The Leading Open Source Backup Solution

2024-03-22

version 15.0.2

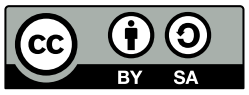
Bacula Community

Bacula® Concept Guide

Arno Lehmann — Philippe Chauvat

March 22, 2024

This manual documents Bacula Community Edition 15.0.2 (21 March 2024)



Contents

Chapter 1

What is Bacula?

Bacula is a set of computer btools that permits the system administrator to manage backup, recovery, and verification of computer data across a network of computers of different kinds. Bacula can also run entirely upon a single computer and can backup to various types of media, including tape and disk.

Bacula provides the following key features:

Network Based All backup and restores are done via the network. This permits Bacula to run on a single server and backup any computer in your data center.

Centralized Administration Bacula administration is centralized in the Director. Centralizing the administration is essential when your network grows beyond a few computers.

Runs Automatically Once setup, Bacula runs automatically. Normally a properly configured Bacula installation requires little maintenance or intervention except when adding new machines or when hardware errors occur.

Performs Bookkeeping Bacula does the hard bookkeeping by maintaining a catalog of what is backed up and where. If you have hundreds or thousands of machines to be backed up, it is essential that the bookkeeping is automatically maintained by a btool rather than requiring human intervention.

Multi-platform Bacula is compatible with a wide range of platforms: *BSD, Unix, Linux, MS Windows, Mac OS X, and others as well as a large range of hardware.

Modular Design means it scales well from small shops (one machine) to very large ones (tens of thousands of machines).

Multiple Backup Media Bacula handles a variety of different media such as disk, tape, autochangers. There is no need for expensive Virtual Tape Libraries (VTL), Bacula knows how to write to disk out of the box.

Reliable Bacula includes advance tools for memory and lock management, it is very common to run it without problems months at a time.

High Performance Bacula's modern multi-threaded design allows running multiple simultaneous backups and can achieve speeds writing to tape at more than 200 Megabytes per second, or faster.

Customizable Bacula can easily be customized to almost any backup / restore need.

Rapid Restores Easy and rapid restores using Bacula's database and graphical user interface.

Advanced Reporting, Notification, Monitoring Bacula has excellent reporting addons such as BWeb; those tools permit to get graphical and raw statistics for custom reports, billings, trends, optimizations and capacity planning.

Bacula provides very good notifications and monitoring capabilities, and can also be extremely well integrated into monitoring tools such as Nagios®.

1.1 Architecture

Bacula is made up of the following five major components or services: Director, Console, Client, Storage, and Catalog.

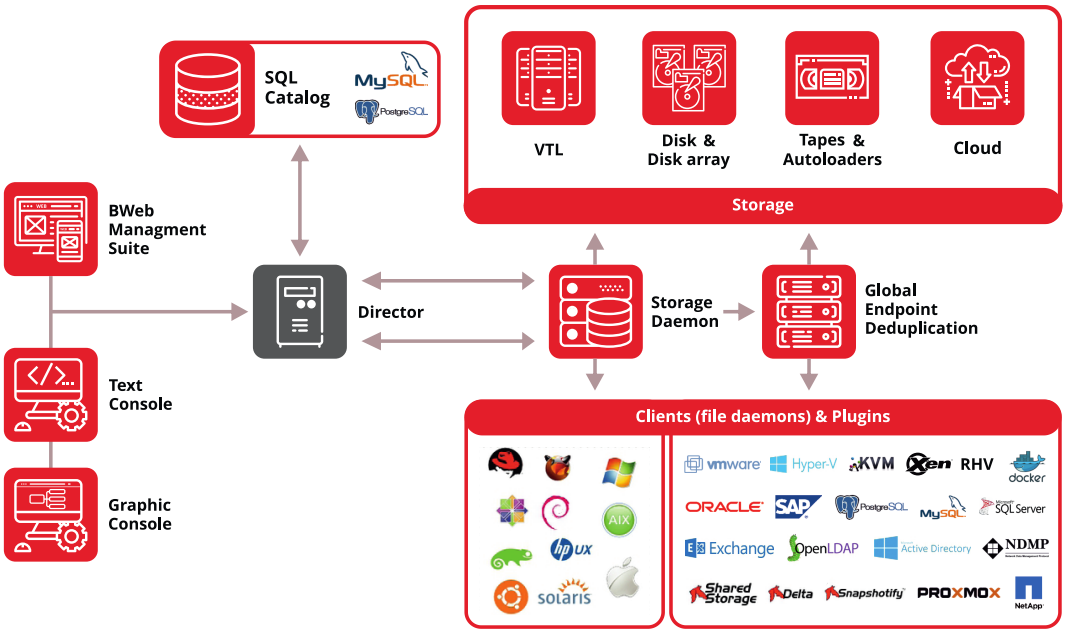


Figure 1.1: Bacula Components

1.2 Bacula Director

The Bacula Director is the btool that supervises all the backup, restore and verify operations. The system administrator uses the Bacula Director to schedule backups and to recover files. The Director runs as a daemon (or service) in the background. If configured by the system administrator, users may access the Director to do backups or restores of their files through restricted consoles.

1.3 Bacula Console

The Bacula Console is the btool that allows the administrator or user to communicate with the Bacula Director. Bacula Consoles are available with different user front ends: text-based console interface, graphical user interface, and web interface. The first and simplest is to run the `bconsole` btool in a shell window (i.e. TTY interface). Most system administrators will find this completely adequate. There are several graphical user interfaces, the most comprehensive being **bat** (Bacula Administration Tool) written using the Qt4 toolkit. There are also several web interfaces, the most complete being **BWeb Management Suite** available in the Bacula Enterprise version.

1.4 Bacula Client

The Bacula Client service (also known as the File Daemon) is the software btool that is installed on each machine to be backed up. It is specific to the operating system on which it runs and is responsible for providing the file attributes and data when requested by the Director. The Client services are also responsible for the file system dependent part of restoring the file attributes and data during a recovery operation. This btool runs as a daemon on the machine to be backed up.

Bacula Enterprise clients, distributed as binary packages, are available for a wide variety of Operating Systems[1]

1.5 Bacula Storage

The Bacula Storage service is the software btool that performs the storage and recovery of the file attributes and data to the physical backup media or volumes. In other words, the Storage Daemon is responsible for reading and writing your tapes (or other storage media, e. g. files). The Storage services runs as a daemon on the machine that has the backup device (for example a tape drive or a large disk).

1.6 Catalog

The Catalog services are comprised of the software btools responsible for maintaining the file indexes and volume databases for all files backed up. The Catalog services permit the system administrator or user to quickly locate and restore any desired file. The Catalog services sets Bacula apart from simple archiver btools like tar and dump, because the catalog maintains a record of all Volumes used, all Jobs run, and all Files saved, permitting efficient restoration and Volume management.

Bacula can run with three SQL database backends: MySQL, PostgreSQL and SQLite. We strongly recommend using SQLite only for testing purposes.[2]. The two others, MySQL and PostgreSQL, provide quite a number of features, including rapid indexing, arbitrary queries, and security.

1.7 Bootstrap File Concept

A *bootstrap file* is an ASCII file containing just the information Bacula needs to find a specific set of data and it is really important in Bacula:

- With a bootstrap file, you can restore without a Catalog;
- Bacula uses bootstrap files in Restore, Migrate or Copy Jobs, and more;
- The bootstrap files are created and updated automatically for each Job defined when they run.

[1] Gnu/Linux flavors, Windows systems, Solaris, Mac OSX, AIX, HP-UX and more
[2] Actually, Bacula Systems does not even provide packages with SQLite as a Catalog database backend.

Chapter 2

How Bacula Works

2.1 Introduction

2.1.1 The Job

The *Job* is probably one of the most important terms to understand when working with Bacula. There are several types of Jobs, the most known and the most used[1] is the "Backup" Job. The most important is, in our opinion, the "Restore" Job. Some others, like "Admin" Jobs are also quite useful. All of them are described in more detail in the chapter 3 on page 17.

2.1.2 The Volume

Another important term is the *Volume*: a Volume is the place where Bacula stores backed up data. When backing up to tapes, a Volume is identical to a tape and when backing up to disks, a Volume is a file. To fully understand Volumes, please read the chapter 4 on page 23 dedicated to their description.

2.1.3 Good to Know

Bacula backs up files By default, when working without special plugins, Bacula backs up files and nothing else.

Failed Jobs When running a backup Job, the Bacula client sends data and metadata to the Storage Daemon. In some situations a backup Job may fail. It is important to know that one can restore the already saved data from failed Jobs.

2.2 Jobs Step by Step

2.2.1 Opening Sessions

Bacula is network based and uses TCP/IP connections to exchange commands, information and data between its components. It needs to open sessions and these sessions are opened from one component to another one as shown figure 2.1 on the following page.

[1] Going by the number of Jobs!

- DIR ⇒ FD
- DIR ⇒ SD
- FD ⇒[2] SD

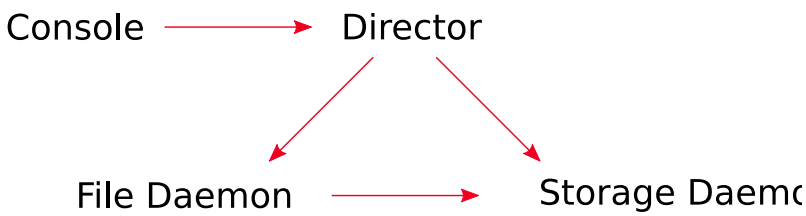


Figure 2.1: TCP/IP session "directions"

2.2.2 Data Stream

When running a backup Job, the stream of data is sent from the File Daemon to the Storage Daemon (data plus metadata). At the end of a backup Job, the Storage daemon sends the metadata to the Director to be inserted into the Catalog (see figure 2.2).

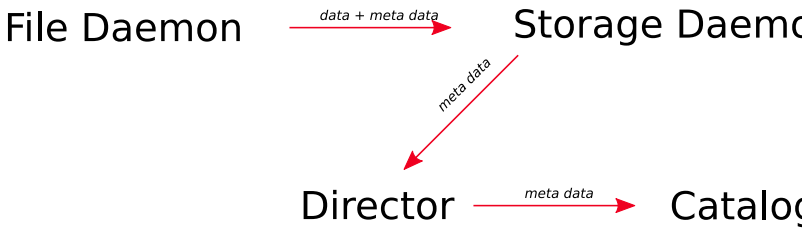


Figure 2.2: Data and metadata streams

2.2.3 Catalog Database Management

As mentioned earlier, Bacula stores its metadata in an SQL Catalog. PostgreSQL, MySQL and SQLite are the database engines that are supported by Bacula. We recommend to use SQLite only for basic testing purposes.

The Catalog management is directly related to keeping backup Jobs data available, and it is often automatically triggered by finishing Jobs.

Retention periods

There are various kinds of retention periods that Bacula uses: File Retention Period, Job Retention Period, and Volume Retention Period. Each of these retention periods defines the minimal time that specific records will be kept in the Catalog database. This should not be confused with the time that the data saved to a Volume is valid and available for restore – in many cases, data will be available much longer[3] than any of the Retention Periods configured.

[2] This is the default. However the session can be opened the other way with the "SD calls Client" directive
[3] Actually the data remains on the Volume until the Volume is recycled or truncated.

The File Retention Period determines the time that File records are kept in the Catalog database. This period is important for two reasons: the first is that as long as File records remain in the database, you can "browse" the database with a console btool and restore any individual file. Once the File records are removed or pruned from the database, the individual files of a backup Job can no longer be "browsed".

The second reason for carefully choosing the File Retention Period is because the database File records typically use the most storage space in the database. As a consequence, you must ensure that regular "pruning" of the database file records is done to keep your database from growing too big.

The Job Retention Period is the length of time that Job records will be kept in the database. Note, all the File records are tied to the Job that saved those files. The File records can be purged, leaving the Job records. In this case, information will be available about the Jobs that ran, but not the details of the files that were backed up. When pruning a Job, Bacula will purge all its File records.

The Volume Retention Period is the minimum time following the last write that a Volume will be kept until the Volume can be reused. Bacula will normally not overwrite a Volume that contains data still inside its Retention period, but if Bacula runs out of usable Volumes, it can select any Volume which is out of its Retention time for *recycling*, at this time automatically removing the related Job and File information from the Catalog.

Pruning

To keep the Catalog to a manageable size, the backup information should be removed (pruned) from the Catalog after the defined File and Job Retention Periods. Bacula by default automatically prunes the catalog database entries according to the retention periods defined.

Purging

Once all the database records that concern a particular Volume have been "pruned" as described above respecting the retention periods, the Volume is said to be "purged" (i.e. has no more catalog entries).

It is, however, possible to submit commands to Bacula to purge specific information, which will not respect configured retention periods. Naturally, this is something that should only be done with the greatest care.

2.2.4 Good to Know

Getting space back Bacula will try to keep your data safe as long as possible, thus purging a volume will not automatically reclaim the used space. If you want to reuse space, you must configure Bacula accordingly[4].

2.3 Bacula Users and Administrators

When considering enterprise backup and recovery environments, it is often useful to distinguish between different types or classes of people interacting with the backup and recovery tool:

[4] See chapter 4 on page 23

Administrators can control all aspects of Bacula, and modify its configuration.

Operators interact with Bacula, following defined procedures, are responsible for certain operational aspects, but do not touch the configuration.

Users or end-users are people who have no access to the Bacula configuration or all Bacula's features, but may access certain of its functions. A typical example is that a user can restore their own data, to their own computer, but can not see other backup data or access computers for which they are not responsible.

Bacula itself does not have the concept of users or administrators, but has "Consoles" (see section 2.4) that are designed to allow some users to have limited permissions regarding Bacula. However the *BWeb Management Suite*, the Bacula Enterprise Web-based tool for management and configuration, includes the concepts of users, groups and permissions.

2.4 Consoles

2.4.1 Overview

To allow interaction from administrators or users, Bacula uses *Consoles*. The Bacula Console (sometimes called the User Agent) is a btool that allows the user or the System Administrator to interact with the Bacula Director Daemon while the daemon is running. Note that, even when managing storage or checking client status, the Console interacts with the Director only, which in turn contacts the other daemons as needed.

The current Bacula Console comes in multiple versions:

- a shell interface (TTY style),
- a Qt GUI interface
- Web GUI interfaces

All permit the administrator or authorized users to interact with Bacula. You can determine the status of a particular Job, examine the contents of the Catalog as well as perform certain tape manipulations with the Console btool. Running Jobs is one of the more central tasks done with the Console.

Since the Console btool interacts with the Director through the network, the Console and Director btools do not necessarily need to run on the same machine. In fact, in an installation containing a single tape drive, a certain minimal knowledge of the Console btool may be needed in order for Bacula to be able to write on more than one Volume, because when Bacula requests a new one, it waits until the user, via the Console btool, indicates that the new Volume is mounted or labeled to be used.

2.4.2 Integration

Because the Console is like other Bacula components, it requires configuration. To know more about Console configuration and/or management, please see the *Console Configuration* chapter of the Bacula main manual (chapter 19).

2.4.3 User Restrictions

If you want to give access to a particular user (not overall Bacula administrator), you need to configure a console for him/her. In particular, it may be desirable to implement specific Access

Control Lists to prevent users from accessing data they are not authorized for. This part is covered in the *Console Configuration* chapter of the Bacula main manual (chapter 19).

Non Bacula Enterprise Consoles

There are many other GUI consoles available designed and proposed by the Bacula community. Here is a non-exhaustive list, ask the community if you want more information on one or another.

- `baculum` (see www.bacula.org[5])
- `bacula-web` (see www.bacula-web.org[6])
- `webacula` (see webacula.sourceforge.net[7])

2.5 Configuration

2.5.1 Installation Directory tree

When installing the Bacula Enterprise version[8] the following directory tree is created[9]:

- **`/opt/bacula/bin`**: Bacula Enterprise binaries directory
- **`/opt/bacula/working`**: Working directory
- **`/opt/bacula/etc`**: Configuration files root directory
- **`/opt/bacula/scripts`**: Scripts directory

2.5.2 Configuration Files

Bacula's configuration is stored in plain text files, and a configuration file can include other files [10], so it is possible to have a structured configuration file repository.

Each daemon has its own configuration consisting of a set of *Resource* definitions. These resources are very similar from one service to another, but may contain different *Directives* (records) depending on the service. For example, in the Director's resource file, the **Director** directive resource defines the name of the Director, a number of global parameters and the password needed to access it from a Console. In the File Daemon configuration file, the **Director** directive resource specifies which Directors are permitted to use the File Daemon.

- Director ⇒ `bacula-dir.conf`
- File Daemon ⇒ `bacula-fd.conf`
- Storage Daemon ⇒ `bacula-sd.conf`
- bconsole ⇒ `bconsole.conf`
- BWeb Management Suite ⇒ `bweb.conf` (this file is **not** structured the same way the others are!)
- Bacula Administration Tool (aka BAT) ⇒ `bat.conf`

[5] http://www.bacula.org/
[6] http://www.bacula-web.org/
[7] http://webacula.sourceforge.net/
[8] Bacula Community version has its own installation process and default locations
[9] Some additional directories under `/opt/bacula` exist and are not presented here
[10] Actually, it is also possible to create configuration parts by btool or script, while the daemon reads its configuration or while a Job is executed.

The configuration files must be written as plain **UTF-8**-encoded text files, which implies that any ASCII file is suitable.

Director

The configuration file defines a lot of resources such as:

- the Director itself
- at least one Restore Job
- Other Job definitions
- Schedules
- FileSets
- Storages
- Clients
- Catalogs
- Messages
- and others

Director definition:

```
Director {
   Name = the-name-of-the-director-dir
   DIRport = 9101
   QueryFile = "/opt/bacula/scripts/query.sql"
   WorkingDirectory = "/opt/bacula/working"
   PidDirectory = "/opt/bacula/working"
   Maximum Concurrent Jobs = 10
   Password = "password-for-the-console-to-access-the-director"
   Messages = Daemon
   Heartbeat Interval = 10
}
```

A Schedule definition to specify one full backup on the Sunday of the 2nd week of every second month starting in January, one differential on each Sunday except the Full's ones and an incremental backup six days a week:

```
Schedule {
   Name = "NightlyCycle"
   Run = Full         jan,mar,may,jul,sep,nov 2nd     sunday  at 21:00
   Run = Differential feb,apr,jun,aug,oct,dec 2nd     sunday  at 21:00
   Run = Differential                      1st,3rd-5th sunday  at 21:00
   Run = Incremental                                   mon-sat at 21:00
}
```

FileSet definition, backing up `/etc`, `/opt`, `/home`, etc. excluding some directories

```
FileSet {
   Name = "fs-websites"
   Include {
      Options {
         Signature = MD5
         Compression = GZIP
      }
      File = /etc
      File = /opt
      File = /root
      File = /home
```

```
      File = /var/log
   }
   Exclude {
      File = /home/websites/tmp
      File = /home/websites/www/tmp/cache
      File = /opt/bacula/working
      File = /opt/bacula/archive
      File = /.journal
      File = /.fsck
   }
}
```

A Storage definition, as required by the Director, i. e. a name, an address, a port, and a media type. The director does not know about the hardware, only the storage daemon does:

```
Storage {
   Name = Remote-Disk-Storage
   Address = sd.bacula6.org
   SDPort = 9103
   Password = "password-for-the-director-to-access-the-storage"
   Device = disk-autochanger
   Media Type = da-mt
   Maximum Concurrent Jobs = 50
   Autochanger = Remote-Disk-Storage
}
```

A client to back up:

```
Client {
   Name = client-to-back-up-fd
   Address = client.bacula6.org
   FDPort = 9102
   Catalog = BaculaCatalog
   Password = "password-for-the-director-to-access-the-client"
   File Retention = 10 days
   Job Retention = 25 days
   AutoPrune = no
}
```

A pool definition:

```
Pool {
   Name = the-pool-name
   Pool Type = Backup
   Recycle = yes
   AutoPrune = no
   Volume Retention = 30 days
   Label Format = "pooldef-"
   Maximum Volume Bytes = 8G
   Maximum Volumes = 6
   Storage = Remote-Disk-Storage
}
```

A JobDef to handle common definitions for several Jobs:

```
JobDefs {
   Name = "common-job-definitions"
   Type = Backup
   Level = Incremental
   Messages = Standard
   Schedule = NightlyCycle
   Priority = 10
   #
   # The following setting saves some time sending
   # all the metadata at the end of the job
   Spool Attributes = yes
   #
```

```
    # A way to keep all of your BSR (bootstrap) files
    # in one place with the same naming conventions
    Write Bootstrap = "/opt/bacula/bsr/%c_%n.bsr"
}
```

And then a Job using the above JobDefs:

```
Job {
    Name = "back-up-job"
    JobDefs = "common-job-definitions"
    #
    # The storage is defined inside the Pool Resource
    # this is a best practice.
    Pool = the-pool-name
    Client = client-to-back-up-fd
    FileSet = "fs-websites"
    Schedule = NightlyCycle
    #
    # Below you have an example of how to include a file
    # notice the "@" sign as first character
    @/opt/bacula/etc/included-configuration-file.conf
}
```

File Daemon

If you read carefully you will notice, on the Bacula client side, a Director Resource similar to this one:

```
Director {
    Name = the-name-of-the-director-dir
    Password = "password-for-the-director-to-access-the-client"
}
```

which authorizes the Director the-name-of-the-director-dir, knowing the client password to access to the client below

```
FileDaemon {
    Name = client-to-back-up-fd
    FDport = 9102
    WorkingDirectory = /opt/bacula/working
    Pid Directory = /opt/bacula/working
    Maximum Concurrent Jobs = 20
}
```

Storage Daemon

Same thing here, the Director Resource

```
Director {
    Name = the-name-of-the-director-dir
    Password = "password-for-the-director-to-access-the-storage"
}
```

authorizes the Director the-name-of-the-director-dir, knowing the storages password to access the storage daemon

```
Storage {                             # definition of myself
    Name = bacula-storage-daemon-definition-sd
    SDPort = 9103                  # Director's port
    WorkingDirectory = "/opt/bacula/working"
```

```
   Pid Directory = "/opt/bacula/working"
   Maximum Concurrent Jobs = 200
   Heartbeat Interval = 10
}
```

and therefore to use the devices defined on the Storage Daemon side for example

```
#
# This is a two drive autochanger definition
Autochanger {
   Name = disk-autochanger
   Device = drive1, drive2
   # No changer command needed for virtual autochangers
   Changer Command = ""
   # No changer device needed either
   Changer Device = "/dev/null"
}
#
# Drive 1 definition
Device {
   Name = drive1
   Archive Device = /bacula/da1
   Media Type = da-mt
   Drive Index = 0
   Label Media = yes
   Random Access = yes
   AutomaticMount = yes
   RemovableMedia = no
   AlwaysOpen = no
   Maximum Concurrent Jobs = 20
}
#
# Drive 2 is pretty much the same in this case
Device {
   Name = drive2
   #
   # This is the same archive device as in drive1 definition
   Archive Device = /bacula/da1
   Media Type = da-mt
   # Another drive, another drive index
   Drive Index = 1
   Label Media = yes
   Random Access = yes
   AutomaticMount = yes
   RemovableMedia = no
   AlwaysOpen = no
   Maximum Concurrent Jobs = 20
}
```

bconsole

The following `bconsole.conf` content allows the Console to connect to the following Director:

```
Director {
   Name = the-name-of-the-director-dir
   DIRport = 9101
   address = localhost
   Password = "password-for-the-console-to-access-the-director"
}
```

2.5.3 BWeb Management Suite considerations

The BWeb Management Suite (BMS) interface allows the administrator to manage the Bacula configuration through a Web interface. When using BMS for this purpose, the

Bacula configuration files are split into pieces, all of them under the `conf.d` directory (`/opt/bacula/etc/conf.d`).

Figure 2.3 on the facing page presents an example of a `conf.d` directory organization while the following is a tree representation of the same example.

```
conf.d/
|--- Console
|   `-- v8-dir
|       |--- bconsole.conf
|       `-- Director
|           `-- v8-dir.cfg
|--- Director
|   `-- v8-dir
|       |--- bacula-dir.conf
|       |--- Catalog
|       |   `-- MyCatalog.cfg
|       |--- Client
|       |   |--- v8-c1-fd.cfg
|       |   |--- v8-c2-fd.cfg
|       |   `-- v8-c3-fd.cfg
|       |--- Console
|       |   `-- v8-mon.cfg
|       |--- Director
|       |   `-- v8-dir.cfg
|       |--- Fileset
|       |   |--- Catalog.cfg
|       |   |--- fs-postgres.cfg
|       |   `-- fs-tls.cfg
|       |--- Job
|       |   |--- job-catalog.cfg
|       |   |--- job-postgres.cfg
|       |   |--- job-RestoreFiles.cfg
|       |   `-- job-tls.cfg
|       |--- JobDefs
|       |   |--- defaultjob.cfg
|       |   `-- common.cfg
|       |--- Messages
|       |   |--- Daemon.cfg
|       |   `-- Standard.cfg
|       |--- Pool
|       |   |--- Dedup.cfg
|       |   |--- File.cfg
|       |   |--- common.cfg
|       |   `-- Scratch.cfg
|       |--- Schedule
|       |   |--- FiveDays.cfg
|       |   |--- WeeklyCycleAfterBackup.cfg
|       |   `-- WeeklyCycle.cfg
|       `-- Storage
|           |--- Remote-Disk-Storage.cfg
|           `-- File.cfg
|--- FileDaemon
|   |--- v8-c1-fd
|   |   |--- bacula-fd.conf
|   |   |--- Director
|   |   |   `-- v8-dir.cfg
|   |   |--- FileDaemon
|   |   |   |-- v8-c1-fd.cfg
|   |   |   `-- v8-mon.cfg
|   |   `-- Messages
|   |       `-- Standard.cfg
|   |--- v8-c2-fd
|   |   |--- bacula-fd.conf
|   |   |--- Director
|   |   |   |--- v8-dir.cfg
|   |   |   `-- v8-mon.cfg
|   |   |--- FileDaemon
|   |   |   `-- v8-c2-fd.cfg
|   |   `-- Messages
|   |       `-- Standard.cfg
|   `-- v8-c3-fd
|       |--- bacula-fd.conf
|       |--- Director
```

```
|       |   `-- v8-dir.cfg
|       |--- FileDaemon
|       |   `-- v8-c3-fd.cfg
|       `-- Messages
|           `-- Standard.cfg
`-- Storage
    `-- v8-sd
        |--- Autochanger
        |   `-- disk-autochanger.cfg
        |--- bacula-sd.conf
        |--- Device
        |   |--- drive1.cfg
        |   |--- drive2.cfg
        |   `-- File.cfg
        |--- Director
        |   |--- v8-dir.cfg
        |   `-- v8-mon.cfg
        |--- Messages
        |   `-- Standard.cfg
        `-- Storage
            `-- v8-sd.cfg
```

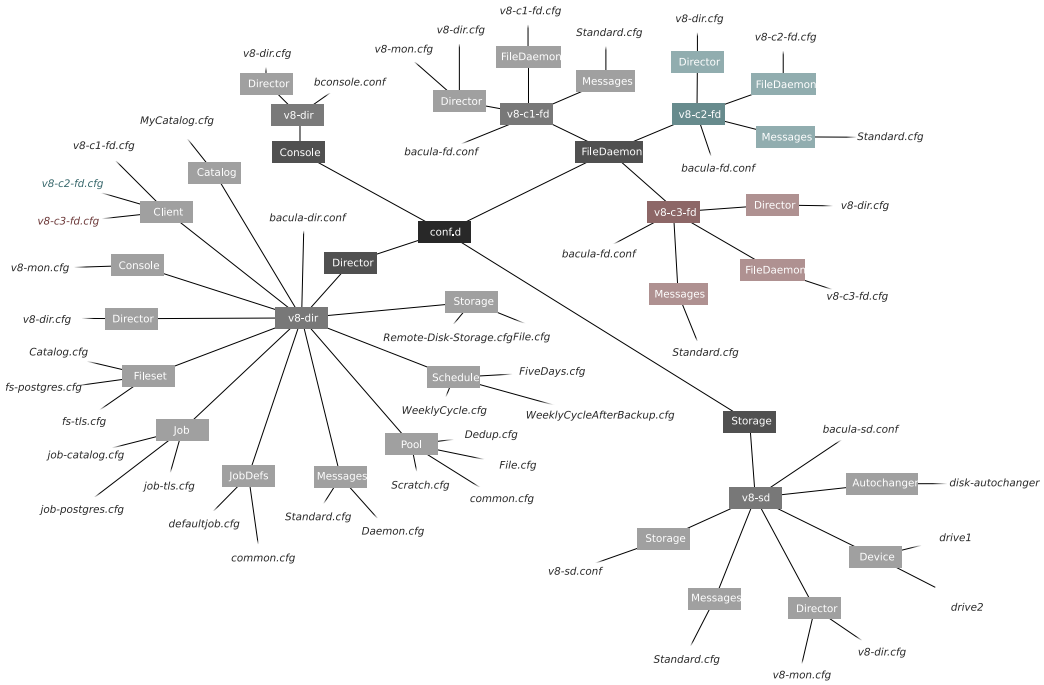


Figure 2.3: `conf.d` directory typical graphical representation example

2.5.4 Good to Know

Installing, understanding and managing Bacula configuration files is covered in-depth during the **Bacula Administration Course I**. Even if this concept guide is here to help and give a quick overview of all concepts, attending this course, given in the US, Europe (Switzerland, Belgium, France) and in Japan is a strong recommendation.

2.6 Encryption

Bacula can encrypt the data it backs up (*Data Encryption*), and independently of that, it can encrypt the network connections it uses (*Transport Encryption*).

Both Data and Transport Encryption make use of industry-standard x509 Public Key Infrastructure, but the requirements differ a bit. In general, Data Encryption needs a minimum of infrastructure and configuration, while Transport Encryption may require much more effort of the Backup Administrator.

2.6.1 Data Encryption

Data Encryption happens on the client and needs a key pair in files stored on the client. Thus, the encrypted data may be inaccessible even to the backup administrators. If a *Master Key* is desirable, so that data can be decrypted even if the client machine owner loses or has changed the keys, Bacula can be configured accordingly.

The client machine owner or administrator can be the only person responsible for the key files, which implies that additional tasks come into their realm of responsibility.

It is important to be aware that, while data itself is encrypted, metadata that is stored in the Catalog remains unencrypted – file and path names, in particular, are easily visible to third parties with access to the network or the catalog database.

2.6.2 Transport Encryption

Transport Encryption, on the other hand, requires a full-blown, managed Public Key Infrastructure including a trusted and securely operated Certification Authority, secure deployment of keys and certificates, explicit configuration of trust relationships and a few lines of configuration in each Bacula configuration resource that refers to any Bacula component. Experience shows that configuring transport encryption for the first time is a challenge sometimes even to experienced Bacula administrators.[11]

The overhead to operate and set up such an environment can be considerable and is, in general, not an area of expertise for backup and storage administrators. However, if a maximum of security is desired, it provides industry-standard protection against any sort of unauthorized data snooping and as such is particularly important when backups are done through insecure networks like the internet.

2.6.3 Good to Know

Data and Transport Encryption can be freely combined, but keep in mind that each layer of encryption adds CPU overhead and can thus decrease throughput and increase resource consumption.

Bacula can also use certificate-based authentication of its components, which is particularly useful if there are already certificates deployed for the involved computers.

[11] Fortunately, experience also shows that once over the first hurdle, things go considerably more smoothly!

Chapter 3

The Job

3.1 Definition

The Job is the basic unit in Bacula and is run by the Director. It ties together the following items:

- **Who**: The *Client*, the machine to backup
- **What**: The *File Set*, which files to backup and not to backup
- **Where**:
  - *Storage*: what physical device to backup to
  - *Pool*: which set of Volumes to use
  - *Catalog*: where to keep track of files
- **When**: *Schedule*: when to run the Job

3.2 Types

There are several types of jobs in Bacula:

- Backup
- Restore
- Admin
- Verify
- Copy
- Migration
- Archive
- Console connection
- Internal system job

3.3 Backup Job Description

As shown in figure 3.1 on the next page, when running a backup job:

- the Bacula Director will connect to the File Daemon and Storage Daemon,
- the Director will then send to the File Daemon the necessary information to actually realize the backup job, like the Level, the File Set, the Storage Daemon to contact, etc.
- then the File Daemon will contact the Storage Daemon, identify the Data to be sent and send them as well as the related meta data (file attributes) to the Storage Daemon
- the Storage Daemon will put them into one or more Bacula Volumes, according to the Pool Resource chosen for the Job
- the Storage Daemon will send metadata back to the Director.

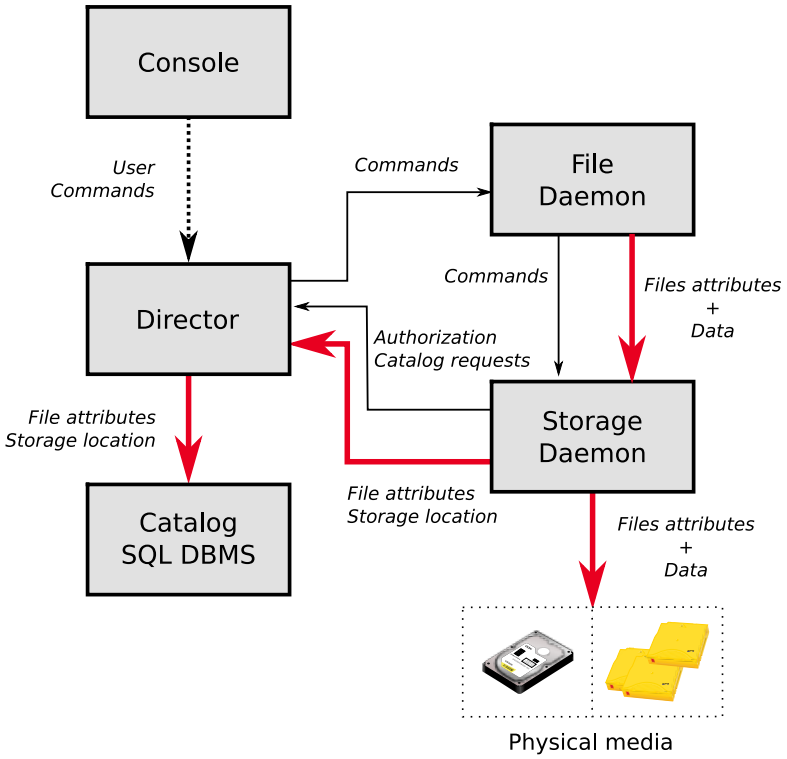


Figure 3.1: The backup job diagram

The moment Bacula will run a Backup Job depends on how this job is started:

- manually through a *Console*, e. g with `bconsole`: `run job=name-of-the-job` command
- automatically by a defined and referenced Schedule
- automatically by an external script or command like
  `bconsole -c bconsole.conf << EOF`
  `run job=name-of-the-job ...`
  `EOF` command

3.3.1 Admin Jobs

An Admin Job is a job which does not backup or move any data, but which can be used to execute scripts through the *Run Script* resource and *Console* or *Command* directives. The following definition will launch the script "/opt/bacula/scripts/my-script.pl" located on the Director machine according to the Schedule "AdminSchedule"

```
Job {
  Type = Admin
  Run Script {
    Runs When = Before
    Runs on Client = No
    Command = "/opt/bacula/scripts/my-script.pl"
  }
  Schedule = AdminSchedule
  ...
}
```

3.3.2 Migration, Copy and Virtual Full Jobs

Bacula provides three kinds of Jobs that do not actually backup data, but move data around. They are doing very similar things:

- Migration Jobs allow the migration of job data from one volume to another one; This is usually interesting when implementing a Disk-to-Disk-to-Tape strategy, or, more general, any sort of multi tiered backup storage system.
- Copy Jobs copy a Job's data to another volume. This is usually used when the requirements include at least two places where backup data is being kept.
- Virtual Full Job which provide a way to consolidate several jobs into one, without requesting any data from the client, based only on existing backup data.

All these kind of Jobs rely on a *Next Pool* directive.

Virtual Full Consideration

Virtual Full backups are a way to do backups in an "Incremental forever" style while continuing to provide Full backups at the same time.

When backing up data at incremental or differential levels, Bacula (by default) does not do anything regarding removed or moved files or directories. Which implies that the result of a restoration could be different than the latest state of the machine. For that reason, we advise using "Accurate" mode which is enabled by the directive of the same name. When set to "yes" in a Job, Bacula will record removed missing files or directories, and depending on additional configuration, Bacula will also consider more criteria than just time stamps to determine if a file needs to be backed up. In this case, Bacula will restore the machine to the exact same state (from a backup content point of view) that it was in during the backups.

Administrators will understand that the "Accurate" mode takes additional resources and time when running backups.

To improve performance, you may use the "Accurate" directive when using Virtual Full backups only for the last incremental before the Virtual Full itself. Activating this directive for every incremental backup would be even better but could increase the backup time.

```
#
# A usual job definition
Job {
  Name = "j-bacula"
  JobDefs = "DefaultJob"
  FileSet = "BaculaFileSet"
  Client = client-fd
  Schedule = s-Data2Disk
  Max Full interval = 19 days
  Run Script {
      Runs When = after
      Runs on Client = no
```

```
      Runs on Failure = yes
      Command = "/opt/bacula/scripts/run_copyjob.pl %l %i %b t-rdx j-copy-full" ;
      # Launching the copy job as soon as the backup is done
      # %l is job level
      # %i is job id
      # %b is job bytes
      # j-copy-full is the job name (see below)
      # The script "run_copyjob.pl" issues a shell command like
      # bconsole -c bconsole.conf << END_OF_DATA
      #          run jobid=%i job=j-copy-full storage=t-rdx yes
      #          quit
      # END_OF_DATA
  }
}
```

And here is the "Copy" job to copy the job from Disk to RDX

```
Job {
    Name = "j-copy-full"
    Type = Copy
    Level = Full
    Client = client-fd
    File Set = "Empty Set"
    Messages = Standard
    Pool = PoolVFull
    Maximum Concurrent Jobs = 1
}
```

And the Schedule used to run the several levels

```
  Schedule {
        Name = s-Data2Disk
        Run = Level=incremental monday-thursday,saturday at 21:00
        Run = Level=incremental accurate=yes friday at 12:30
        Run = Level=VirtualFull priority=15 friday at 12:35
}
```

3.4 Pruning

Pruning by default occurs at the end of a Job. When doing some tests or if you only have few jobs a day, this could be fine. But as soon as the number of jobs is growing, you would prefer to manage pruning another way to let your Catalog do its best for the backup jobs, not for the database administrative tasks.

In such a situation, you will configure your clients not to activate the pruning algorithm, using the "Auto Prune" Directive such as the following:

```
  Client {
    Name = client-fd
    Address = bacula.example.com
    FDPort = 9102
    Catalog = Catalog
    Password = "do-you-want-a-very-strong-password?"
    File Retention = 15 days
    Job Retention = 50 days
    AutoPrune = no
         # No automatic pruning at the end of a job for this client
  }
```

If you don't do anything more, your Catalog will grow infinitely. To keep it at its best, you should define an "Admin" job, like the following:

```
Job {
   Name = "admin-manual-pruning"
   Type = Admin
   JobDefs = "DefaultJob"
   RunScript {
      Runs When = Before
      # below command relies on proper PATH!
      Command = "/bin/sh -c \"echo prune expired volume yes\" | bconsole"
      Runs On Client = no
   }
   Schedule = s-Prune
}
```

As a Bacula volume can contain one or more jobs (or parts of jobs) and a job contains one or more files, the pruning process will have side effects:

- when pruning a Volume, all the jobs related to the Volume are pruned
- when pruning a Job, all the files related to this jobs are pruned

That is the reason why you should have the following inequality:

$$\text{File Retention} \leqslant \text{Job Retention} \leqslant \text{Volume Retention}$$

Chapter 4

Volumes

Bacula manages all its storage compartments – Volumes in our terminology – in Pools. In particular, a Job is not configured to write to any particular Volume, but to a set of Volumes called a Pool. Volumes of Bacula are always members of only one Pool. There can however be multiple pools. Accordingly, we present the conceptual overview here starting with the higher-level Pool view.

4.1 The "Pool"

4.1.1 The Concept

Pools in Bacula are just a way to manage volumes and to manage collections of similar volumes. The following are just examples and you might have either different requirements or even no requirement to separate volumes into several pools:

- Pools "short-term" and "long-term" to handle short and long retention periods.
- Pools "big-customer" and "usual-customers" to keep all the backup jobs from your client "Big Customer" in a specific pool while putting all "usual" customers into another one.
- Pools "r-and-d" and "accounting" to keep R&D backups together and the Accounting ones in another pool, separated from each other.

There can be many reasons to use or not use several pools; the Bacula configuration gives the Backup Administrator the ability to match the defined requirements.

Due to how Bacula manages Volume Retention periods, the jobs put into the same pool should have the same retention periods.

4.1.2 Pool Types

Bacula allows several kinds of pools including[1]:

- Scratch
- Backup
- Recycle

[1] Listed in the logical order

A *Scratch* pool contains volumes that may be with any pool. If no volumes are available for a particular pool Bacula will look in the Scratch pool and move the volume to the pool in question.

A *Backup* pool contains volumes intended to keep data from backups for a defined retention period. It can also include Recycled or Purged volumes, i. e. Volumes that are eligible to be overwritten.

A *Recycle* pool contains volumes after the purging process freed them, **after** they have been used in a *Backup* pool.

The Bacula configuration allows the Administrator to attach a *Scratch* Pool and a *Recycle* Pool to a *Backup* pool. The volume cycle management process is described by the figure 4.1.

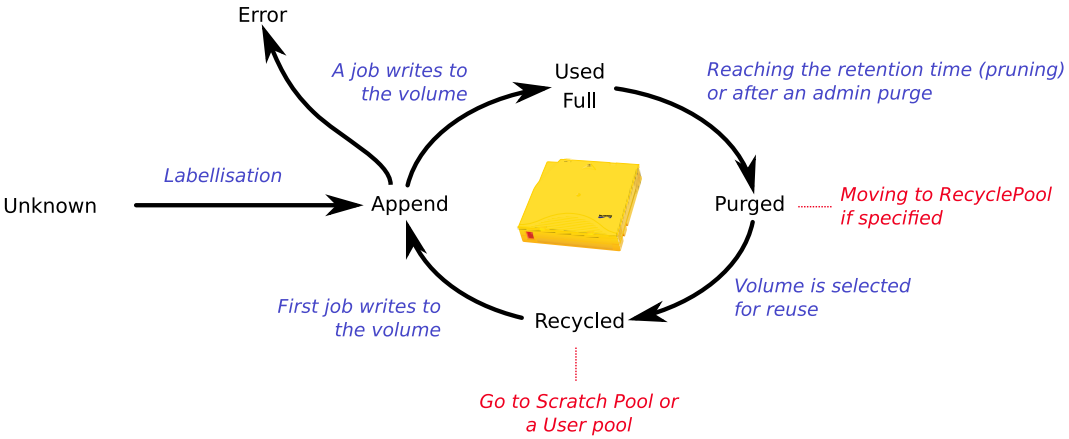


Figure 4.1: Volume Cycling Process

The following example is a pool definition to manage Volumes for day-to-day incremental backup jobs:

```
# Pool to handle day-to-day backup jobs
Pool {
   Name = PoolFile
   Pool Type = Backup
   Recycle = yes
   AutoPrune = yes
   Volume Retention = 15 days
   Label Format = "file-"
         # Auto-labeling new volumes
   Action on Purge = Truncate
         # Backup to disk. We get back the space ASAP.
   Next Pool = PoolVFull
         # This is the pool where Virtual Full will take place
   Storage = t-disk
   Volume Use Duration = 23h
         # Low duration time, better volume retention period.
   Maximum Volumes = 750
   Maximum Volume Bytes  = 1GB
         # Small volumes. A lot of them. This is for incrementals.
}
```

In the next example you will find a pool definition to keep the Virtual Full backups for the previously described incremental backup jobs:

```
#
# Virtual Full in this pool
Pool {
   Name = PoolVFull
   Pool Type = Backup
   Recycle = yes
   AutoPrune = yes
```

```
    Volume Retention = 20 days
    Label Format = "file-vfull-"
         # Auto-labeling new volumes
    Action on Purge = Truncate
         # Backup to disk. We get back the space ASAP.
    Next Pool = PoolRDX
         # This is the pool where Copy jobs will be sent
    Storage = t-migrate
    Volume Use Duration = 23h
         # Low duration time, better volume retention period.
    Maximum Volume Bytes = 10GB
    Maximum Volumes = 20
         # Few volumes, bigger than the incremental ones. For ‘‘Full’’ (VF) purpose.
}
```

And below, we show a pool definition to store the Copy volumes for the previous Full backup jobs:

```
#
# Copy of Fulls in this pool
Pool {
   Name = PoolRDX
   Pool Type = Backup
   Recycle = yes
   AutoPrune = yes
   Volume Retention = 40 days
   Action on Purge = Truncate
        # Backup to disks. We get back the space ASAP.
   Volume Use Duration = 20h
        # Low duration time, better volume retention period.
   Storage = t-rdx
   Maximum Volume Bytes = 20GB
        # 320GB RDX removable disk. 20GB per RDX volume.
}
```

4.2 Media Type

Even if Media Types may be used for several different reasons with Bacula, you must use them when dealing with several storages or several locations on the same storage.

In particular when Bacula is searching for a volume to be used for a backup, it will go through the available volumes. A Bacula job uses a distinct Storage device and the volume selection must be restricted to the actually available volumes. For example, it is quite common to see different disk storage arrays, mounted at different mount points and used by different storage devices, where volumes located in one directory are not available through the other storage device.

More obvious (and less easy to fix!) is a case of mixed media – for example, a disk storage device will never be able to write to a physical tape. So, all storage devices that cannot share their media must use distinct Media Types.

4.3 Storage Capacity Management

If you want to overwrite a backup you need to overwrite the corresponding volume(s). Additionally, to be able to reuse a Volume, its retention period should first be expired.

4.3.1 Volume Retention

The Volume retention period is defined as the time Bacula will not overwrite a Volume after it was last written to. Therefore, if you keep writing to a volume with a retention period of seven

days for six months, your volume will not be overwritten for a total of 187 days – perhaps even more: remember that Bacula will try to keep your data safe for as long as possible.

4.3.2 Limiting the Volume Size

Tapes

On tapes, we usually don't need to explicitly limit the volume size, because the volume is identical to the tape which has a limited and defined maximum capacity.

Disks

On disks, on the other hand, as volumes are handled with files, there is a risk of filling the disk with only one volume, so we need to limit the volume size and number of volumes. We explained above how the volume retention periods are handled by Bacula. However, often we need to get back our volumes as soon as possible to be able to reuse the storage space they are using during their retention periods. In the previous case (see chapter 4.3.1 on the previous page) we should have restricted the use duration of the Volume.

There are several directives available to limit the occupied storage space in a Pool:

- Maximum Volumes: the pool will contain at most this number of volumes
- Maximum Volume Bytes: a volume won't have a size above this value, even if it handles concurrent jobs
- Maximum Volume Jobs: this is a way to limit a volume usage duration. As soon as the volume contains this number of jobs, Bacula will refuse to use it and will use another one
- Maximum Use Duration: this is also a way to limit the use duration of a volume.

These limitations can be managed manually (by the Administrator with console commands) or automatically (through scripts or Admin jobs, for example).

4.3.3 Space Management

As we already noted, Bacula will try to keep your data safe as long as possible. This means that you can be in a situation (disk backups only) where your storage space is too small even if you defined correct retention periods. In such cases, you can use the "Action on Purge" directive requesting Bacula to actually `truncate` the disk volumes and thus to get back the corresponding storage space freed by the truncate.

4.4 Volumes and Pool Modifications

Because of its great flexibility, Bacula can be puzzling. Let's say you defined Maximum Volume Bytes to 10 GB and then realize that you want to move it to 15 GB. Then you will modify your "Pool" resource definition. This is correct and will work for all new volumes created into the Pool. But the existing volumes are not modified by this change. To update the existing volumes already in the Pool with the new values you must update the existing volumes' meta-data:

- In `bconsole`, enter the `update volume` command and answer the subsequent questions Bacula will ask, or
- use `update volume allfrompool=the-pool-name` command, or
- use one of the graphical interfaces.

Chapter 5

Going Further

5.1 Installing Bacula Enterprise

Before implementing Bacula Enterprise version, please be sure to read all product specifications in the related White Papers so that you choose compatible platforms for your backup infrastructure. For example, some plugins need specific operating systems to run.

5.2 How your Download Area is Organized

When contracting with Bacula Systems, you will receive a Welcome Package including documentation describing how to contact the Support team and where your Bacula Enterprise software is available. This place is called your **download area**.

Depending on what you subscribed to at Bacula Systems, your download area gives you access to several entries:

- debs/
  Where you will find all the Debian and related flavors packages
- demo/
  Where you will find the demo packages as well as the kickstart
- manuals/
  The Bacula Enterprise core documentation (cf. 5.3)
- RPMs/
  Where you will find all the Red Hat, SUSE and related flavors packages
- white papers/
  Where you will find the complementary documentation (cf. 5.3)
- Bacula_Enterprise_Edition_Installation_Guide.pdf
  The Bacula Enterprise Edition installation guide.
- BaculaSystems-Public-Signature.asc
  To be used with the `yum`, `aptitude`, `yast`, etc. tools.
- ...

5.3 How Documentation is Organized

The Bacula documentation is composed of the following elements:

- Core manual suite:
  - Main Manual
    This is the manual where you can find the core documentation: concept, getting started, configuration.
  - Console Manual
    This manual is intended to present the Console's fundamentals. Many useful Console features are described here.
  - Utility Manual
    Before using Bacula Enterprise or to validate your regex / wildcard expressions, you may want to use the Bacula utilities like `btape`, `bregex`, `bwild`, etc.
  - Problems Manual
    As a Bacula Systems customer, you certainly should open a ticket if you have an issue with Bacula Enterprise. If you want to know more about how Bacula operates or how to find a solution for some issues, this manual is for you.
  - Developer's Manual
    Together with the policies defined by the Bacula team describing how to write the code Bacula users are expecting, this document also presents the Bacula storage format.
- Installation Guide
  To help you during the installation process of Bacula Enterprise version, you will find an installation guide at the root directory of your download area. This document will show you how to use the download area the support team has already prepared for you.
- White papers
  When setting up your Bacula Enterprise Backup and Restore Solution, you may need some advice to tune the Bacula catalog, manage Disk backups, do Virtual Full, Copy or Migration Jobs. The usage of such features is detailed into our general white papers.
- Plugin documentation
  When designing a new plugin, Bacula Systems creates a dedicated technical document. For each plugin you will find a corresponding white paper to help you set it up.

5.4 About Plugins

There are several kinds of plugins. Some of them are related to the storage daemon and some others to the file daemon. You can distinguish between them by their names:

- a storage daemon plugin is named `xxxx-sd.so`
- a file daemon plugin is named `yyyy-fd.so`

Be sure to install the plugin in the right place (on the File Daemon or the Storage Daemon), to read the related documentation and to set it up correctly before use.