Multiple Ethernet links aggregated with failover detection using bonding kernel module, these links should be connected to different and independent network equipment
- Hot plug and redundant power supply.

In this architecture, each server that is used for your Bacula installation should be protected by second one located in the other data center. Each couple of cluster nodes should have a dedicated direct Ethernet fiber optic link and may implement a STONITH mechanism. If your Storage Daemons has a single point of failure (because your disks are not mirrored between data centers for example, or you have an Autochanger directly connected), you won't necessarily need to protect them at the same level, and some spare hardware should be sufficient.

Cluster Resources

The role of a resource agent is to abstract the service it provides and present a consistent view to the cluster, which allows the cluster to be agnostic about the resources it manages. The cluster doesn't need to understand how the resource works because it relies on the resource agent to do the right thing when given a start, stop or monitor command.

Typically resource agents come in the form of shell scripts, however they can be written using any technology (such as C, Python or Perl) that the author is comfortable with. With Bacula, the following default resources will be used in the resource manager tool:

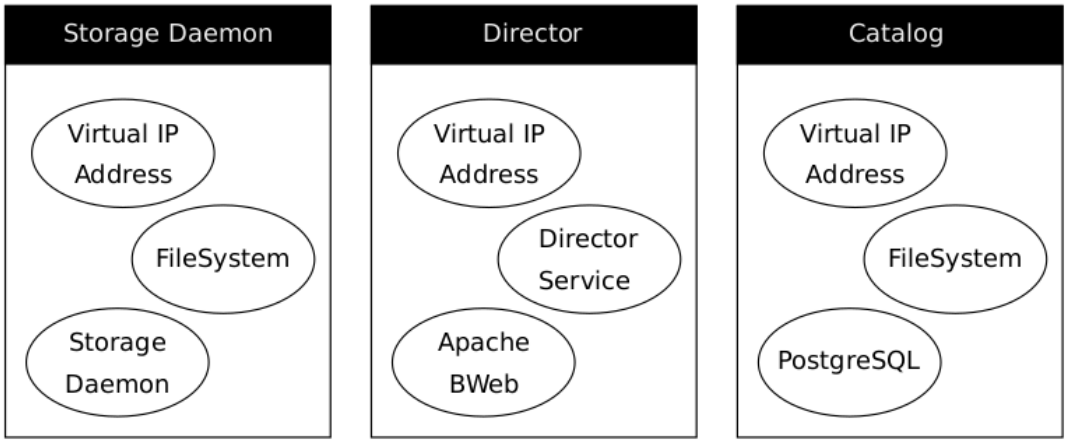

Fig. 6: Pacemaker/Heartbeat service definition

- Virtual IP Addresses
- Bacula Director service
- Bacula Storage services
- Backup storage filesystems
- PostgreSQL catalog service
- PostgreSQL data and configuration filesystem

When using cluster techniques, a very common way to ensure that you can move or restart a service elsewhere on your internal network without having to reconfigure all your clients is to use virtual IP addresses for all your components. All Bacula components should have their own virtual IP address,

the resource manager (Heartbeat) will ensure that only one primary node is using it at a time on your network.

Since Bacula isn't designed to reconnect automatically when a TCP connection drops, running jobs will fail when a resource is moved from one location to another. Make sure that Bacula is stopped before moving services between hosts.

For example, the Director (host bacula-dir1) HeartBeat resource definition will look like:

```
bacula-dir1 IPAddr::10.0.0.2/24 bacula-dir httpd
```

Bacula Configuration Synchronization

In this solution, the resource manager (Heartbeat) will detect if a node or a service has a failure and will restart it at the right place, but it won't ensure that Bacula's configuration is synchronized between nodes. A simple and flexible way to do that is to use **rsync** at regular intervals on the master node and automatically after a reload command.

PostgreSQL Catalog

Protecting your SQL Catalog (PostgreSQL or MySQL) is a very large subject, there are dozens of techniques that accomplish the job.

The PostgreSQL High Availability configuration is the most complex part of this setup, to be able to restart the service on the second node after an outage, data should be replicated between nodes. This replication can use high end hardware, standard PostgreSQL replication or DRBD replication layer (RAID1 over the network, see http://linbit.com)

Clusters using HeartBeat, DRBD and PostgreSQL are very common in the OpenSource world, and it's rather easy to find knowledge and resources about them.

Data Replication

DRBD is OpenSource and has been in development for over 10 years and continues to undergo feature upgrades, it has been officially accepted into the Linux Kernel 2.6.33, it is simple, fast and flexible, it has transaction safe technology; this means that DRBD is designed to replicate data in a reliable, secure and safe method no matter how sensitive your payload is. DRBD has support options: it may be installation assistance, 24/7 support or a single support incident, LINBIT can help.

The performance cost of this block level replication is around 10/15% of the overall disk throughput, but it has the major advantage to be very safe and simple during standby/takeover operations, it's almost impossible to loose your data with bad sequence of commands. Streaming replication using PostgreSQL is faster, but requires taking careful steps before reactivating the replication and being protected again.

Since DRBD is well integrated with HeartBeat (LINBIT is now the official maintainer of HeartBeat resource manager), once volumes are initialized and synchronized, the HeartBeat resource definition on the bacula-sql1 host will look like:

```
bacula-sql1 IPAddr::10.0.0.1/24 drbddisk::postgres \
            Filesystem::/dev/drbd0::/pgdata::ext3 postgresql
```


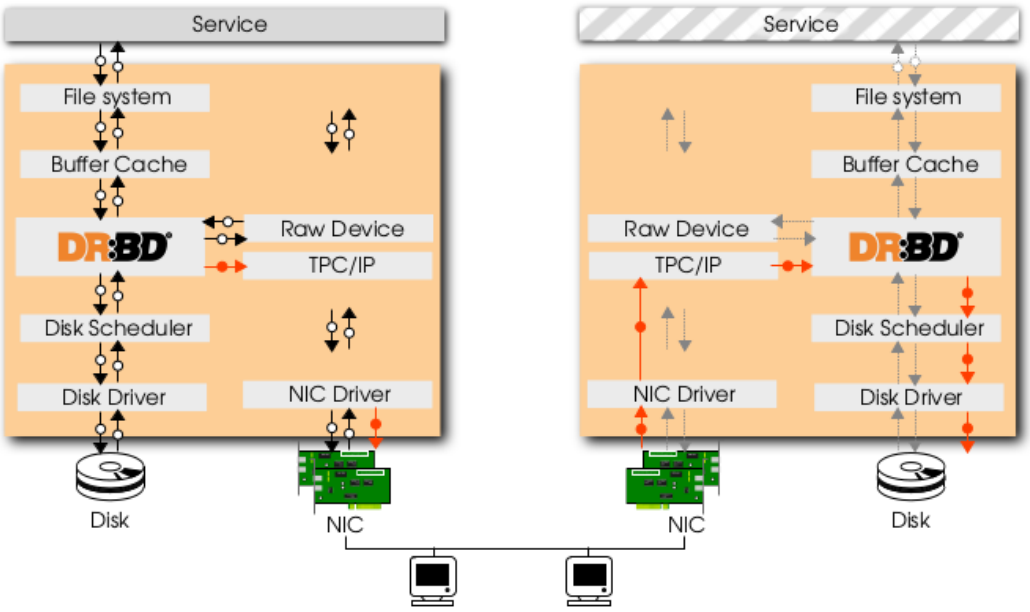

Fig. 7: DRBD architecture

3 Bootstrap File

The information in this chapter is provided so that you may either create your own bootstrap files, or so that you can edit a bootstrap file produced by **Bacula**. However, normally the bootstrap file will be automatically created for you during the Bacula Console or by using a Write Bootstrap record in your Backup Jobs, and thus you will never need to know the details of this file.

The **bootstrap** file contains ASCII information that permits precise specification of what files should be restored, what volume they are on, and where they are on the volume. It is a relatively compact form of specifying the information, is human readable, and can be edited with any text editor.

3.1 Bootstrap File Format

The general format of a **bootstrap** file is:

<keyword>= <value>

Where each **keyword** and the **value** specify which files to restore. More precisely the **keyword** and their **values** serve to limit which files will be restored and thus act as a filter. The absence of a keyword means that all records will be accepted.

Blank lines and lines beginning with a pound sign (#) in the bootstrap file are ignored.

There are keywords which permit filtering by Volume, Client, Job, FileIndex, Session Id, Session Time, etc.

The more keywords that are specified, the more selective the specification of which files to restore will be. In fact, each keyword is **AND**ed with other keywords that may be present.

For example:


```
Volume = Test-001

VolSessionId = 1

VolSessionTime = 108927638
```

directs the Storage daemon (or the **bextract** program) to restore only those files on Volume Test-001 **AND** having VolumeSessionId equal to one **AND** having VolumeSession time equal to 108927638.

The full set of permitted keywords presented in the order in which they are matched against the Volume records are:

Volume The value field specifies what Volume the following commands apply to. Each Volume specification becomes the current Volume, to which all the following commands apply until a new current Volume (if any) is specified. If the Volume name contains spaces, it should be enclosed in quotes. At lease one Volume specification is required.

Count The value is the total number of files that will be restored for this Volume. This allows the Storage daemon to know when to stop reading the Volume. This value is optional.

VolFile The value is a file number, a list of file numbers, or a range of file numbers to match on the current Volume. The file number represents the physical file on the Volume where the data is stored. For a tape volume, this record is used to position to the correct starting file, and once the tape is past the last specified file, reading will stop.

VolBlock The value is a block number, a list of block numbers, or a range of block numbers to match on the current Volume. The block number represents the physical block within the file on the Volume where the data is stored.

VolSessionTime The value specifies a Volume Session Time to be matched from the current volume.

VolSessionId The value specifies a VolSessionId, a list of volume session ids, or a range of volume session ids to be matched from the current Volume. Each VolSessionId and VolSessionTime pair corresponds to a unique Job that is backed up on the Volume.

JobId The value specifies a JobId, list of JobIds, or range of JobIds to be selected from the current Volume. Note, the JobId may not be unique if you have multiple Directors, or if you have reinitialized your database. The JobId filter works only if you do not run multiple simultaneous Jobs. This value is optional and not used by to restore files.

Job The value specifies a Job name or list of Job names to be matched on the current Volume. The Job corresponds to a unique VolSessionId and VolSessionTime pair. However, the Job is perhaps a bit more readable by humans. Standard regular expressions (wildcards) may be used to match Job names. The Job filter works only if you do not run multiple simultaneous jobs. The value specifies a Client name or list of Clients to be matched on the current Volume.

Client The value specifies a Client name or list of Clients to will be matched on the current Volume.

The Client filter works only if you do not run multiple simultaneous Jobs. This value is optional and not used by to restore files.

FileIndex The value specifies a FileIndex, list of FileIndexes, or range of FileIndexes to be selected from the current Volume. Each file (object) stored on a Volume within a Session has a unique FileIndex. For each Session, the first file written is assigned FileIndex equal to one and the value is incremented for each file backed up.

Thus for a given Volume, the triple VolSessionId, VolSessionTime, and FileIndex uniquely identifies a file stored on the Volume. Multiple copies of the same file may be stored on the same Volume, but for

each file, the triple VolSessionId, VolSessionTime, and FileIndex will be unique. This triple is stored in the Catalog database for each file.

To restore a particular file, this value (or a range of FileIndexes) is required.

FileRegex The value is a regular expression. When specified, only matching filenames will be restored.

```
FileRegex=^/etc/passwd(.old)?
```

Slot The value specifies the autochanger slot. There may be only a single **Slot** specification for each Volume.

Stream The value specifies a Stream, a list of Streams, or a range of Streams to be selected from the current Volume. Unless you really know what you are doing (the internals of **Bacula**), you should avoid this specification. This value is optional and not used by Bacula to restore files.

***JobType** Not yet implemented.

***JobLevel** Not yet implemented.

The **Volume** record is a bit special in that it must be the first record. The other keyword records may appear in any order and any number following a Volume record.

Multiple Volume records may be specified in the same bootstrap file, but each one starts a new set of filter criteria for the Volume.

In processing the bootstrap file within the current Volume, each filter specified by a keyword is **AND**ed with the next. Thus,

```
Volume = Test-01

Client = "My machine"

FileIndex = 1
```

will match records on Volume **Test-01 AND** Client records for **My machine AND** FileIndex equal to **one**.

Multiple occurrences of the same record are **OR**ed together. Thus,

```
Volume = Test-01

Client = "My machine"

Client = "Backup machine"

FileIndex = 1
```

will match records on Volume **Test-01 AND** (Client records for **My machine OR Backup machine**) **AND** FileIndex equal to **one**.

For integer values, you may supply a range or a list, and for all other values except Volumes, you may specify a list. A list is equivalent to multiple records of the same keyword. For example,

```
Volume = Test-01

Client = "My machine", "Backup machine"

FileIndex = 1-20, 35
```


will match records on Volume **Test-01 AND (**Client records for **My machine OR Backup machine) AND** (FileIndex 1 **OR** 2 **OR** 3 . . . **OR** 20 **OR** 35).

As previously mentioned, there may be multiple Volume records in a bootstrap file. Each new Volume definition begins a new set of filter conditions that apply to that Volume and will be **OR**ed with any other Volume definitions.

As an example, suppose we query for the current set of tapes to restore all files on Client **Rufus** using the query command in the console program:

```
Using default Catalog name=MySQL DB=bacula
\*query
Available queries:
    1: List Job totals:

    2: List where a file is saved:

    3: List where the most recent copies of a file are saved:

    4: List total files/bytes by Job:

    5: List total files/bytes by Volume:

    6: List last 10 Full Backups for a Client:

    7: List Volumes used by selected JobId:

    8: List Volumes to Restore All Files: Choose a query (1-8): 8

    Enter Client Name: Rufus
```

JobId	StartTime	VolumeName	StartFile	VolSesId	VolSesTime
154	2002-05-30 12:08	test-02	0	1	1022753312
202	2002-06-15 12:08	test-02	0	2	1024128917
203	2002-06-15 12:08	test-02	3	1	1024132350
204	2002-06-18 12:08	test-02	4	1	1024380678

The output shows us that there are four Jobs that must be restored. The first one is a Full backup, and the following three are all Incremental backups.

The following bootstrap file will restore those files:

```
Volume=test-02

VolSessionId=1

VolSessionTime=1022753312

Volume=test-02

VolSessionId=2

VolSessionTime=1024128917
```

(continues on next page)

(continued from previous page)

```
Volume=test-02

VolSessionId=1

VolSessionTime=1024132350

Volume=test-02

VolSessionId=1

VolSessionTime=1024380678
```

As a final example, assume that the initial Full save spanned two Volumes. The output from might look like:

JobId	StartTime	VolumeName	StartFile	VolSesId	VolSesTime
242	2002-06-25 16:50	File0003	0	1	1025016612
242	2002-06-25 16:50	File0004	0	2	1025016612
243	2002-06-25 16:52	File0005	3	1	1025016612
246	2002-06-25 19:19	File0006	4	1	1025025494

and the following bootstrap file would restore those files:

```
Volume=File0003

VolSessionId=1

VolSessionTime=1025016612

Volume=File0004

VolSessionId=1

VolSessionTime=1025016612

Volume=File0005

VolSessionId=2

VolSessionTime=1025016612

Volume=File0006

VolSessionId=2

VolSessionTime=1025025494
```


3.2 Automatic Generation of Bootstrap Files

One thing that is probably worth knowing: the bootstrap files that are generated automatically at the end of the job are not as optimized as those generated by the **restore** command. This is because during Incremental and Differential jobs, the records pertaining to the files written for the Job are appended to the end of the bootstrap file. As consequence, all the files saved to an Incremental or Differential job will be restored first by the Full save, then by any Incremental or Differential saves.

When the bootstrap file is generated for the **restore** command, only one copy (the most recent) of each file is restored.

So if you have spare cycles on your machine, you could optimize the bootstrap files by doing the following:

```
./bconsole

restore client=xxx select all

done

no

quit

Backup bootstrap file.
```

The above will not work if you have multiple Filesets because that will be an extra prompt. However, the restore client=xxx select all builds the in-memory tree, selecting everything and creates the bootstrap file.

The **no** answers the **Do you want to run this (yes/mod/no)** question.

3.3 Bootstrap for bscan

If you have a very large number of Volumes to scan with **bscan**, you may exceed the command line limit (511 characters). I that case, you can create a simple bootstrap file that consists of only the volume names. An example might be:

```
Volume="Vol001"

Volume="Vol002"

Volume="Vol003"

Volume="Vol004"

Volume="Vol005"
```


3.4 Final Bootstrap Example

If you want to extract a single Job, you can do it by applying VolSessionTime and the VolSessionId taken from a Job report or the Catalog. A .bsr file might look like the following:

```
Volume="Vol001"

VolSessionId=10

VolSessionTime=1080847820
```

If you know how many files are backed up (on the job report), you can enormously speed up the selection by adding (let's assume there are 157 files):

```
FileIndex=1-157

Count=157
```

Finally, if you know the logical file number where the Job starts, you can also cause bextract to forward space to the right file without reading every record:

```
VolFile=20
```

There is nothing magic or complicated about a .bsr file. Parsing it and properly applying it within *is* close to magic, but you don't need to worry about that.

If you want to see a **real** bsr file, simply fire up the restore command in the console program, select something, and while the "yes/mod/no" menus is shown, have a look at the .bsr file reported with the menu prompt. Eventually, you can copy that file and, in bconsole, answer "no" to the prompt.

