Bacula Enterprise Advanced Features Usage

Bacula Systems Documentation

Contents

Contents

The following article aims at presenting various information on how to use advanced features provided by Bacula Enterprise.

1 Replication: Copy/Migration Jobs

The term Migration, as used in the context of Bacula, means moving data from one Volume to another. In particular it refers to a Job (similar to a backup job) that reads data that was previously backed up to a Volume and writes it to another Volume. As part of this process, the File catalog records associated with the first backup job are purged. In other words, Migration moves Bacula Job data from one Volume to another by reading the Job data from the Volume it is stored on, writing it to a different Volume in a different Pool, and then purging the database records for the first Job.

The Copy process is essentially identical to the Migration feature with the exception that the Job that is copied is left unchanged. This essentially creates two identical copies of the same backup. However, the copy is treated as a copy rather than a backup job, and hence is not directly available for restore. If bacula finds a copy when a job record is purged (deleted) from the catalog, it will promote the copy as *real* backup and will make it available for automatic restore. Note: in the text below, to simplify it, we usually speak of a migration job. This, in fact, means either a migration job or a copy job.

The Copy and the Migration jobs run without using the File daemon by copying the data from the old backup Volume to a different Volume in a different Pool. It is not possible to run commands on the defined Client via a RunScript from within the Migration or Copy Job.

The selection process for which Job or Jobs are migrated can be based on quite a number of different criteria such as:

- a single previous Job

- a Volume

- a Client

- a regular expression matching a Job, Volume, or Client name

- the time a Job has been on a Volume

- high and low water marks (usage or occupation) of a Pool

- Volume size.

The details of these selection criteria will be defined below.

To run a Migration job, you must first define a Job resource very similar to a Backup Job but with **Type = Migrate** instead of **Type = Backup**. One of the key points to remember is that the Pool that is specified for the migration job is the only pool from which jobs will be migrated, with one exception noted below. In addition, the Pool to which the selected Job or Jobs will be migrated is defined by the **Next Pool = …** in the Pool resource specified for the Migration Job.

Bacula permits Pools to contain Volumes with different Media Types. However, when doing migration, this is a very undesirable condition. For migration to work properly, you should use Pools containing only Volumes of the same Media Type for all migration jobs.

The migration job normally is either manually started or starts from a Schedule much like a backup job. It searches for a previous backup Job or Jobs that match the parameters you have specified in the migration Job resource, primarily a **Selection Type** (detailed a bit later). Then for each previous backup JobId found, the Migration Job will run a new Job which copies the old Job data from the previous Volume to a new Volume in the Migration Pool. It is possible that no prior Jobs are found for migration, in which case, the Migration job will simply terminate having done nothing, but normally at a minimum, three jobs are involved during a migration:

- The currently running Migration control Job. This is only a control job for starting the migration child jobs.

- The previous Backup Job (already run). The File records for this Job are purged if the Migration job successfully terminates. The original data remains on the Volume until it is recycled and rewritten.

- A new Migration Backup Job that moves the data from the previous Backup job to the new Volume. If you subsequently do a restore, the data will be read from this Job.

If the Migration control job finds a number of JobIds to migrate (e.g. it is asked to migrate one or more Volumes), it will start one new migration backup job for each JobId found on the specified Volumes. Please note that Migration doesn't scale too well since Migrations are done on a Job by Job basis. This if you select a very large volume or a number of volumes for migration, you may have a large number of Jobs that start. Because each job must read the same Volume, they will run consecutively (not simultaneously).

See the directives for migration and copy:

- MigrationAndCopyDirectives

1.1 Important Migration Considerations

- The Job(s) resource configuration of the job(s) that will be migrated shouldn't be deleted. Otherwise, the migration job will fail.

- Each Pool into which you migrate Jobs or Volumes **must** contain Volumes of only one Media Type. Jobs on Volumes will be considered for Migration ** only if the Volume is marked Full, Used, or Error**. Volumes that are still marked Append will not be considered for migration. This prevents Bacula from attempting to read the Volume at the same time it is writing it. It also reduces other deadlock situations, as well as avoids the problem that you migrate a Volume and later find new files appended to that Volume.

- Migration takes place on a JobId by JobId basis. That is each JobId is migrated in its entirety and independently of other JobIds. Once the Job is migrated, it will be on the new medium in the new Pool, but for the most part, aside from having a new JobId, it will appear with all the same characteristics of the original job (start, end time, etc.). The column RealEndTime in the catalog Job table will contain the time and date that the Migration terminated, and by comparing it with the EndTime column you can tell whether or not the job was migrated. The original job is purged of its File records, and its Type field is changed from "B" to "M" to indicate that the job was migrated.

- As noted above, for the Migration High Bytes, the calculation of the bytes to migrate is somewhat approximate.

- If you keep Volumes of different Media Types in the same Pool, it is not clear how well migration will work. We recommend only one Media Type per pool.

- It is possible to get into a resource deadlock where Bacula Enterprise does not find enough drives to simultaneously read and write all the Volumes needed to do Migrations. For the moment, you must take care as all the resource deadlock algorithms are not yet implemented. You can also use the **Maximum Concurrent Read Jobs** directive to control the drive assignment behavior to reduce resource deadlocks. The MaximumConcurrentReadJobs should have a maximum of one less than one half the number of drives that are available for reading and writing. Doing so, will assure that for each read drive there is one that can write so that the Migration jobs will not stall due to a deadlock.

- Migration is done only when you run a Migration job. If you set a Migration High Bytes and that number of bytes is exceeded in the Pool no migration job will automatically start. You must schedule the migration jobs, and they must run for any migration to take place.

- If you migrate a number of Volumes, a very large number of Migration jobs may start.

- Figuring out what jobs will actually be migrated can be a bit complicated due to the flexibility provided by the regex patterns and the number of different options. Turning on a debug level of 100 or more will provide a limited amount of debug information about the migration selection process.

- Bacula Enterprise currently does only minimal Storage conflict resolution, so you must take care to ensure that you don't try to read and write to the same device or Bacula may block waiting to reserve a drive that it will never find. In general, ensure that all your migration pools contain only one Media Type, and that you always migrate to pools with different Media Types.

- The **Next Pool = ...** directive must be defined in the Pool referenced in the Migration Job to define the Pool into which the data will be migrated.

- Pay particular attention to the fact that data is migrated on a Job by Job basis, and for any particular tape Volume, only one Job can read that Volume at a time (no simultaneous read), so migration jobs that all reference the same Volume will run sequentially. This can be a potential bottle neck

and does not scale very well to large numbers of jobs. For disk Volumes, multiple simultaneous Jobs can read the same Volume at the same time, so the above restriction does not apply.

Available as of version 10.0.0

Since Bacula Enterprise version 10.0.0, most plugins are compatible with Copy/Migration jobs. Please refer to a plugin's specific documentation for more information.

1.2 Migration and Copy Directives

Job Resource Directives

The following directives can appear in a Director's Job resource, and they are used to define a Copy or Migration job.

Pool = <Pool-name> The Pool specified in the Copy/Migration control Job is not a new directive for the Job resource, but it is particularly important because it determines what Pool will be examined for finding JobIds to migrate. The exception to this is when **Selection Type = SQLQuery**, and although a Pool directive must still be specified, no Pool is used, unless you specifically include it in the SQL query. Note, in any case, the Pool resource defined by the Pool directive must contain a **Next Pool = …** directive to define the Pool to which the data will be migrated.

Type = Migrate Migrate is a new type that defines the job that is run as being a Migration Job. A Migration Job is a sort of control job and does not have any Files associated with it, and in that sense they are more or less like an Admin job. Migration jobs simply check to see if there is anything to Migrate then possibly start and control new Backup jobs to migrate the data from the specified Pool to another Pool. Note, any original JobId that is migrated will be marked as having been migrated, and the original JobId can nolonger be used for restores; all restores will be done from the new migrated Job.

Type = Copy Copy is a new type that defines the job that is run as being a Copy Job. A Copy Job is a sort of control job and does not have any Files associated with it, and in that sense they are more or less like an Admin job. Copy jobs simply check to see if there is anything to Copy then possibly start and control new Backup jobs to copy the data from the specified Pool to another Pool. Note that when a copy is made, the original JobIds are left unchanged. The new copies can not be used for restoration unless you specifically choose them by JobId. Also you subsequently delete a JobId that has a copy, the copy will be automatically upgraded to a Backup rather than a Copy, and it will subsequently be used for restoration.

Once again, a Copy Job cannot be used to restore unless you explicitly specify the Copy JobId during the restore command. If the original backup Job is deleted and there is a Copy of that backup Job, the Copy JobIds will be changed to be backup Jobs that can then be restored.

Selection Type = <Selection-type-keyword> The **<Selection-type-keyword>** determines how the copy/migration job will go about selecting what JobIds to migrate. In most cases, it is used in conjunction with a **Selection Pattern** to give you fine control over exactly what JobIds are selected. The possible values for **<Selection-type-keyword>** are:

- **SmallestVolume** This selection keyword selects the volume with the fewest bytes from the Pool to be migrated. The Pool to be migrated is the Pool defined in the Copy/Migration Job resource. The copy/migration control job will then start and run one copy/migration backup job for each of the Jobs found on this Volume. The Selection Pattern, if specified, is not used.

- **OldestVolume** This selection keyword selects the volume with the oldest last write time in the Pool to be migrated. The Pool to be migrated is the Pool defined in the Copy/Migration Job resource.

The copy/migration control job will then start and run one copy/migration backup job for each of the Jobs found on this Volume. The Selection Pattern, if specified, is not used.

- **Client** The Client selection type, first selects all the Clients that have been backed up in the Pool specified by the Copy/Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Client names, giving a filtered Client name list. All jobs that were backed up for those filtered (regexed) Clients will be migrated. The copy/migration control job will then start and run one copy/migration backup job for each of the JobIds found for those filtered Clients.

- **Volume** The Volume selection type, first selects all the Volumes that have been backed up in the Pool specified by the Copy/Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Volume names, giving a filtered Volume list. All JobIds that were backed up for those filtered (regexed) Volumes will be migrated. The copy/migration control job will then start and run one copy/migration backup job for each of the JobIds found on those filtered Volumes.

- **Job** The Job selection type, first selects all the Jobs (as defined on the **Name** directive in a Job resource) that have been backed up in the Pool specified by the Copy/Migration Job resource, then it applies the **Selection Pattern** (defined below) as a regular expression to the list of Job names, giving a filtered Job name list. All JobIds that were run for those filtered (regexed) Job names will be migrated. Note, for a given Job named, they can be many jobs (JobIds) that ran. The copy/migration control job will then start and run one copy/migration backup job for each of the Jobs found.

- **SQLQuery** The SQLQuery selection type, used the **Selection Pattern** as an SQL query to obtain the JobIds to be migrated. The Selection Pattern must be a valid SELECT SQL statement for your SQL engine, and it must return the JobId as the first field of the SELECT.

- **PoolOccupancy** This selection type will cause the Migration job to compute the total size of the specified pool for all Media Types combined. If it exceeds the **Migration High Bytes** defined in the Pool, the Migration job will migrate all JobIds beginning with the oldest Volume in the pool (determined by Last Write time) until the Pool bytes drop below the **Migration Low Bytes** defined in the Pool. This calculation should be consider rather approximate because it is made once by the Migration job before migration is begun, and thus does not take into account additional data written into the Pool during the migration. In addition, the calculation of the total Pool byte size is based on the Volume bytes saved in the Volume (Media) database entries. The bytes calculate for Migration is based on the value stored in the Job records of the Jobs to be migrated. These do not include the Storage daemon overhead as is in the total Pool size. As a consequence, normally, the migration will migrate more bytes than strictly necessary.

- **PoolTime** The PoolTime selection type will cause the Migration job to look at the time each JobId has been in the Pool since the job ended. All Jobs in the Pool longer than the time specified on **Migration Time** directive in the Pool resource will be migrated.

- **PoolUncopiedJobs** This selection which copies all jobs from a pool to an other pool which were not copied before is available only for copy Jobs.

Selection Pattern = <Quoted-string> The Selection Patterns permitted for each Selection-type-keyword are described above.

For the OldestVolume and SmallestVolume, this Selection pattern is not used (ignored).

For the Client, Volume, and Job keywords, this pattern must be a valid regular expression that will filter the appropriate item names found in the Pool.

For the SQLQuery keyword, this pattern must be a valid SELECT SQL statement that returns JobIds.

Purge Migration Job = <yes|no> This directive may be added to the Migration Job definition in the Director configuration file to purge the job migrated at the end of a migration.

Pool Resource Directives

The following directives can appear in a Director's Pool resource, and they are used to define a Migration job.

Migration Time = <time-specification> If a PoolTime migration is done, the time specified here in seconds (time modifiers are permitted – e.g. hours, etc.) will be used. If the previous Backup Job or Jobs selected have been in the Pool longer than the specified PoolTime, then they will be migrated.

Migration High Bytes = <byte-specification> This directive specifies the number of bytes in the Pool which will trigger a migration if a **PoolOccupancy** migration selection type has been specified. The fact that the Pool usage goes above this level does not automatically trigger a migration job. However, if a migration job runs and has the PoolOccupancy selection type set, the Migration High Bytes will be applied. Bacula does not currently restrict a pool to have only a single Media Type, so you must keep in mind that if you mix Media Types in a Pool, the results may not be what you want, as the Pool count of all bytes will be for all Media Types combined.

Migration Low Bytes = <byte-specification> This directive specifies the number of bytes in the Pool which will stop a migration if a **PoolOccupancy** migration selection type has been specified and triggered by more than Migration High Bytes being in the pool. In other words, once a migration job is started with **PoolOccupancy** migration selection and it determines that there are more than Migration High Bytes, the migration job will continue to run jobs until the number of bytes in the Pool drop to or below Migration Low Bytes.

Next Pool = <pool-specification> The Next Pool directive specifies the pool to which Jobs will be migrated. This directive is required to define the Pool into which the data will be migrated. Without this directive, the copy/migration job will terminate in error. For additional details, see the NextPool resource Chapter.

Storage = <storage-specification> The Storage directive specifies what Storage resource will be used for all Jobs that use this Pool. It takes precedence over any other Storage specifications that may have been given such as in the Schedule Run directive, or in the Job resource. We highly recommend that you define the Storage resource to be used in the Pool rather than elsewhere (job, schedule run, etc.).

1.3 Example Migration Job

When you specify a Migration Job, you must specify all the standard directives as for a Job. However, certain such as the Level, Client, and Fileset, though they must be defined, are ignored by the Migration job because the values from the original job used instead.

As an example, suppose you have the following Job that you run every night. To note: there is no Storage directive in the Job resource; there is a Storage directive in each of the Pool resources; the Pool to be migrated (File) contains a Next Pool directive that defines the output Pool (where the data is written by the migration job).

```
# Define the backup Job
Job {
    Name = "NightlySave"
    Type = Backup
    Level = Incremental     # default
    Client = rufus-fd
```

```
    Fileset = "Full Set"
    Schedule = "WeeklyCycle"
    Messages = Standard
    Pool = Default
}
# Default pool definition

Pool {
    Name = Default
    Pool Type = Backup
    AutoPrune = yes
    Recycle = yes
    Next Pool = Tape
    Storage = File
    LabelFormat = "File"
}
# Tape pool definition
Pool {
    Name = Tape
    Pool Type = Backup
    AutoPrune = yes
    Recycle = yes
    Storage = DLTDrive
}
# Definition of File storage device
Storage {
    Name = File
    Address = rufus
    Password = "xxx"
    Device = "File"  # same as Device in Storage daemon
    Media Type = File  # same as MediaType in Storage daemon
}
# Definition of DLT tape storage device
Storage {
    Name = DLTDrive
    Address = rufus
    Password = "yyy"
    Device = "HP DLT 80" # same as Device in Storage daemon
    Media Type = DLT8000 # same as MediaType in Storage daemon
}
```

Where we have included only the essential information – i.e. the Director, Fileset, Catalog, Client, Schedule, and Messages resources are omitted.

As you can see, by running the NightlySave Job, the data will be backed up to File storage using the Default pool to specify the Storage as File.

Now, if we add the following Job resource to this conf file.

```
Job {
    Name = "migrate-volume"
    Type = Migrate
    Level = Full
```

```
    Client = rufus-fd
    Fileset = "Full Set"
    Messages = Standard
    Pool = Default
    Maximum Concurrent Jobs = 4
    Selection Type = Volume
    Selection Pattern = "File"
}
```

and then run the job named **migrate-volume**, all volumes in the Pool named Default (as specified in the migrate-volume Job that match the regular expression pattern **File** will be migrated to tape storage DLTDrive because the **Next Pool** in the Default Pool specifies that Migrations should go to the pool named **Tape**, which uses Storage **DLTDrive**.

If instead, we use a Job resource as follows:

```
Job {
    Name = "migrate"
    Type = Migrate
    Level = Full
    Client = rufus-fd
    Fileset = "Full Set"
    Messages = Standard
    Pool = Default
    Maximum Concurrent Jobs = 4
    Selection Type = Job
    Selection Pattern = ".*Save"
}
```

All jobs ending with the name Save will be migrated from the File Default to the Tape Pool, or from File storage to Tape storage.

1.4 Example Copy Job

A copy Job, as its name indicates, creates a copy of an already existing backup Job. This can be used to change retention policy, storage media and/or where the data is physically stored. Bacula detects whether the reading and writing devices are in the same storage daemon and proceeds accordingly. In either case, the Storage Daemon currently holding the data to be copied acts as a File Daemon to the destination storage daemon.

Suppose these device definitions in `bacula-dir.conf`:

```
Autochanger {
   Name = Local-Storage
   Address = local-sd.bacula6.com
   SDPort = 9103
   Password = "SomeFancyPassword"
   Device = Local-FileChgr1
   Media Type = Local-MediaType
   Maximum Concurrent Jobs = 10
   Autochanger = Local-Storage
}
```

```
Autochanger {
    Name = Remote-Storage
    Address = remote-sd.bacula6.com
    SDPort = 9903
    Password = "SomeOtherFancyPassword"
    Device = Remote-FileChgr1
    Media Type = Remote-MediaType
    Maximum Concurrent Jobs = 10
    Autochanger = Remote-Storage
}
```

This Job copies the Jobs stored in Local-Storage to Remote-Storage:

```
Job {
    Name = "CopyLocal2Remote"
    Type = "Copy"
    JobDefs = "DefaultJob"
    NextPool = "Remote-Pool"
    Pool = "Local-Pool"
    SelectionType = PoolUncopiedJobs
    Storage = "Local-Storage"
    WriteBootstrap = "/opt/bacula/bsr/%c_%n.bsr"
}
```

Be careful to be sure the Pools use the right storages:

```
Pool {
    Name = Local-Pool
    Pool Type = Backup
    Storage = Local-Storage
    Recycle = yes
    AutoPrune = no
    Volume Retention = 1 year
    Maximum Volume Bytes = 50G
    Maximum Volumes = 100
    Label Format = "Local-Vol-"
}

Pool {
    Name = Remote-Pool
    Pool Type = Backup
    Storage = Remote-Storage
    Recycle = yes
    AutoPrune = no
    Volume Retention = 2 years
    Maximum Volume Bytes = 50G
    Maximum Volumes = 100
    Label Format = "Remote-Vol-"
}
```

The important definitions in local-sd.bacula6.com's `bacula-sd.conf`:

```
Director {
   Name = director.bacula6.com-dir
   Password = "SomeFancyPassword"
   ...
}

Autochanger {
   Name = Local-FileChgr1
   Device = Local-FileChgr1-Dev1, Local-FileChgr1-Dev2
   Changer Command = ""
   Changer Device = /dev/null
}

Device {
   Name = Local-FileChgr1-Dev1
   Media Type = Local-MediaType
   Archive Device = /backup/volumes
   LabelMedia = yes
   Random Access = yes
   AutomaticMount = yes
   RemovableMedia = no
   AlwaysOpen = no
   Maximum Concurrent Jobs = 5
}

Device {
   Name = Local-FileChgr1-Dev2
   Media Type = Local-MediaType
   Archive Device = /backup/volumes
   LabelMedia = yes
   Random Access = yes
   AutomaticMount = yes
   RemovableMedia = no
   AlwaysOpen = no
   Maximum Concurrent Jobs = 5
}
```

As well as the important definitions in the remote storage daemon's configuration (remote-sd.bacula6.com):

```
Director {
   Name = director.bacula6.com-dir
   Password = "SomeOtherFancyPassword"
   ...
}

Autochanger {
   Name = Remote-FileChgr1
   Device = Remote-FileChgr1-Dev1, Remote-FileChgr1-Dev2
   Changer Command = ""
   Changer Device = /dev/null
}
```

```
Device {
   Name = Remote-FileChgr1-Dev1
   Media Type = Remote-MediaType
   Archive Device = /some/other/folder
   LabelMedia = yes
   Random Access = Yes
   AutomaticMount = yes
   Volume Poll Interval = 0
   RemovableMedia = no
   AlwaysOpen = no
   Maximum Concurrent Jobs = 5
}

Device {
   Name = Remote-FileChgr1-Dev2
   Media Type = Remote-MediaType
   Archive Device = /some/other/folder
   LabelMedia = yes
   Random Access = Yes
   AutomaticMount = yes
   RemovableMedia = no
   Volume Poll Interval = 0
   AlwaysOpen = no
   Maximum Concurrent Jobs = 5
}
```

The Job output would be similar to:

```
08-Feb 10:15 director.bacula6.com-dir JobId 53: The following 1 JobId was
                                                chosen to be copied: 52
08-Feb 10:15 director.bacula6.com-dir JobId 53: Copying using JobId=52
                                                Job=BackupClient1.2017-02-08_
→10.14.41_10
08-Feb 10:15 director.bacula6.com-dir JobId 53: Start Copying JobId 53,
                                                Job=CopyLocal2Remote.2017-02-
→08_10.15.14_11
08-Feb 10:15 director.bacula6.com-dir JobId 53: Using Device "Local-FileChgr1-
→Dev1"
                                                to read.
08-Feb 10:15 director.bacula6.com-dir JobId 54: Created new Volume="Remote-
→Vol-0006",
                                                Pool="Remote-Pool", MediaType=
→"Remote-MediaType"
                                                in catalog.
08-Feb 10:15 director.bacula6.com-dir JobId 54: Using Device "Remote-
→FileChgr1-Dev1"
                                                to write.
08-Feb 10:15 remote-sd.bacula6.com JobId 54: Labeled new Volume "Remote-Vol-
→0006"
                                                on file device "Remote-FileChgr1-
→Dev1" (/some/other/folder).
```

```
08-Feb 10:15 remote-sd.bacula6.com JobId 54: Wrote label to prelabeled Volume
                                          "Remote-Vol-0006" on file device
                                          "Remote-FileChgr1-Dev1" (/some/
→other/folder)
08-Feb 10:15 sd.bacula6.com JobId 53: Ready to read from volume
                                      "Local-Vol-0005"
                                      on file device "Local-FileChgr1-Dev1" (/
→backup/volumes).
08-Feb 10:15 sd.bacula6.com JobId 53: Forward spacing Volume "Local-Vol-0005"
                                      to file:block 0:256.
08-Feb 10:15 sd.bacula6.com JobId 53: End of Volume at file 0 on device
                                      "Local-FileChgr1-Dev1" (/backup/
→volumes),
                                      Volume "Local-Vol-0005"
08-Feb 10:15 sd.bacula6.com JobId 53: Elapsed time=00:00:01,
                                      Transfer rate=19.16 M Bytes/second
08-Feb 10:15 remote-sd.bacula6.com JobId 54: Elapsed time=00:00:01,
                                          Transfer rate=19.16 M Bytes/
→second
08-Feb 10:15 remote-sd.bacula6.com JobId 54: Sending spooled attrs to
                                          the Director. Despooling 6,114␣
→bytes ...
08-Feb 10:15 director.bacula6.com-dir JobId 53: Bacula Enterprise
director.bacula6.com-dir 8.6.5 (10Aug16):
  Build OS:              x86_64-redhat-linux-gnu-bacula-enterprise redhat
  Prev Backup JobId:     52
  Prev Backup Job:       BackupClient1.2017-02-08_10.14.41_10
  New Backup JobId:      54
  Current JobId:         53
  Current Job:           CopyLocal2Remote.2017-02-08_10.15.14_11
  Backup Level:          Full
  Client:                fd.bacula6.com
  Fileset:               "Full Set" 2015-03-03 17:12:31
  Read Pool:             "Local-Pool" (From Job resource)
  Read Storage:          "Local-Storage" (From Pool resource)
  Write Pool:            "Remote-Pool" (From Job resource)
  Write Storage:         "Remote-Storage" (From Job resource)
  Catalog:               "MyCatalog" (From Client resource)
  Start time:            08-Feb-2017 10:15:16
  End time:              08-Feb-2017 10:15:18
  Elapsed time:          0 mins 1 secs
  Priority:              10
  SD Files Written:      22
  SD Bytes Written:      19,167,642 (19.16 MB)
  Rate:                  19.16 MB/s
  Volume name(s):        Remote-Vol-0006
  Volume Session Id:     3
  Volume Session Time:   1486558493
  Last Volume Bytes:     19,183,189 (19.18 MB)
  SD Errors:             0
  SD termination status: OK
  Termination:           Copying OK
```

2 Verify Jobs

This document explains the uses of Verify Jobs in **Bacula Enterprise**. One of them, found in most other backup packages, is used to verify that the data on the backup volumes is actually usable, which is done by reading backup data and checking if it's intact and matches the cataloged information. The other is the special functionality of **Bacula Enterprise** to work like a file integrity scanner, finding files that have changed unexpectedly, which might indicate a security issue.

The reader of this paper is expected to have a good understanding of **Bacula** in general, i. e. working with the configuration files and bconsole should be expected. Also, general system administration knowledge as required for the backed up environment is assumed. The terminology used with **Bacula** needs also to be known.

Bacula Enterprise's Verify Jobs allow not only to ensure integrity of the data written to storage media, but also to ensure that what exists in the file system still matches what was backed up. This feature can be used to detect changes to critical files, that are expected to remain static.

In addition, **Bacula Enterprise** also allows to run jobs that only, without actually backing up data, check file integrity against information previously collected by **Bacula**. Being highly configurable and inherently secure, this can be one important building block of an intrusion detection system.

With these features, **Bacula Enterprise** offers features not found in other backup software, while utilizing its own infrastructure that is required anyway. The area where **Bacula Enterprise** can be used extends well beyond a plain backup and recovery software into a tool for ensuring data integrity at several levels – not only creating redundancy, managing existing backup copies, and recovering data, but also ensuring data integrity both on its own media and on the source file systems.

Running Verify Jobs and properly tracking the job results can accordingly become an important part of auditing procedures and allows system administrators new levels of confidence into their data's security.

2.1 Verify Jobs Functions

In **Bacula**, Verify Jobs perform three important tasks to ensure data integrity:

1. Verify Volume Data Integrity

2. Verify Volume Metadata Integrity

3. Verify File System Integrity

While the goals are quite different, the setup for all types of Verify Jobs share a good deal of the configuration (which is due to the fact that they also share a good deal of program code). However, it is important to understand the differences between these types of verification.

Once data is backed up, it is important to ensure that the backed up data is stored correctly. Today, modern tape drives do *Read-after-Write* to detect errors writing data and re-write that data automatically, but this approach can only find errors occuring between the drive and the tape media. Data corruption happening at earlier stages, for example while transferring data from the host computer to the storage device can not be found using this technology.

As there are many – and, often, very complex – layers of data processing between the application and the storage media, data can become corrupted long before it arrives at the storage device.[1] This sort of problems can only be detected by the software generating the data. To do this detection, the stored data needs to be read and compared to what is expected on the storage media.

---

[1] Failing RAM, HBAs and interface cables are common causes for data corruption, but software failures or even malicious software may also be the cause of data corruption.

Verify Volume Data Integrity

Verifying the data written to a Volume is done using a `level = data` for a Verify Job of **Bacula**. Such a Job will read all data records from a Volume, and for each object encountered, will calculate the checksum and record the size. That information is then compared against the metadata as stored on the Volume. Doing so, the Storage Daemon is able to detect data corruption on the storage media.

These Verify Jobs do not compare the data against what is stored in the Catalog, but it does read all job-related data from the storage media, passes them to the Storage daemon, and does the checksum processing, so they can be an integral part of ensuring reliability of the whole stack of interfaces and hardware involved.

Data level Verify Jobs are most useful with less reliable storage media, for example traditional file systems which do not provide redundancy, error correction, and "scrubbing" features. So-called "Next-Generation File Systems" like ZFS or btrfs, or highly reliable tape systems, are typically less prone to data corruption on media, so these Verify Jobs may be less important with them.

Verify Volume Metadata Integrity

As reading the complete stored data is expensive in terms of network bandwidth, CPU cycles, and storage I/O used, Bacula allows to compare the metadata read from the media against what it stored in the catalog. With Bacula, this comparison is done on a Job-by-Job basis; other backup systems often verify complete volumes.

This sort of Verify Job is called a **VolumeToCatalog** verification. How to set up and run this sort of Job is explained in detail in chapter v2cjob.

These verifications are a bit more efficient in terms of storage bandwith consumption than the previously described data verifications, and they also help ensuring the catalog matches what is stored on the Volumes, i. e. they get a step further towards actual end-to-end data verification, including the middle layer of the **Bacula** catalog.

Verify File System Integrity

The term "File System Integrity" can be interpreted in two ways: First, the integrity of the file system at the technical level, to ensure that file metadata like permissions, or the file contents, do not change unexpectedly. In case of a disk failing, or faults at the file system driver level, comparing what is stored on disk against what was last backed up may indicate problems before other checks trigger – for example, a file system check is usually only run at system startup, and S.M.A.R.T. [2]_-monitoring is well known to **not** trigger in cases of failing hard disks.

Bacula implements a verification of the actual file system contents against what was last backed up as a **DiskToCatalog** verification. Again, not the contents of files is compared but only metadata stored in the Catalog; this keeps the "cost" of those verifications manageable as the backed up data does not need to be read from the storage media, but using checksums file contents changes are likely to be noticed. These verification Jobs are discussed in more detail in section d2cjob.

The other interpretation of "File System Integrity" aims at a higher level of integrity. The goal here is to detect changes to files that should not change during normal operations. For many unix / linux systems, the contents of the `/etc/` and `/sbin/`-directories are typical candidates: These are system configuration files and programs, which, except for very special cases, will only be intentionally modified by an administrator. As such, unexpected modifications to these files may indicate security issues like break-ins and unathorized attempts to change the system's behaviour or to install malicious software.

Bacula can keep track of those files by collecting their metadata in its catalog without actually saving them, and later compare the system's files against the cataloged information, which is done by a pair of jobs called **InitCatalog** and **Catalog**, respectively. This sort of verification is explained in cjob.

When comparing **Bacula Enterprise**'s verify features against those of other backup packages, a user may miss the ability to compare the file system's data against what is stored on backup media. We believe this feature is, in practice, not needed: A byte-per-byte comparison will only rarely be more reliable than the checksum-based verification **Bacula** implements, and the ability to separate the comparison into two distinct steps – Catalog vs. File System and vs. Volume data – allows more flexibility and better utilization of the available resources.

Linking together Bacula's **DiskToCatalog** and **VolumeToCatalog** jobs results in a "near-end-to-end" comparison, where discrepancies between the source file system and the data on the backup media will be reported. If this linkage is desired, it is possible to schedule the needed jobs in a way that they are executed directly after each other.

More In-Depth Verification

If real "end-to-end" data comparison is required, the only reliable way is to restore to a new file system location on the source system and compare data using tools like `diff`, as only then the data flow through all the involved components is verified. This scenario is the only one that can help finding problems at the source system's operating system and hardware level, for example.

Regularly running test restores and comparing the files against their originals is a procedure recommended by **Bacula Systems** to ensure proper operation of your backup environment. Verify jobs should only be used in addition to this approach, not to replace it.

In sensitive environments, it may be required to do those tests quite often, which results in the goal to do these tests automatically. In cases where this is required, some simple scripts may be created to trigger the needed restore, launch the comparison after the restore is completed, and finally report the results and remove no longer needed data from the set of restored files.

2.2 Verification Criteria

Bacula Enterprise can be configured to compare or to ignore different parts of a file's metadata during Verify Jobs. This configuration affects all variants of Bacula's Verify Jobs.

It is important to be aware that the actual contents of a file will never be used; instead, cryptographic checksums can be generated when reading a file, and are stored along with the other metadata in the catalog database. We will discuss those checksums in section *Checksums*.

The complete set of metadata stored in the catalog, and available for verification purposes, is described in detail in the Bacula Enterprise manual. [4]_ File ownership, permissions and times are available. Note that permissions only cover the basic unix file system permission bits; in particular, ACLs as used by linux, unix or Windows file systems are not included. [5]_

```
File Set {
  Name = LinuxSys
  Include {
    Options {
      Verify = pnugsicm1
    }
  File = /sbin
  File = /usr/sbin
```

(continues on next page)

```
  File = /boot
  File = /etc
  File = /lib


  }
}
```

Configuring which data to use during verification is done in the `File Set` resource used, in particular in the `Options` sub-resources of the `Include` part of each File Set. A File Set might look like the one shown above, where rather paranoid `Verify` options are specified. In this example, the SHA1 checksum is used. Using `MD5` instead – for example to save a bit of the client's CPU cycles – would be achieved by replacing 1 with 5. `SHA1` and `MD5` are mutually exclusive – you can use only one of them, while all the other option letters may be freely added (or concatented).

Bacula Systems recommends to always include ownership, permissions, size, inode and creation time metadata as well as a cryptographic checksum for security-related verification (i.e. **InitCatalog / Catalog** verifies), and ownership, permissions, and a checksum for integrity-related verification.

Checksums

Cryptographic checksums are an efficient way to verify data identity without actually doing byte-per-byte comparison. The underlying idea is to use a hash function that generates a checksum that is considerably shorter than the full data. As the checksum is shorter than the input data, different input data will generate identical checksums. The art of choosing the right checksumming algorithm is to use one that will generate a different checksum for small as well as large changes to the input data. Bacula supports two checksumming algorithms that both were developed for use in cryptograhic applications, and as such are well understood and known to minimize the risk of hash collisions.

MD5 is the older one, creating a hash of 128 bits length. Even though attacks are possible today, for purposes of data integrity checking it's still considered reasonably safe.

SHA1 is a newer algorithm, generating a checksum of 160 bits length. SHA1 is more cpu-intensive, but reduces the risk of different data producing identical hashes, so that it's considered more secure for data integrity purposes.

When using either of these algorithms on a really large number of files, hash collisions – different files with identical checksums – may happen. Adding other metadata, like file size, file times, or inode number as verification criteria is assumed to reduce the risk of incorrect identification of files. For data integrity verification, this is less of a problem, but it needs to be considered when using **Data Deduplication** with **Base Jobs** in **Bacula Enterprise** and using the same file sets for both purposes. [6]_

2.3 Volume Data Verification

These Jobs are not described in detail as the feature is not released.

It will, however, be important to use only one signature throughout all the ob, i. e. using a `File Set` with different `signature` = settings for differently matched files is not going to be supported.

Also, each file should have a signature generated.

These requirements, however, are not difficult to implement as it results when following best practices of **Bacula** usage.

2.4 Volume to Catalog Verification

If, after a backup job finished, you want to ensure that the backed up data was stored correctly and can be used by Bacula, a **VolumeToCatalog** verification job is the tool needed. In the simplest form, shown below, this sort of Verify Job verifies the last job run on the given client, but it is possible to configure which job to verify.

```
Job {
  Name = SomeJob
  JobDefs = DefaultJob
  Client = one-fd
  File Set = FullSet
  Schedule = SomeSch
}

Job {
  Name = Verify
  JobDefs = one-fd
  Client = one-fd
  Type = Verify
  Level = VolumeToCatalog
  Schedule = SomeSch
  Priority = 11
}
```

We assume that the priority of the ``DefaultJob`` is set to 10; the configuration is not complete.
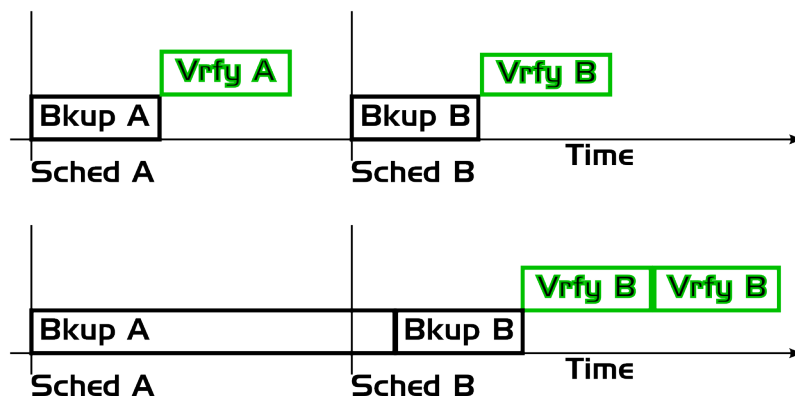
Fig. 1: Scheduling/ordering problems

Automatic selection of the job to verify fails if more than one job is run on a client and the jobs and their verifications get re-ordered, for example due to one job running longer than planned. Top: Planned scheduling. Bottom: Backup Job A runs beyond the scheduled time for backup and verify Job B, and eventually B gets verified twice, and A not at all.

As the simple approach requires a verify job directly after each backup job run (to ensure the correct order of jobs, priorities can be used), it is rather inflexible and can have the result that backup windows are exceeded.

The requirement of strict order of Backup and Verify Jobs brings lots of problems when more than one backup job per client is used, as it needs to be enforced by using dedicated schedules for each pair of job and related verification, and fails once the order is disturbed – which may easily happen when a single job runs longer than expected. See figure *Scheduling/ordering problems*.

To prevent this, Verify Jobs can also be run later, on a dedicated **Bacula** system typically during the working hours. In the configuration, this is done by defining which job is to be verified. The job with the given name that was run last will be chosen for verification. The above example configuration changes to the example configuration below, accordingly. Using this approach, verification of a large number of jobs on a specific client is more easily possible, and the risk of not verifying the correct jobs vanishes. Using priorities it's easily possible to ensure all verifications run after the backups. The downside here is that a higher number of jobs needs to be created and maintained. As **Bacula Systems** recommends to generate configurations for environments with many jobs automatically, adding the generation of the required verify jobs will not add much to the maintenance process in those cases.

```
Job {
  Name = SomeJob
  JobDefs = DefaultJob
  Client = one-fd
  File Set = FullSet
  Schedule = SomeSch
}

Job {
  Name = SomeVerify
  JobDefs = DefaultJob
  Type = Verify
  VerifyJob = SomeJob
  Level = VolumeToCatalog
  Schedule = SomeSch
  Priority = 11
}
```

The `File Set` configured in the verify job resource is, although required by the configuration file parser, not actually used; the comparison uses the file set actually used for the original backup. This simplifies the maintenance of Verify Jobs a lot, although Job Reports may appear misleading.

Verify Any Specified Job

It is possible to tell Bacula which exact Job should be verified instead of only automatically verifying the latest one.

This feature can be used with all Verify Job levels.

To verify a given job, just specify the Job's jobid in the argument when starting the verify job.

```
*run job=VerifyVolume jobid=1 level=VolumeToCatalog
Run Verify job
JobName:     VerifyVolume
Level:       VolumeToCatalog
Client:      127.0.0.1-fd
Fileset:     Full Set
Pool:        Default (From Job resource)
Storage:     File (From Job resource)
Verify Job:  VerifyVol.2010-09-08_14.17.17_03
Verify List: /tmp/regress/working/VerifyVol.bsr
When:        2010-09-08 14:17:31
Priority:    10
OK to run? (yes/mod/no):
```

Operational Considerations

Quite naturally, a Verify Job consumes time, network bandwidth and CPU cycles. On the client, resource usage during a **VolumeToCatalog** verification is much less than when running a backup, as the file system is not accessed for each file verified, and no files are read as during a backup.

The main resources used for this verification are the catalog database and the storage device needed to read the job data. The critical element, typically, is the storage device which is unavailable for other tasks during Verify Jobs. For that reason, **Bacula Systems** recommends to run Verify Jobs outside the regular backup time window, so backup jobs are not negatively affected.

It is also possible to not run these Verify Jobs daily or after each backup, but only at selected times. One useful approach is to verify the jobs that have data stored on volumes that are to be removed for off-site storage. Implementing such a scheme might require a bit of additional scripting, but may be a valuable addition to existing volume handling procedures.

2.5 Job Results

A **VolumeToCatalog** Verify Job that does not find any problems creates a job report that looks similar to the one shown below.

```
main-dir JobId 2: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build 0s:               x86_64-unknown-linux-gnu suse 11.1
  JobId:                  2
  Job:                    Verify-Last -RH1-Job-Volume.2010-10-17_00.04.16_21
  Fileset:                Test Set
  Verify Level:           VolumeToCatalog
  Client:                 s-rhi-fd
  Verify JobId:           1
  Verify Job:
  Start time:             17-Oct-2010 00:04:18
  End time:               17-Oct-2010 00:04:25
  Files Expected:         2,216
  Files Examined:         2,216
  Non-fatal FD errors:    0
  FD termination status:  OK
  SD termination status:  OK
  Termination:            Verify OK
```

```
dir JobId 3: Verifying against JobId=2 Job=RH1-config.2010-10-17_00.03.08_20
dir JobId 3: Bootstrap records written to /opt/bacula/main-dir.restore.bsr
dir JobId 3: Start Verify JobId=3 Level=VolumeToCatalog Job=Verify-Last-RH1-
→Job-Volume.2010-10-17_00.16.06_43
dir JobId 3: Using Device "FileStorage"
sd JobId 3: Ready to read from volume "F-8" on device "FileStorage" (/data)
sd JobId 3: Forward spacing Volume "F-8" to file: block 0:1359
sd JobId 3: Error: block.c:1098 Volume data error at 0:1359! Short block of␣
→2646 bytes on device "FileStorage" (/data) discarded
sd JobId 3: Error: read_record.c:150 block.c:1098 Volume data error at 0:1359!
→ Short block of 2646 bytes on device "FileStorage" (/data) discarded
sd JobId 3: End of Volume at file 0 on device "FileStorage" (/data), Volume
→"F-8"
sd JobId 3: End of all volumes.
```

```
dir JobId 3: Warning: The following files are in the Catalog but not on the␣
↪Volume(s):
dir JobId 3:          /etc/sestatus conf
    ... many lines skipped
dir JobId 3:          /etc/yum/yum-updatesd.conf
dir JobId 3: Error: Bacula Enterprise s-su-1-dir 4.0.4 (04Sep10):
17-Oct-2010 00:16:10
  Build OS:              x86_64-unknown-linux-gnu suse 11.1
  JobId:                 24473
  Job:                   Verify-Last-RH1-Job-Volume.2010-10-17_00.16.06_43
  Fileset:               Test Set
  Verify Level:          VolumeToCatalog
  Client:                s-rhl-fd
  Verify JobId:          24458
  Verify Job:
  Start time:            17-Oct-2010 00:16:08
  End time:              17-Oct-2010 00:16:10
  Files Expected:        2,216
  Files Examined:        0
  Non-fatal FD errors:   0
  FD termination status: OK
  SD termination status: OK
  Termination:           *** Verify Error ***
```

We show the job report for the same volume to catalog verification, but this time the volume has been damaged. Initially, some informational messages from the **storage daemon (sd)** show up, indicating the volume problem. As a result of the inability to properly read that volume, lateron, most of the backed up files could not be found in the volume. Accordingly, the job fails with "*** Verify Error ***".

Note that in the job reports introduced above, the job to verify against is not set, i.e. it is selected automatically. Setting the job to verify against by adding `Verify Job = <job name>` to the job resource results in a job report as shown below.

```
main-dir JobId 3: Verifying against JobId=2 Job=RHi-sbin.2010-10-17_01.25.35_
↪48
main-dir JobId 3: Bootstrap records written to /opt/bacula/main-dir.restore.
↪bsr
main-dir JobId 3: Start Verify JobId=3 Level=VolumeToCatalog Job=Verify-RH1-
↪sbin-Job-Volume .2010-10-17_01.30.39_52
main-dir JobId 3: Using Device "FileStorage"
main-sd JobId 3: Ready to read from volume "F-2" on device "FileStorage" (/
↪data)
main-sd JobId 3: Forward spacing Volume "F-2" to file:block 0:191.
main-dir JobId 3: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build OS:              x86_64-unknown-linux-gnu suse 11.1
  JobId:                 3
  Job:                   Verify-RH1-sbin-Job-Volume.2010-10-17_01.30.39_52
  Fileset:               Test Set
  Verify Level:          VolumeToCatalog
  Client:                s-rhi-fd
  Verify JobId:          2
  Verify Job:            RHi-sbin
```

```
 Start time:             17-Oct-2010 01:30:41
 End time:               17-Oct-2010 01:30:43
 Files Expected:         279
 Files Examined:         279
 Non-fatal FD errors:    0
 FD termination statu:   OK
 SD termination status:  OK
 Termination:            Verify OK

main-dir JobId 24478: Begin pruning Jobs older than 3 hours.
main-dir JobId 24478: No Jobs found to prune.
main-dir JobId 24478: Begin pruning Jobs.
main-dir JobId 24478: No Files found to prune.
main-dir JobId 24478: End auto prune.
```

This is the approach to use if there is more than one job defined for a client, and the backup and verify jobs need to be separated from each other. The configuration used to create the above examples is shown in figures below.

```
Job {
  Name = Verify-Last-RH1-Job-Volume
  Type = Verify
  Level = VolumeToCatalog
  Client = s-rh1-fd
  JobDefs = Default Job
}

Job {
  Name = Verify-RH1-sbin-Job-Volume
  Type = Verify
  Verify Job = RH1-sbin
  Level = VolumeToCatalog
  Client = s-rh1-fd
  JobDefs = Default Job
}

Job {
  Name = RH1-config
  Type = Backup
  Level = Full
  Client = s-rh1-fd
  File Set = etc-fs
  JobDefs = Default Job
}

Job {
  Name = RH1-sbin
  Type = Backup
  Level = Full
  Client = s-rh1-fd
  File Set = sbin-fs
  JobDefs = Default Job
```

```
}
```

```
Fileset {
  Name = etc-fs
  Include {
    Options {
      Verify = pnugsicm1
      Signature = SHA1
    }
    File = /etc
  }
}

Fileset {
  Name = sbin-fs
  Include {
    Options {
      Verify = pinsmc5
      Signature = MD5
    }
    File /sbin
  }
}
```

```
JobDefs {
  Name = "Default Job"
  Type = Backup
  Level = Incremental
  Client = s-su-1-fd
  Fileset = "Test Set"
  Schedule = "HourlyCycle"
  Storage = File
  Messages = Standard
  Pool = Tier1
  Priority = 10
  Write Bootstrap = "/var/lib/bacula/%n.bsr"
}
Pool {
  Name = Tier1
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 1 days
  Maximum Volume Jobs = 5
  Volume Use Duration = 1 days
  Maximum Volumes = 50
  Maximum Volume Bytes = 4G
  Scratch Pool = Scratch
  Recycle Pool = Scratch
  Next Pool = Tier2 # not needed here
  Storage = File # should not need details
```

```
}
Client {
  Names-rh1-fd
  Address = 1sb-254.1sb.os.bacula systems.com
  FDPort = 9102
  Catalog = MyCatalog
  Password = "rhifd"
  File Retention = 2 hours
  Job Retention = 3 hours
  Auto Prune = Yes
}
```

This configuration will be re-used and enhanced for the following discussions of other Verify Job types.

With **VolumeToCatalog** verification set up and providing good results, the first important step towards a continuously verified backup solution has been taken. [7]_ In the following chapters, we will build on the knowledge gained and the existing configuration, adding more verifications.

2.6 Disk to Catalog Verification

We will now add the next step towards "end-to-end" integrity checking of your backups – the **DiskTo-Catalog** verification. We first complete our verification job of the /etc directory on the client, and then add the needed job to verify the source files against the catalog by adding the following job resources to our configuration:

```
Job {
  Name = Verify-RH1-etc-Job-Volume
  Type = Verify
  Verify Job = RH1-config
  Level = VolumeToCatalog
  Client = s-rh1-fd
  JobDefs = DefaultJob
}

Job {
  Name = Verify-RH1-etc-Job-Disk
  Type = Verify
  Verify Job = RH1-config
  Level = DiskToCatalog
  Client = s-rh1-fd
  JobDefs = DefaultJob
}
```

The configuration should be rather self-explanatory by now – `Verify-RH1-etc-Job-Volume` ensures we can run the verification of the latest backup on media any time, and `Verify-RH1-etc-Job-Disk` allows us to run the corresponding verification against the source files.

If we run the **CatalogToDisk** verification, we get a result as shown in the figure below.

```
main-dir JobId 28: Verifying against JobId=26 Job=RHi-config.2010-10-17_01.25.
→44 _49
main-dir JobId 28: Start Verify JobId=28 Level=DiskToCatalog Job=Verify-
```

```
→RHi-etc-Job-Disk.2010-10-17 _02.21.23_57
main-dir JobId 28: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build:                 x86_64-unknown-linux-gnu suse 11.1
  JobId:                 28
  Job:                   Verify-RH1-etc-Job-Disk.2010-10-17 02.21.23 _57
  Fileset:               etc-fs
  Verify Level:          DiskToCatalog
  Client:                s-rhi-fd
  Verify JobId:          26
  Verify Job:            RH1-config
  Start time:            17-Oct-2010 02:2
  End time:              17-Oct-2010 02:2
  Files Examined:        2,216
  Non-fatal FD errors:   0
  FD termination status: OK
  Termination:           Verify OK


main-dir JobId 24481: Begin pruning Jobs older than 3 hours.
main-dir JobId 24481: No Jobs found to prune.
main-dir JobId 24481: Begin pruning Jobs.
main-dir JobId 24481: Pruned Files from 2 Jobs for client s-rhi-fd from
→catalog.
main-dir JobId 24481: End auto prune.
```

We will now change the verified metadata of one of the files verified, add a new file, and remove a file
that existed at the time of the latest backup:

```
[root@lsb-254 etc]# pwd
/etc
[root@lsb-254 etc]# touch resolv.conf
[root@lsb-254 etc]# touch NEWFILE
[root@lsb-254 etc]# mv dnsmasq.conf /tmp/
```

The job report from a subsequent Verify-RH1-etc-Job-Disk job clearly indicates the differences
we created (actually, it does more – it implies that /etc/resolv.conf and /etc/sysconfig/
networking/profiles/default/resolv.conf are hard links to the same data).

```
main-dir JobId 25: Verifying against JobId=24 Job=RH1-config.2010-10-17_01.25.
→44_49
main-dir JobId 25: Start Verify JobId=25 Level=DiskToCatalog
                  Job=Verify-RH1-etc-Job-Disk.2010-10-17_02.31.25_58
main-dir JobId 25: File: /etc/sysconfig/networking/profiles/default/resolv.
→conf
main-dir JobId 25: st_ctime differs
main-dir JobId 25: st_mtime differs
main-dir JobId 25: File: /etc/resolv.conf
main-dir JobId 25: st_ctime differs
main-dir JobId 25: st_mtime differs
main-dir JobId 25: New file: /etc/NEWFILE
main-dir JobId 25: File: /etc/
main-dir JobId 25: st_ctime differs
main-dir JobId 25: st_mtime differs
```

```
main-dir JobId 25: Warning: The following files are in the Catalog but not on␣
↪disk:
main-dir JobId 25: /etc/dnsmasq. conf
main-dir JobId 25: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build:                  x86_64-unknown-linux-gnu suse 11.1
  JobId:                  25
  Job:                    Verify-RH1-etc-Job-Disk.2010-10-17 02.31.25_58
  Fileset:                etc-fs
  Verify Level:           DiskToCatalog
  Client:                 s-rhi-fd
  Verify JobId:           24
  Verify Job:             RH1-config
  Start time:             17-Oct-2010 02:31:27
  End time:               17-Oct-2010 02:31:40
  Files Examined:         2,216
  Non-fatal FD errors:    0
  FD termination status:  OK
  Termination:            Verify Differences
```

This verification just checked "real" metadata and does not prove that file contents changes will be noticed. However, a job report containing lines like

```
File: /etc/hosts
      st_size  differ. Cat: 187 File: 222
      st_ctime differs
      st_mtime differs
      SHA1 digest differs. File=CRPld0oeAo4ugQ2/HCV0/JJe/EY␣
↪Cat=rZUB6O05vQ3P28HUvi9B/O7917Y
```

indicates just those sort of changes. (With the File Set used in the example, the size and file time changes are also noted, of course.)

We have seen how easy it is to set up **CatalogToDisk** verification; it gets even easier once the configuration is generated automatically, as only very minimal information in the job resource needs to be adjusted. We have also seen that changes to the files on disk are noticed and reported, including the actual information that has changed.

Operational Considerations

A Bacula administrator must be aware that a verification against the source files uses resources on the client system – files are searched for using essentially the same process as during a backup, and when checksumming is done (Signature in the File Set options being set), these files are also entirely read from disk and the checksum is generated, which uses considerable CPU power.

The needed network interaction between **director (dir)** and **file daemon (fd)** is usually not that critical.

To do all the configured checks, the catalog database will be required. The queries it has to process for verify jobs are usually not more challenging than running backup and restore jobs.

Bacula Systems, for the reasons outlined, recommends to run **CatalogToDisk** verifications out of the business hours of the backed up machine (if possible, run them during the backup window), and to only run them for jobs where the additional effort is really required. It also may not be necessary to verify each single job – in many cases, only verifying the (monthly) full backups or jobs that are going to be used for long-term archival may be sufficient.

If you want to ensure that both data integrity verifications – catalog to disk and to volume – happen close to each other, several approaches can be used.

The one that integrates most easily with normal **Bacula** operations is a schedule to run those jobs, possibly scheduling to the same time the backup job runs, but using a lower priority setting.

Alternatively, it would seem possible to explicitly link the jobs using the `Run` directive in the backup job. This, unfortunately, will not give the desired result as jobs started like this are run **before** the original job, which would verify against an older backup. The solution seems to be using `Run Script`s and use `Console` commands to start the desired jobs. Again, this does not work, as the `run` command is not allowed in `Run Script` sub-resources.

So, the most reasonable way is to use a simple shell script to be called by **Bacula** after the backup job finishes, which starts the two verify jobs.

```
Job {
  Name = RH1-config
  Type = Backup
  Level = Full
  Run Script {
    Runs When = After
    Fail Job on Error = No
    Runs on Client = No
    Command = "/opt/bacula/scripts/rh1-vrfy.sh"
  }
  Client = s-rh1-fd
  File Set = etc-fs
  JobDefs = DefaultJob
}
```

Fig. 2: Adding a Run Script to the Backup Job

```
#!/bin/bash
echo "run job=Verify-RH1-etc-Job-Volume yes
run job=Verify-RH1-etc-Job-Disk yes
quit" | /opt/bacula/bin/bconsole 2>&1 >/dev/null
echo "Verify jobs started!"
```

Fig. 3: Script to Execute Jobs through bconsole

In the DIRs configuration, you would change the job configuration to the one shown in figure *Adding a Run Script to the Backup Job*, and the referenced script could be something as simple as the one shown in figure *Script to Execute Jobs through bconsole*.

If using **Bacula**'s integrated scheduler is not desired, the script mentioned above could also be used to trigger the needed jobs through `cron` or any other job scheduler.

The solution we now have verifies both "ends" of a backup job shortly after the original backup was run – delays may happen when other jobs block the needed resources for the Verify Jobs, or when controlling the job execution order through priorities to ensure all backups run as quickly as possible, making best use of the available backup window.

We have seen that the Verify Jobs we set up detect changes both on the backup media (ensuring volume integrity) and on the original file system (ensuring data integrity). This sounds like we've already implemented some **Tripwire**-like functionality[8] to detect tampering of critical files. Why this is not the case, and how to better implement that functionality is discussed in the next chapter.

---

[8] Tripwire is a well-known data integrity tool available both as open-source or commercial software. See www.tripwire.org for more information.

2.7 InitCatalog/Catalog Verification

The one shortcoming of using **DiskToCatalog** verification, if integrity checking of important files is desired, is that the verification process shown above only looks at files actually being saved. This approach may fail when a changed file is not actually being backed up, which will happen if

1. A file is modified in a way that does not change its `ctime` or `mtime` metadata,

2. Full backups are not run against the interesting set of files **[9]_**, and

3. if Differential or Incremental backups are run, those are not set up to not use `accurate` mode with settings capturing those files.

In this situation, there simply may not exist a backup which includes the modified file, and accordingly, that file would not be verified.

The simple solution would be to run Full backups of the interesting files daily (or even more often). This would require more backup jobs to exist than required per an organizations backup policy, and it will also waste storage space. For those reasons, **Bacula Enterprise** offers another solution to this challenge: **InitCatalog / Catalog** verification.

Essentially, a Verify Job at level InitCatalog goes through a `File Set`, records all the metadata as configured, but does not actually back up any data. The result is that all the needed information to verify exists in the catalog. The next step then is to run a Verify Job against that original job. This is done with Verify Jobs at level **Catalog**.

```
Job {
  Name = Verify-RH1-sbin-Job-Catalog
  Type = Verify
  Level = Catalog
  Client = s-rh1-fd
  File Set = sbin-fs
  JobDefs = DefaultJob
}
```

A Job to implement this might look like the one shown in figure above.

We first generate the initial information in the catalog by issuing a command like `run level=InitCatalog job=Verify-RH1-sbin-Job-Catalog yes` to bconsole. The resulting Job Report should look similar to the one shown in figure **fig:cvinit**.

```
main-dir JobId 4: Start Verify JobId=4 Level=InitCatalog Job=Verify-RH1-sbin-
↪Job-Catalog.2010-10-17_22.28.10_12
main-dir JobId 4: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build:                x86_64-unknown-linux-gnu suse 11.1
  JobId:                24489
  Job:                  Verify-RH1-sbin-Job-Catalog.2010-10-17_22.28.10_12
  Fileset:              sbin-fs
  Verify Level:         InitCatalog
  Client:               s-rh1-fd
  Verify JobId:         0
  Verify Job:
  Start time:           17-Oct-2010 22:28:12
  End time:             17-Oct-2010 22:28:14
  Files Examined:       279
  Non-fatal FD errors:  0
```

(continues on next page)

```
  FD termination status:  OK
  Termination:            Verify OK
```

Running the actual verification job can be done manually, through `bconsole`, `BAT`, or `BWeb`, or by scheduling it as usual (typically outside the backup window, or using priorities to prevent it from real backup jobs running in time). Note that there's no need to schedule **InitCatalog** levels of this job – when verifying files that are expected to be static this would severely reduce the ability to detect file changes!

A Job Report for a Catalog verification without any changes detected is shown in figure **fig:vcok**, and one with discrepancies is shown in figure **fig:vcnok**.

```
main-dir JobId 29: Verifying against JobId=28 Job=Verify-RH1-sbin-Job-Catalog.
↪2010-10-17_22.28.10_12
main-dir JobId 29: Start Verify JobId=29 Level=Catalog Job=Verify-RH1-sbin-
↪Job-Catalog.2010-10-17_22.39.47_14
main-dir JobId 29: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build:                  x86_64-unknown-linux-gnu suse 11.1
  JobId:                  24490
  Job:                    Verify-RH1-sbin-Job-Catalog.2010-10-17_22.39.47_14
  Fileset:                sbin-fs
  Verify Level:           Catalog
  Client:                 s-rh1-fd
  Verify JobId:           24489
  Verify Job:
  Start time:             17-Oct-2010 22:39:49
  End time:               17-Oct-2010 22:39:50
  Files Examined:         279
  Non-fatal FD errors:    0
  FD termination status:  OK
  Termination:            Verify OK
```

```
main-dir JobId 29: Verifying against JobId=28 Job=Verify-RH1-sbin-Job-Catalog.
↪2010-10-17_22.28.10_12
main-dir JobId 29: Start Verify JobId=29 Level=Catalog Job=Verify-RH1-sbin-
↪Job-Catalog.2010-10-17_22.47.10_15
main-dir JobId 29: File: /sbin/grub-install
main-dir JobId 29:        st_ino   differ. Cat: 229424 File: 229651
main-dir JobId 29:        st_size  differ. Cat: 18521 File: 18538
main-dir JobId 29:        st_mtime differs
main-dir JobId 29:        st_ctime differs
main-dir JobId 29:        MD5 digest differs. File=U8f9+qfyGoNZY9Fmf0j9hQ↪
↪Cat=7cLXtYBAGwQNMU4yz+UFoQ
main-dir JobId 29: New file: /sbin/ls
main-dir JobId 29: File: /sbin/
main-dir JobId 29:        st_mtime differs
main-dir JobId 29:        st_ctime differs
main-dir JobId 29: Warning: The following files are in the Catalog but not on↪
↪disk:
main-dir JobId 29:        /sbin/mkinitrd
main-dir JobId 29: Bacula Enterprise main-dir 4.0.4 (04Sep10):
  Build:                  x86_64-unknown-linux-gnu suse 11.1
  JobId:                  29
```

```
Job:                    Verify-RH1-sbin-Job-Catalog.2010-10-17_22.47.10_15
Fileset:                sbin-fs
Verify Level:           Catalog
Client:                 s-rh1-fd
Verify JobId:           28
Verify Job:
Start time:             17-Oct-2010 22:47:12
End time:               17-Oct-2010 22:47:13
Files Examined:         279
Non-fatal FD errors:    0
FD termination status:  OK
Termination:            Verify Differences
```

Only after explicitly allowed changes to the files covered by those Verify Jobs the job has to be run at **InitCatalog** level again. This would be the case after configuration changes or system updates. Notifying system or application administrators about this requirement is easily done and adds only minimal overhead to your existing maintenance procedures.

This sort of jobs allows to verify data integrity of files not being backed up at all, or not backed up recently. For security-aware organizations, this functionality may be essential. Seeing that **Bacula** can be one tool to do this, the question is: **Why use Bacula for File Integrity checking?**

There are several good reasons for it:

- **Bacula Enterprise** is based on open source, and as such may be more trusted than a purely closed source solution.

- You may already use **Bacula Enterprise** for backups, and it may be desirable not to install and maintain another software package.

- **Bacula Enterprise** keeps its metadata, which it uses for integrity checking, off the checked systems, which is the only reliable way to prevent tampering with the baseline information once a system is compromised.

- As a backup software is security relevant, a Bacula Enterprise installation, its data (like the catalog) and access to it should be closely controlled and well audited anyway; not having to maintain the environment of another software system under tight control may be a noticeable reduction of management effort.

- Verification reports from **Bacula Enterprise** can be easily fed into monitoring or ticketing systems, which may not be easily possible for alternative integrity scanners. [10]_

- **Bacula Enterprise** is highly customizable; it may allow you more flexibility and more detailed checks than other integrity solutions.

Bacula Systems accordingly believes that **Bacula** may be a good choice of a software package to use for file integrity scanning.

2.8 Using Verify Jobs to Improve Computer Security

Since **Bacula** maintains a Catalog of files, their attributes, and either SHA1 or MD5 signatures, it can be an ideal tool for improving computer security. This is done by making a snapshot of your system files with a **Verify** Job, and then, on a regular basis (e.g. nightly), checking the current state of your system against the snapshot.

The first step is to set up a **Verify** Job and to run it with:

```
Level = InitCatalog
```

The **InitCatalog** level tells **Bacula** simply to get the information on the specified files and to put it into the Catalog. That is, your database is initialized and no comparison is done. The **InitCatalog** is normally run one time manually.

Thereafter, you will run a Verify Job on a daily (or other) basis with:

```
Level = Catalog
```

The **Level = Catalog** level tells **Bacula** to compare the current state of the files on the Client to the last **InitCatalog** that is stored in the Catalog and to report any differences.

You decide what files you want to form your "snapshot" by specifying them in a **Fileset** resource, and normally, they will be system files that do not change, or that only their certain features change.

Then, you decide what attributes of each file you want compared by specifying comparison options on the **Include** statements that you use in the **Fileset** resource of your **Catalog** Jobs.

Details

In the discussion that follows, we will make reference to the Verify Configuration Example section. You might want to look over it now to get an idea of what it does.

The main elements consist of adding a schedule, which will normally be run daily, or perhaps more often. This is provided by the **VerifyCycle** Schedule, which runs at 5:05 in the morning every day.

Then, you must define a Job. We recommend that the Job name contain the name of your machine as well as the word "Verify" or "Check". In the example, we named it "MatouVerify". This will permit you to easily identify your Job when running it from the Console.

You will notice that most records of the Job are quite standard, but that the **Fileset** resource contains **verify=pins1** option in addition to the standard **signature=SHA1** option. If you don't want SHA1 signature comparison, and we cannot imagine why not, you can drop the **signature=SHA1** and none will be computed nor stored in the Catalog. Or alternatively, you can use **verify=pins5** and **signature=MD5**, which will use the MD5 hash algorithm. The MD5 hash computes faster than SHA1, but is cryptographically less secure.

The **verify=pins1** is ignored during the **InitCatalog** Job, but is used during the subsequent **Catalog** Jobs to specify what attributes of the files should be compared to those found in the Catalog. **pins1** is a reasonable set to begin with, but you may want to look at the details of these and other options. They can be found in the Fileset Resource section. Briefly, however, the **p** of the **pins1** tells Verify to compare the permissions bits, the **i** is to compare inodes, the **n** causes comparison of the number of links, the **s** compares the file size, and the **1** compares the SHA1 checksums (this requires the **signature=SHA1** option to have been set also).

You must also specify the **Client** and the **Catalog** resources for your Verify job, but you probably already have them created for your Client and do not need to recreate them, they are included in the example for completeness.

As mentioned above, you will need to have a **Fileset** resource for the Verify Job, which will have the additional **verify=pins1** option. You will want to take some care in defining the list of files to be included in your **Fileset**. Basically, you will want to include all system (or other) files that should not change on your system. If you select files, such as log files or mail files, which are constantly changing, your automatic Verify Job will be constantly finding differences. The objective in forming the Fileset is to choose all unchanging important system files. Then, if any of those files have changed, you will be notified, and you can determine if it changed because you loaded a new package, or because someone has broken into your computer and modified your files.

Running the Verify

The first thing you will want to do is to run an **InitCatalog** level Verify Job. This will initialize the Catalog to contain the file information that will later be used as a basis for comparisons with the actual file system, thus allowing you to detect any changes (and possible intrusions into your system).

The easiest way to run the **InitCatalog** is manually with the Console program by simply entering **run**. You will be presented with a list of Jobs that can be run, and you will choose the one that corresponds to your Verify Job, **MatouVerify** in this example.

```
The defined Job resources are:
    1: MatouVerify
    2: kernsrestore
    3: Filetest
    4: kernsave
Select Job resource (1-4): 1
```

Next, the console program will show you the basic parameters of the Job and ask you:

```
Run Verify job
JobName: MatouVerify
Fileset: Verify Set
Level:
Catalog
Client:
MatouVerify
Storage: DLTDrive
Verify Job:
Verify List: /tmp/regress/working/MatouVerify.bsr
OK to run? (yes/mod/no): mod
```

Here, you want to respond **mod** to modify the parameters because the Level is by default set to **Catalog** and we want to run an **InitCatalog** Job. After responding **mod**, the Console will ask:

```
Parameters to modify:
    1: Level
    2: Storage
    3: Job
    4: Fileset
    5: Client
    6: When
    7: Priority
    8: Pool
    9: Verify Job
Select parameter to modify (1-5): 1
```

you should select number 2 to modify the **Level**, and it will display:

```
Levels:
    1: Initialize Catalog
    2: Verify Catalog
    3: Verify Volume to Catalog
    4: Verify Disk to Catalog
    5: Verify Volume Data
Select level (1-4): 1
```

Choose item 1, and you will see the final display:

```
Run Verify job
JobName: MatouVerify
Fileset: Verify Set
Level:
Initcatalog
Client:
MatouVerify
Storage: DLTDrive
Verify Job:
Verify List: /tmp/regress/working/MatouVerify.bsr
OK to run? (yes/mod/no): yes
```

at which point you respond **yes**, and the Job will begin.

Thereafter, the Job will automatically start according to the schedule you have defined. If you wish to immediately verify it, you can simply run a Verify **Catalog** which will be the default. No differences should be found.

To use a previous Job, you can add `jobid=xxx` option in **run** command line. It will run the Verify Job against the specified Job.

```
*run jobid=1 job=MatouVerify
Run Verify job
JobName: MatouVerify
Level: Catalog
Client: 127.0.0.1-fd
Fileset: Full Set
Pool: Default (From Job resource)
Storage: File (From Job resource)
Verify Job: MatouVerify.2010-09-08_15.33.33_03
Verify List: /tmp/regress/working/MatouVerify.bsr
When: 2010-09-08 15:35:32
Priority: 10
OK to run? (yes/mod/no):
```

What to Do When Differences Are Found

If you have setup your messages correctly, you should be notified if there are any differences and exactly what they are. For example, below is the email received after doing an update of OpenSSH:

```
HeadMan: Start Verify JobId 83 Job=RufusVerify.2002-06-25.21:41:05
HeadMan: Verifying against Init JobId 70 run 2002-06-21 18:58:51
HeadMan: File: /etc/pam.d/sshd
HeadMan: st_ino differ. Cat: 4674b File: 46765
HeadMan: File: /etc/rc.d/init.d/sshd
HeadMan: st_ino differ. Cat: 56230 File: 56231
HeadMan: File: /etc/ssh/ssh_config
HeadMan: st_ino differ. Cat: 81317 File: 8131b
HeadMan: st_size differ. Cat: 1202 File: 1297
HeadMan: SHA1 differs.
HeadMan: File: /etc/ssh/sshd_config
HeadMan: st_ino differ. Cat: 81398 File: 81325
HeadMan: st_size differ. Cat: 1182 File: 1579
HeadMan: SHA1 differs.
HeadMan: File: /etc/ssh/ssh_config.rpmnew
HeadMan: st_ino differ. Cat: 812dd File: 812b3
HeadMan: st_size differ. Cat: 1167 File: 1114
HeadMan: SHA1 differs.
HeadMan: File: /etc/ssh/sshd_config.rpmnew
HeadMan: st_ino differ. Cat: 81397 File: 812dd
HeadMan: st_size differ. Cat: 2528 File: 2407
HeadMan: SHA1 differs.
HeadMan: File: /etc/ssh/moduli
HeadMan: st_ino differ. Cat: 812b3 File: 812ab
HeadMan: File: /usr/bin/scp
HeadMan: st_ino differ. Cat: 5e07e File: 5e343
HeadMan: st_size differ. Cat: 26728 File: 26952
HeadMan: SHA1 differs.
HeadMan: File: /usr/bin/ssh-keygen
HeadMan: st_ino differ. Cat: 5df1d File: 5e07e
HeadMan: st_size differ. Cat: 80488 File: 84648
HeadMan: SHA1 differs.
HeadMan: File: /usr/bin/sftp
HeadMan: st_ino differ. Cat: 5e2e8 File: 5df1d
HeadMan: st_size differ. Cat: 46952 File: 46984
HeadMan: SHA1 differs.
HeadMan: File: /usr/bin/slogin
HeadMan: st_ino differ. Cat: 5e359 File: 5e2e8
HeadMan: File: /usr/bin/ssh

HeadMan: st_mode differ. Cat: 89ed File: 81ed
HeadMan: st_ino differ. Cat: 5e35a File: 5e359
HeadMan: st_size differ. Cat: 219932 File: 234440
HeadMan: SHA1 differs.
HeadMan: File: /usr/bin/ssh-add
HeadMan: st_ino differ. Cat: 5e35b File: 5e35a
HeadMan: st_size differ. Cat: 76328 File: 81448
HeadMan: SHA1 differs.
```

```
HeadMan: File: /usr/bin/ssh-agent
HeadMan: st_ino differ. Cat: 5e35c File: 5e35b
HeadMan: st_size differ. Cat: 43208 File: 47368
HeadMan: SHA1 differs.
HeadMan: File: /usr/bin/ssh-keyscan
HeadMan: st_ino differ. Cat: 5e35d File: 5e96a
HeadMan: st_size differ. Cat: 139272 File: 151560
HeadMan: SHA1 differs.
HeadMan: 25-Jun-2002 21:41
JobId: 83
Job: RufusVerify.2002-06-25.21:41:05
Fileset: Verify Set
Verify Level: Catalog
Client: RufusVerify
Start time: 25-Jun-2002 21:41
End time: 25-Jun-2002 21:41
Files Examined: 4,258
Termination: Verify Differences
```

At this point, it was obvious that these files were modified during installation of the update. If you want to be super safe, you should run a **Verify Level=Catalog** immediately before installing new software to verify that there are no differences, then run a **Verify Level=InitCatalog** immediately after the installation.

To keep the above email from being sent every night when the Verify Job runs, after a software update, simply re-run the Verify Job setting the level to **InitCatalog** (as in the very beginning). This will re-establish the current state of the system as your new basis for future comparisons. Make sure that you don't do an **InitCatalog** after someone has placed a Trojan horse on your system!

If you have included in your **Fileset** a file that is changed by the normal operation of your system, you will get false matches, and you will need to modify the **Fileset** to exclude that file (or not to Include it), and then re-run the **InitCatalog**.

Verify Configuration Example

```
Schedule {
    Name = "VerifyCycle"
    Run = Level=Catalog sun-sat at 5:05
}
Job {
    Name = "MatouVerify"
    Type = Verify
    Level = Catalog
    # default level
    Client = MatouVerify
    Fileset = "Verify Set"
    Messages = Standard
    Storage = DLTDrive
    Pool = Default
    Schedule = "VerifyCycle"
}
#
```

```
# The list of files in this Fileset should be carefully
# chosen. This is a good starting point.
#
Fileset {
    Name = "Verify Set"
        Include {
            Options {
                verify=pins1
                signature=SHA1
            }
        File = /boot
        File = /bin
        File = /sbin
        File = /usr/bin
        File = /lib
        File = /root/.ssh
        File = /home/kern/.ssh
        File = /var/named
        File = /etc/sysconfig
        File = /etc/ssh
        File = /etc/security
        File = /etc/exports
        File = /etc/rc.d/init.d
        File = /etc/sendmail.cf
        File = /etc/sysctl.conf
        File = /etc/services
        File = /etc/xinetd.d
        File = /etc/hosts.allow
        File = /etc/hosts.deny
        File = /etc/hosts
        File = /etc/modules.conf
        File = /etc/named.conf
        File = /etc/pam.d
        File = /etc/resolv.conf
    }
    Exclude = { }
}
Client {
    Name = MatouVerify
    Address = lmatou
    Catalog = Bacula
    Password = ""
    File Retention = 80d   # 80 days
    Job Retention = 1y  # one year
    AutoPrune = yes  # Prune expired Jobs/Files
}
Catalog {
    Name = Bacula
    dbname = verify; user = bacula; password = ""
}
```

2.9 Limitations

There exists one critical element of a **Bacula Enterprise** instance the reliability of which may affect the reliability of any integrity checking done with **Bacula**: The catalog database.

Not only does the catalog have to be configured to prevent unauthorized access and modifications to it, the catalog must also remain available, consistent and correct. Otherwise, if you can not ensure the catalog's security, you can not rely on the results of any Verify Jobs. The most important steps are

1. Create a dedicated role for access to **Bacula Enterprise**'s** catalog database. The credentials are stored in the **director (dir)**'s configuration, so this needs to be protected against unauthorized access as well.

2. Do not assign permissions to modify the catalog to applications that do not need them, like reporting tools. Use a role with restricted permissions for those tools.

3. Ensure that your **Bacula** as well as your catalog database server(s) are inaccessible to unauthorized persons – both remotely and physically.

4. Create and keep backups of your catalog database. Keep those backups safe.

5. Consider running your catalog in a cluster to minimize the risk of failure and data loss.

While `Unix` and `Gnu/Linux` systems systems typically have a file system layout that allows to identify critical files easily, Microsoft Windows is much more difficult in that respect. One reason is that a lot of the configuration information is kept in the registry, not easily accessible at the file level. Also, in the registry, system configuration, critical application data, and non-critical, changing data are mixed, which makes verification of the complete registry impossible. Furthermore, it's difficult – if not impossible – to identify critical system files on a windows system with reasonable effort, and the way Windows itself manages its system files makes it very difficult to apply Verify Jobs to Windows operating systems.

3 Virtual Full Jobs

Available as of version 14.0.4

Since Bacula Enterprise version 14.0.4, most plugins are compatible with VirtualFull jobs. Refer to a specific plugin's article for more information.

When the Job Level is set to VirtualFull, it permits you to consolidate the previous Full backup plus the most recent Differential backup and any subsequent Incremental backups into a new Full backup. This new Full backup will then be considered as the most recent Full for any future Incremental or Differential backups. The VirtualFull backup is accomplished without contacting the client by reading the previous backup data and writing it to a volume in a different pool.

Bacula Enterprise virtual backup feature is often called Synthetic Backup or Consolidation in other backup products.

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.

In some respects the Virtual Backup feature works similar to a Migration job, in that Bacula normally reads the data from the pool specified in the Job resource, and writes it to the **Next Pool** specified in

the Job resource. Note, this means that usually the output from the Virtual Backup is written into a different pool from where your prior backups are saved. Doing it this way guarantees that you will not get a deadlock situation attempting to read and write to the same volume in the Storage daemon. If you then want to do subsequent backups, you may need to move the Virtual Full Volume back to your normal backup pool. Alternatively, you can set your **Next Pool** to point to the current pool. This will cause Bacula to read and write to Volumes in the current pool. In general, this will work, because Bacula will not allow reading and writing on the same Volume. In any case, once a VirtualFull has been created, and a restore is done involving the most current Full, it will read the Volume or Volumes by the VirtualFull regardless of in which Pool the Volume is found.

A typical Job resource definition might look like the following:

```
Job {
    Name = "MyBackup"
    Type = Backup
    Client = localhost-fd
    Fileset = "Full Set"
    Storage = File
    Messages = Standard
    Pool = Default
    SpoolData = yes
}
# Default pool definition
Pool {
    Name = Default
    Pool Type = Backup
    Volume Retention = 365d  # one year
    NextPool = Full
    Storage = File
}
Pool {
    Name = Full
    Pool Type = Backup
    Volume Retention = 365d  # one year
    Storage = DiskChanger
}
# Definition of file storage device
Storage {
    Name = File
    Address = localhost
    Password = "xxx"
    Device = FileStorage
    Media Type = File
    Maximum Concurrent Jobs = 5
}
# Definition of DDS Virtual tape disk storage device
Storage {
    Name = DiskChanger
    Address = localhost # N.B. Use a fully qualified name here
    Password = "yyy"
    Device = DiskChanger
    Media Type = DiskChangerMedia
    Maximum Concurrent Jobs = 4
```

```
     Autochanger = yes
}
```

Then in bconsole or via a Run schedule, you would run the job as:

```
run job=MyBackup level=Full
run job=MyBackup level=Incremental accurate=yes
run job=MyBackup level=Differential accurate=yes
run job=MyBackup level=Incremental accurate=yes
run job=MyBackup level=Incremental accurate=yes
```

So providing there were changes between each of those jobs, you would end up with a Full backup, a Differential, which includes the first Incremental backup, then two Incremental backups. Please notice that incremental and differential jobs require the option **accurate=yes** to work properly. All the above jobs would be written to the **Default** pool.

To consolidate those backups into a new Full backup, you would run the following:

```
run job=MyBackup level=VirtualFull
```

And it would produce a new Full backup without using the client, and the output would be written to the **Full** Pool which uses the Diskchanger Storage.

If the Virtual Full is run, and there are no prior Jobs, the Virtual Full will fail with an error.

Note, the Start and End time of the Virtual Full backup is set to the values for the last job included in the Virtual Full (in the above example, it is an Increment). This is so that if another incremental is done, which will be based on the Virtual Full, it will backup all files from the last Job included in the Virtual Full rather than from the time the Virtual Full was actually run.

3.1 Manual Jobs Selection

By default Bacula is selecting jobs automatically, however, you may want to choose any point in time to create the Virtual backup.

For example, if you have the following Jobs in your catalog:

JobId	Name	Level	JobFiles	JobBytes	JobStatus
1	Vbackup	F	1754	50118554	T
2	Vbackup	I	1	4	T
3	Vbackup	I	1	4	T
4	Vbackup	D	2	8	T
5	Vbackup	I	1	6	T
6	Vbackup	I	10	60	T
7	Vbackup	I	11	65	T
8	Save	F	1758	50118564	T

If you want to consolidate only the first 3 jobs and create a virtual backup equivalent to Job 1 + Job 2 + Job 3, you will use `jobid=3` in the **run** command, then Bacula will select the previous Full backup, the previous Differential (if any) and all subsequent Incremental jobs.

```
run job=Vbackup jobid=3 level=VirtualFull
```

If you want to consolidate a specific job list, you must specify the exact list of jobs to merge in the **run** command line. For example, to consolidate the last Differential and all subsequent Incremental, you will use `jobid=4,5,6,7` or `jobid=4-7` in the **run** command line. As one of the Job in the list is a Differential backup, Bacula will set the new job level to Differential. If the list is composed only with Incremental jobs, the new job will have a level set to Incremental.

```
run job=Vbackup jobid=4-7 level=VirtualFull
```

When using this feature, Bacula will automatically discard jobs that are not related to the current Job. For example, specifying `jobid=7,8`, Bacula will discard the jobid 8.

If you know what you are doing and still want to consolidate jobs that have different names (so probably different clients, filesets, etc.), you must use `alljobid=` keyword instead of `jobid=`.

```
run job=Vbackup alljobid=1-3,6-8 level=VirtualFull
```

3.2 Limitations

- Virtual Full Backup is not supported for the use of two different Storage Daemons.
- Virtual Full Backup is not supported with all the plugins.

4 Tape Autochanger Setup

The following article provides information on tape autochanger setup.

4.1 Supported Tape Drives

Bacula uses standard operating system calls (`read`, `write`, `ioctl`) to interface to tape drives. As a consequence, it relies on having a correctly written OS tape driver. Bacula is known to work perfectly well with SCSI tape drivers on FreeBSD, Linux, Solaris, and Windows machines, and it may work on other *nix machines.

Recently there are many new drives that use IDE, ATAPI, or SATA interfaces rather than SCSI. On Linux the OnStream drive, which uses the OSST driver is one such example, and it is known to work with Bacula. In addition a number of such tape drives (i.e. OS drivers) seem to work on Windows systems. However, non-SCSI tape drives (other than the OnStream) that use `ide-scsi`, `ide-tape`, or other non-scsi drivers do not function correctly with Bacula (or any other demanding tape application) as of today (April 2007). If you have purchased a non-SCSI tape drive for use with Bacula on Linux, there is a good chance that it will not work. We are working with the kernel developers to rectify this situation, but it will not be resolved in the near future.

Generally any modern tape drive (i.e. after 2010) will work out of the box with Bacula using the standard Bacula Device specification in the `bacula-sd.conf` file.

Please check the Tape Testing for procedures that you can use to verify if your tape drive will work with Bacula. If your drive is in fixed block mode, it may appear to work with Bacula until you attempt to do a restore and Bacula wants to position the tape. You can be sure only by following the procedures suggested above and testing.

Note: If your tape hardware and operating system are relatively new (<10 years), they will cooperate smoothly.

4.2 Setting Up a Tape Autochanger Using Bweb

This document is meant to be a walk-though guide to help you set up an Autochanger using the Bacula Systems' Bweb Management Suite.

For Bacula Systems subscribers, support may be requested by sending an email to support@baculasystems.com or by opening a ticket in our Customer Portal.

This document is specific to Bacula Systems' Bweb Management Suite version 12.x and there may be minor discrepancies with newer versions.

Go through the following steps:

Determine Node Names for Drives and Autochanger

First, we need to determine the unique device "by-id" nodes that udev has assigned to the tape drives and the tape library's Autochanger device.

In a shell prompt:

```
# ls -la /dev/tape/by-id
total 0
drwxr-xr-x 2 root root 220 Apr 14 13:00 .
drwxr-xr-x 4 root root  80 Apr 14 13:00 ..
lrwxrwxrwx 1 root root   9 Apr 14 13:00 scsi-350223344ab000100 -> ../../st0
lrwxrwxrwx 1 root root  10 Apr 14 13:00 scsi-350223344ab000100-nst -> ../../
→nst0  <-----
lrwxrwxrwx 1 root root   9 Apr 14 13:00 scsi-350223344ab000200 -> ../../st1
lrwxrwxrwx 1 root root  10 Apr 14 13:00 scsi-350223344ab000200-nst -> ../../
→nst1
lrwxrwxrwx 1 root root   9 Apr 14 13:00 scsi-350223344ab000300 -> ../../st2
lrwxrwxrwx 1 root root  10 Apr 14 13:00 scsi-350223344ab000300-nst -> ../../
→nst2
lrwxrwxrwx 1 root root   9 Apr 14 13:00 scsi-350223344ab000400 -> ../../st3
lrwxrwxrwx 1 root root  10 Apr 14 13:00 scsi-350223344ab000400-nst -> ../../
→nst3
lrwxrwxrwx 1 root root   9 Apr 14 13:00 scsi-SSTK_L700_XYZZY_A -> ../../sg7  ↵
→     <-----
```

We are interested in the longer "by-id" device node names which point to our "/dev/nstX" (non-rewind) tape devices.

For the tape drive, we will use the unique "by-id" node name **"/dev/tape/by-id/scsi-350223344ab000100-nst"** instead of "/dev/nst0" in the configuration.

If the tape library has more than one drive in it, as in the example above, make sure to note the correct "by-id" name of each tape drive, they will be needed later.

In the example output above, there is also a "by-id" device which points to "/dev/sg7". This is the library's autochanger device node, and we will use its "by-id" name **"/dev/tape/by-id/scsi-SSTK_L700_XYZZY_A"** in the configurations.

Throughout the rest of this document, these two "by-id" device node names will be used. Be sure to use the correct device nodes identified on your specific system, and not the device nodes in this document.

Determine the Password for the Storage Daemon (SD)

The password for the SD will be needed in several places below, so let's get it now:

1. At the left menu, click: Configuration –> Storage Daemon –> Directors.

2. Click on the item with a name that ends with "-dir" - For example: "bacula-dir", not "bacula-mon".

3. Click "Show/Hide Password".

4. Copy this password to your clipboard it will be needed later.

5. Click on "Cancel" to be sure that no changes are accidentally made.

Create Drive Device(s) in the Storage Daemon (SD)

1. In the left menu, click: Configuration –> Storage Daemon –> Devices

   - Click the "+" sign to add a new device.

   - Click the "Set optimal LTO Settings" button in the "Actions" box on the left <- This is important. Do not skip this step.

   - Fill in the fields:

     – **Name:** Give the device a name that includes the name of our Library. e.g.: Quantum-Lib1_Drv0

     – **Description:** Leave blank, or put something descriptive here. It is not parsed. e.g.: Quantum Library 1 - LTO6 - Drive 0

     – **Media Type:** Can be any test string (no spaces or special characters), but it must be unique to the Library that this device is in. This Media Type must be the same for all of the devices in a Library. e.g.: QuantumLib1-LTO6

     – **Device Type:** Select "Tape" from the drop-down menu

     – **Archive Device:** Set this to the by-id name we identified above for the /dev/nst0 device. e.g.: /dev/tape/by-id/scsi-350223344ab000100-nst

     – **Autochanger:** Check this box

     – Click the arrow next to **"Tape Settings"** to expand this section

     – **Drive Index:** Set this first drive device's index to **0**, the next drive device to **1**, and so on

     – At the top of the "Add Device" dialog box, click the "+ Add" and you should see this new **QuantumLib1_Drv0** device in your device listing

     – Repeat these steps for each drive device, then continue with the next steps to commit the workset and reload the Director configuration.

2. If auto-commit is not enabled, click on the "Bell" icon at the top right.

3. Click on the "Commit" button.

4. You will see a "Services restart/reload" dialog box showing the Storage component and a check box under "Restart". Check this "Restart" box, then click "Restart/Reload" just below and confirm the operation.

Create an Autochanger in the Storage Daemon (SD)

1. In the left menu, click: Configuration –> Storage Daemon –> Autochangers.

   - Click the "+" sign to add a new Autochanger

   - Fill in the fields:

     - **Name:** Give this Autochanger a sensible name to match our drive device created in the previous section. e.g.: **QuantumLib1**

     - **Description:** Leave blank, or put something descriptive here. It is not parsed. e.g.: **Quantum Library 1 - LTO6**

     - **Changer Command:** Cut and paste this into the field: **/opt/bacula/scripts/mtx-changer %c %o %S %a %d**

     - **Changer Device:** Put the by-id device name we identified in the very first section. e.g.: **/dev/tape/by-id/scsi-SSTK_L700_XYZZY_A**

     - **Device:** Scroll and locate the drive device(s) created in the previous section (ctrl-click to select more than one drive device)

     - At the top of this "Add Autochanger" dialog box, click the "+ Add" and you should see this new QuantumLib1 autochanger in your Autochanger listing.

2. If auto-commit is not enabled, click on the "Bell" icon at the top right.

3. Click on the "Commit" button.

4. You will see a "Services restart/reload" dialog box showing the Storage component and a check box under "Restart". Check this "Restart" box, then click "Restart/Reload" just below and confirm the operation.

Test the Tape Drive Device With Btape Utility

> **Warning:** Once the drive device(s) and the Autochanger device have been created in the SD, STOP HERE. Each tape drive in the library must be tested with the btape utility by following the instructions in the box below before moving on to the rest of this document.

The goal is first to check the correct link between the tape Autochanger and btape which uses your bacula-sd.conf to test your Autochanger.

To run btape, please follow this procedure:

1. Load a tape in the drive you want to test. The following "mtx" command will load a tape from slot 1 into drive 0:

```
# mtx -f /dev/tape/by-id/dev/tape/by-id/scsi-SSTK_L700_XYZZY_A load 1 0
```

Important: Be sure to load an empty tape or one you can erase because all data will be lost on this tape during the test.

---

2. Type the following command to start btape, then type "test" at the "*" prompt:

```
# /opt/bacula/bin/btape QuantumLib1_Drv0

Tape block granularity is 1024 bytes.

19-Apr 09:41 btape JobId 0: Loaded plugin: dedup-sd.so

btape: butil.c:295-0 Using device: "QuantumLib1_Drv0" for writing.

btape: btape.c:472-0 open device "QuantumLib1_Drv0"

(/dev/tape/by-id/scsi-350223344ab000100-nst): OK   <- This is good, btape can␣
→connect to the tape drive


*test   <- Type test and hit enter to begin the tests

=== Write, rewind, and re-read test ===

I'm going to write 10000 records and an EOF
then write 10000 records and an EOF, then rewind,
and re-read the data to verify that it is correct.

This is an *essential* feature ...
```

[. . .] Long list of output snipped here [. . .]

Each of these tests must succeed before going forward Type "quit" to return the the shell prompt If the library contains more than 1 drive, repeat these btape steps on each drive. Once all drives have been tested with btape, continue with the rest of this document.

More detailed information about testing your tape drives to obtain best performance may be found in the btape chapter and in the Testing Your Tape Drive with Bacula.

Create a Storage Resource in the Director (DIR) for Each Drive Device

In order to be able to manage the Autochanger and its drive devices from within Bweb, each drive device that was created in the SD for the Autochanger also needs to be created as a distinct Storage resource in the Director.

1. In the left menu, click: Configuration –> Director –> Storage.

    - Click the "+" sign to add a new Storage resource in the Director

    - Fill in the fields:

        - **Storage Name:** You may use the same name as was configured in the SD for the drive device. e.g.: **QuantumLib1_Drv0**

        - **Description:** Leave blank, or put something descriptive here. It is not parsed. e.g.: **Quantum Library 1 - LTO6 - Drive 0**

        - **Address:** This should be an IP address or, preferably a FQDN of your server running the SD. Do not use localhost or 127.0.0.1 here

        - **Password:** This is the SD password determined above

- **Device:** This must match the name used for the drive device created in the SD. e.g.: **QuantumLib1_Drv0**

- **Media Type:** This must match the Media Type of the drive device(s) in the SD. e.g.: **QuantumLib1-LTO6**

- **Allow Compression:** Un-check this for tape drive devices since they should use hardware compression

- **Autochanger:** Select the **"QuantumLib1"** Autochanger from the drop-down list

- At the top of this "Add Storage" dialog box, click the "+ Add" and you should see this new **QuantumLib1_Drv0** device in your Storages listing

- Repeat these steps for each drive device, then continue with the next steps to commit the workset and reload the Director configuration.

2. If auto-commit is not enabled, click on the "Bell" icon at the top right.

3. Click on the "Commit & Reload" button.

Create an Autochanger Storage Resource in Director (DIR)

1. In the left menu, click: Configuration –> Director –> Storage.

   - Click the "+" sign to add a new Storage resource in the Director

   - Fill in the fields:

     - **Storage Name:** You may use the same name as was configured in the SD for the Autochanger device. e.g.: **QuantumLib1**

     - **Description:** Leave blank, or put something descriptive here. It is not parsed. e.g.: **Quantum Library 1 - LTO6**

     - **Address:** This should be an IP address or, preferably a FQDN of your server running the SD. Do not use localhost or 127.0.0.1 here

     - **Password:** This is the SD password determined in this step

     - **Device:** This must match the name used for the Autochanger created in the SD. e.g.: **QuantumLib1**

     - **Media Type:** This must match the Media Type of the drive device(s) in the SD. e.g.: **QuantumLib1-LTO6**

     - **Allow Compression:** Un-check this for tape drive devices since they should use hardware compression

     - **Autochanger:** If the name defined on the field Storage Name above gets listed, select it, otherwise, select "yes"

     - Click on **"Limits"** to expand this section:

       * Maximum Concurrent Jobs: Set this to at least the number of drive devices in the Autochanger

       * Leave everything else alone in this "Add Storage" dialog box for now

     - At the top of the "Add Storage" dialog box, click the "+ Add" and you should see this new **QuantumLib1** autochanger in your Storages listing.

2. If auto-commit is not enabled, click on the "Bell" icon at the top right.

3. Click on the "Commit & Reload" button.

Configure Bweb to Manage an Autochanger and Its Drive Devices

1. In the left menu, click: Storage and Media –> Tapes –> + Add Tape Library.

2. Fill in the fields:

   - **Name:** Select the new "QuantumLib1" Autochanger from the drop-down list.

   - **Pre-command:** Leave blank (if you are connecting to the remote SD, then setup the key-based ssh authentication in between the Director and the SD host and place "ssh root@<SH_host>" to be executed as the pre-command)

   - **mtx command:** Leave as "/usr/sbin/mtx"

   - **Device:** We need to use its "by-id" name "/dev/tape/by-id/scsi-SSTK_L700_XYZZY_A" as identified on the SD host

   - **Drives:**

     – Locate the new drive device that was created (QuantumLib1_Drv0), click the check box next to its name and set an index (starting at 0) that matches the index of the drive device that was previously created in the SD configuration.

3. Click save. You will be returned the the Bweb "Main Configuration" page.

4. The new QuantumLib1 Autochanger should now be listed under Storage and Media –> Tapes.

5. In the left menu, click: Storage and Media –> Tapes –> Storage Overview, choose the new "QuantumLib1" Autochanger from the drop-down list.

6. This page shows a listing of the QuantumLib1 Autochanger with tape media and their barcodes listed. On the right side, the drives in the library are listed.

Check if Scratch Pool Exists

The Scratch pool is a special pool that eases the management of tape volumes, specially. Be sure that it exists before continuing:

1. At the left menu, click: Configuration –> Director –> Pools.

2. Search for the pool **"Scratch"**. If it exists, please go to the next step, otherwise:

   - Click the "+" sign to add a new Pool resource in the Director

   - Set the pool name as **"Scratch"**

   - Fill no other parameter and click the "+ Add" and you should see this new **Scratch** pool in your Pools listing

   - If auto-commit is not enabled, click on the "Bell" icon at the top right

   - Click on the "Commit & Reload" button.

Label Tapes in an Autochanger

Before the tapes are usable by Bacula, they must have a Bacula label written onto them.

1. In the left menu, click: Storage and Media –> Tapes –> "QuantumLib1".

2. Click the check box in the "Select" column for each tape you want to have labeled.

3. Click the "Label" icon - Note: If you do not have any cleaning tapes in the Autochanger, you can simply click the "Label" icon without selecting each tape.

4. This process can take some time as Bacula loads each tape in the Autochanger into the drive and writes a Bacula label to it.

5. When Bacula is finished labeling the tapes, click on Storage and Media –> Tapes –> "Quantum-Lib1" to get back to the Autochanger status page.

6. Notice that the tapes now have a slot listed, a certain number of bytes written to them (the Bacula label), a Media Type (QuantumLib1-LTO6) and have been added to the pool "Scratch".

Configure a Pool for Full Backups

Before running a backup job using the new QuantumLib1 Autochanger, a backup pool needs to be created for the tapes in the Autochanger. The first backup pool we create will be used for Full backups in the QuantumLib1 Autochanger. You may wish to create additional pools for Incremental or Differential jobs. Only a Full pool will created in this example.

1. In the left menu, click: Configuration –> Director –> Pools.

   - Click the "+" sign to add a new Pool resource in the Director

   - Click "Set as tape pool" in the "Actions" box on the left. <- This is important. Do not skip this step

   - Fill in the fields:

      – **Pool Name:** Fill in a sensible name for this pool. e.g.: **QuantumLib1-Full**

      – **Description:** Leave blank, or give a sensible description. e.g.: **Quantum Library 1 - LTO6 - Full Pool**

      – **Label Format:** Leave blank for tape media because each tape's barcode label will be used as its Bacula label and name

      – **Storage:** From the drop-down, choose the Autochanger created in the Director configuration: **"QuantumLib1"**

      – Label Type: Leave as "Bacula"

      – Cleaning Prefix: Leave as "CLN" unless your cleaning tapes have another prefix

      – Expand the Recycling pane and ensure that both fields, **"Recycle Pool"** and **"Scratch Pool"** are set to **"Scratch"** in their respective drop-down list

      – At the top of the "Add Pool" dialog box, click the "+ Add" and you should see this new **QuantumLib1-Full** pool in your Pools listing.

2. If auto-commit is not enabled, click on the "Bell" icon at the top right.

3. Click on the "Commit & Reload" button.

Test a Full Backup Job Using a New Autochanger

1. In the left menu, click: Run Backup.

2. From the "Job Name" drop-down list, choose any backup job available. e.g.: **BackupClient1**.

3. Click "Run now".

4. In the "Run job" dialog box:

   • Select **"QuantumLib1-Full"** from the Pool drop-down list

   • Select **"QuantumLib1"** from the Storage drop-down list

   • Select **"Full"** from the Level drop-down list

   • Click "Run now" at the bottom.

At this point you will be taken to the "Running job details" screen for this job.

When the job finishes, check the job summary to verify the Pool, Storage and Volumes used are all expected values. That is to say, make sure the new QuantumLib1 Autochanger was used and that tape media from this library was used.

The next step is to create new jobs and/or modify your existing jobs to utilize this new, fully tested Autochanger.

4.3 Tape Autochanger Setup with CLI

This chapter is concerned with testing and configuring your tape drive to make sure that it will work properly with **Bacula** using the `btape` program.

Get Your Tape Drive Working

Specifying Device Name For Tape

`btape` uses the **device-name** to identify the device where the Volume can be found. In the case of a tape, this is the physical device name such as **/dev/nst0** or **/dev/rmt/0ubn** depending on your system that you specify on the Archive Device directive. For the program to work, it must find the identical name in the Device resource of the configuration file. If the name is not found in the list of physical names, the utility program will compare the name you entered to the Device names (rather than the Archive device names).

When specifying a tape device, it is preferable that the "non-rewind" variant of the device file name be given. In addition, on systems such as Sun, which have multiple tape access methods, you must be sure to specify to use Berkeley I/O conventions with the device. The **b** in the Solaris (Sun) archive specification **/dev/rmt/0mbn** is what is needed in this case. **Bacula** does not support SysV tape drive behavior.

See below for specifying Volume names.

Specifying Device Name For File

If you are attempting to read or write an archive file rather than a tape, the **device-name** should be the full path to the archive location including the filename. The filename (last part of the specification) will be stripped and used as the Volume name, and the path (first part before the filename) must have the same entry in the configuration file. So, the path is equivalent to the archive device name, and the filename is equivalent to the volume name.

btape

This program permits a number of elementary tape operations via a `tty` command interface. The `test` command, described below, can be very useful for testing tape drive compatibility problems. Aside from initial testing of tape drive compatibility with **Bacula**, `btape` will be mostly used by developers writing new tape drivers.

`btape` can be dangerous to use with existing **Bacula** tapes because it will relabel a tape or write on the tape if so requested regardless of whether or not the tape contains valuable data, so please be careful and use it only on blank tapes.

To work properly, `btape` needs to read the Storage daemon's configuration file. As a default, it will look for **bacula-sd.conf** in the current directory. If your configuration file is elsewhere, please use the **-c** option to specify where.

The physical device name or the Device resource name must be specified on the command line, and this same device name must be present in the Storage daemon's configuration file read by `btape`:

```
Usage: btape [options] device_name
    -b <file> specify bootstrap file
    -c <file> set configuration file to file
    -d <nn> set debug level to nn
    -p proceed inspite of I/O errors
    -s turn off signals
    -v be verbose
    -? print this message.
```

Using btape to Verify Tape Drive

An important reason for this program is to ensure that a Storage daemon configuration file is defined so that **Bacula** will correctly read and write tapes.

It is highly recommended that you run the `test` command before running your first **Bacula** job to ensure that the parameters you have defined for your storage device (tape drive) will permit **Bacula** to function properly. You only need to mount a blank tape, enter the command, and the output should be reasonably self explanatory.

For example:

```
(ensure that Bacula is not running)
./btape -c /usr/bin/bacula/bacula-sd.conf /dev/nst0
```

The output will be:

```
Tape block granularity is 1024 bytes.
btape: btape.c:376 Using device: /dev/nst0
*
```

Enter the test command:

```
test
```

The output produced should be something similar to the following:

```
=== Append files test ===
This test is essential to Bacula.
I'm going to write one record in file 0,
two records in file 1,
and three records in file 2
btape: btape.c:387 Rewound /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:855 Wrote one record of 64412 bytes.
btape: btape.c:857 Wrote block to device.
btape: btape.c:410 Wrote EOF to /dev/nst0
btape: btape.c:387 Rewound /dev/nst0
btape: btape.c:693 Now moving to end of media.
btape: btape.c:427 Moved to end of media
We should be in file 3. I am at file 3. This is correct!
Now the important part, I am going to attempt to append to the tape.
...
=== End Append files test ===
```

If you do not successfully complete the above test, please resolve the problem(s) before attempting to use **Bacula**. Depending on your tape drive, the test may recommend that you add certain records to your configuration. We strongly recommend that you do so and then re-run the above test to insure it works the first time.

Some of the suggestions it provides for resolving the problems may or may not be useful. If at all possible avoid using fixed blocking. If the test suddenly starts to print a long series of:

```
Got EOF on tape.
Got EOF on tape.
...
```

then almost certainly, you are running your drive in fixed block mode rather than variable block mode. See below for more help of resolving fix versus variable block problems.

It is also possible that you have your drive set in SysV tape drive mode. The drive must use BSD tape conventions. See the section above on setting your **Archive device** correctly.

Testing Tape Drive Speed

To determine the best configuration of your tape drive, you can run the `speed` command available in the `btape` program.

This command can have the following arguments:

file_size=n
> Specify the Maximum File Size for this test (between 1 and 5GB). This counter is in GB.

nb_file=n
> Specify the number of file to be written. The amount of data should be greater than your memory (*file_size * nb_file*).

skip_zero
> This flag permits to skip tests with constant data.

skip_random
> This flag permits to skip tests with random data.

skip_raw
> This flag permits to skip tests with raw access.

skip_block
> This flag permits to skip tests with **Bacula** block access.

```
*speed file_size=3 skip_raw
btape.c:1078 Test with zero data and bacula block structure.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 44.128 MB/s
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 43.531 MB/s
btape.c:1090 Test with random data, should give the minimum throughput.
btape.c:956 Begin writing 3 files of 3.221 GB with blocks of 129024 bytes.
++++++++++++++++++++++++++++++++++++++++++
btape.c:604 Wrote 1 EOF to "Drive-0" (/dev/nst0)
btape.c:406 Volume bytes=3.221 GB. Write rate = 7.271 MB/s
++++++++++++++++++++++++++++++++++++++++++
...
btape.c:383 Total Volume bytes=9.664 GB. Total Write rate = 7.365 MB/s
```

When using compression, the random test will give your the minimum throughput of your drive. The test using constant string will give you the maximum speed of your hardware chain (cpu, memory, scsi card, cable, drive, tape).

You can change the block size in the Storage Daemon configuration file.

Linux SCSI Tricks

You can find out what SCSI devices you have by doing:

```
lsscsi
```

Typical output is:

```
[0:0:0:0] disk ATA ST3160812AS 3.AD /dev/sda
[2:0:4:0] tape HP Ultrium 2-SCSI F6CH /dev/st0
[2:0:5:0] tape HP Ultrium 2-SCSI F6CH /dev/st1
[2:0:6:0] mediumx OVERLAND LXB 0107 -
[2:0:9:0] tape HP Ultrium 1-SCSI E50H /dev/st2
[2:0:10:0] mediumx OVERLAND LXB 0107 -
```

There are two drives in one autochanger: **/dev/st0** and **/dev/st1** and a third tape drive at **/dev/st2**. For using them with **Bacula**, one would normally reference them as **/dev/nst0** … **/dev/nst2**. Note also, there are two different autochangers identified as "mediumx OVERLAND LXB". They can be addressed via their **/dev/sgN** designation, which can be obtained by counting from the beginning as 0 to each changer. In the above case, the two changers are located on **/dev/sg3** and **/dev/sg5**. The one at **/dev/sg3**, controls drives **/dev/nst0** and **/dev/nst1**; and the one at **/dev/sg5** controls drive **/dev/nst2**.

If you do not have the `lsscsi` command, you can obtain the same information as follows:

```
cat /proc/scsi/scsi
```

For the above example with the three drives and two autochangers, I get:

```
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA Model: ST3160812AS Rev: 3.AD
  Type: Direct-Access ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: HP Model: Ultrium 2-SCSI Rev: F6CH
  Type: Sequential-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 05 Lun: 00
  Vendor: HP Model: Ultrium 2-SCSI Rev: F6CH
  Type: Sequential-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 06 Lun: 00
  Vendor: OVERLAND Model: LXB Rev: 0107
  Type: Medium Changer ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 09 Lun: 00
  Vendor: HP Model: Ultrium 1-SCSI Rev: E50H
  Type: Sequential-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 10 Lun: 00
  Vendor: OVERLAND Model: LXB Rev: 0107
  Type: Medium Changer
```

As an additional example, I get the following (on a different machine from the above example):

```
Attached devices:
Host: scsi2 Channel: 00 Id: 01 Lun: 00
  Vendor: HP Model: C5713A Rev: H107
  Type: Sequential-Access ANSI SCSI revision: 02
```

```
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: SONY Model: SDT-10000 Rev: 0110
  Type: Sequential-Access ANSI SCSI revision: 02
```

The above represents first an autochanger and second a simple tape drive. The HP changer (the first entry) uses the same SCSI channel for data and for control, so in **Bacula**, you would use:

```
Archive Device = /dev/nst0
Changer Device = /dev/sg0
```

If you want to remove the SDT-10000 device, you can do so as `root` with:

```
echo "scsi remove-single-device 2 0 4 0">/proc/scsi/scsi
```

and you can put add it back with:

```
echo "scsi add-single-device 2 0 4 0">/proc/scsi/scsi
```

where the 2 0 4 0 are the Host, Channel, Id, and Lun as seen on the output from `cat /proc/scsi/scsi`. Note, the Channel must be specified as numeric.

Below is a slightly more complicated output, which is a single autochanger with two drives, and which operates the changer on a different channel from from the drives:

```
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA Model: WDC WD1600JD-75H Rev: 08.0
  Type: Direct-Access ANSI SCSI revision: 05
Host: scsi2 Channel: 00 Id: 04 Lun: 00
  Vendor: HP Model: Ultrium 2-SCSI Rev: F6CH
  Type: Sequential-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 05 Lun: 00
  Vendor: HP Model: Ultrium 2-SCSI Rev: F6CH
  Type: Sequential-Access ANSI SCSI revision: 03
Host: scsi2 Channel: 00 Id: 06 Lun: 00
  Vendor: OVERLAND Model: LXB Rev: 0106
  Type: Medium Changer ANSI SCSI revision: 02
```

The above tape drives are accessed on **/dev/nst0** and **/dev/nst1**, while the control channel for those two drives is **/dev/sg3**.

Troubleshooting

Bacula Cannot Open the Device

If you get an error message such as:

```
dev open failed: dev.c:265 stored: unable to open
device /dev/nst0:> ERR=No such device or address
```

the first time you run a job, it is most likely due to the fact that you specified the incorrect device name on your **Archive Device**.

Sometimes the tape handler gets confused. This can happen if your tape drive door is locked (Bacula locks it when writing a tape), and Bacula crashes or some other problem occurs. After this kind of a problem, the kernel driver will not recognize that a tape is loaded and you may see messages such as:

```
19-Sep 16:39 groschat-sd JobId 16072: Warning: mount.c:219 Open of tape
device "LTO9-Drive" (/dev/tape/by-id/scsi-3500143801403cf22-nst) Volume
"AAJ372" failed: ERR=tape_dev.c:162 Unable to open device "LTO9-Drive"
(/dev/tape/by-id/scsi-3500143801403cf22-nst): ERR=No medium found

19-Sep 16:39 groschat-sd JobId 16072: Please mount append Volume "AAJ372"
 or label a new one for:
     Job:          Rufus.2016-09-19_16.34.22_03
     Storage:      "LTO9-Drive" (/dev/tape/by-id/scsi-3500143801403cf22-nst)
     Pool:         Default
     Media type:   LTO-4
```

in that case, the best thing to do is unload the tape by hand with:

```
(stop Bacula)
mtx -f /dev/sgx unload
(restart Bacula)
```

where you replace /dev/sgx by your scsi control device name.

Incorrect File Number

When **Bacula** moves to the end of the medium, it normally uses the **ioctl(MTEOM)** function. Then **Bacula** uses the **ioctl(MTIOCGET)** function to retrieve the current file position from the **mt_fileno** field. Some SCSI tape drivers will use a fast means of seeking to the end of the medium and in doing so, they will not know the current file position and hence return a **-1**. As a consequence, if you get **"This is NOT correct!"** in the positioning tests, this may be the cause. You must correct this condition in order for **Bacula** to work.

There are two possible solutions to the above problem of incorrect file number:

- Figure out how to configure your SCSI driver to keep track of the file position during the MTEOM request. This is the preferred solution.

- Modify the **Device** resource of your **bacula-sd.conf** file to include:

  ```
  Hardware End of File = no
  ```

This will cause **Bacula** to use the MTFSF request to seek to the end of the medium, and **Bacula** will keep track of the file number itself.

Incorrect Number of Blocks or Positioning Errors

Bacula's preferred method of working with tape drives (sequential devices) is to run in variable block mode, and this is what is set by default. You should first ensure that your tape drive is set for variable block mode (see below).

If your tape drive is in fixed block mode and you have told **Bacula** to use different fixed block sizes or **variable block sizes** (default), you will get errors when **Bacula** attempts to forward space to the correct block (the kernel driver's idea of tape blocks will not correspond to **Bacula**'s).

Even in variable block mode, with the exception of the first record on the second or subsequent volume of a multi-volume backup, **Bacula** will write blocks of a fixed size. However, in reading a tape, **Bacula** will assume that for each read request, exactly one block from the tape will be transferred. This the most common way that tape drives work and is well supported by **Bacula**.

Drives that run in fixed block mode can cause serious problems for **Bacula** if the drive's block size does not correspond exactly to **Bacula's** block size. In fixed block size mode, drivers may transmit a partial block or multiple blocks for a single read request. From **Bacula's** point of view, this destroys the concept of tape blocks. It is much better to run in variable block mode, and almost all modern drives run in variable block mode. In order for **Bacula** to run in fixed block mode, you must include the following records in the Storage daemon's Device resource definition:

```
Minimum Block Size = nnn
Maximum Block Size = nnn
```

where **nnn** must be the same for both records and must be identical to the driver's fixed block size.

We recommend that you avoid this configuration if at all possible by using variable block sizes.

If you must run with fixed size blocks, make sure they are not 512 bytes. This is too small and the overhead that **Bacula** has with each record will become excessive.

To recover files from tapes written in fixed block mode, see below.

Linux: Setting Tape Modes Properly

If you have a modern SCSI tape drive and you are having problems with the `test` command as noted above, it may be that some program has set one or more of your SCSI driver's options to non-default values. For example, if your driver is set to work in SysV manner, **Bacula** will not work correctly because it expects BSD behavior. To reset your tape drive to the default values, you can try the following, but **ONLY** if you have a SCSI tape drive on a **Linux** system:

```
become super user
mt -f /dev/nst0 rewind
mt -f /dev/nst0 stoptions buffer-writes async-writes read-ahead
```

The above commands will clear all options and then set those specified. None of the specified options are required by **Bacula**, but a number of other options such as SysV behavior must not be set. **Bacula** does not support SysV tape behavior. On systems other than Linux, you will need to consult your `mt` man pages or documentation to figure out how to do the same thing. This should not really be necessary though – for example, on both Linux and Solaris systems, the default tape driver options are compatible with **Bacula**. On Solaris systems, you must take care to specify the correct device name on the **Archive device** directive. See above for more details.

You may also want to ensure that no prior program has set the default block size by explicitly turning it off with:

```
mt -f /dev/nst0 defblksize 0
```

If you are running a Linux system, and the above command does not work, it is most likely because you have not loaded the appropriate `mt` package, which is often called `mt_st`, but may differ according to your distribution.

If you would like to know what options you have set before making any of the changes noted above, you can now view them on Linux systems. Do the following:

```
become super user
mt -f /dev/nst0 stsetoptions 0
grep st0 /var/log/messages
```

and you will get output that looks something like the following:

```
kernel: st0: Mode 0 options: buffer writes: 1, async writes: 1, read ahead: 1
kernel: st0: can bsr: 0, two FMs: 0, fast mteom: 0, auto lock: 0,
kernel: st0: defs for wr: 0, no block limits: 0, partitions: 0, s2 log: 0
kernel: st0: sysv: 0 nowait: 0
```

Note: The beginning of the line with the date and machine name was chopped off for presentation purposes.

Some people find that the above settings only last until the next reboot, so check this otherwise you may have unexpected problems.

Tape Hardware Compression and Blocking Size

You should be able to verify the tape compression status with sysfs on Linux.

```
cat /sys/class/scsi_tape/nst0/default_compression
```

You can, turn it on by using (on Linux):

```
become super user
mt -f /dev/nst0 defcompression 1
```

and of course, if you use a zero instead of the one at the end, you will turn it off.

You can use `tapeinfo` to get quite a bit of information about your tape drive even if it is not an autochanger. This program is called using the SCSI control device. On Linux for tape drive **/dev/nst0**, this is usually **/dev/sg0**, while on FreeBSD for **/dev/nsa0**, the control device is often **/dev/pass2**. For example on my DDS-4 drive (**/dev/nst0**), I get the following:

```
tapeinfo -f /dev/sg0
Product Type: Tape Drive
Vendor ID: 'HP '
Product ID: 'C5713A '
Revision: 'H107'
Attached Changer: No
MinBlock:1
MaxBlock:16777215
```

```
SCSI ID: 5
SCSI LUN: 0
Ready: yes
BufferedMode: yes
Medium Type: Not Loaded
Density Code: 0x26
BlockSize: 0
```

where the **DataCompEnabled: yes** means that tape hardware compression is turned on. You can turn it on and off (yes|no) by using the `mt` commands given above. Also, this output will tell you if the **BlockSize** is non-zero and hence set for a particular block size. **Bacula** is not likely to work in such a situation because it will normally attempt to write blocks of 64,512 bytes, except the last block of the job which will generally be shorter. The first thing to try is setting the default block size to zero using the `mt -f /dev/nst0 defblksize 0` command as shown above. On FreeBSD, this would be something like: `mt -f /dev/nsa0 blocksize 0`.

On some operating systems with some tape drives, the amount of data that can be written to the tape and whether or not compression is enabled is determined by the density usually the `mt -f /dev/nst0 setdensity xxx` command. Often `mt -f /dev/nst0 status` will print out the current density code that is used with the drive. Most systems, but unfortunately not all, set the density to the maximum by default. On some systems, you can also get a list of all available density codes with: `mt -f /dev/nst0 densities` or a similar `mt` command. Note, for DLT and SDLT devices, no-compression versus compression is very often controlled by the density code. On FreeBSD systems, the compression mode is set using `mt -f /dev/nsa0 comp xxx` where xxx is the mode you want. In general, see `man mt` for the options available on your system.

Note, some of the above `mt` commands may not be persistent depending on your system configuration. That is they may be reset if a program other than **Bacula** uses the drive or, as is frequently the case, on reboot of your system.

If your tape drive requires fixed block sizes (very unusual), you can use the following records:

```
Minimum Block Size = nnn
Maximum Block Size = nnn
```

in your Storage daemon's Device resource to force **Bacula** to write fixed size blocks (where you sent nnn to be the same for both of the above records). This should be done only if your drive does not support variable block sizes, or you have some other strong reasons for using fixed block sizes. As mentioned above, a small fixed block size of 512 or 1024 bytes will be very inefficient. Try to set any fixed block size to something like 64,512 bytes or larger if your drive will support it.

Also, note that the **Medium Type** field of the output of `tapeinfo` reports **Not Loaded**, which is not correct. As a consequence, you should ignore that field as well as the **Attached Changer** field.

To recover files from tapes written in fixed block mode, click *here*.

Using btape to Simulate Filling Tape

Because there are often problems with certain tape drives or systems when end of tape conditions occur, `btape` has a special command `fill` that causes it to write random data to a tape until the tape fills. It then writes at least one more **Bacula** block to a second tape. Finally, it reads back both tapes to ensure that the data has been written in a way that **Bacula** can recover it. Note, there is also a single tape option as noted below, which you should use rather than the two tape test. See below for more details.

This can be an extremely time consuming process to fill a full tape. Note, that btape writes random data to the tape when it is filling it. This has two consequences:

1. it takes a bit longer to generate the data, especially on slow CPUs.

2. the total amount of data is approximately the real physical capacity of your tape, regardless of whether or not the tape drive compression is on or off. This is because random data does not compress very much.

To begin this test, you enter the `fill` command and follow the instructions. There are two options: the simple single tape option and the multiple tape option. Use only the simple single tape option first. If the single tape option does not succeed, you should correct the problem before using **Bacula**.

Recovering Files Written with Fixed Block Sizes

If you have been previously running your tape drive in fixed block mode (default 512) and **Bacula** with variable blocks (default) **Bacula** will fail to recover files because it does block spacing, and because the block sizes don't agree between your tape drive and **Bacula** it will not work.

The long term solution is to run your drive in variable block mode as described above. However, if you have written tapes using fixed block sizes, this can be a bit of a pain. The solution to the problem is: while you are doing a restore command using a tape written in fixed block size, ensure that your drive is set to the fixed block size used while the tape was written. Then when doing the `restore` command in the Console program, do not answer the prompt **yes/mod/no**. Instead, edit the bootstrap file (the location is listed in the prompt) using any ASCII editor. Remove all **VolBlock** lines in the file. When the file is re-written, answer the question, and **Bacula** will run without using block positioning, and it should recover your files.

Autochanger Errors

If you are getting errors such as:

```
3992 Bad autochanger "load slot 1, drive 1": ERR=Child exited with code 1.
```

and you are running your Storage daemon as non-root, then most likely you are having permissions problems with the control channel. Running as `root`, set permissions on **/dev/sgX** so that the userid and group of your Storage daemon can access the device.

Syslog Errors

If you are getting errors such as:

```
: kernel: st0: MTSETDRVBUFFER only allowed for root
```

you are most likely running your Storage daemon as non-root, and **Bacula** is attempting to set the correct OS buffering to correspond to your Device resource. Most OSes allow only `root` to issue this ioctl command. In general, the message can be ignored providing you are sure that your OS parameters are properly configured as described earlier. If you are running your Storage daemon as `root`, you should not be getting these system log messages, and if you are, something is probably wrong.

4.4 Data Spooling

When performing a backup directly to tape, Bacula allows you to specify that you want the Storage daemon to initially write your data to disk and then subsequently to tape. This serves several important purposes.

- It takes a long time for data to come in from the File daemon during an Incremental backup. If it is directly written to tape, the tape will start and stop or shoe-shine as it is often called causing tape wear. By first writing the data to disk, then writing it to tape, the tape can be kept in continual motion.

- While the spooled data is being written to the tape, the despooling process has exclusive use of the tape. This means that you can spool multiple simultaneous jobs to disk, then have them very efficiently despooled one at a time without having the data blocks from several jobs intermingled, thus substantially improving the time needed to restore files. While despooling, all jobs spooling continue running.

- Writing to a tape can be slow. By first spooling your data to disk, you can often reduce the time the File daemon is running on a system, thus reducing downtime, and/or interference with users. Of course, if your spool device is not large enough to hold all the data from your File daemon, you may actually slow down the overall backup.

Data spooling is exactly that: "spooling". It is not a way to first write a "backup" to a disk file and then to a tape. When the backup has only been spooled to disk, it is not complete yet and cannot be restored until it is written to tape.

Bacula version 1.39.x and later supports writing a backup to disk then later **Migrating** or moving it to a tape (or any other medium). For details on this, please see the Migration chapter for more details.

The remainder of this chapter explains the various directives that you can use in the spooling process.

Data Spooling Directives

The following directives can be used to control data spooling.

- To turn data spooling on/off at the Job level in the Job resource in the Director's conf file (default **no**).

SpoolData = <yes|no>

- To override the Job specification in a Schedule Run directive in the Director's conf file.

SpoolData = <yes|no>

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.

- To override the Job specification in a bconsole session using the command. Please note that this does **not** refer to a configuration statement, but to an argument for the run command.

SpoolData= <yes|no>

- To limit the the maximum spool file size for a particular job in the Job resource **Spool Size = size** Where size is a the maximum spool size for this job specified in bytes.

- To limit the maximum total size of the spooled data for a particular device. Specified in the Device resource of the Storage daemon's conf file (default unlimited).

  Maximum Spool Size = size Where size is a the maximum spool size for all jobs specified in bytes.

- To limit the maximum total size of the spooled data for a particular device for a single job. Specified in the Device Resource of the Storage daemon's configuration file (default unlimited).

  Maximum Job Spool Size = size Where size is the maximum spool file size for a single job specified in .

- To specify the spool directory for a particular device. Specified in the Device Resource of the Storage daemon's configuration file (default, the working directory).

Spool Directory = directory

Fileset Consideration

Please be very careful to exclude the spool directory from any backup, otherwise, your job will write enormous amounts of data to the Volume, and most probably terminate in error. This is because in attempting to backup the spool file, the backup data will be written a second time to the spool file, and so on *ad infinitum*.

Another advice is to always specify the maximum spool size so that your disk doesn't completely fill up. In principle, data spooling will properly detect a full disk, and despool data allowing the job to continue. However, attribute spooling is not so kind to the user. If the disk on which attributes are being spooled fills, the job will be canceled. In addition, if your working directory is on the same partition as the spool directory, then jobs will fail possibly in bizarre ways when the spool fills.

Other Points

- When data spooling is enabled, automatically turns on attribute spooling. In other words, it also spools the catalog entries to disk. This is done so that in case the job fails, there will be no catalog entries pointing to non-existent tape backups.

- Attribute despooling occurs near the end of a job. The Storage daemon accumulates file attributes during the backup and sends them to the Director at the end of the job. The Director then inserts the file attributes into the catalog. During this insertion, the tape drive may be inactive. When the file attribute insertion is completed, the job terminates.

- Attribute spool files are always placed in the working directory of the Storage daemon.

- When begins despooling data spooled to disk, it takes exclusive use of the tape. This has the major advantage that in running multiple simultaneous jobs at the same time, the blocks of several jobs will not be intermingled.

- It probably does not make a lot of sense to enable data spooling if you are writing to disk files.

- It is probably best to provide as large a spool file as possible to avoid repeatedly spooling/despooling. Also, while a job is despooling to tape, the File daemon must wait (i.e. spooling stops for the job while it is despooling).

- If you are running multiple simultaneous jobs, will continue spooling other jobs while one is despooling to tape, provided there is sufficient spool file space.

5 Bacula Enterprise Continuous Data Protection

Continuous Data Protection (CDP) also called continuous backup or real-time backup, refers to backup of Client data by automatically saving a copy of every change made to that data, essentially capturing every version of the data that the user saves. It allows the user or administrator to restore data to any point in time.
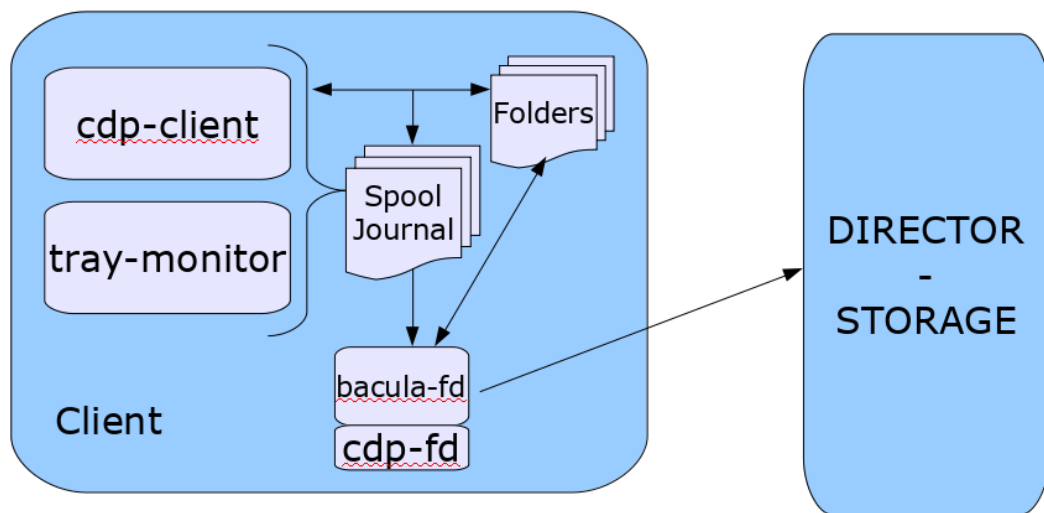
The CDP feature is composed of two components. A application (**cdp-client** or **tray-monitor**) that will monitor a set of directories configured by the user and a **Bacula** FileDaemon plugin responsible to secure the data on a **Bacula** infrastructure.

The user application (**cdp-client** or **tray-monitor**) is responsible to monitor files and directories. When a modification is detected, a copy of the new data is done into a **spool** directory. At a regular interval, a **Bacula** backup job will contact the FileDaemon and will save all the files archived by the **cdp-client**. The local data can be restored at any time without a network

5.1 CDP Plugin Installation

On the **Bacula** File Daemon that you want to configure the CDP feature, the repository file for your package manager should contain a section for the main programs.

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.

CPD Installation on RHEL

In RHEL:

/etc/yum.repos.d/bacula.repo:

```
[Bacula]
name=Bacula Enterprise Edition
baseurl=https://www.baculasystems.com/dl/@customer-string@/rpms/bin/@version@/
↪rhel7-64/
enabled=1
protect=0
gpgcheck=0
```

Once the repository is configured, perform a **yum update**, then the package *bacula-enterprise-cdp-plugin* (and optionally the Tray Monitor package *bacula-enterprise-bat* on RHEL/SLES) can be installed with **yum install**.

```
# yum install bacula-enterprise-cdp-plugin bacula-enterprise-bat
```

CPD Installation on Debian

On Debian Stretch:

/etc/apt/sources.list.d/bacula.list:

```
#Bacula Enterprise deb
https://www.baculasystems.com/dl/@customer-string@/debs/bin/@version@/stretch-
↪64/stretch main
```

Once the repository is configured, perform a **apt-get update**, then the package *bacula-enterprise-cdp-plugin* (and optionally the Tray Monitor package *bacula-enterprise-tray-monitor*) can be installed with **apt-get install**.

```
# apt-get update
# apt-get install bacula-enterprise-cdp-plugin bacula-enterprise-tray-monitor
```

Manual Installation of CPD

If you prefer to manually install the packages, you may download them directly from your download area, and use one of the low level package manager tools (**rpm** or **dpkg**) to do the plugin installation.

5.2 CDP Plugin Configuration

The following article aims at presenting information on CDP Plugin configuration.

Plugin Parameters

In order to configure the CDP Plugin, one of the following parameters must be set.

The following parameters are used in the **Plugin** line of the Fileset that is utilized by the CDP job.

- user=<string> specifies a user in the system. The CDP Plugin will guess the location of the journal file for this user. **Not available on Windows**.
- group=<string> specifies a group in the system. The CDP Plugin will guess the location of the journal files of the users from this group. **Not available on Windows**.
- userHome=<string> specifies the path to the cdp User Home. This is where the CDP Plugin will look for the journal file. **On Windows, this is the User AppData/Roaming folder, for example: "C:/Users/Auser/AppData/Roaming"**.

Components Configuration

CPD File Daemon Configuration

As with all **Bacula** plugins, the **Plugin Directory** directive in the **FileDaemon** resource configuration of the file **bacula-fd.conf** needs to be set:

```
FileDaemon {
    Name = test-fd
    ...
    Plugin Directory = /opt/bacula/plugins
}
```

CPD Director Configuration

Here is an example of the **Fileset** and a **Job** resource configuration:

```
Fileset {
    Name = cdp Include
    Include {
        Options {
            compression = LZO
        }
    Plugin = "cdp: userHome=/home/user1"
    }
}
Job {
    Name = CDP
    Client = myclient-fd
    Type = Backup
    Pool = Default
    Schedule = Hourly
    Messages = Standard
    Fileset = cdp
}
```

CDP Client Configuration

These are the command line options you can use with the **cdp-client** command:

-j <dir> sets journal directory to **<dir>**, default value is **<HOME>**

-s <dir> sets spool directory to **<dir>**, default value is **<HOME>/.cdp-sdir**

-f <dir> watches the directory **<dir>** for changes

-d <nn> set debug level to **<nn>**

-? print help message

Important: It's necessary to keep either the **cdp-client** or the **tray-monitor** running in order to protect the directories being watched.

Tray Monitor Configuration

You can setup the folders you wish to watch for changes by using the Tray Monitor.

1. Open the Tray Monitor options and click in the option **Watch...**:

2. You should see a window displaying the watched folders. Click in the button **Add**:

3. Select the folder you wish to watch:

4, You should see the watched folder listed, as in the image below:
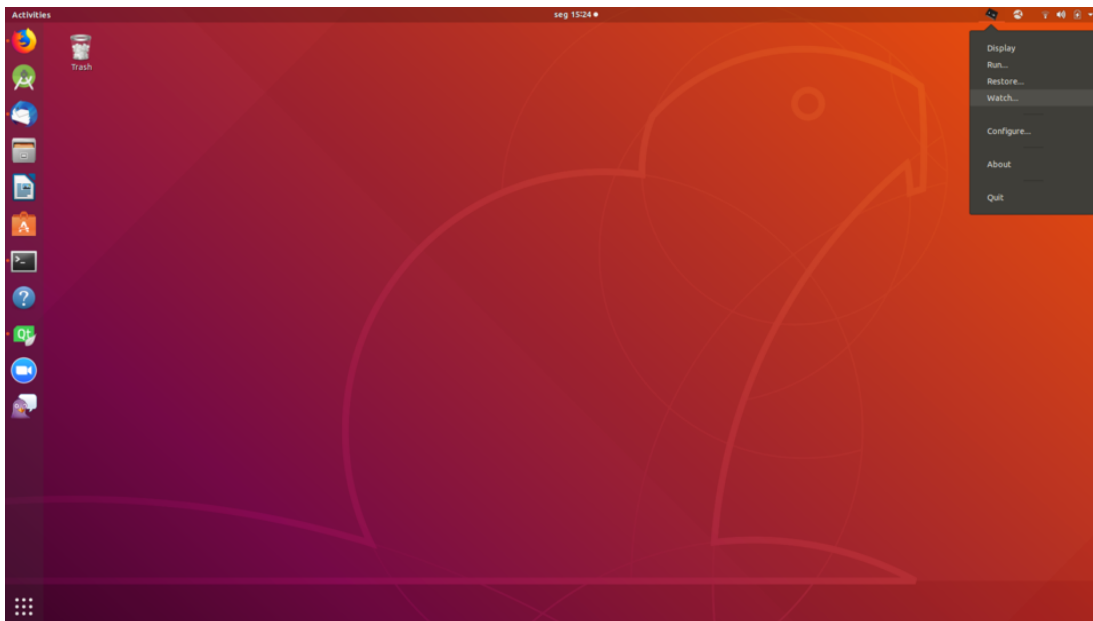

Important: It's necessary to keep either the **cdp-client** or the **tray-monitor** running in order to protect the directories being watched.

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.
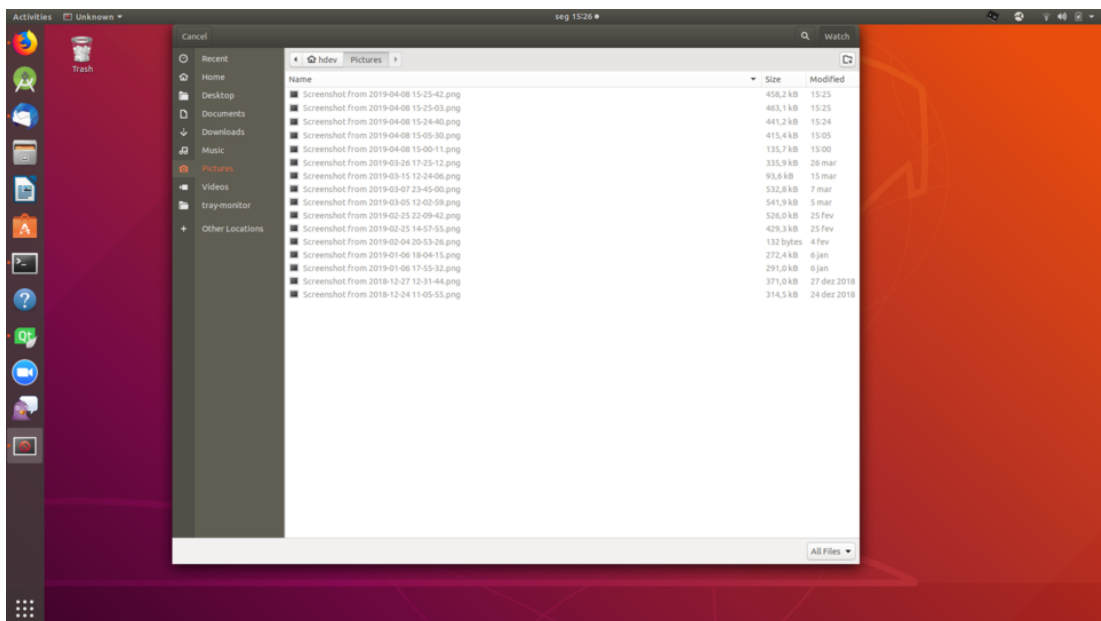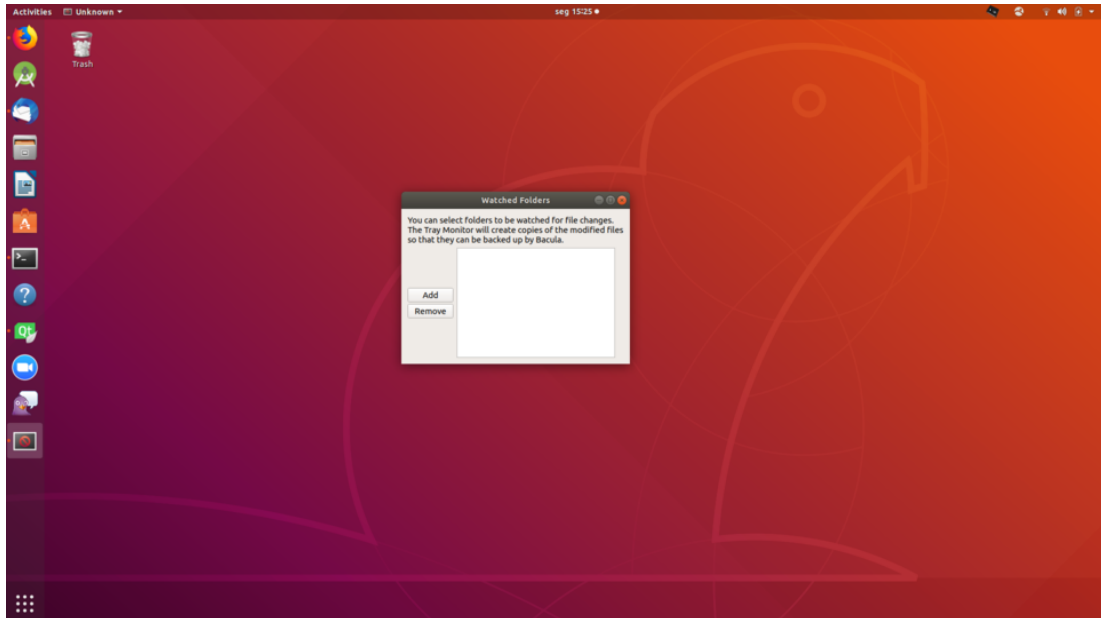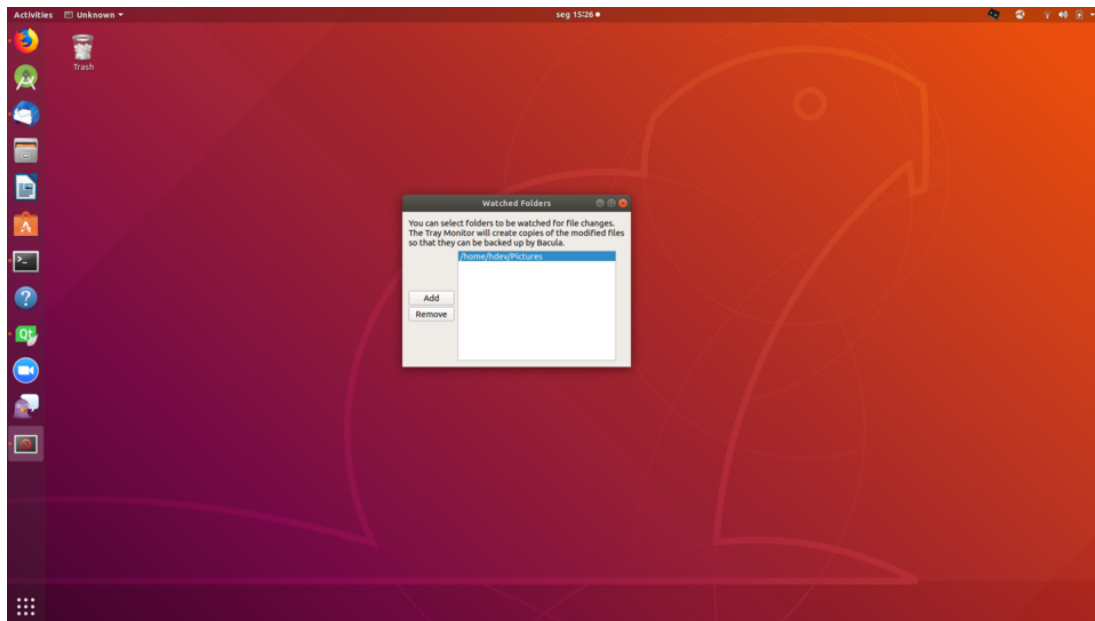
Spool Directory and Scheduled Backups

The Spool Directory contains all files from the watched folders that were created and/or changed. It keeps a copy of every version of the files.

An example of this directory can be seen below:

You must schedule backup jobs with a proper Files in order to backup those files with Bacula. Following this example, the job configuration would be:

```
Fileset {
    Name = cdp
    Include {
        Options {
            compression = LZO
        }
        Plugin = "cdp: userHome=/home/user1"
    }
}
Job {
    Name = CDP
    Client = myclient-fd
    Type = Backup
    Pool = Default
    Schedule = Hourly
    Messages = Standard
    Fileset = cdp
}
```

The Job reads the journal file to select which files from the Spool Directory should be backed up. After that, the files will be available to be restored at any time:
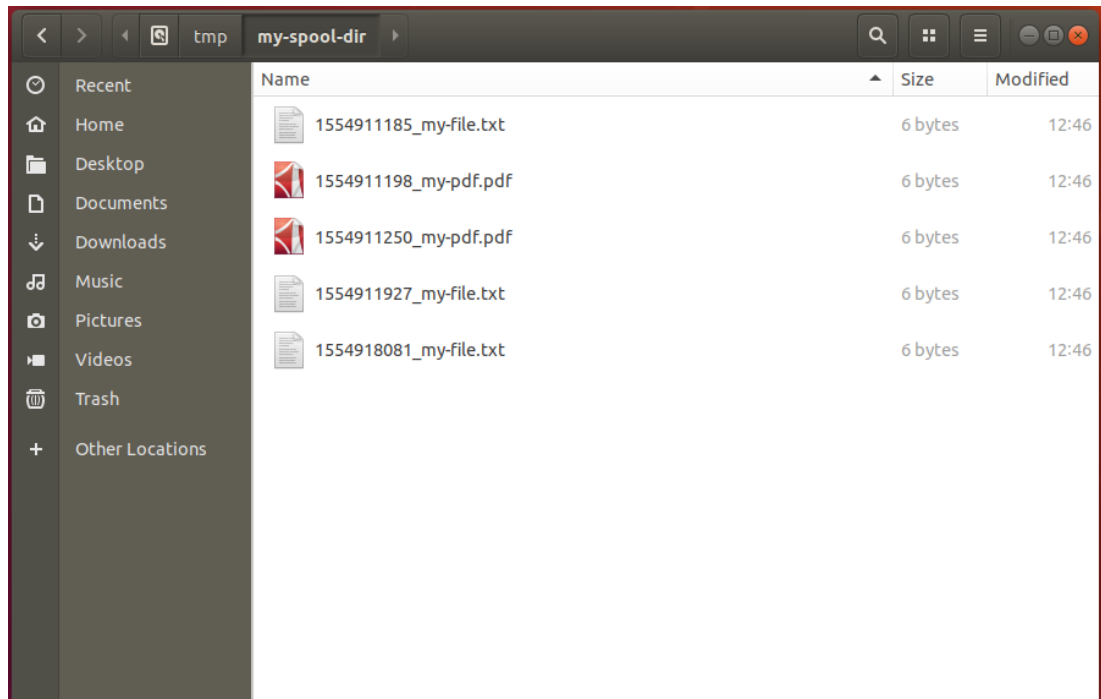
```
list files jobid=1
Using Catalog "MyCatalog"
+----------------------------------------------------------------+
| filename                                                       |
+----------------------------------------------------------------+
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-file.txt |
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-pdf.pdf  |
| /home/hdev/bswift/regress/tmp/testUserHome/spool-dir/my-png.png  |
+----------------------------------------------------------------+
```

Limitations

- There is a limit in the number of folders that can be watched by default, which varies depending on your Operating System. You can increase that number by checking the correct procedure for your Operating System.

- On Linux OSes, the CDP client implementation relies on the **inotify API**, therefore being subject to it's limitations (for example, it does not work with NFS). On Windows, the client relies on the **ReadDirectoryChangesW API**.

6 Data Encryption

6.1 Storage Daemon Data Volume Encryption

The Storage Daemon is responsible for encrypting data stored in the Bacula Volume by using the `Encryption Command` specified in the Storage resource configured within the Storage Daemon.

Each time a new volume is initialized or mounted on a device with encryption enabled, the Storage Daemon invokes the `Encryption Command`. A very simple example script - the `key-manager.py` script - is available to assist in managing Volume encryption keys.

Bacula uses symmetric keys to encrypt the Volumes, meaning that the same key is used for both encryption and decryption of the Volume's content. To improve security, each Volume is expected to be encrypted using a different key. The `key-manager.py` script provides these keys to the Storage Daemon when needed. These keys must be generated for a new Volume or when a Volume is recycled, which can lead to a large amount of key files that need to be managed and securely stored.

To streamline this process, the Storage Encryption can use a Master Key, implemented using a public/private key pair.

The symmetric keys used for encrypting the data within the Volume are encrypted using the Master Key, thereby eliminating the necessity to store the symmetric keys directly.

Note: If Data Spooling is enabled, the data located in the specified *Spool Directory* is not subject to encryption.

Important: Ensure that the spooled data is not saved on insecure storage.

What is Encrypted

The main goal of encryption is to safeguard the Volume's data from unauthorized access by individuals lacking the Volume key. Bacula effectively achieves this, however, it is important to note that encryption alone does not guard against alterations to the Volume.

The initial block of the Volume is the *Volume label*, which remains unencrypted. Certain information is necessary for the effective management of the Volume. The user-provided data within the *Volume label* are: the *hostname*, *volumename*, *poolname*. The *hostname* can be obscured through the use of the *STRONG* encryption mode.

Important: Data in your Catalog database, e.g. directories, filenames, and *JobLog* are not encrypted.

Attackers may also make undetected modifications to the Volumes. To safeguard Volumes against modifications, it is advisable to utilize the immutable feature known as Volume Protection.

The *XXH64* checksum inside each Volume is encrypted using the encryption key. This is not as strong as using a certified signature, but it provides substantial confidence that the block cannot be modified easily.

In conclusion, with Volume encryption activated, one can be assured that:

- An attacker cannot read any of your data

- An attacker cannot substitute a Volume with another one

- An attacker cannot modify the contents of a Volume

Important: Volume encryption should be complemented by the addition of Volume protection.

Compatibility between Encrypted and Unencrypted Volumes

The Volume contains a flag in the header of the first block which indicates whether the Volume is encrypted or not. This status is represented by the `VolEncrypted` field in the Catalog.

Volumes initialized using Bacula Enterprise Edition version 14.0 or earlier, characterized by the BB02 label format, can be read and appended in version 16 and later.

Volumes initialized or appended using version 16.0 and later, characterized by the BB03 label format, cannot be read with version 14.0 or prior versions.

Note: Bacula does not mix encrypted and unencrypted data inside the same Volume.

Volumes that have been labeled or relabeled on a Device with the directive Volume Encryption are classified as encrypted. Other Volumes, including those initialized with the BB02 format using version 14.0 or earlier, are regarded as unencrypted.

New Storage Daemon Directives

The following article presents directives for Storage Daemon Encryption.

You should first set up the Encryption command and install the needed script. Then, you will set the directives to enable the desirable encryption.

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.

Volume Encryption

The following directive must be incorporated into all devices used to encrypt Volumes:

Volume Encryption = <no|yes|strong>
> It allows you to enable encryption for a given device. The encryption can be of 3 different types:
>
> **no** Default, indicating that the device does not perform encryption.
>
> **yes** The device encrypts the data, but all information in the *Volume label* remains in clear text.
>
> **strong** The device encrypts the data, and obfuscate any information in the *Volume label* except for the details necessary for Volume management. The fields that are obfuscated are: *hostname*.

You can change the value of the directive Volume Encryption to `Yes` or `Strong` at any time. Volumes that are not encrypted can be read on this device, but cannot be appended until they are recycled. Switching between `Yes` and `Strong` is permissible, as the distinction lies solely in the unencrypted label rather than in the blocks that follow.

Changing the value of the directive back to `No` will deactivate the encryption and make all encrypted Volumes unreadable and ineffective on this device until they are recycled.

When the Device directive is set to `Volume Encryption = yes`, the blocks in the Volumes are encrypted using the *BLOCK_CIPHER_AES_128_XTS* or *BLOCK_CIPHER_AES_256_XTS* cipher algorithms. These symmetric ciphers are efficient and widely employed by most applications requiring symmetric block encryption.

Each block is encrypted using a *key* that is unique for the Volume and an *IV* (Initialization Vector), which corresponds to the block number stored in the block header. The *XTS* ciphers are specifically designed to support an *IV* with a low entropy.

The initial *block* of the Volume that holds the *Volume Label* is not encrypted, as certain fields, such as the *Volume Name*, are required to manage the Volume and the encryption. Users have an option to obfuscate non-essential fields that may contain sensitive information, e.g. *hostname*. These fields are replaced by the string *"OBFUSCATED"*.

The header of the block is not encrypted. This 24-byte header does not hold user information. See the content of the header:

- 32bit header option bit field
- 32bit block length
- 32bit block number
- `BB03` string
- 32bit Volume session id
- 32bit Volume session time.

The *Volume session time* indicates the time recorded by the Storage Daemon upon startup. The *Volume session id* is reset to zero when the daemon begins operation and is incremented with each backup performed by the Storage Daemon.

The 64bit *XXH64* checksum is encrypted alongside the data. To verify the checksum, the *block* must be decrypted. If the checksum matches, Bacula uses the right encryption key and the block remains unchanged. It is currently not possible to verify the integrity of the block without the encryption key.

Encryption Command

A directive has been introduced in the Storage resource of the Storage Daemon:

Encryption Command = <command>
> The **command** specifies an external program that must provide the keys used to encrypt the Volume.

The Storage Daemon invokes the Encryption Command each time a new Volume is either initialized or mounted on a device that has encryption enabled. We offer a straightforward example script designed to assist in the management of Volume encryption keys.

An example of the Encryption Command setting within the Storage resource in the Storage Daemon configuration is as follows:

```
Encryption Command = "/opt/bacula/scripts/key-manager.py getkey"
```

Important: The command is limited to 127 characters. The variable substitutions applicable to the *Autochanger Command* can also be utilized in the script.

The program can serve as an interface with your existing key management system or independently manage keys.

The sample script named `key-manager.py` can be installed through the `bacula-enterprise-storage-key-manager` package, which will place it in the `/opt/bacula/scripts` directory. Additionally, the `install-key-manager.sh` script, also part of the `bacula-enterprise-storage-key-manager` package, is designed to assist in setting up a Master Key:

```
# sudo -u bacula /opt/bacula/scripts/install-key-manager.sh check

# sudo -u bacula /opt/bacula/scripts/install-key-manager.sh install
```

The `key-manager.py` script can be used with the following options:

> **cipher** This is the cipher that Bacula must use. Bacula knows the following ciphers: *AES_128_XTS* and *AES_256_XTS*. The key lengths varies depending on the selected cipher. By default, the script uses the *AES_128_XTS* cipher.

> **key-dir** This is where the symmetric keys are stored. By default, they are stored in the `/opt/bacula/etc/keydir` directory.

For example, to use the *AES_256_XTS* cipher, type:

```
Encryption Command = "/opt/bacula/scripts/key-manager.py getkey --cipher AES_
→256_XTS"
```

key-manager Script and Using Master Key

The task of the key manager script is to provide the symmetric keys to the Storage Daemon to encrypt the Volumes.

The same symmetric key is used for both encryption and decryption of the Volume's content. To improve security, each volume is expected to be encrypted using a unique symmetric key. When needed, these keys must be generated for each new Volume or when an existing Volume is recycled. This process can lead to a large amount of key files that need to be managed, which must also be backed up or securely stored. To know more about backing up the keys, click here.

The master key is implemented using a public/private key pair. The public key is stored on the Storage Daemon and is used to generate the encrypted version of the symmetric keys by the `key-manager.py` script.

The private key may be stored outside the Storage Daemon server. It is not necessary for the private key to reside on the Storage Daemon in order to perform backups.

The symmetric key, along with a unique identifier for the Master Key, are stored in the Volume label and in the `KEYDIR` directory. In this case, the symmetric key is stored in plaintext.

At restore time, these two pieces of information and the Volume name are provided to the `key-manager.py` script which is responsible for delivering the correct symmetric key and for data decryption.

It is possible to store the symmetric key encrypted in the Volume label and in the `KEYDIR` directory by using the *stealth* mode.

The Master Key is intended to offer two primary benefits:

- To be able to decrypt multiple Volumes using one, or a set of Master Keys

- When the **stealth** mode is used, the symmetric key is securely stored in an encrypted format alongside the master key on disk. Consequently, any attempt to restore data without the private key becomes impossible. In the event of a server being compromised or stolen, unauthorized individuals will be unable to access your data.

The *stealth* mode can be configured in the key-manager.py script configuration file.

The `key-manager.py` script provided with Bacula uses GnuPG to manage the public/private key.

For further information regarding the `key-manager.conf` file and and the implementation of the `key-manager.py` script, refer to the relevant documentation:

key-manager.conf File Format

When using the Master Key feature, it is possible to configure your master keys in the `key-manager.conf` file located in `/opt/bacula/etc`. This file is automatically populated with a single master key at installation time. If you do not intend to use a master key, this file is not necessary.

This is an example of the `key-manager.conf` file generated at installation time:

```
[default]
gnupghome="/opt/bacula/etc/gnupg"

[0378FB9C839FF9F207834D89DB856A1A513B7AB4]
volume_regex=Volume[0-9]+|TestVolume[0-9]+
uid=bacula@localhost
passphrase=xm3ynBi7MfHTUovls4QV8OtBT5rfAnXwT8Wb7wjVRyCT
stealth=off
```

It is essential to define a section for each master key intended for use. The name of the section is the *Key-ID* of your public/private key. In the above example, there is one section for the *Key-ID* 0378FB9C839FF9F207834D89DB856A1A513B7AB4.

The supported fields are:

- **gnupghome**: This is where GnuPG stores the public and private keys.

When the `bacula-enterprise-storage-key-manager` package is installed, the default `gnupghome` directory is set to `/opt/bacula/etc/gnupg`.

Thus, if you plan to use GnuPG commands in the Storage Daemon host, it is necessary to employ the `--homedir` option like: `--homedir /opt/bacula/etc/gnupg`, or set the `GNUPGHOME` environment variable to `/opt/bacula/etc/gnupg`:

```
export GNUPGHOME="/opt/bacula/etc/gnupg"
```

To properly configure the `key-manager.conf` file with the master key values, you can get the *Key-ID* values by listing your keys:

```
bacula $ GNUPGHOME=/opt/bacula/etc/gnupg gpg -k
/opt/bacula/etc/gnupg/pubring.kbx
--------------------------------------------------
pub   rsa3072 2023-01-11 [SC]
      0378FB9C839FF9F207834D89DB856A1A513B7AB4
uid           [ultimate] Bacula <bacula@localhost>
sub   rsa3072 2023-01-11 [E]
```

Notice that the public key has a *subkey* that allows encryption. This is the *[E]* in the last line. If your public/private key does not own such a key, you cannot use it with the `key-manager.py` script.

- **volume_regex**: The `key-manager.py` script will use the master key

to encrypt the symmetric keys for the Volumes that match the regular expression specified here.

The **volume_regex** values are checked sequentially. The first regex that matches the Volume name submitted to the `key-manager.py` script will be used even if another *volume_regex* matches the name of the Volume. By default, the *volume_regex* is commented out to disable the master key feature.

- **uid**: The User ID of the key, which is simply the user name and

email address. The uid assists in identifying the key.

The **uid** serves only as a reference and is not used by the `key-manager.py` script.

- **passphrase**: This is the passphrase of the key.

When the **passphrase** of the public/private key is not set but required by the `key-manager.py` script, it depends on the gpg-agent to provide the key.

- **stealth**: When the stealth mode is used, the `key-manager.py` script stores the symmetric keys encrypted in the `KEYDIR` directory. The `volume_regex` option must be set when using the stealth mode.

When **stealth** is set to *on*, the `key-manager.py` does not keep the symmetric keys in clear text in the `KEYDIR` directory. When *on*, it is advisable not to specify a *passphrase*.

The following scenarios are possible:

- *stealth* mode is not used: the `key-manager.py` script can use the passphrase of the private key that you have provided in the configuration file to decrypt the symmetric key coming from the Volume, or use the clear version of the symmetric key stored locally.

- *stealth* mode is used: the `key-manager.py` script asks GnuPG to decrypt the symmetric key coming from the Volume.

  In this case, there are two potential outcomes:

  – The passphrase of the private key has been preset or is still in the cache of the *gpg* agent, allowing *gpg* to successfully decrypt the symmetric key.

  – Bacula waits until the user provides the passphrase.

This is an example to use the same master key to all Volumes:

```
[default]
gnupghome="/opt/bacula/etc/gnupg"

[0378FB9C839FF9F207834D89DB856A1A513B7AB4]
volume_regex=.*
uid=bacula@localhost
passphrase=xm3ynBi7MfHTUovls4QV8OtBT5rfAnXwT8Wb7wjVRyCT
stealth=on
```

key-manager Script Implementation

Example interaction using the sample `key-manager.py` script:

```
$ OPERATION=LABEL VOLUME_NAME=Volume0001 ./key-manager.py getkey --cipher AES_
↪128_XTS --key-dir /opt/bacula/keys
cipher: AES_128_XTS
cipher_key: G6HksAYDnNGr67AAx2Lb/vecTVjZoYAqSLZ7lGMyDVE=
volume_name: Volume0001

$ OPERATION=READ VOLUME_NAME=Volume0001 ./key-manager.py getkey --cipher AES_
↪128_XTS --key-dir /opt/bacula/keys
cipher: AES_128_XTS
cipher_key: G6HksAYDnNGr67AAx2Lb/vecTVjZoYAqSLZ7lGMyDVE=
volume_name: Volume0001

$ cat /opt/bacula/keys/Volume0001
cipher: AES_128_XTS
cipher_key: G6HksAYDnNGr67AAx2Lb/vecTVjZoYAqSLZ7lGMyDVE=
volume_name: Volume0001

$ OPERATION=READ VOLUME_NAME=DontExist ./key-manager.py getkey --cipher AES_
↪128_XTS --key-dir /opt/bacula/keys 2>/dev/null
error: no key information for volume "DontExist"
$ echo $?
0

$ OPERATION=BAD_CMD VOLUME_NAME=Volume0002 ./key-manager.py getkey --cipher
↪AES_128_XTS --key-dir /opt/bacula/keys 2>/dev/null
error: environment variable OPERATION invalid "BAD_CMD" for volume "Volume0002
↪"
$ echo $?
0
```

In the command above, notice that the keys are kept in a single directory, specifically `/opt/bacula/keys`, while the arguments are conveyed through the use of *environment variables*.

Bacula passes the following variables via the *environment*:

OPERATION This is set to *LABEL* when the Volume is being labeled. In this case the script is expected to generate a new key. Alternatively, it may be set to *READ* when the Volume already has a label, necessitating the Storage Daemon to use the existing key to read or append data to the Volume.

VOLUME_NAME This is the name of the Volume.

Some variables already exist to support a Master Key in the future. Although this feature is not currently available, it is anticipated to be introduced later:

ENC_CIPHER_KEY This is a base64 encoded version of the key encrypted by the master key.

MASTER_KEYID This is a base64 encoded version of the key ID of the master key that was used to encrypt the *ENC_CIPHER_KEY* above.

Bacula anticipates the following values in response:

volumename This is a repetition of the name of the Volume that is given to the script. This field is optional and ignored by Bacula.

cipher This is the cipher that Bacula must use. Bacula knows the following ciphers: *AES_128_XTS* and *AES_256_XTS*. The key lengths will vary based on the cipher selected.

cipher_key This is a symmetric *key* in *base 64* format.

comment This is a single line of text that is optional and ignored by Bacula.

error This is a single line error message. This is optional, but when provided, Bacula considers that the script returned an error and displays this error in the job log.

Bacula expects an *exit code* of 0. Should the script terminate with any other error code, all output will be disregarded, and Bacula will log a generic message along with the exit code.

To signal an error to Bacula, the script must use the *error* field and return an error code of 0.

Best Practices with Volume Encryption

The following chapters presents best practices articles regarding Storage Daemon Encryption.

Backup Your Keys

In the event that you lose your symmetric keys and/or your Master Key(s), the recovery of your data will not be possible. Therefore, it is important to backup your keys. Each time a new Volume is generated or recycled with encryption activated, a new symmetric key is created, therefore it is advisable to synchronize this key to a secure location immediately upon its creation.

By default, your symmetric keys are located in the `/opt/bacula/etc/keydir` directory. Thus, it is important to backup or synchronize the contents of this directory to a secure location.

For example, this is the `Vol-0004` file created in the `/opt/bacula/etc/keydir` directory:

```
# cat keydir/Vol-0004
cipher: AES_128_XTS
cipher_key: X9m2cWwekR+6xuPj3+XNbthXdSZkQ9qjW9CDfojzNrA=  <--- this is the␣
↪symmetric key
volume_name: Vol-0004
```

This file must exist in the Storage Daemon to recover your data stored in the `Vol-0004`.

The default directory for the keys, located at `/opt/bacula/etc/keydir`, can be modified using the `--key-dir` option in the command line, which is configured with the Encryption Command directive defined. This directory must be backed up regularly.

By default, your Master Keys are kept at the `/opt/bacula/etc/gnupg` directory. This location is defined in the key manager configuration file, typically found at `/opt/bacula/etc/key-manager.conf`, within the *[default]* section under the `gnupghome` directive as seen above. The `key-manager.conf` file can be moved using the `--config` option in the command associated with the Encryption Command directive. The default passphrase is also stored in the `key-manager.conf`. You just need to backup the `key-manager.conf` file, and the `/opt/bacula/etc/gnupg` directory.

You can export your default private key using the command:

```
# gpg --homedir /opt/bacula/etc/gnupg --output private.pgp --armor --export-
↪secret-key bacula@localhost
```

It requests the passphrase that is saved in you `key-manager.conf` file. This exports an ASCII armored version of your private key into the `private.pgp` file. You can print it and/or save it on a USB drive or in another location.

6.2 File Daemon Data Encryption

Important: This section describes file data encryption and signing within the File Daemon. If you are interested in encryption within the Storage Daemon at the volume level (encryption of data at rest) more information can be found in volumeencryption.

Bacula permits file data encryption and signing within the File Daemon (or Client) prior to sending data to the Storage Daemon. Upon restoration, file signatures are validated and any mismatches are reported. At no time does the Director or the Storage Daemon have access to unencrypted file contents.

It is very important to specify what this implementation does NOT do:

- There is one important restore problem to be aware of, namely, it's possible for the director to restore new keys or a **Bacula** configuration file to the client, and thus force later backups to be made with a compromised key and/or with no encryption at all. You can avoid this by not changing the location of the keys in your **Bacula** File daemon configuration file, and not changing your File daemon keys. If you do change either one, you must ensure that no restore is done that restores the old configuration or the old keys. In general, the worst effect of this will be that you can no longer connect the File daemon.

- The implementation does not encrypt file metadata such as file path names, permissions, and ownership. Extended attributes are also currently not encrypted. However, Mac OSX resource forks are encrypted.

Encryption and signing are implemented using RSA private keys coupled with self-signed X.509 public certificates. This is also sometimes known as PKI.

Each File Daemon should be given its own unique private/public key pair. In addition to this key pair, any number of "Master Keys" may be specified – these are key pairs that may be used to decrypt any backups should the File Daemon key be lost. Only the Master Key's public certificate should be made available to the File Daemon. Under no circumstances should the Master Private Key be shared or stored on the Client machine.

The Master Keys should be backed up to a secure location. The Master Keys should never be kept on the same machine as the Storage Daemon or Director if you are worried about an unauthorized party compromising either machine and accessing your encrypted backups.

While less critical than the Master Keys, File Daemon Keys are also a prime candidate for off-site backups.

Important: If you lose your encryption keys, backups will be unrecoverable. **ALWAYS** store a copy of your master keys in a secure, off-site location.

The basic algorithm used for each backup session (Job) is:

1. The File daemon generates a session key.

2. The FD encrypts that session key via PKE for all recipients (the file daemon, any master keys).

3. The FD uses that session key to perform symmetric encryption on the data.

Encryption Technical Details

The implementation uses 128bit AES-CBC, with RSA encrypted symmetric session keys. The RSA key is user supplied. If you are running OpenSSL 0.9.8 or later, the signed file hash uses SHA-256 – otherwise, SHA-1 is used.

End-user configuration settings for the algorithms are not currently exposed – only the algorithms listed above are used. However, the data written to Volume supports arbitrary symmetric, asymmetric, and digest algorithms for future extensibility, and the back-end implementation currently supports:

- Symmetric Encryption:
    - 128, 192, and 256-bit AES-CBC
    - Blowfish-CBC
- Asymmetric Encryption (used to encrypt symmetric session keys):
    - RSA
- Digest Algorithms:
    - MD5
    - SHA-1
    - SHA-256
    - SHA-512

The various algorithms are exposed via an entirely re-usable, OpenSSL-agnostic API (ie, it is possible to drop in a new encryption backend). The Volume format is DER-encoded ASN.1, modeled after the Cryptographic Message Syntax from RFC 3852. Unfortunately, using CMS directly was not possible, as at the time of coding a free software streaming DER decoder/encoder was not available.

Generating Private/Public Encryption Keys

Generate a Master Key Pair (once) with:

```
openssl genrsa -out master.key 2048
openssl req -new -key master.key -x509 -out master.cert
```

Save these files (**master.key** and **master.cert**) as you will need them to create File Daemon keys or in case you loose the File Daemon keys.

Generate a File Daemon Key Pair for each FD:

```
openssl genrsa -out fd-example.key 2048
openssl req -new -key fd-example.key -x509 -out fd-example.cert
cat fd-example.key fd-example.cert >fd-example.pem
```

When configuring the File Daemon (see this section), you will need the keys you just generated here, so you will need to transmit them to the machine where the FD is installed.

Note, there seems to be a lot of confusion around the file extensions given to these keys. For example, a **.pem** file can contain all the following: private keys (RSA and DSA), public keys (RSA and DSA) and (X.509) certificates. It is the default format for OpenSSL. It stores data Base64 encoded DER format, surrounded by ASCII headers, so is suitable for text mode transfers between systems. A **.pem** file may contain any number of keys either public or private. We use it in cases where there is both a public and a private key.

Typically, above we have used the **.cert** extension to refer to X.509 certificate encoding that contains only a single public key.

Data Encryption Configuration

Data encryption is configured in **Bacula** only in the File Daemon. The Job report that is produced at the end of each job has a line indicating whether or not encryption was enabled:

```
VSS: no
Encryption: no/yes <===
Accurate: no
...
```

Example Data Encryption Configuration

When configuring the FD, use the keys generated above in a FD configuration file that will look something like the following:

FileDaemon **bacula-fd.conf**

```
FileDaemon {
    Name = example-fd
    FDport = 9102     # where we listen for the director
    WorkingDirectory = /opt/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 20

    PKI Signatures = Yes   # Enable Data Signing
```

```
    PKI Encryption = Yes   # Enable Data Encryption
    PKI Keypair = "/opt/bacula/etc/fd-example.pem"   # Public and␣
→Private Keys
    PKI Master Key = "/opt/bacula/etc/master.cert"   # ONLY the␣
→Public Key
}
```

You must restart your File Daemon after making this change to the file.

Note: The PKIMasterKey directive is not mandatory, but if used will allow you to decrypt the files if ever the FD PKIKeypair is lost. If you loose the FD's PKIKeypair, you will not be able to recover your data unless you have used a PKIMasterKey.

Decrypting with Master Key

It is preferable to retain a secure, non-encrypted copy of the client's own encryption keypair. However, should you lose the client's keypair, recovery with the master keypair is possible.

First create a keypair with:

```
cat master.key master.cert >master.pem
```

Then modify your File Daemons configuration file to use the master keypair:

```
FileDaemon {
    Name = example-fd
    FDport = 9102         # where we listen for the director
    WorkingDirectory = /opt/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 20

    PKI Signatures = Yes   # Enable Data Signing
    PKI Encryption = Yes   # Enable Data Encryption
    PKI Keypair = "/opt/bacula/etc/master.pem" # Master Public and Private␣
→Keys
}
```

Restart your File Daemon and you should be able to recover your lost files.

7 Bacula TLS - Communications Encryption

Bacula TLS is built-in network encryption code to provide secure network transfer and mutual authentication similar to that offered by other TLS-enabled services such as web servers (**https://**) or email (**starttls**). The **Bacula** TLS encryption applies only to information transmitted across a network, so the data written to Volumes by the Storage daemon is not encrypted by this code. For data encryption, see the PKI options described in Data Encryption chapter.

Note: TLS is now implemented by default.

Supported features of this code include:

- Client/Server TLS Requirement Negotiation

- TLSv1.3 Connections with Server and Client Certificate Validation

  Depending on particular environments and configuration, the Bacula software may fall back to or even support only earlier versions of TLS. We strongly recommend to always use recent software builds.

- Forward Secrecy Support via Diffie-Hellman Ephemeral Keying

- TLS-PSK is used by default when TLS certificates are not configured.

This document will refer to both "server" and "client" contexts. These terms refer to the accepting and initiating peer, respectively. In addition, each of the three daemons (Director, File daemon, Storage daemon) as well as the user interface programs (**bconsole, tray monitor**, etc.) use the same TLS configuration directives. When we are speaking of one or all of these daemons/programs, we will generally refer to them as a "component".

Diffie-Hellman anonymous ciphers are not supported by this code. The use of DH anonymous ciphers increases the code complexity and places explicit trust upon the two-way CRAM-MD5 implementation. CRAM-MD5 is subject to known plaintext attacks, and it should be considered less secure than PKI certificate-based authentication.

Appropriate autoconf macros have been added to detect and use OpenSSL if enabled on the **./configure** line with **--with-openssl**; the use of OpenSSL is required for other functionality as well, and it should be included for all builds.

Note: Bacula Enterprise packages are always built with a recent, distribution specific version of OpenSSL and will therefore always support TLS transport encryption.

Stricter security requirements can be met by using FIPS conforming, validated OpenSSL modules.

7.1 TLS Configuration Directives

Additional configuration directives have been added to all the components (daemons) (Director, File daemon, and Storage daemon) as well as the various different Console programs).

Note that for the connection between a Storage Daemon and a File Daemon or between two Storage Daemons, the TLS directives that will be used are the ones defined in the FileDaemon resource on the client, and in the Storage resource on the Storage Daemon.

The default value of the directive **TLS-PSK Enable** is **yes**, if both **TLS Enable** and **TLS-PSK Enable** are enable on both side, then Bacula will use TLS certificates.

If none of TLS or TLS-PSK are enabled, then the TLS directives have no effect.

These new directives are defined as follows:

TLS Enable = <yes|no> Enable TLS support. If TLS is not enabled, none of the other TLS directives have any effect. In other words, even if you set **TLS Require = yes** you need to have TLS enabled or TLS will not be used.

TLS PSK Enable = <yes|no> Enable or Disable automatic TLS PSK support. TLS PSK is enabled by default between all **Bacula** components. The Pre-Shared Key used between the programs is the **Bacula** password. If both **TLS Enable** and **TLS PSK Enable** are enabled, the system will use TLS certificates.

TLS Require = \<yes|no\> Require TLS or TLS-PSK encryption. This directive is ignored unless one of **TLS Enable** or **TLS PSK Enable** is set to **yes**. If TLS is not required while TLS or TLS-PSK are enabled, then the **Bacula** component will connect with other components either with or without TLS or TLS-PSK

If TLS or TLS-PSK is enabled and TLS is required, then the **Bacula** component will refuse any connection request that does not use TLS.

TLS Authenticate = \<yes|no\> When **TLS Authenticate** is enabled, after doing the CRAM-MD5 authentication, **Bacula** will also do TLS authentication, then TLS encryption will be turned off, and the rest of the communication between the two **Bacula** components will be done without encryption. If TLS-PSK is used instead of the regular TLS, the encryption is turned off after the TLS-PSK authentication step.

If you want to encrypt communications data, use the normal TLS directives but do **not** turn on **TLS Authenticate**.

TLS Certificate = \<Filename\> The full path and filename of a PEM encoded TLS certificate. It will be used as either a client or server certificate, depending on the connection direction. PEM stands for Privacy Enhanced Mail, but in this context refers to how the certificates are encoded. This format is used because PEM files are base64 encoded and hence ASCII text based rather than binary. They may also contain encrypted information.

This directive is required in a server context, but it may not be specified in a client context if **TLS Verify Peer** is set to **no** in the corresponding server context.

Example:

File Daemon configuration file (**bacula-fd.conf**), resource configuration has **TLS Verify Peer=no:**

```
Director {
    Name = bacula-dir
    Password = "password"
    Address = director.example.com
    # TLS configuration directives
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = no
    TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
    TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
    TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

Having **TLS Verify Peer=no**, means the File Daemon, server context, will not check Director's public certificate, client context. There is no need to specify **TLS Certificate File** neither **TLS Key** directives in the Client resource, director configuration file. We can have the below client configuration in **bacula-dir.conf**:

```
Client {
    Name = client1-fd
    Address = client1.example.com
    FDPort = 9102
    Catalog = MyCatalog
    Password = "password"
    ...
    # TLS configuration directives
    TLS Enable = yes
```

```
    TLS Require = yes
    TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
}
```

TLS Key = <Filename> The full path and filename of a PEM encoded TLS private key. It must correspond to the TLS certificate.

TLS Verify Peer = <yes|no> Verify peer certificate. Instructs server to request and verify the client's X.509 certificate. Any client certificate signed by a known-CA will be accepted. Additionally, the client's X509 certificate Common Name must meet the value of the **Address** directive. If the **TLSAllowedCN** configuration option is used, the client's x509 certificate Common Name must also correspond to one of the CN specified in the **TLS Allowed CN** directive. This directive is valid only for a server and not in client context. The default is **yes**.

TLS Allowed CN = <string list> Common name attribute of allowed peer certificates. This directive is valid for a server and in a client context. If this directive is specified, the peer certificate will be verified against this list. In the case this directive is configured on a server side, the allowed CN list will not be checked if **TLS Verify Peer** is set to **no** (**TLS Verify Peer** is **yes** by default). This can be used to ensure that only the CN-approved component may connect. This directive may be specified more than once.

In the case this directive is configured in a server side, the allowed CN list will only be checked if **TLS Verify Peer = yes** (default). For example, in **bacula-fd.conf**, Director resource definition:

```
Director {
    Name = bacula-dir
    Password = "password"
    Address = director.example.com
    # TLS configuration directives
    TLS Enable = yes
    TLS Require = yes
    # if TLS Verify Peer = no, then TLS Allowed CN will not be checked.
    TLS Verify Peer = yes
    TLS Allowed CN = director.example.com
    TLS CA Certificate File = /opt/bacula/ssl/certs/root_cert.pem
    TLS Certificate = /opt/bacula/ssl/certs/client1_cert.pem
    TLS Key = /opt/bacula/ssl/keys/client1_key.pem
}
```

In the case this directive is configured in a client side, the allowed CN list will always be checked.

```
Client {
    Name = client1-fd
    Address = client1.example.com
    FDPort = 9102
    Catalog = MyCatalog
    Password = "password"
    ...
    # TLS configuration directives
    TLS Enable = yes
    TLS Require = yes
    # the Allowed CN will be checked for this client by director
    # the client's certificate Common Name must match any of
    # the values of the Allowed CN list
```

```
    TLS Allowed CN = client1.example.com
    TLS CA Certificate File = /opt/bacula/ssl/certs/ca_client1_cert.pem
    TLS Certificate = /opt/bacula/ssl/certs/director_cert.pem
    TLS Key = /opt/bacula/ssl/keys/director_key.pem
}
```

If the client doesn't provide a certificate with a Common Name that meets any value in the **TLS Allowed CN** list, an error message will be issued:

```
16-Nov 17:30 bacula-dir JobId 0: Fatal error: bnet.c:273 TLS certificate
verification failed. Peer certificate did not match a required commonName
16-Nov 17:30 bacula-dir JobId 0: Fatal error: TLS negotiation failed with FD␣
 ↪at
"192.168.100.2:9102".
```

TLS CA Certificate File = <Filename> The full path and filename specifying a PEM encoded TLS CA certificate(s). Multiple certificates are permitted in the file. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** (see above) is set to **no**, and are always required in a client context.

TLS CA Certificate Dir = <Directory> Full path to TLS CA certificate directory. In the current implementation, certificates must be stored PEM encoded with OpenSSL-compatible hashes, which is the subject name's hash and an extension of **.0**. One of **TLS CA Certificate File** or **TLS CA Certificate Dir** are required in a server context, unless **TLS Verify Peer** is set to **no**, and are always required in a client context.

TLS DH File = <Directory> Path to PEM encoded Diffie-Hellman parameter file. If this directive is specified, DH key exchange will be used for the ephemeral keying, allowing for forward secrecy of communications. DH key exchange adds an additional level of security because the key used for encryption/decryption by the server and the client is computed on each end and thus is never passed over the network if Diffie-Hellman key exchange is used. Even if DH key exchange is not used, the encryption/decryption key is always passed encrypted. This directive is only valid within a server context.

To generate the parameter file, you may use **openssl**:

```
openssl dhparam -out dh4096.pem -5 4096
```

7.2 Creating Self-Signed Certificate

A self-signed certificate for use with the **Bacula** TLS that will work, but will not allow proper certificate validation can be created easily. The .pem file containing both the certificate and the key valid for about ten years can be created as follows:

```
openssl req -new -x509 -nodes -out bacula.pem -keyout bacula.pem -days 3650
```

The above command will ask a number of questions. You may simply answer each of them by entering a return, or if you wish you may enter your own data.

Note, however, that self-signed certificates will only work for the outgoing end of connections. For example, in the case of the Director making a connection to a File Daemon, the File Daemon may be configured to allow self-signed certificates, but the certificate used by the Director must be signed by a certificate that is explicitly trusted on the File Daemon end.

This is necessary to prevent "man in the middle" attacks from tools such as ettercap. Essentially, if the Director does not verify that it is talking to a trusted remote endpoint, it can be tricked into talking to any malicious 3rd party who is relaying and capturing all traffic by presenting its own certificates to the Director and File Daemons. The only way to prevent this is by using trusted certificates, so that the man in the middle is incapable of spoofing the connection using his own.

7.3 Getting CA-Signed Certificate

The process of getting a certificate that is signed by a CA is quite a bit more complicated. You can purchase one from a number of PKI vendors, but that is not at all necessary for use with **Bacula**.

To get a CA signed certificate, you will either need to find a friend that has setup his own, trusted CA, or to become a CA yourself, and thus you can sign all your own certificates. The book OpenSSL by John Viega, Matt Mesier & Pravir Chandra from O'Reilly explains how to do it, or you can read the documentation provided in the Open-source PKI Book.

7.4 Example TLS Configuration Files

A few examples of the TLS portions of configuration files are shown, which should help you setting up your own.

For the examples in this section, we will consider three hosts named darkstar, arrakis and caladan, in order to have examples with components running in different machines. The following private key and public certificate files will be used:

```
* host darkstar.example.com:
/opt/bacula/ssl/keys/darkstar_key.pem
/opt/bacula/ssl//certs/darkstar_cert.pem
* host arrakis.example.com:
/opt/bacula/ssl/keys/arrakis_key.pem
/opt/bacula/ssl//certs/arrakis_cert.pem
* host caladan.example.com:
/opt/bacula/ssl/keys/caladan_key.pem
/opt/bacula/ssl//certs/caladan_cert.pem
```

The **TLS Verify Peer = yes** is present in the below examples to emphasize where it can be configured, but there is no need since this is the **default value used by Bacula**. This means that both server and client certificates will be checked in the TLS communication handshake. The CN in the peer certificate will be checked against the **Address** directive configured for the corresponding resource.

The **TLS Allowed CN = <FQDN! or IP address>** is configured to ensure that only the peer certificates with a CN Subject field listed here are authorized to communicate. For example, consider the below certificate:

```
# openssl x509 -in ../ssl/certs/arrakis_cert.pem -text -noout
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=US, ST=Yverdon-les-Bains, L=Vaud, O=Bacula Systems, CN=ca.
↪baculasystems.com/emailAddress=example@baculasystems.com
Validity
```

(continues on next page)

```
    Not Before: Sep 7 09:50:27 2016 GMT
    Not After : Dec 31 23:59:00 2021 GMT
Subject: C=CH, ST=Madrid, O=BaculaSystems, CN=arrakis.example.com
Subject
Public Key Info:
Public Key Algorithm: rsaEncryption Public-Key: (4096 bit)
Modulus:
    00:ae:7f:3f:6f:22:27:c3:c5:f5:d3:d6:f9:fd:8f:
    a0:7c:40:33:4c:90:1e:43:b2:14:fa:d3:82:00:b0:
    88:df:71:43:29:f7:7b:de:b4:87:0a:52:80:43:f7:
    1a:c9:1f:0f:c1:7c:27:00:45:5b:c4:a3:c3:a0:4a:
    14:53:7b:f6:33:54:44:70:41:8c:50:70:a1:eb:50:
    7a:52:87:6d:6c:84:2e:e5:48:86:9c:60:48:96:8e:
    48:2c:4b:13:87:af:94:cf:56:c1:74:e9:f5:4b:f9:
    16:32:fe:3a:65:a1:ef:eb:99:97:1a:10:e6:7f:6e:
    f3:18:1f:1f:67:f7:2a:38:0b:4b:00:70:0b:34:37:
    8c:04:56:02:ec:41:43:35:0c:c5:0c:8a:b2:91:00:
    7c:39:95:65:1e:b4:49:44:f6:af:87:d9:27:1e:a2:
    74:b3:5d:98:ac:da:53:fe:f6:9b:ee:15:fa:34:29:
    14:48:90:d6:61:31:43:3e:c0:30:f6:59:bb:b8:00:
    3b:98:a3:e8:d5:73:f1:ff:f5:23:d5:ac:87:57:ce:
    18:9f:35:1f:c2:4d:ba:44:05:b4:e3:ba:47:17:6a:
    76:5c:84:5c:f9:ce:83:19:87:ff:67:5b:82:24:4a:
    35:99:b1:91:7c:43:c9:84:2f:d4:1d:cf:6f:23:84:
    13:ae:59:28:66:a5:da:a4:5d:14:a4:04:69:21:4b:
    dd:d3:68:a3:16:0e:5f:23:4d:51:12:10:e3:f5:24:
    38:51:67:bf:cf:73:10:9a:b3:44:ff:87:bf:23:47:
    db:36:f0:c4:4a:6a:16:21:1f:ec:3c:1e:a9:c8:84:
    b4:2e:e5:d7:a1:d7:29:57:69:be:67:e9:1f:f5:66:
    dc:4b:c8:89:58:53:cd:c3:63:5e:58:86:48:db:71:
    3b:9f:25:1a:91:27:93:bf:1f:20:49:fc:b9:5b:ea:
    ba:b5:29:28:f2:a2:10:c5:ed:1c:fa:75:11:d2:22:
    7a:fd:50:be:56:e7:13:b1:a6:59:3b:aa:4b:8e:54:
    4b:1a:10:d8:6b:9c:46:ce:d8:7e:9a:f5:e7:28:62:
    6b:25:7c:ad:e6:64:4c:c0:4e:dc:1a:d8:c6:20:68:
    8b:3a:7a:8d:86:df:2e:e5:ab:39:7d:a1:3a:84:19:
    55:9b:46:2c:81:19:77:2c:2f:ca:6f:49:e0:92:98:
    c5:36:5a:db:4c:d8:58:04:4c:af:17:38:0e:2c:b1:
    21:1d:8b:88:69:69:fa:de:e7:fc:f0:9a:1a:71:1b:
    5e:68:51:b1:ef:44:1c:d6:a4:2d:55:93:b8:4c:e7:
    e2:dc:5b:99:ed:79:3a:02:6e:4e:61:32:62:03:a8: be:b2:4d
Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints:
    CA:FALSE
Netscape Comment:
    OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
    EC:FC:DC:5C:A5:50:1B:AE:B5:04:40:E3:C6:72:63:9F:F8:34:A3:89
X509v3 Authority Key Identifier:
    keyid:3B:36:FF:19:37:DF:12:D1:78:CC:F7:88:13:25:3D:CD:62:59:0F:9B
Signature Algorithm: sha256WithRSAEncryption
```

```
      6d:f0:74:43:a7:9a:1b:99:ee:9e:51:e8:9c:5f:4c:f3:c3:f4:
      ff:f5:5c:b4:e2:33:07:ed:fd:d5:09:e8:6f:b5:83:51:66:58:
      53:66:04:75:46:6b:5b:2a:ac:d8:54:78:df:de:d0:fc:6c:57:
      6e:0c:c9:3c:cf:62:64:63:cd:f3:a2:5b:fe:57:8c:12:7a:08:
      6a:53:b0:de:0e:c2:c7:35:9c:78:d5:27:98:05:b3:8c:54:fa:
      4f:0a:34:fc:4d:5f:14:93:5a:b1:86:2d:4a:74:e4:d8:83:e7:
      99:dd:36:56:a0:11:ff:d8:68:d2:62:13:72:96:4f:66:08:7c:
      f7:ce:7f:8f:2a:7d:c0:fc:fe:bf:1a:38:43:cc:01:aa:46:e4:
      ac:d7:be:65:51:a6:bc:6c:80:ed:54:76:0b:b8:b8:4b:a1:85:
      d5:9b:14:2b:65:19:cf:0f:65:5f:32:b8:9e:36:5c:c2:6f:92:
      7c:c5:db:90:11:40:db:e7:37:23:c5:78:1f:c0:09:c5:11:b8:
      06:30:3b:95:82:17:3e:49:51:7e:91:45:2e:a6:d1:89:1f:49:
      dc:6d:a9:1d:92:53:bd:ff:a2:cf:7b:8e:40:ab:96:4a:82:fd:
      5e:c7:04:53:2d:55:0c:78:57:03:5c:c1:6c:b1:12:ce:5f:be:
      10:9f:76:2c:5a:79:5a:a3:3c:68:ee:8c:69:c8:ac:cc:c5:09:
      fb:19:50:cb:5e:61:8a:9b:47:74:60:68:2a:94:49:6f:0f:22:
      78:6d:9b:ff:76:24:60:d1:ee:c6:59:59:e2:6f:f2:69:3f:ae:
      43:53:54:a1:ab:e0:ae:30:13:82:64:9c:ce:c1:fa:de:6a:7d:
      bb:94:a4:05:6e:03:84:d2:f1:f2:5d:1d:45:4c:bb:88:0b:6a:
      b6:74:8b:4b:fb:cf:99:6e:1d:ed:df:67:97:92:a7:0d:08:a7:
      57:be:7e:6e:06:13:f7:42:12:7a:05:aa:a9:1c:1b:1d:86:73:
      00:66:a7:07:88:d0:2d:eb:2b:dc:6e:e0:61:15:d9:ff:43:65:
      2d:99:c1:e6:80:ea:26:c4:08:ae:3c:12:ef:f0:6e:15:00:33:
      53:6a:c1:e3:14:5c:f3:ec:df:72:c1:ee:ca:ff:f6:c9:51:22:
      79:62:af:00:07:92:7c:0c:75:17:98:1d:b2:43:b2:fa:a7:41:
      d4:64:8e:3e:47:1c:f6:a0:aa:e3:30:d1:d7:5c:91:5c:ea:80:
      dc:31:8f:a2:40:fc:ac:4d:05:02:cb:c3:b4:f6:b5:98:58:7e:
      31:c2:0c:85:1b:a4:95:ed:77:bb:dc:95:12:81:45:6c:5c:2f:
      c4:7e:d3:86:9c:b0:1a:d4
```

If a component (director, file daemon, storage daemon or console) has **TLS Allowed CN = arrakis.example.com,**, it will accept only connections from peers that have "CN=arrakis.example.com" in the subject field of their certificate. The **TLS Allowed CN** directive can have a list of values like **TLS Allowed CN = darkstar.example.com, arrakis.example.com.**. This will permit the clients with CN Subject field equal to **darkstar.example.com** or **arrakis.example.com** to be allowed for connection. The below examples are configured with **TLS Allowed CN** to show where and how to configure this directive.

Enable TLS Communications Encryption between Daemons

Enable TLS Communications Encryption between Console and Director

Director and Console on the Same "darkstar" Host

1. If you are using an anonymous console: You only need to define the TLS directives in the resource of both **bacula-dir.conf** and **bconsole.conf** files.

   • In **bacula-dir.conf**:

   ```
   Director {
       Name = darkstar-dir
   ```

```
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = darkstar.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
→darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bconsole.conf**:

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
→darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

#. If a named console is used: You only need to define the TLS directives in the resource of both **bacula-dir.conf** and **bconsole.conf**.

- In **bacula-dir.conf:**: There is no need to configure TLS in the Director resource for a named console:

```
Director {
    Name = darkstar-dir
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
}
```

Instead, the named Console resource has the TLS configuration:

```
Console {
    Name = darkstar-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
↪certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.
↪pem
}
```

- In **bconsole.conf:**: There is no need to configure TLS in the Director resource as it was in the case of the anonymous console above:

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
}
```

Instead, the Console resource contains the TLS configuration:

```
Console {
    Name = darkstar-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
↪certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.
↪pem
}
```

Director and Console on Different Hosts

If you have your `bconsole` installed on another host than the Director one, then it is more likely that different public certificate and private key files for director and for console are used. Let's consider "darkstart-dir" director on "darkstar.example.com" and "arrakis-con" console on "arrakis.example.com".

1. If you're using an anonymous console:

   - In **bacula-dir.conf**:

```
Director {
    Name = darkstar-dir
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = darkstar.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bconsole.conf**:

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

2. If you are using a named console:

- In **bacula-dir.conf**: There is no need to configure TLS in the Director resource:

```
Director {
    Name = darkstar-dir
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
}
```

Instead, the Console resource has the TLS configurations:

```
Console {
    Name = arrakis-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Allowed CN = arrakis.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
↪certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.
↪pem
}
```

- In **bconsole.conf**: It is not needed to configure TLS in the Director resource:

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
}
```

Instead, the Console resource needs the TLS configuration:

```
Console {
    Name = arrakis-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
↪certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪arrakis_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.
↪pem
}
```

Enable TLS Communications Encryption between Director and File Daemon

Director and File Daemon on the Same "darkstar" Host

- In **bacula-dir.conf**:

```
Client {
    Name = darkstar-fd
    Address = darkstar.example.com
    FD Port = 9112
    Catalog = MyCatalog
    Password = "password"
```

(continues on next page)

```
    AutoPrune = no
    Maximum Concurrent Jobs = 4
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bacula-fd.conf**:

```
Director {
    Name = darkstar-dir
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = darkstart.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Director and File Daemon on Different Hosts

Let's consider "darkstar-dir" director at "darkstar.example.com" and the "arrakis-fd" file daemon on "arrakis.example.com".

- In **bacula-dir.conf:**:

```
Client {
    Name = arrakis-fd
    Address = arrakis.example.com
    FD Port = 9112
    Catalog = MyCatalog
    Password = "password"
    AutoPrune = no
    Maximum Concurrent Jobs = 4
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bacula-fd.conf**:

```
Director {
    Name = darkstar-dir
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = arrakis.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

Enable TLS Communications Encryption between Director and Storage Daemon

Director and Storage Daemon on the Same "darkstar" Host

- In **bacula-dir.conf**:

```
Storage {
    Name = VTL-storage
    SD Port = 9113
    Address = darkstar.example.com
    Password = "password"
    Device = "Virtual Tape Library"
    Autochanger = yes
    Media Type = VTL
    Maximum Concurrent Jobs = 30
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
    Name = darkstar-dir
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = darkstart.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
```

```
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Director and Storage Daemon on Different Hosts

Let's consider the "darkstart-dir" director on "darkstar.example.com" and the "caladan-sd" Storage Daemon running on "caladan.example.com".

- In **bacula-dir.conf**:

```
Storage {
    Name = VTL-storage
    SD Port = 9113
    Address = caladan.example.com
    Password = "password"
    Device = "Virtual Tape Library"
    Autochanger = yes
    Media Type = VTL
    Maximum Concurrent Jobs = 30
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
    Name = darkstar-dir
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Allowed CN = caladan.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Enable TLS Communications Encryption between File Daemon and Storage Daemon

Let's consider "darkstar-fd" (on "darkstar.example.com") and "arrakis-fd" (at "arrakis.example.com") clients, needing to connect to "caladan-sd" storage daemon at "caladan.example.com" using TLS.

- In **bacula-fd.conf** file at darkstar.example.com:

```
FileDaemon {
    Name = darkstar-fd
```

```
    FD Port = 9112
    FD Address = darkstar.example.com
    WorkingDirectory = /usr/local/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 10
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bacula-fd.conf** file at "arrakis.example.com":

```
FileDaemon {
    Name = arrakis-fd
    FD Port = 9112
    FD Address = arrakis.example.com
    WorkingDirectory = /usr/local/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 10
    TLS Enable = yes
    TLS Require = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

- In bacula-sd.conf file at "caladan.example.com":

```
Storage {
    Name = caladan-sd
    SD Port = 9113
    SD Address = caladan.example.com
    WorkingDirectory = "/usr/local/bacula/working"
    Pid Directory = "/var/run"
    Maximum Concurrent Jobs = 40
    TLS Enable = yes
    TLS Require = yes
    TLS Allowed CN = darkstar.example.com , arrakis.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Note: The **TLS Allowed CN** directive is only configured on the Storage Daemon side. This is

due to the fact that the communication in this case is always between the Storage Daemon acting as a "TLS server" and the File Daemon acting as a "TLS client". In this case, the client is the peer in the TLS communications context. The CN in the client's certificate subject will be checked by the Storage Daemon if it is an *Allowed CN*. in the case of the above example, **TLS AllowedCN = darkstar.example.com, arrakis.example.com** is allowing "darkstar-fd" (at darkstar.example.com) and "arrakis-fd" (at arrakis.example.com) clients to connect to "caladan-sd" storage daemon at "caladan.example.com" using TLS.

Enable TLS Authentication Between Daemons

Bacula by default uses SCRAM-SHA256 or CRAM-MD5 authentication using the **Password** directive as a shared secret for each daemon. In addition to this, you can also use TLS to provide a more secure authentication between the daemons. This is achieved by using the **TLS Authenticate = yes** directive in the main daemon configuration resource in addition to configuring **TLS Enable, TLS Require, TLS Certificate CA File, TLS Certificate** and **TLS Key** directives. Please find examples below.

Notice a very important feature of enabling **TLS Authenticate** to your daemons: **if you enable the TLS authentication, the TLS encryption will be turned off and communication between the daemons will be done without Encryption**.

Enabling TLS Authentication between Director and Console

#. If you're using an anonymous console: You only need to define the TLS directives in the resource of both **bacula-dir.conf** and **bconsole.conf** files.

In **bacula-dir.conf**:

```
Director {
    Name = darkstar-dir
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
→certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
→darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.
→pem
}
```

In **bconsole.conf**:

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_
↪cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.
↪pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

#. If you are using a named console: You only need to define the TLS directives in the resource of both **bacula-dir.conf** and **bconsole.conf** .

- In **bacula-dir.conf**: There is no need to configure TLS in the resource as for option
  SEE:

```
Director {
    Name = darkstar-dir
    DIR Port = 9111
    DIR Address = darkstar.example.com
    QueryFile = "/usr/local/bacula/scripts/query.sql"
    WorkingDirectory = "/usr/local/bacula/working"
    PidDirectory = "/var/run"
    Maximum Concurrent Jobs = 10
    Password = "password"
    Messages = Daemon
}
```

Instead, the resource has the TLS configurations:

```
Console {
    Name = darkstar-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/
↪certs/root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/
↪darkstar_cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.
↪pem
}
```

- In **bconsole.conf**: There is no need to configure TLS in the resource as for option
  SEE:

```
Director {
    Name = darkstar-dir
    DIRport = 9111
    Address = darkstar.example.com
    Password = "password"
}
```

Instead, the resource has the TLS configurations:

```
Console {
    Name = darkstar-con
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_
→cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_cert.
→pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

Enabling TLS Authentication between Director and Client

Let's consider "darkstart-dir" director at "darkstar.example.com and" and the "arrakis-fd" file daemon at "arrakis.example.com".

- In **bacula-dir.conf**:

```
Client {
    Name = arrakis-fd
    Address = arrakis.example.com
    FD Port = 9112
    Catalog = MyCatalog
    Password = "password"
    AutoPrune = no
    Maximum Concurrent Jobs = 4
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bacula-fd.conf**:

```
Director {
    Name = darkstar-dir
```

```
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Authenticate = yes
    TLS Allowed CN = arrakis.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

Enabling TLS Authentication between Director and Storage

Let's consider "darkstart-dir" director at "darkstar.example.com" and the "caladan-sd" storage daemon on "caladan.example.com".

- In **bacula-dir.conf**:

```
Storage {
    Name = VTL-storage
    SD Port = 9113Address = caladan.example.com
    Password = "password"
    Device = "Virtual Tape Library"
    Autochanger = yes
    Media Type = VTL
    Maximum Concurrent Jobs = 30
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In bacula-sd.conf:

```
Director {
    Name = darkstar-dir
    Password = "password"
    TLS Enable = yes
    TLS Require = yes
    TLS Verify Peer = yes
    TLS Authenticate = yes
    TLS Allowed CN = caladan.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_
```

```
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Enabling TLS Authentication between Client and Storage

Let's consider "darkstar-fd" (on "darkstar.example.com") and "arrakis-fd" (at "arrakis.example.com") clients need to connect to the "aladan-sd" storage daemon running on "caladan.example.com" using TLS.

- In **bacula-fd.conf** file at darkstar.example.com:

```
FileDaemon {
    Name = darkstar-fd
    FD Port = 9112
    FD Address = darkstar.example.com
    WorkingDirectory = /usr/local/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 10
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/darkstar_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/darkstar_key.pem
}
```

- In **bacula-fd.conf** file at "arrakis.example.com":

```
FileDaemon {
    Name = arrakis-fd
    FD Port = 9112
    FD Address = arrakis.example.com
    WorkingDirectory = /usr/local/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 10
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
→root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/arrakis_
→cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/arrakis_key.pem
}
```

- In bacula-sd.conf file at "caladan.example.com":

```
Storage {
    Name = caladan-sd
    SD Port = 9113
    SD Address = caladan.example.com
    WorkingDirectory = "/usr/local/bacula/working"
    Pid Directory = "/var/run"
    Maximum Concurrent Jobs = 40
    TLS Enable = yes
    TLS Require = yes
    TLS Authenticate = yes
    TLS Allowed CN = darkstar.example.com , arrakis.example.com
    TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/
↪root_cert.pem
    TLS Certificate = /usr/local/bacula/etc/ssl/certs/caladan_
↪cert.pem
    TLS Key = /usr/local/bacula/etc/ssl/keys/caladan_key.pem
}
```

Using Certificates Issued by Different Root

It is possible to have a TLS environment that uses certificates issued by different CAs. In previous examples, we have been using only one root CA:

```
# openssl genrsa -out ./keys/root_key.pem 4096
# openssl req -new -x509 -batch -config ./openssl.cnf -key ./keys/root_key.
↪pem -days 36500 -out ./certs/root_cert.pem
```

In the case of the use of multiple root CA in your **Bacula** environment, there are two possible ways to configure:

1. to concatenate all the root CA certificates into one **.pem** file used in the **TLS CA Certificate File** directive:

   ```
   # cat root_cert_ca1.pem root_cert_ca2.pem root_cert_ca3.
   ↪pem > root_cert_ca.pem
   ```

   - use **TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert_ca.pem** in resources TLS directives definitions.

2. use the **TLS CA Certificate Dir** directive instead. In this case, the certificates should have OpenSSL-compatible hashes. Please find below an example:

```
root@darkstar:/opt/bacula/ssl/certs# ls -l
...
lrwxrwxrwx 1 root root 19 Oct 25 23:10 7293a8c5.0 -> root_ca1_cert.
↪pem
lrwxrwxrwx 1 root root 19 Oct 25 23:10 8fb0c2b0.0 -> root_ca2_cert.
↪pem
lrwxrwxrwx 1 root root 18 Oct 25 23:10 a6476ecf.0 -> root_ca3_cert.
↪pem
-rw-r--r-- 1 root root 2134 Sep 7 16:03 root_ca1_cert.pem
-rw-r--r-- 1 root root 2134 Sep 7 15:56 root_ca2_cert.pem
-rw-r--r-- 1 root root 2134 Sep 7 11:47 root_ca3_cert.pem
```

- **TLS CA Certificate File = /usr/local/bacula/etc/ssl/certs/root_cert_ca.pem** in resources TLS directives definitions.

8 File System Snapshot with bsnapshot

Since Bacula Enterprise version 8.2, it is possible for Clients to perform automatic File System Snapshots backup. It is also possible to manage Snapshots from Bacula's bconsole tool through a unique interface.

The plugin provides file-level restore at the snapshot state.

The following Snapshot backends are supported:

- BTRFS
- ZFS
- LVM

Note: Some restrictions described *here* apply to the LVM backend.

By default, Snapshots are mounted (or directly available) under .snapshots directory on the root filesystem (On ZFS, the default is .zfs/snapshots).

The Snapshot backend program is called bsnapshot and is available in the bacula-enterprise-snapshot package. In order to use the Snapshot Management feature, the package must be installed on the Client.

The bsnapshot program can be configured using `/opt/bacula/etc/bsnapshot.conf` file. The following parameters can be adjusted in the configuration file:

- `sudo=<yes|no>` Use sudo to run commands
- `disabled=<yes|no>` Disable snapshot support
- `retry=` Configure the number of retries for some operations
- `snapshot_dir=` Use a custom name for the Snapshot directory. (.SNAPSHOT, .snapdir, etc.)
- `lvm_snapshot_size=` Specify a custom snapshot size for a given LVM volume
- `mountopts=` Specify a custom mount option for a given device (available since 10.0.4)
- `trace=` Specify a trace file
- `debug=` Specify a debug level

There is an example as follows:

```
# cat /opt/bacula/etc/bsnapshot.conf
trace=/tmp/snap.log
debug=10
lvm_snapshot_size=/dev/ubuntu-vg/root:5%
mountopts=nouuid
mountopts=/dev/ubuntu-vg/root:nouuid,nosuid
```

8.1 Application Quiescing

When using Snapshots, it is very important to quiesce applications that are running on the system. The simplest way to quiesce an application is to stop it. Usually, taking the Snapshot is very fast, and the downtime is only about a couple of seconds.

If downtime is not possible and/or the application provides a way to quiesce, directives RunBeforeJob and RunAfterJob can be used.

8.2 Director Directives

The use of the Snapshot Plugin on the FileDaemon is determined by the Enable Snapshot Fileset directive. The default is no.

```
Fileset {
  Name = LinuxHome
  Enable Snapshot = yes
  Include {
    Options = { Compression = LZO }
      File = /home
  }
}
```

By default, Snapshots are deleted from the Client at the end of the backup. To keep Snapshots on the Client and record them in the Catalog for a determined period, it is possible to use the Snapshot Retention directive in the Client or in the Job resource. The default value is 0 seconds. If, for a given Job, both Client and Job Snapshot Retention directives are set, the Job directive will be used.

```
Client {
  Name = linux1
  ...
  Snapshot Retention = 5 days
}
```

To automatically prune Snapshots, it is possible to use the following RunScript command:

```
Job {
  ...
  Client = linux1
  ...
  RunScript {
    RunsOnClient = no
    Console = "prune snapshot client=%c yes"
    RunsAfter = yes
  }
}
```

In RunScripts, the AfterSnapshot keyword for the RunsWhen directive will allow a command to be run just after the Snapshot creation. AfterSnapshot is a synonym for the AfterVSS keyword.

```
Job {
  ...
  RunScript {
```

```
      Command = "/etc/init.d/mysql start"
      RunsWhen = AfterSnapshot
      RunsOnClient = yes
   }
  RunScript {
      Command = "/etc/init.d/mysql stop"
      RunsWhen = Before
      RunsOnClient = yes
   }
}
```

8.3 Job Output Information

Information about Snapshots are displayed in the Job output. The list of all devices used by the Snapshot Engine is displayed, and the Job summary indicates if Snapshots were available.

```
JobId 3:     Create Snapshot of /home/build
JobId 3:     Create Snapshot of /home/build/subvol
JobId 3:     Delete snapshot of /home/build
JobId 3:     Delete snapshot of /home/build/subvol
...
JobId 3: Bacula 127.0.0.1-dir 7.2.0 (23Jul15):
  Build OS:               x86_64-unknown-linux-gnu archlinux
  JobId:                  3
  Job:                    Incremental.2015-02-24_11.20.27_08
  Backup Level:           Full
...
  Snapshot/VSS:           yes
...
  Termination:            Backup OK
```

8.4 Snapshot Bconsole Commands

The snapshot command will display by default the following menu:

```
*snapshot
Snapshot choice:
     1: List snapshots in Catalog
     2: List snapshots on Client
     3: Prune snapshots
     4: Delete snapshot
     5: Update snapshot parameters
     6: Update catalog with Client snapshots
     7: Done
Select action to perform on Snapshot Engine (1-7):
```

The snapshot command can also have the following parameters:

```
[client=<client-name> | job=<job-name> | jobid=<jobid>]
 [delete | list | listclient | prune | sync | update]
```

It is also possible to use traditional list, llist, update, prune or delete commands on Snapshots.

```
*llist snapshot jobid=5
 snapshotid: 1
       name: NightlySave.2015-02-24_12.01.00_04
 createdate: 2015-02-24 12:01:03
     client: 127.0.0.1-fd
    fileset: Full Set
      jobid: 5
     volume: /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
     device: /home/btrfs
       type: btrfs
  retention: 30
    comment:

* snapshot listclient
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:8102
Snapshot       NightlySave.2015-02-24_12.01.00_04:
  Volume:      /home/.snapshots/NightlySave.2015-02-24_12.01.00_04
  Device:      /home
  CreateDate:  2015-02-24 12:01:03
  Type:        btrfs
  Status:      OK
  Error:
```

In order to update the catalog with Client snapshots option (or snapshot sync), the Director contacts the FileDaemon, lists snapshots of the system and creates catalog records of the Snapshots.

```
*snapshot sync
Automatically selected Client: 127.0.0.1-fd
Connecting to Client 127.0.0.1-fd at 127.0.0.1:8102
Snapshot       NightlySave.2015-02-24_12.35.47_06:
  Volume:      /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
  Device:      /home
  CreateDate:  2015-02-24 12:35:47
  Type:        btrfs
  Status:      OK
  Error:
Snapshot added in Catalog

*llist snapshot
 snapshotid: 13
       name: NightlySave.2015-02-24_12.35.47_06
 createdate: 2015-02-24 12:35:47
     client: 127.0.0.1-fd
    fileset:
      jobid: 0
     volume: /home/.snapshots/NightlySave.2015-02-24_12.35.47_06
     device: /home
       type: btrfs
  retention: 0
    comment:
```

8.5 LVM Backend Restrictions

LVM Snapshots are quite primitive compared to ZFS, BTRFS, NetApp and other systems. For example, it is not possible to use Snapshots if the Volume Group (VG) is full. The administrator must keep some free space in the VG to create Snapshots.

The amount of free space required depends on the activity of the Logical Volume (LV). bsnapshot uses 10% of the LV by default. This number can be configured per LV in the bsnapshot.conf file.

```
[root@system1]# vgdisplay
  --- Volume group ---
 VG Name               vg_ssd
 System ID
 Format                lvm2
...
 VG Size               29,81 GiB
 PE Size               4,00 MiB
 Total PE              7632
 Alloc PE / Size       125 / 500,00 MiB
 Free  PE / Size       7507 / 29,32 GiB
...
```

It is also not advisable to leave snapshots on the LVM backend. Having multiple snapshots of the same LV on LVM will slow down the system.

Only Ext4, XFS and EXT3 filesystems are supported with the Snapshot LVM backend.

8.6 Debug Options

To get low level information about the bsnapshot, the debug tag "snapshot" should be used in the setdebug command.

```
*setdebug level=10 tags=snapshot client
*setdebug level=10 tags=snapshot dir
```

9 File Deduplication using Base Jobs

A base job is sort of like a Full save except that you will want the Fileset to contain only files that are unlikely to change in the future (i.e. a snapshot of most of your system after installing it). After the base job has been run, when you are doing a Full save, you specify one or more Base jobs to be used. All files that have been backed up in the Base job/jobs but not modified will then be excluded from the backup. During a restore, the Base jobs will be automatically pulled in where necessary.

This can be a very nice optimization for your backups. Basically, imagine that you have 100 nearly identical Windows or Linux machines containing the OS and user files. Now for the OS part, a Base job will be backed up once, and rather than making 100 copies of the OS, there will be only one. If one or more of the systems have some files updated, no problem, they will be automatically saved and restored.

The Job directive *Base=Jobx, Joby...* permits you to specify the list of jobs that will be used during a Full backup as base.

```
Job {
    Name = BackupLinux
    Level= Base
    ...
}

Job {
    Job Name = BackupZog4
    Base = BackupZog4, BackupLinux
    Accurate = yes
    ...
}
```

In this example, the job *BackupZog4* will use the most recent version of all files contained in *BackupZog4* and *BackupLinux* jobs. Base jobs should have run with *level=Base* to be used.

By default, Bacula will compare permissions bits, user and group fields, modification time, size and the checksum of the file to choose between the current backup and the BaseJob file list. You can change this behavior with the BaseJob Fileset option. This option works like the verify= one, that is described in the Fileset chapter.

```
Fileset {
    Name = Full
    Include = {
        Options {
            BaseJob = pmugcs5
            Accurate = Ms
            Verify = pin5
        }
        File = /
    }
}
```

Important note : The current implementation doesn't permit to scan a Volume with the **bscan** facility. The result does not permit the restore of files. It is therefore recommended to not prune File or Job records with Basejobs in use.

The "M" option letter for the Accurate directive in the Fileset Options block is the recommended option for Jobs that refer to BaseJobs as it will backup files based on the last backup time, which is more useful, because the mtime/ctime timestamps ("m" and "c" accurate option letters, respectively) may differ on various Clients, causing unnecessary files to be backed up.

```
Job {
    Name = USR
    Level = Base
    Fileset = BaseFS
    ...
}

Job {
    Name = Full
    Fileset = FullFS
    Base = USR
    ...
```

```
}

Fileset {
    Name = BaseFS
    Include {
        Options {
            Signature = MD5
        }
        File = /usr
    }
}

Fileset {
    Name = FullFS
    Include {
        Options {
            Accurate = Ms # check for mtime/ctime and Size
            Signature = MD5
        }
        File = /home
        File = /usr
    }
}
```

Base Jobs are not compatible with Copy or Migration Jobs.

10 REST API

The Bacula REST API specification permits web GUI programmers to interface with Bacula and to obtain information about Bacula objects, create objects, modify objects, and delete objects. Objects are either configuration information in the various Bacula component configuration files, or objects that are stored in the Bacula catalog database. The API also permits the programmer to use a command interface to run Bacula.

We assume that the person reading this document is thoroughly familiar with Bacula and experienced in using REST APIs to build GUI applications.

10.1 Available Requests Methods

API commands may be send by two request methods:

- **POST** - for getting object data using OAuth2
- **GET** - for getting object data with OAuth2 disabled

Note, if you are using the OAuth2 authentication, you must use POST requests since only POST requests provide the support needed for OAuth2 authentication. By default OAuth2 authentication is enabled. If you manually disable OAuth2, not recommended, then you can use a web browser to access the REST API and GET requests can be used. How to disable OAuth2 is discussed in the Manual Installation chapter.

10.2 Authentication Methods

There are two possible authentication methods with the Bacula REST-API. One is basic http authentication, and the other is OAuth2 authentication. By default, OAuth2 authentication is enabled.

The recommended form of authentication is done using OAuth2 protocol via a standard **"Username and Password"** login dialog. After the login the REST-API will use a *bconsole.conf* configuration file that is specifically defined by the REST-API administrator for each Username, thus permitting various privileges.

If you are running a secure site, it is possible to disable the OAuth2 authentication. However, we do not recommend doing so.

Please be aware that by using restricted Bacula consoles, you can allow certain users to access this REST API and only see data concerning their Client or their Jobs, depending on how you configure the restricted console. However, the code in the REST API that interfaces directly to the Bacula catalog database does not have Bacula access control lists, so we strongly recommend that you do not let untrusted users use the REST API interface directly, or that require untrusted users to use cPanel or some other Web GUI where your program handles the catalog security issues. For more information on the use of restricted consoles with the REST-API, see section restrictedcons.

In a future versions, we will implement plugins that will allow you to more easily control data access. For more information and several diagrams please see the Architecture section architecture.

10.3 Input Data Format

The data format for sending requests to the Bacula REST server is a URL encoded format compatible with the common URLs that are used by the HTTPS protocol.

```
https://<host>/<category>/<class>[/subclass/...]?option=value[&...]
```

where:

<host>
>   is the host name

<category>
>   is one of the following:

>   **cat**
>>   for catalog data. The class is the name of the catalog SQL table concerned. Upper/lower case is important.

>   **res**
>>   for resource data. The class is the name of the resource concerned. Upper/lower case is ignored.

>   **status**
>>   for status information. Status has no class.

>   **cmd**
>>   for bconsole commands.

>   **conf**
>>   for configure resources.

<class>
>   is the class or type of object desired.

<subclass>

is an additional specification of the class. At the current time, no subclass is implemented, but it is planned in a future release.

<option>

is a keyword that selects or limits the object class (see tables below). The option is specific to the class specified.

<value>

is the value associated with the keyword. Not all keywords have values.

To specify a particular Bacula component (Director, Client, Storage), use `option=value`, where the option is: **director, client, storage, console, bat**, and the value is the name of the component. If no component name is given, then the default `director` is assumed.

A concrete example of input that would obtain a list of all the Client objects from the Bacula catalog would be:

```
https://localhost/cat/Client
```

Input Options

There are several special options that can be used on input to limit the number of items that are returned and to set a start offset of the first item to return. They have corresponding values in the Output Data Format shown below.

The special input options are:

limit=nn

sets the limit to nn for the number of entries to return. For example: limit=10 will return a maximum of 10 items, but if there are not 10 items, it may return fewer. The default if not specified is 50.

offset=nn

sets the offset to the first item to be returned to nn. An offset=0 corresponds to the default and will returns items from the beginning. An offset=1 will return the second and subsequent items.

count

returns the total number of items that can be referenced, but it does not return any data. The returned value is returned in the limit field of the Output Data Structure (see below).

regex

applies the specified regex agains the name of the item (`/cat/` or `/res/`).

Note, the exact implementation of the above input options depends on the category. For example for the category status, none of the above are implemented, while for the category cmd, only the limit is implemented.

10.4 Output Data Format

Output data is always returned in JSON format.

data

contains returned objects and data from the Bacula server. If an error occurred, this field value will contain an error message.

error

contains integer value corresponding to the exit code from REST API server side. If there is no error then value is always set to 0 (zero).

limit

contains the limit that was specified in the call or the default (50) if none was specified.

offset

contains the offset used when returning data.

Example:

```
{
  "abc": "def",
  "ghi": 123,
  "extra_variables": ["one", "two", "three"],
  "last": 34.124
}
```

As noted above, `limit` and `offset` do not apply to `cmd` and `status` output.

Bacula REST API Error Codes

- **0** - No errors.
- **1** - Invalid command.
- **2** - Error connecting to the database.
- **3** - Error during write to the database. (not implemented)
- **4** - Error connecting to bconsole.
- **5** - Invalid director.
- **10** - Table Client does not exist.
- **11** - Table BaseFiles do not exist.
- **12** - Table Counters do not exist.
- **13** - Table File does not exist.
- **14** - Table Fileset does not exist.
- **15** - Table Job does not exist.
- **16** - Table JobHisto does not exist.
- **17** - Table JobMedia does not exist.
- **18** - Table Location does not exist.
- **19** - Table LocationLog does not exist.

- **20** - Table Log does not exist.

- **21** - Table Media does not exist.

- **22** - Table Path does not exist.

- **23** - Table Pool does not exist.

- **24** - Table RestoreObject does not exist.

- **25** - Table Storage does not exist.

- **26** - The version of the Bacula SQL tables format does not exist.

- **27** - Table Status does not exist.

Error Examples

Example with an error:

```
{
  "data": "Problem with connection to bconsole.",
  "error": 4
}
```

Example of a single item with no errors:

```
{
  "error": 0,
  "limit": 50,
  "data": [
    {
      "clientid": 3,
      "name": "darkstarus-fd",
      "uname": "5.2.3 (16Dec11) x86_64-unknown-linux-gnu,ubuntu,11.10",
      "autoprune": 1,
      "fileretention": 5184000,
      "jobretention": 15552000
    }
  ],
  "offset": 0
}
```

OAuth2 Error Codes

OAuth2 Error Codes (according to RFC 6749):

- invalid_request

- unauthorized_user (in RFC6749 it is error named unauthorized_client)

- access_denied

- unsupported_response_type

- invalid_scope

- server_error

- temporarily_unavailable

For each OAuth2 error is returned HTTP status code 400.

Above OAuth2 errors descriptions are available at:

```
http://tools.ietf.org/html/rfc6749#section-4.1.2.1.
```

Multiple Item Output Example

```json
{
  "data": [
    {
      "clientid": 3,
      "name": "Polymatou",
      "uname": "i686-pc-linux-gnu,redhat,(Stentz)",
      "autoprune": 1,
      "fileretention": 2592000,
      "jobretention": 31536000
    },
    {
      "clientid": 4,
      "name": "Minou",
      "uname": "2.4.2 (26Jul08) Linux,Cross-compile,Win32",
      "autoprune": 1,
      "fileretention": 2592000,
      "jobretention": 31536000
    },
    {
      "clientid": 5,
      "name": "Tibs",
      "uname": "3.0.3a (06Dec09) Linux,Cross-compile,Win32",
      "autoprune": 1,
      "fileretention": 5184000,
      "jobretention": 31536000
    }
  ],
  "error": 0,
  "limit": 50,
  "offset": 0
}
```

Limit and Offset Example

The following input:

```
https://localhost/cat/Client?limit=1&offset=1
```

will produce:

```json
{
  "error": 0,
```

```
  "limit": 1,
  "data": [
    {
     "clientid": 4,
     "name": "Minou",
     "uname": "2.4.2 (26Jul08) Linux,Cross-compile,Win32",
     "autoprune": 1,
     "fileretention": 2592000,
     "jobretention": 31536000
    }
  ],
  "offset": 1
}
```

which is the same as the second item of the full listing above.

10.5 Catalog Category

The **<category>** cat always refers to a Bacula defined table in the Catalog database, and can have the following values for **<class>**:

- Client
- Counters
- Job
- JobHisto
- Location
- LocationLog
- Log
- Media
- Pool
- Storage
- Version

In addition to the above mentioned tables, there are also certain tables that we strongly recommend not to access as doing so will lock you into a particular version of Bacula and might make it very difficult to update your application to new Bacula versions. The list of tables that we recommend against using are:

- BaseFiles
- File
- Fileset
- JobMedia
- Path
- RestoreObject

In addition to all the above, there are a few other catalog tables that are used internally by Bacula and thus are not listed here.

Below we show the options (in addition to the global limit and offset options) that can be applied to specific request to limit the output or select particular items. These options will vary depending on the class of the item being requested.

We also show a sample output of the data you can expect to have returned for each class.

General cat Options

The `catalog=xxx` option applies to all cat category requests, where xxx is the name of the database. If no `catalog=xxx` option is specified, then the API will use the first catalog name listed in the resources and specified in the **dbname** directive of the *bacula-dir.conf* configuration file. Note, this is by default **bacula**.

The `catalog=xxx` also applies to the cmd category (/cmd/. . .), and in this case the xxx is the named of the Catalog resource in the Director's configuration file.

Storage cat Options

Option	Value
name	

Storage cat Output

```
{
    "storageid": 1,
    "name": "example-storage-name1",
    "autochanger": 0
}
```

Pool cat Output

```
{
  "poolid": 19,
  "name": "Incremental-VTL",
  "numvols": 10,
  "maxvols": 0,
  "useonce": 0,
  "usecatalog": 1,
  "acceptanyvolume": 0,
  "volretention": 1296000,
  "voluseduration": 0,
  "maxvoljobs": 1,
  "maxvolfiles": 0,
  "maxvolbytes": 0,
  "autoprune": 1,
```

```
  "recycle": 1,
  "actiononpurge": 1,
  "pooltype": "Backup",
  "labeltype": 0,
  "labelformat": "*",
  "enabled": 1,
  "scratchpoolid": 12,
  "recyclepoolid": 12,
  "nextpoolid": 0,
  "migrationhighbytes": 0,
  "migrationlowbytes": 0,
  "migrationtime": 0
}
```

Job cat Options

Option	Value
level	
pool	
age	
type	
client	string
fileset	string

Job cat Output

```
{
  "jobid": 1130,
  "job": "robotki_masmiki.2013-05-13_20.10.00_22",
  "name": "robotki_masmiki",
  "type": "B",
  "level": "F",
  "clientid": 2,
  "jobstatus": "R",
  "schedtime": "2013-05-13 20:10:00",
  "starttime": "2013-05-15 04:01:15",
  "endtime": null,
  "realendtime": null,
  "jobtdate": 1368583275,
  "volsessionid": 0,
  "volsessiontime": 0,
  "jobfiles": 0,
  "jobbytes": 0,
  "readbytes": 0,
  "joberrors": 0,
  "jobmissingfiles": 0,
  "poolid": 18,
```

```
  "filesetid": 25,
  "priorjobid": 0,
  "purgedfiles": 0,
  "hasbase": 0,
  "hascache": 0,
  "reviewed": 0,
  "comment": ""
}
```

Media cat Output

```
{
  "mediaid": 120,
  "volumename": "VTL_0004_0015",
  "slot": 60,
  "poolid": 18,
  "mediatype": "Plik",
  "mediatypeid": 0,
  "labeltype": 0,
  "firstwritten": "2013-04-22 02:57:50",
  "lastwritten": "2013-04-22 02:57:51",
  "labeldate": "2013-04-22 02:57:50",
  "voljobs": 1,
  "volfiles": 0,
  "volblocks": 619,
  "volmounts": 6,
  "volbytes": 39899073,
  "volparts": 0,
  "volerrors": 0,
  "volwrites": 2115760,
  "maxvolbytes": 0,
  "volcapacitybytes": 0,
  "volstatus": "Used",
  "enabled": 1,
  "recycle": 1,
  "actiononpurge": 1,
  "volretention": 2592000,
  "voluseduration": 0,
  "maxvoljobs": 1,
  "maxvolfiles": 0,
  "inchanger": 1,
  "storageid": 1,
  "deviceid": 0,
  "mediaaddressing": 0,
  "volreadtime": 0,
  "volwritetime": 1539071050,
  "endfile": 0,
  "endblock": 39899072,
  "locationid": 0,
  "recyclecount": 5,
```

```
  "initialwrite": null,
  "scratchpoolid": 0,
  "recyclepoolid": 12,
  "comment": null,
  "pool": null,
  "whenexpire": null
}
```

Client cat Options

Client cat Input Example

```
https://localhost/cat/Client?name=Rufus
```

with the corresponding output:

```
{
  "error": 0,
  "limit": 50,
  "data": [
    {
     "clientid": 1,
     "name": "Rufus",
     "uname": "6.3.17 (12May13) x86_64-unknown-linux-gnu,ubuntu,12.04",
     "autoprune": 1,
     "fileretention": 6912000,
     "jobretention": 31536000
    }
  ],
  "offset": 0
}
```

```
/cat/Client?catalog=mycat
```

The above returns all the Client items in the catalog named mycat.

/cat/Client?limit=15 – returns data of the first 15 clients in the default catalog

/cat/Client?offset=10&limit=15 – returns the data of the 15 clients starting at the 10th client.

/cat/Client?clientid=2 – returns all the data for client with **clientid** equal 2.

/cat/Client?name=Rufus – returns all the data for client with client name Rufus

/cat/Client/ - returns the data of all clients in the first catalog defined in the resources

/cat/Client?option=value[&. . .] – see the table above

Client cat Output Example

```
{
    "clientid": 1,
    "name": "example-client-name1",
    "uname": "client environment specific information",
    "autoprune": 0,
    "fileretention": 5184000,
    "jobretention": 15552000
}
```

/cat/Media/ - returns the name of all media in the first catalog defined
/cat/Media?parameter=value[&...] - see the table below

Media cat keywords:

10.6 Commands Category

These commands those that are available in the **bconsole** program and are used to control the Bacula Director.

The `option=value` are not checked by the REST interface, but are sent directly to Bacula where only error checking is done. If multiple keyword=value are specified, they must be separated by the ampersand (&) character.

Note that the `offset` and `limit` options shown above for the `cat` category do not apply to the `cmd` category.

The **<category>** `cmd` can have the following values for **<class>**:

```
- pool=<pool-name> storage=<storage> jobid=<JobId>

- on off

jobid=<number-list> job=<job-name> ujobid=<unique-jobid> all

pool=<pool-name> storage=<storage> jobid=<JobId>

volume=<vol-name> pool=<pool-name> jobid=<id>

job=<name> batch

job=<name> batch

fileset=<fs> client=<cli> level=<level> accurate=<yes/no> job=<job>
listing

add autodisplay automount cancel create delete disable enable
estimate exit gui label list llist messages memory mount prune purge
python quit query restore relabel release reload run status
```

(continues on next page)

```
setbandwidth setdebug setip show sqlquery time trace mount umount
update use var version wait

storage=<storage> volume=<vol> pool=<pool> slot=<slot> barcodes

pools jobs jobtotals volume media <pool=pool-name> files jobid=<nn>
copies jobid=<nn

>

pools jobs jobtotals volume media <pool=pool-name> files jobid=<nn>
copies jobid=<nn>

pools jobs jobtotals volume media <pool=pool-name> files jobid=<nn>
copies jobid=<nn>

files jobs pool=<pool> client=<client-name> [ expired ]

volume=<volume-name>

files jobs volume=<vol> [action=<action> devicetype=<type>
pool=<pool> allpools storage=<st> drive=<num>

where=</path> client=<client> storage=<storage> bootstrap=<file>
restorejob=<job> comment=<text> jobid=<jobid> copies done select all

storage=<storage-name> oldvolume=<old-volume-name>
volume=<newvolume-name> pool=<pool>

storage=<storage-name>

job=<job-name> client=<client-name>\\n\\tfileset=<Fileset-name>
level=<level-keyword> storage=<storage-name> where=<directory-prefix>
when=<universal-time-specification>
pool=<pool-name>\\n\\tcomment=<text> accurate=<bool> spooldata=<bool>
yes

incomplete job=<job-name> client=<client-name> fileset=<Fileset-name>
level=<level-keyword> storage=<storage -name>
when=<universal-time-specification> comment=<text> spooldata=<bool>
jobid=<jobid>

all dir=<dir-name> director client=<client-name>
storage=<storage-name> slots days=nnn

jobid=<number-list> job=<job-name> ujobid=<unique-jobid> all

level=<nn> trace=0/1 client=<client-name> dir storage=<storage-name>
all

limit=<nn-kbs> client=<client-name> jobid=<number> job=<job-name>
ujobid=<unique-jobid>
```

```
limit=<nn-kbs> client=<client-name> jobid=<number> job=<job-name>
ujobid=<unique-jobid>

on off

storage=<storage-name> [ drive=<num> ] jobid=<id> job=<job-name>

storage=<storage-name> [ drive=<num> ] jobid=<id> job=<job-name>

stats pool=<poolname>\\n\\tslots storage=<storage> scan
volume=<volname> volstatus=<status> volretention=<time-def>
pool=<pool> recycle=<yes/no> slot=<number> inchanger=<yes/no>
maxvolbytes=<size> maxvolfiles=<nb> maxvoljobs=<nb> enabled=<yes/no>
recyclepool=<pool> actiononpurge=<action>
```

10.7 Status Category

/status?director=<director-name> - returns status of the specified director

/status?storage=<storage-name> - returns status of the specified storage

/status?client=<client-name> - returns status of the specified client

Director Status

Units

Possible values of binary prefixes:

- Ki (kibi) - 1 KiB = 1024 B
- Mi (mebi) - 1 MiB = 1024^2 B
- Gi (gibi) - 1 GiB = 1024^3 B.

Possible values of decimal prefixes:

- k (kilo) - 1 kB = 1000 B,
- M (mega) - 1 MB = 1000^2 B,
- G (giga) - 1 GB = 1000^3 B.

Job Status

Jobs status as short code:

- Canceled by user
- Blocked
- Created but not yet running
- Verify differences
- Job terminated in error
- Waiting on File daemon
- Incomplete Job
- Committing data
- Waiting for Mount
- Running
- Waiting on the Storage daemon
- Terminated normally
- Terminated normally with warnings
- SD despooling attributes
- Waiting for Client resource
- Waiting for maximum jobs
- Non-fatal error
- Fatal error
- Doing batch insert file records
- Waiting for job resource
- Doing data despooling
- waiting for new media
- Waiting for higher priority jobs to finish
- Queued waiting for device
- Waiting for storage resource
- Waiting for start time

Full Status Director Output

```
POST /status/director/?name=speedstar-dir

OUTPUT:
{
   "data": [
      {
      "name": "speedstar-dir",
      "version": "6.6.0 (01Nov13)",
      "uname": "x86_64-pc-linux-gnu-bacula-enterprise ubuntu 12.04",
      "started": "2013-11-10 13:16",
      "reloaded": "2013-11-10 14:41",
      "jobs_run": "2",
      "jobs_running": "1",
      "nclients": "2",
      "nstores": "2",
      "npools": "3",
      "ncats": "2",
      "nfsets": "3",
      "nscheds": "2",
      "jobs": {
         "scheduled": [
         {
            "name": "SpeedstarJob",
            "level": "differential",
            "type": "backup",
            "priority": 10,
            "schedtime": "2013-11-10 20:05:00",
            "volume": "file-volume-workstations-V-0032",
            "pool": "MyPool"
         },
         {
            "name": "gani-Full-Backup",
            "level": "differential",
            "type": "backup",
            "priority": 10,
            "schedtime": "2013-11-10 20:05:00",
            "volume": "file-volume-workstations-V-0032",
            "pool": "MyPool"
         }
         ],
      "running": [
      {
         "jobid": 79,
         "level": "incremental",
         "type": "B",
         "jobfiles": 0,
         "jobbytes": 0,
         "jobbytes_formatted": "0 B",
         "binaryprefix": "",
         "name": "gani-Full-Backup",
```

```json
        "fileset": "gani-Fileset",
        "starttime": "2013-11-10 13:50:20",
        "clientname": "gani-fd",
        "status": "F",
        "status_desc": "is waiting for Client gani-fd to connect",
        "errors": 0
    },
    {

        "jobid": 80,
        "level": "incremental",
        "type": "B",
        "jobfiles": 32181,
        "jobbytes": 8781060636,
        "jobbytes_formatted": "8.178 GiB",
        "binaryprefix": "Gi",
        "name": "gani-Full-Backup",
        "fileset": "gani-Fileset",
        "starttime": "2013-11-10 13:50:18",
        "clientname": "gani-fd",
        "status": "R",
        "status_desc": "is running",
        "errors": 0
    }
],
"terminated": [
    {
        "name": "SpeedstarJob",
        "jobid": 69,
        "level": "incremental",
        "jobfiles": 0,
        "jobbytes": 0,
        "jobbytes_formatted": "0 B",
        "binaryprefix": "",
        "status": "T",
        "status_desc": "OK",
        "starttime": "2013-11-10 16:13:02",
        "endtime": "2013-11-10 16:13:10",
        "errors": 0
    },
    {
        "name": "SpeedstarJob",
        "jobid": 70,
        "level": "incremental",
        "jobfiles": 0,
        "jobbytes": 0,
        "jobbytes_formatted": "0 B",
        "binaryprefix": "",
        "status": "T",
        "status_desc": "OK",
        "starttime": "2013-11-10 16:18:02",
        "endtime": "2013-11-10 16:18:01",
        "errors": 0
```

```
        }
    ]
    }
  }
 ],
 "error": 0
}
```

Scheduling Job List for Director

```
POST /status/director/?name=speedstar-dir&jobs=scheduled

OUTPUT:
{
    "data": [
        {
            "name": "SpeedstarJob",
            "level": "differential",
            "type": "backup",
            "priority": 10,
            "schedtime": "2013-11-10 20:05:00",
            "volume": "file-volume-workstations-V-0032",
            "pool": "MyPool"
        },
        {
            "name": "gani-Full-Backup",
            "level": "differential",
            "type": "backup",
            "priority": 10,
            "schedtime": "2013-11-10 20:05:00",
            "volume": "file-volume-workstations-V-0032",
            "pool": "MyPool"
        }
    ],
    "error": 0,
    "offset": 0,
    "limit": 10
}
```

Running Jobs for Director

```
POST /status/director/?name=speedstar-dir&jobs=running

OUTPUT:
{
    "data": [
        {
            "jobid": 79,
```

```
            "level": "incremental",
            "jobfiles": 0,
            "jobbytes": 0,
            "jobbytes_formatted": "0 B",
            "binaryprefix": "",
            "name": "gani-Full-Backup",
            "fileset": "gani-Fileset",
            "starttime": "2013-11-10 13:50:20",
            "clientname": "gani-fd",
            "status": "F",
            "status_desc": "is waiting for Client gani-fd to connect to Storage␣
↪File",
            "errors": 0
        },
        {
            "jobid": 80,
            "level": "incremental",
            "jobfiles": 32181,
            "jobbytes": 8781060636,
            "jobbytes_formatted": "8.178 GiB",
            "binaryprefix": "Gi",
            "name": "gani-Full-Backup",
            "fileset": "gani-Fileset",
            "starttime": "2013-11-10 13:50:18",
            "clientname": "gani-fd",
            "status": "R",
            "status_desc": "is running",
            "errors": 0
        }
    ],
    "error": 0,
    "offset": 0,
    "limit": 10
}
```

Getting Last 10 Terminated Jobs for Director

```
POST /status/director/?name=speedstar-dir&jobs=terminated

OUTPUT:
{
    "data": [
        {
            "name": "SpeedstarJob",
            "jobid": 69,
            "level": "incremental",
            "jobfiles": 0,
            "jobbytes": 0,
            "jobbytes_formatted": "0 B",
            "binaryprefix": "",
```

```
        "status": "T",
        "status_desc": "OK",
        "starttime": "2013-11-10 16:13:02",
        "endtime": "2013-11-10 16:13:10"
      },
      {
        "name": "gani-Full-Backup",
        "jobid": 78,
        "level": "incremental",
        "jobfiles": 1,
        "jobbytes": 26429,
        "jobbytes_formatted": "25.81 KiB",
        "binaryprefix": "Ki",
        "status": "T",
        "status_desc": "OK",
        "starttime": "2013-11-10 16:31:02",
        "endtime": "2013-11-10 18:31:08"
      }
    ],
    "error": 0,
    "offset": 0,
    "limit": 10
}
```

Getting Last 10 Terminated Jobs for Director with Unified Unit binaryprefix

```
POST /status/director/?name=speedstar-dir&jobs=terminated&binaryprefix=Ki
OUTPUT:
{
    "data": [
      {
        "name": "SpeedstarJob",
        "jobid": 69,
        "level": "incremental",
        "jobfiles": 0,
        "jobbytes": 0,
        "jobbytes_formatted": "0 KiB",
        "binaryprefix": "Ki",
        "status": "T",
        "status_desc": "OK",
        "endtime": "2013-11-10 16:13:10"
      },
      {
        "name": "gani-Full-Backup",
        "jobid": 78,
        "level": "incremental",
        "jobfiles": 1,
        "jobbytes": 26429,
        "jobbytes_formatted": "25.80 KiB",
        "binaryprefix": "Ki",
```

```
            "status": "T",
            "status_desc": "OK",
            "endtime": "2013-11-10 18:31:08"
        }
    ],
    "error": 0,
    "offset": 0,
    "limit": 10
}
```

Running Incremental Jobs for Director

```
POST /status/director/?name=speedstar-dir&jobs=running&level=incremental

OUTPUT:
{
    "data": [
        {
            "jobid": 79,
            "level": "incremental",
            "jobfiles": 0,
            "jobbytes": 0,
            "jobbytes_formatted": "0 B",
            "binaryprefix": "",
            "name": "gani-Full-Backup",
            "fileset": "gani-Fileset",
            "starttime": "2013-11-10 13:50:20",
            "clientname": "gani-fd",
            "status": "F",
            "status_desc": "is waiting for Client gani-fd to connect to Storage␣
→File"
        },
        {
            "jobid": 80,
            "level": "incremental",
            "jobfiles": 32181,
            "jobbytes": 8781060636,
            "jobbytes_formatted": "8.178 GiB",
            "binaryprefix": "Gi",
            "name": "gani-Full-Backup",
            "fileset": "gani-Fileset",
            "starttime": "2013-11-10 13:50:18",
            "clientname": "gani-fd",
            "status": "R",
            "status_desc": "is running",
            "errors": 0
        }
    ],
    "error": 0,
    "offset": 0,
```

```
    "limit": 10
}
```

Status Storage Output

Possible values of binary and decimal prefixes are described at unit.

```
POST /status/storage/?name=VTL
OUTPUT:
{
    "data": [
        {
            "name": "speedstar-sd",
            "version": "6.6.0 (01Nov13)",
            "uname": "x86_64-pc-linux-gnu-bacula-enterprise ubuntu 12.04",
            "started": "2013-11-28 00:40:00",
            "jobs_run": "3",
            "jobs_running": "1",
            "ndevices": "8",
            "nautochgr": "1",
            "jobs": {
                "running": [
                    {
                        "jobid": 274,
                        "type": "backup",
                        "name": "gani-Full-Backup",
                        "level": "full",
                        "volume": "VTL_0001_0015",
                        "pool": "Workstations-Differential",
                        "device": "Drive-1",
                        "spooling": "0",
                        "despooling": "0",
                        "despool_wait": "0",
                        "files": 2403,
                        "bytes": 736885328,
                        "bytes_formatted": "702.74 MiB",
                        "avebytes_sec": 245628442,
                        "avebytes_sec_formatted": "234.24 MiB/s",
                        "lastbytes": 245628442,
                        "lastbytes_formatted": "234.24 MiB/s",
                        "binaryprefix": "Mi",
                        "errors": 0
                    },
                    {
                        "jobid": 297,
                        "type": "restore",
                        "name": "RestoreFiles",
                        "level": "",
                        "volume": "VTL_0001_0027",
                        "pool": "Workstations-full",
```

```
                    "device": "Drive-3"
                }
            ],
            "terminated": [
                {
                    "jobid": 263,
                    "name": "gani-Full-Backup",
                    "level": "incremental",
                    "files": 3,
                    "jobbytes": 1167065,
                    "jobbytes_formatted": "1.133 MiB",
                    "binaryprefix": "Mi",
                    "status": "T",
                    "status_desc": "OK",
                    "starttime": "25-Nov-13 20:29",
                    "endtime": "25-Nov-13 20:30",
                    "errors": 0
                },
                {
                    "jobid": 266,
                    "name": "gani-Full-Backup",
                    "level": "full",
                    "files": 0,
                    "jobbytes": 0,
                    "jobbytes_formatted": "0 B",
                    "binaryprefix": "",
                    "status": "f",
                    "status_desc": "Error",
                    "starttime": "25-Nov-13 17:14",
                    "endtime": "26-Nov-13 17:15",
                    "errors": 0
                }
            ]
        },
        "devices": [
            {
                "autochangers": [
                    {
                        "name": "VirtualTapeLibrary",
                        "devices": [
                            {
                                "name": "Drive-1",
                                "device": "/media/backup/VTL/0/drive0"
                            },
                            {
                                "name": "Drive-2",
                                "device": "/media/backup/VTL/1/drive1"
                            },
                            {
                                "name": "Drive-3",
                                "device": "/media/backup/VTL/2/drive2"
                            }
```

```
                    ]
                }
            ],
            "devices": [
                {
                    "name": "FileStorage",
                    "archive_device": "/mnt/bacula-storage",
                    "mounted": 0,
                    "volume": "",
                    "pool": "",
                    "media_type": "Plik",
                    "slot":    0,
                    "bytes":  0,
                    "blocks": 0
                },
                {
                    "name": "Drive-1",
                    "archive_device": "/media/backup/VTL/0/drive0",
                    "mounted": 1,
                    "volume": "VTL_0001_0015",
                    "pool": "Workstations-Differential",
                    "media_type": "Plik",
                    "slot": 15,
                    "bytes": 737436867,
                    "blocks": 11431
                },
                {
                    "name": "Drive-2",
                    "archive_device": "/media/backup/VTL/1/drive1",
                    "mounted": 1,
                    "volume": "VTL_0001_0028",
                    "pool": "Workstations-Differential",
                    "media_type": "Plik",
                    "slot": 28,
                    "bytes": 25095372,
                    "blocks": 389
                },
                {
                    "name": "Drive-3",
                    "archive_device": "/media/backup/VTL/2/drive2",
                    "mounted": 1,
                    "volume": "VTL_0001_0027",
                    "pool": "Workstations-Differential",
                    "media_type": "Plik",
                    "slot": 27,
                    "bytes": 310818816,
                    "blocks": 4818
                }
            ]
        }
    ]
}
```

```
    ]
}
```

Status Client Output

Possible values of binary and decimal prefixes are described in unit

```
POST /status/client/?name=gani-fd
OUTPUT:
{
    "data": [
        {
            "client-daemon": "gani-fd",
            "version": "6.6.0",
            "uname": "x86_64-pc-linux-gnu-bacula-enterprise ubuntu 12.04",
            "started": "2013-12-01 01:02:00",
            "jobs_run": "30",
            "jobs_running": "1",
            "jobs": {
                "running": [
                    {
                        "jobid": 330,
                        "name": "RestoreFiles",
                        "type": "restore",
                        "starttime": "2013-11-29 16:41:15",
                        "status": "R",
                        "status_desc": "is running",
                        "jobbytes": 435505249,
                        "jobbytes_formatted": "415.33 MiB",
                        "avebytes_sec": 48389472,
                        "avebytes_sec_formatted": "46.14 MiB/s",
                        "lastbytes_sec": 48389472,
                        "lastbytes_sec_formatted": "46.14 MiB/s",
                        "binaryprefix": "Mi",
                        "bwlimit": 0,
                        "jobfiles": 0,
                        "restored": 2093,
                        "expected_files": 29521,
                        "percent_completed": "7%",
                        "files_examined": 2093,
                        "processing_file": "/tmp/bacula-restore/home/gani/
→DSC00183.jpg",
                        "errors": 0
                    },
                    {
                        "jobid": 331,
                        "name": "gani-Full-Backup",
                        "type": "backup",
                        "starttime": "2013-11-29 16:41:19",
                        "status": "R",
```

```
                        "status_description": "is running",
                        "jobbytes": 435505249,
                        "jobbytes_formatted": "415.33 MiB",
                        "avebytes_sec": 48389472,
                        "avebytes_sec_formatted": "46.14 MiB/s",
                        "lastbytes_sec": 48389472,
                        "lastbytes_sec_formatted": "46.14 MiB/s",
                        "binaryprefix": "Mi",
                        "bwlimit": 0,
                        "jobfiles": 2210,
                        "restored": 0,
                        "expected_files": 0,
                        "percent_completed": "",
                        "files_examined": 2210,
                        "processing_file": "/home/gani/Bacula/tapeta_bacula.
→png",
                        "errors": 0
                        }
                }
            ],
            "terminated": [
                {
                    "jobid": 319,
                    "name": "gani-Full-Backup",
                    "type": "backup",
                    "level": "incremental",
                    "files": 0,
                    "jobbytes": 0,
                    "jobbytes_formatted": "0 B",
                    "binaryprefix": "",
                    "starttime": "2013-11-29 16:25:00",
                    "endtime": "2013-11-29 16:26:00",
                    "status": "T",
                    "status_description": "OK",
                    "errors": 0
                }
            ]
        }
    ],
    "error": 0
}
```

Running Jobs on Client

```
POST /status/client/?name=gani-fd&jobs=running

OUTPUT:
{
    "data": [
        {
            "jobid": 330,
            "name": "RestoreFiles",
            "type": "restore",
            "starttime": "2013-11-29 16:41:15",
            "status": "R",
            "status_description": "is running",
            "jobbytes": 435505249,
            "jobbytes_formatted": "415.33 MiB",
            "avebytes_sec": 48389472,
            "avebytes_sec_formatted": "46.14 MiB/s",
            "lastbytes_sec": 48389472,
            "lastbytes_sec_formatted": "46.14 MiB/s",
            "binaryprefix": "Mi",
            "bwlimit": 0,
            "jobfiles": 0,
            "restored": 2093,
            "expected_files": 29521,
            "percent_completed": "7%",
            "files_examined": 2093,
            "processing_file": "/tmp/bacula-restore/home/gani/DSC00183.jpg",
            "errors": 0
        },
        {
            "jobid": 331,
            "name": "gani-Full-Backup",
            "type": "backup",
            "starttime": "2013-11-29 16:41:19",
            "status": "R",
            "status_description": "is running",
            "jobbytes": 564739793,
            "jobbytes_formatted": "538.57 MiB",
            "avebytes_sec": 141184948,
            "avebytes_sec_formatted": "134.64 MiB",
            "lastbytes_sec": 141184948,
            "lastbytes_sec_formatted": "134.64 MiB/s",
            "bwlimit": 0,
            "backed_up": 2210,
            "restored": 0,
            "expected_files": 0,
            "percent_completed": "",
            "files_examined": 2210,
            "processing_file": "/home/gani/Obrazy/Bacula/tapeta_bacula.png",
            "errors": 0
        }
```

```
    ],
    "error": 0
}
```

10.8 Resource Category

There are separate resources for each of the Bacula components, Director, File daemon, and the Storage daemon.

Although these resources can be obtained dynamically from the Director or any other component with the `show` command, the examples given below are obtained by executing the appropriate Bacula `json` program that reads the Bacula configuration file and returns the JSON output shown below. The programs are named `bdirjson`, `bfdjson`, and `bsdjson`. When no options are given to each of those programs they simply list the full Bacula conf file in JSON format. They also have a series of options that permit obtaining individual resources.

Please note that if you have updated the *bacula-dir.conf* file, but have not restarted the Director or done a reload command, the information that is returned to you will not correspond to that which the Director is using. Likewise if you have update *bacula-fd.conf* or *bacula-sd.conf* and have not restarted the appropriate File daemon or Storage daemon, the information you get from the REST-API will not correspond to what the particular daemon is using.

Director Resource Category

The <category> `res` for the Director may have any of the following values:

- director
- client
- job
- storage
- catalog
- schedule
- fileset
- pool
- messages
- counter
- console
- jobdefs
- device
- autochanger

File Daemon Resource Category

The <category> **res** for the File daemon may have any of the following values:

- director

- filedaemon

- messages

- client

Storage Daemon Resource Category

The <category> `res` for the Storage daemon may have any of the following values:

- director

- storage

- device

- messages

- autochanger

bconsole and bat Resource Category

The <category> **res** for the bconsole and bat programs may have any of the following values:

- console

- director

Director Director Resource Category

The following output is similar to what would be output from:

```
/opt/bacula/bin/bdirjson -r Director -D
```

```
[
 {
    "Name": "rufus-dir",
    "Messages": "Standard",
    "DirPort": 8101,
    "QueryFile": "/opt/bacula/bin/query.sql",
    "WorkingDirectory": "/opt/bacula/working",
    "PluginDirectory": "/opt/bacula/bin",
    "PidDirectory": "/opt/bacula/working",
    "SubsysDirectory": "/opt/bacula/working",
    "MaximumConcurrentJobs": 4,
    "Password": "my-password",
    "HeartbeatInterval": 330
 }
]
```

Resource Client Input Examples

/res/director/?resource=Client - returns the name of all clients in resources

/res/director/?option=value - see the table below

Client Resource Options

Director Client Resource Output

```
[
 {
    "Name": "rufus-fd",
    "Address": "rufus",
    "FdPort": 8102,
    "Password": "xevrjURYoCHhn26RaJoWbeWXEY/a3VqGKp/37tgWiuHc",
    "Catalog": "MyCatalog",
    "FileRetention": 2592000,
    "JobRetention": 15552000,
    "AutoPrune": true,
    "MaximumConcurrentJobs": 10
 }
]
```

Director Job Resource Output

```
{
 {
    "Name": "BackupCatalog",
    "Type": "Backup",
    "Messages": "Standard",
    "Storage": ["File"],
    "Pool": "Default",
    "Client": "rufus-fd",
    "Fileset": "Catalog",
    "WriteBootstrap": "/opt/bacula/bsr/BackupCatalog.bsr",
    "MaxRunTime": 1800,
    "Runscript": [
      {
        "RunsWhen": "Before",
        "RunsOnClient": false,
        "Command": "/opt/bacula/scripts/make_catalog_backup -u regress"
      },
      {
        "RunsWhen": "After",
        "RunsOnClient": false,
        "Command": "/opt/bacula/scripts/delete_catalog_backup"
      }
    ]
 }
]
```

Director Storage Resource Output

```
[
 {
    "Name": "File",
    "SdPort": 8103,
    "Address": "rufus",
    "Password": "xxxx",
    "Device": ["FileStorage"],
    "MediaType": "File",
    "HeartbeatInterval": 330,
    "MaximumConcurrentJobs": 10
 }
]
```

Director Catalog Resource Output

```
{
  "Catalog": {
    "Name": "MyCatalog",
    "Password": "",
    "User": "regress",
    "DbName": "regress"
  }
}
```

Director Schedule Resource Output

```
[
 {
    "Name": "WeeklyCycle",
    "Run": [
     {
      "Level": "Full",
      "Minute": 5,
      "Hour": [1],
      "Day": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,␣
↪18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
      "Month": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
      "DayOfWeek": [0],
      "WeekOfMonth": [0],
      "WeekOfYear": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
↪ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 2
     },
     {
      "Level": "Differential",
      "Minute": 5,
      "Hour": [1],
      "Day": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,␣
↪18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
```

```
      "Month": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
      "DayOfWeek": [0],
      "WeekOfMonth": [1, 2, 3, 4],
      "WeekOfYear": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
↪ ...  48, 49, 50, 51, 52, 53]
    },
    {
     "Level": "Incremental",
     "Minute": 5,
     "Hour": [1],
     "Day": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
↪18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
     "Month": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
     "DayOfWeek": [1, 2, 3, 4, 5, 6],
     "WeekOfMonth": [0, 1, 2, 3, 4],
     "WeekOfYear": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
↪ ... 48, 49, 50, 51, 52, 53]
    }
    ]
 }
]
```

Director Fileset Resource Output

```
[
 {
    "Name": "DeltaSet",
    "Include": [{
      "File": ["</opt/bacula/scripts/file-list"],
      "Options": [
       {
         "Signature": "Md5",
         "Deduplication": "Yes",
         "Plugin": "delta"
       }
      ]
    }]
 }
]
```

Director Pool Resource Output

```
[
 {
    "Name": "Default",
    "PoolType": "Backup",
    "VolumeRetention": 31536000,
    "AutoPrune": true,
    "Recycle": true
```

```
 }
]
```

Director Messages Resource Output

```
[
 {
    "Name": "Standard",
    "MailCommand": "/opt/bacula/bin/bsmtp -h localhost -f \"(Bacula␣
→regression) %r\" -s \"Regression: %t %e of %c %l\" %r",
    "OperatorCommand": "/opt/bacula/bin/bsmtp -h localhost -f \"(Bacula␣
→regression) %r\" -s \"Regression: Intervention needed for %j\" %r",
    "Destinations": [
      {
        "Type": "Append",
        "MsgTypes": ["All","!Skipped"],
        "Where": ["/opt/bacula/working/log"]
      },
      {
        "Type": "Console",
        "MsgTypes": ["All","!Skipped","!Terminate","!Restored"]
      },
      {
        "Type": "Catalog",
        "MsgTypes": ["All","!Skipped"]
      }
    ]
 }
]
```

Director JobDefs Resource Output

```
[
 {
    "Name": "BackupJob",
    "Type": "Backup",
    "Messages": "Standard",
    "Storage": ["File"],
    "Pool": "Default",
    "MaxRunTime": 1800,
    "Priority": 10
 }
]
```

Client Director Resource Output

```
[
 {
    "Name": "rufus-dir",
    "Messages": "Daemon",
    "DirPort": 8101,
    "QueryFile": "/opt/bacula/scripts/query.sql",
    "WorkingDirectory": "/opt/bacula/working",
    "PidDirectory": "/opt/bacula/working",
    "MaximumConcurrentJobs": 40,
    "Password": "NonSecurePassword"
 }
]
```

Client Client Resource Output

```
[
 {
    "Name": "rufus-fd",
    "FdPort": 8102,
    "WorkingDirectory": "/opt/bacula/working",
    "PidDirectory": "/opt/bacula/working",
    "PluginDirectory": "/opt/bacula/bin/plugins",
    "MaximumConcurrentJobs": 100
 }
]
```

Client Messages Resource Output

```
[
 {
    "Name": "Standard",
    "Destinations": [
      {
        "Type": "Director",
        "MsgTypes": ["All","!Terminate","!Restored"],
        "Where": ["rufus-dir"]
      }
    ]
 }
]
```

Storage Director Resource Output

```
[
 {
    "Name": "rufus-dir",
    "Password": "secret-password"
 }
]
```

Storage Storage Resource Output

```
[
 {
    "Name": "rufus-sd",
    "SdPort": 8103,
    "WorkingDirectory": "/opt/bacula/working",
    "PidDirectory": "/opt/bacula/working",
    "SubsysDirectory": "/opt/bacula/working",
    "PluginDirectory": "/opt/bacula/bin/plugins",
    "MaximumConcurrentJobs": 100,
    "DseEnable": true,
    "DseArchdir": "/opt/bacula/working"
 }
]
```

Storage Device Resource Output

```
[
 {
    "Name": "FileStorage",
    "MediaType": "File",
    "ArchiveDevice": "/opt/bacula/volumes",
    "RemovableMedia": false,
    "RandomAccess": true,
    "AutomaticMount": true,
    "LabelMedia": true,
    "AlwaysOpen": false
 }
]
```

Storage Messages Resource Output

```
[
 {
    "Name": "Standard",
    "Destinations": [
      {
        "Type": "Director",
        "MsgTypes": ["All","!Terminate"],
        "Where": ["rufus-dir"]
      }
    ]
 }
]
```

10.9 Configure Category

This `conf` category enables to configure the Bacula configuration resources via the Bacula REST API interface. Before using it, it is required to install and configure the BWeb Management Suite.

Note: The `conf` category can be used only when the OAuth2 authorization is enabled.

You must configure the `BConfig URI`, the web auth option the `BConfig User` and the `BConfig Password`. These fields are highlighted by the red arrows on the screen shot below.
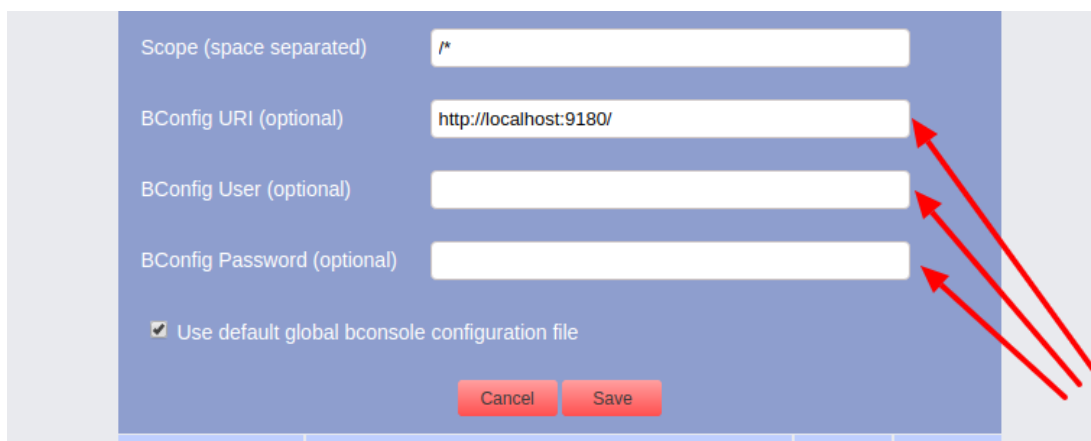
Fig. 4: REST API BWeb configuration

In general, the `conf` category request format consists in four items:

- component type (director, storage, filedaemon, console),
- component name (ex. mydirector-dir, storage-sd ... etc.),
- resource type (ex. job, client, pool, storage ... etc.),
- resource name (ex. MyJob123, FavClient ... etc.).

The `conf` category request has the following form:

```
/conf/<component_type>/<component_name>/<resource_type>/<resource_name>/
```

There are two `conf` request methods available:

- PUT method to update resource configuration. If resource doesn't exist it is created.

- DELETE method to delete resource.

Example request to update configuration job "MyJob":

```
PUT /conf/director/mysystem-dir/job/MyJob/
{
    "Name": "MyJob",
    "Type": "Backup",
    "Client": "MyClient",
    "Fileset": "MyFileset",
    "Pool": "MyPool",
    "Schedule": "MySchedule",
    "Storage": "MyStorage",
    "Messages": "MyMessages"
}
```

Example request to delete job "MyJob" configuration:

```
DELETE /conf/director/mysystem-dir/job/MyJob/
```

After changing configuration it can be useful to commit these changes in the BWeb workset by the following request:

```
DELETE /conf/director/mysystem-dir/job/MyJob/

with POST param:
commit=1
```

It is also possible to reload the Bacula Director by:

```
GET /restart?reload_dir=1
```

or restart a component by:

```
GET /restart?restart_dir=1
GET /restart?restart_sd=1
GET /restart?restart_fd=1
```

10.10 Installation

Normally, the Bacula REST API is installed from a package such as a .deb or .rpm. The package name is of the following form:

```
bacula-enterprise-rest-api-6.6.0.xxx.(rpm/deb)
```

where the 6.6.0 must correspond to the Bacula Enterprise version you installed, and the xxx depends on the package and the architecture of the machine on which you are installing the package.

The easiest way to do the installation is to use the Bacula Enterprise repository and select the package using `yum` on systems with rpm packages and `apt-get` on systems with deb packages.

For example, on systems using rpms:

```
yum install bacula-enterprise-rest-api-6.6.0-1.noarch.rpm
```

or if you are using the Bacula Enterprise Yum repository:

```
yum install bacula-enterprise-rest-api
```

Likewise on systems using debs:

```
dpkg -i bacula-enterprise-rest-api-6.6.0.deb
apt-get -f install
```

where the second command (apt-get) is only needed if you get unresolved package errors on the first (dpkg) command. And if you are using the Bacula Enterprise Debian repository:

```
apt-get install bacula-enterprise-rest-api
```

The installation process will create the directory /opt/bacula/rest-api. Under the rest-api directory, you will find the following additional directories:

- **bin**

  where any binaries or start scripts are placed.

- **etc**

  where configuration files are placed (e.g. lighttpd.conf).

- **log**

  where you can find lighttpd log files.

- **scripts**

  where you can find various useful scripts.

- **uploads**

  where your app can upload files if necessary (not recommended).

- **www**

  where the rest-api web files reside.

10.11 Configuration

After you have successfully installed the Bacula REST API, you will need to do a bit of customization of the configuration files that are appropriate for your system. This is a three step process:

1. Configure the lighttpd web server.

2. Configure the REST API authentication

3. Configure the Users allowed to access the REST API interface

lighttpd Configuration

The first part to configure is the `lighttpd` web server that the Bacula REST API uses. Note, it is possible to configure the REST API server on an Apache server, but due to the complexity of doing so on different systems, we have standardized on lighttpd and do not support Apache. For a sys admin that uses Apache, the Lighttpd examples should be very instructive for configuring Apache.

The default configuration file is located in:

```
/opt/bacula/rest-api/etc/lighttpd.conf
```

Points to consider are:

1. The default server port is 443. If that port is already used, you must change it to an unused port.

2. The default installation uses the secure https protocol. If you do not wish to use encrypted communications, you will need to disable the ssl engine.

3. The installation process creates a default ssl certificate for you, providing you have **openssl** installed on your system. If you do not have openssl installed, the installation process uses a pre-prepared stock certificate. If you would like to use a certificatee of your own, you will need to copy it to the appropriate place in the directory:

```
/opt/bacula/rest-api/etc
```

   or change the server ssl definitions to point to the appropriate directory and file.

Once, you have checked and possibly modified the server conf file, you can start it with one of the two following commands.

```
/etc/init.d/bacula-enterprise-rest-api start
```

or

```
/opt/bacula/rest-api/bin/lighttpd.redhat start
```

REST API Configuration

If the above `lighttpd` configuration is successful or has been successfully done during the package installation, you should enter the following URL into your web browser:

```
https://localhost/
```

where you should see initial configuration wizard for REST API. If you get an error, you will need to turn on debugging in the lighttpd server, and restart it, then examine the lighttpd log files to find the problem. The Trouble Shooting chapter explains how to do so in detail.

Language Selection
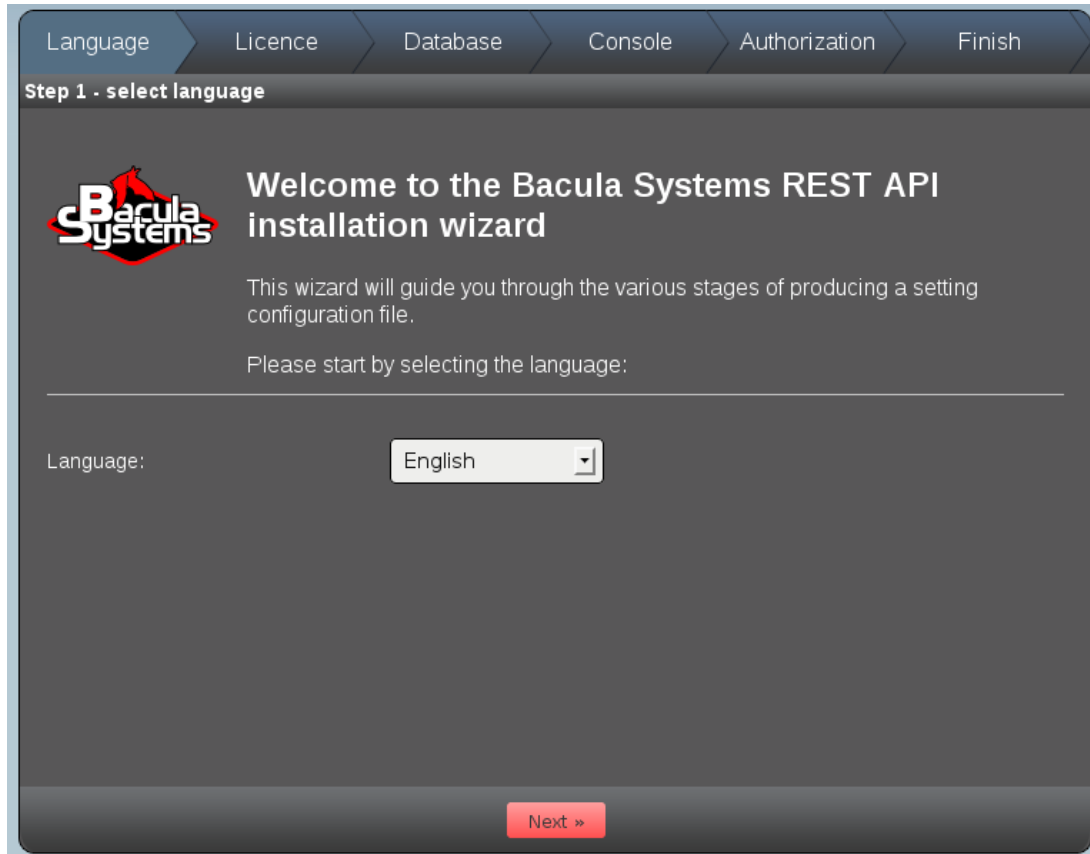
Fig. 5: Welcome Panel

If the `https://localhost/` command works, you should get a screen that looks like Figure *Welcome Panel*.

At the moment, English is the only language, so you should click on the `Next` >> button.

License Acceptance

You will then be presented with the License (Figure *License Panel*, which you should read, then check the box saying you have read and agree with it, and finally click on the `Next` >> button.

Database Selection

You should then be at the database panel (Figure *Database Panel*).

The first thing to do is to select a database, which is the type of database your Bacula Director uses. Then, if you do not use the standard database name or login (bacula), you should change those fields as well as enter the database password.

If you do not enter an **IP address** or hostname or a **Port**, the REST API will assume that the database is located on the machine you are running on and will attempt to connect to the database through the

Fig. 6: License Panel

Fig. 7: Database Panel

standard Unix socket. If you database resides on another machine, you should fill in the IP address or hostname and possibly choose a different port.

Please note: on some systems, notably Ubuntu Precise, the packagers have incorrectly placed the Unix socket in /tmp rather than in the standard location in **/var**, and consequently, this often produces database connection problems.

If you click on the `test` button, the REST API will immediately attempt to connect to your SQL server, and if it does, it will show a green check mark with a green OK. See Figure *Completed Database Panel*, which shows how to connect to a database on a different machine (with no password).

Fig. 8: Completed Database Panel

Click on the `Next` >> button to go to the Console configuration panel.

bconsole and bxxjson Tools

The screen shown in Figure *Console Panel*. This is where you configure REST API's access to bconsole and the bxxxjson tools that it needs to work with the Bacula Enterprise Director. The configuration fields are all filled with the default, which for a standard installation should work. If you want to see if the REST API has a proper connection to your Director, click on the `test` button. If you do not get the green check mark and OK, you will need to fix the problem. Most often, it is due to permissions.

Once bconsole is working, click on the `Next` button and you will get the Authorization panel.

Fig. 9: Console Panel

Authorization Panel

Fig. 10: Authorization Panel

The Authorization Panel (Figure *Authorization Panel* is where you define the name of the Administrator and his/her password. The name and password can be anything you want, but please use a strong password (upper/lower case characters, plus a number and special character at least 8 characters long), because this person will have access to all the Authentication accounts that you will create.

Click on the `Next >>` button to go to the Finish panel.

Finish Panel

The Finish Panel will show you a summary of everything that you have configured. If it looks correct click on the `Save` button to save the results. If you need to modify any of the configuration items later, you can simply re-execute the `https://localhost/installation-wizard` command to go through the Installation Wizard again.

After clicking on the `Save` button, you will be asked to confirm that you want to save the data (create the *settings.conf* file) as well as install the OAuth2 SQL tables in the the catalog database.

Creating User OAuth2 Accounts

If clicking the Save button on the Finish Panel is successful, you will arrive at the **login** panel that looks like Figure *Login Panel*.
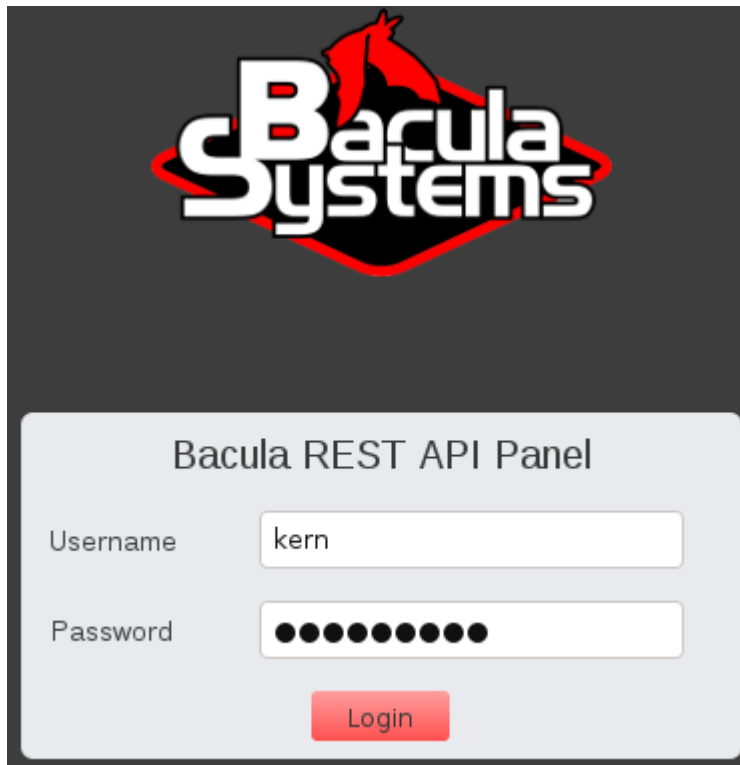

Fig. 11: Login Panel

If your browser has not already filled in your administrator's name and password that you just put into the Authorization Panel, please fill them in and click on the **Login** button.

Now you should be on a screen where you can create a REST API User account. The user account ID and User Secret key will be generate that will permit that user to have access to the REST API functions. You can create as many users as you want. To begin, click on the **Add REST API User** button and you will get a screen as shown in Figure *Filled Add User Panel*.

Please fill in the User Name, then click on the red generate buttons for User Id and User Secret. The REST API will create appropriate random strings. These strings (especially the User Secret Key) should be kept secret and will be needed in the OAuth2 process to get access to the REST API.

Also, please enter the Redirect URI which if you are testing or using the built-in REST-API OAuth2 server, should be:

```
https://localhost/
```

Please include the trailing forward slash. In other case, such as interfacing to cPanelthe Redirect URI must point to your application for authentication.

Unless you have your own or a special OAuth server. Using localhost here will indicate to the REST API to use its own built-in OAuth server. Leave the Scope field at the default value (full access). Scopes can be used to limit the REST API access for a particular user to a particular or set of Categories. For more details on Scopes, see section scopes.

Fig. 12: Filled Add User Panel

Then click on the `Add user` button at the bottom of the panel to actually create the user (Figure *Filled Add User Panel*).

If the user is correctly created, you will have a screen as shown in Figure *User Panel*.

If you need to know the User ID and/or User Secret Key, simply click on the **edit** Action, and you will get the screen shown in Figure *Edit User Panel*.

If you want to create a new User Secret Key, in case the user lost his/her key or the key was compromised, click on the `generate` button. All the fields that are available in this panel with the exception of the User ID, which is the way the user is identified by the system. If you wish to change the User ID, you must first delete the User entry, then create a new one.

OAuth2 Testing

Once you have create at least one REST API User, you can proceed to testing the API. To do so, you must put the User ID (random string) and the User Secret Key into the program that will be used to access the REST API and that uses web POST requests. If you have such an application, you can try it.

You can test the REST API interface by going to:

```
https://localhost/panel/
```

and loging in as the administrator, and then click on the **console** item under the **Action** column for any user. This will bring up a console Window that will allow you to enter REST API commands. The results of those commands will be displayed in the Window. See *OAuth2 Console Test Panel*.

Finally, another way to test the REST API is to use the **oauth2cli.sh** shell script that comes with the REST API. It is installed in:

```
/opt/bacula/rest-api/scripts
```

Normally, you will only need to enter the User Id and User Secret and other information on the following lines that are near the top of the **oauth2cli.sh** program:
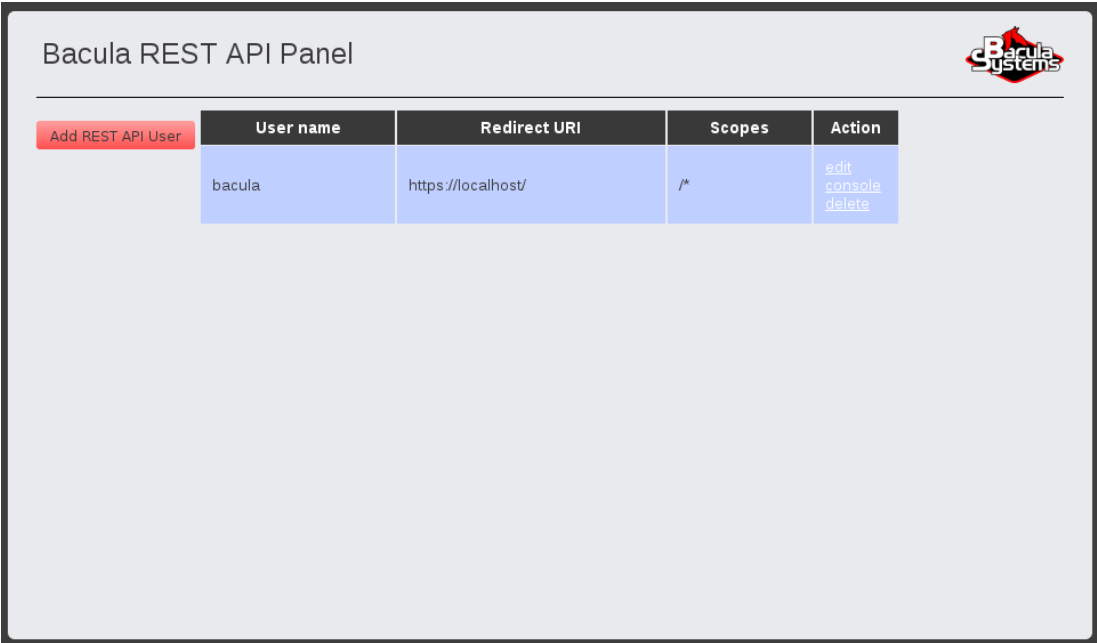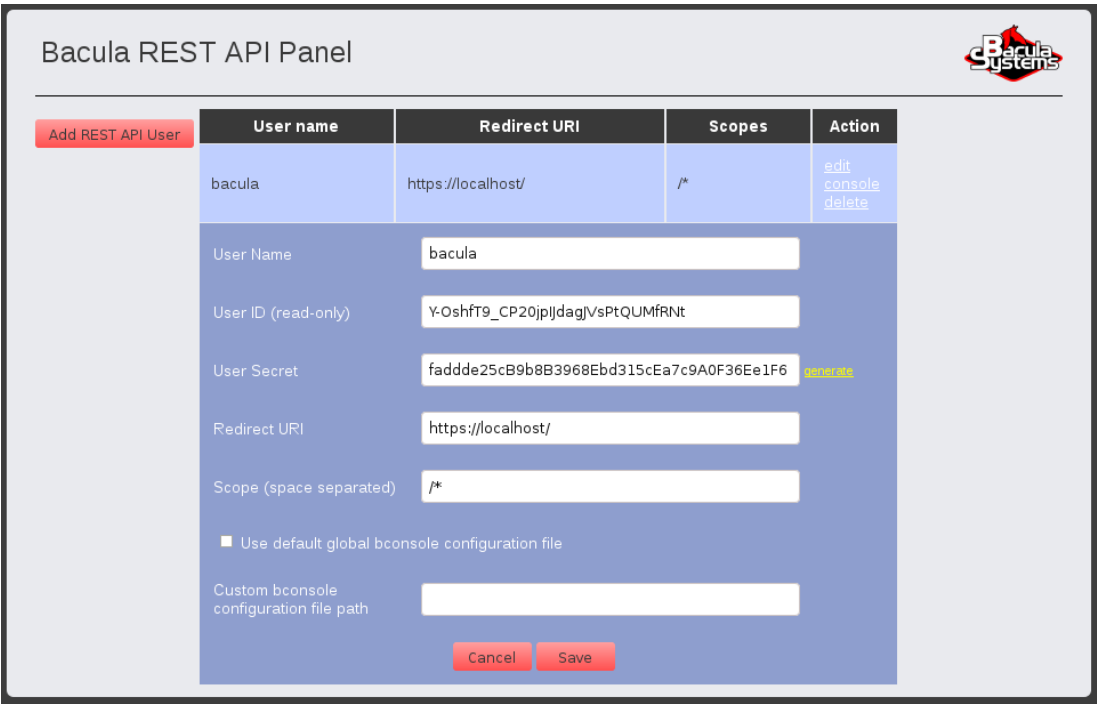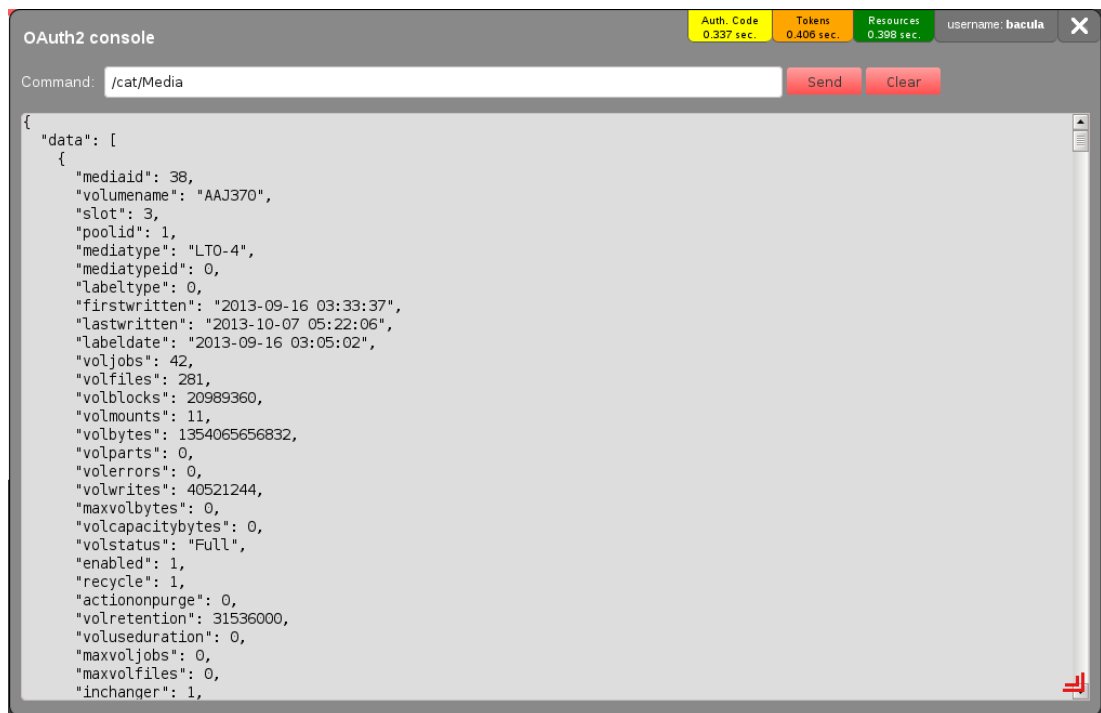
Fig. 13: User Panel

Fig. 14: Edit User Panel

Fig. 15: OAuth2 Console Test Panel

```
readonly USER_ID='Y-OshfT9_CP20jpIJdagJVsPtQUMfRNt'
readonly USER_SECRET='faddde25cB9b8B3968Ebd315cEa7c9A0F36Ee1F6'
readonly USER_REDIRECT_URI='https://localhost/'
readonly USER_SCOPE='/*'
readonly USER_STATUS=3
readonly SERVER_RESOURCES='https://localhost/'
```

where the above two lines have been modified to have the same values created during the creation of the user **bacula**.

You can then execute the script with:

```
/opt/bacula/rest-api/scripts/oauth2cli.sh
```

and it should produce the following output:

```
Please input request for send to Bacula REST API Server.
Requests should be input WITHOUT http://yourhost/ prefix (please look into
below examples).

Examples:
        GOOD:
                /cat/Media/
                /cmd/list?jobs
                /cat/Client?regex=^abc
        BAD:
                cat/Media/
```

```
            https://yourhost/cat/Media/
            /yourhost/cat/Media/


==========================================


Bacula REST API UserApp https://localhost>
```

at which point, you can enter on of the suggested commands and if your Bacula database has some jobs, for the first two, you should get output, and for the third one, you will probably get and error since **abc** is unlikely to be a valid Client name. You will probably get something like:

```
{
    "error" : 10,
    "data" : "Client does not exist."
}
Status: OK (tokens refreshed)

Bacula REST API UserApp https://localhost>
```

Use ctl-C to stop the shell script.

OAuth2 compatible endpoints

Since version 8.8.3, a new "v2" OAuth2 interface has been introduced because of compatibility issues in the initial interface of the OAuth2 specification.

Initial OAuth2 endpoints:

- Authorization URL: /authorize

- Access token URL: /request_token

- Refresh token URL: /refresh_token

New "v2" compatible OAuth2 endpoints:

- Authorization URL: /v2/authorize

- Access token URL: /v2/request_token

- Refresh token URL: /v2/refresh_token

10.12 Advanced Topics

Restricted Consoles

As shown above the REST-API administrator defines which *bconsole.conf* configuration file that will be used by each user defined to use the REST-API. The *bconsole.conf* file can either be unrestricted or restricted. In the case of a restricted console, REST-API must have a few minimum permissions. At a minimum, you must define in the *bacula-dir.conf* file the following ACLs for the user:

```
CatalogACL = <your-catalog>
CommandACL = "use, .catalogs"
```

Of course, for the REST-API to be useful, you will generally also need to include the JobACL and possible ClientACL. A simple example in *bacula-dir.conf* might be:

```
Console {
  Name = "JohnDoe"
  Password = "secret"
  JobACL = "Johns-NightlySave"
  CatalogACL = MyCatalog
  CommandACL = "use, .catalogs, list"
}
```

Basic Authentication

As an alternative to use OAuth2 authentication or as an addition to OAuth2, with lighttpd, you may add basic authentification, similar to what Apache permits.

> **Warning:** The Basic authentication should be used for tests purposes. In production environments please use the OAuth2 authorization.

The following lighttpd configuration example implements basic authentication.

```
# in file /opt/bacula/rest-api/etc/lighttpd.conf
...
server.modules += ( "mod_auth")     # if not already done

auth.backend = "htpasswd"
auth.backend.htpasswd.userfile =  "/opt/bacula/rest-api/etc/htpasswd.rest-api"
auth.require = ( "/" =>
        (
        "method" => "basic",
        "realm" => "Password protected area",
        "require" => "valid-user"
        )
   )
...
```

The ligghtpd command `htpasswd` allows you to manage the list of users that can authenticate.

```
# htpasswd -c /opt/bacula/rest-api/etc/htpasswd.rest-api admin
New password:
Re-type new password:
Adding password for user admin
```

Using Scopes

Scopes are an additional feature that permits you to limit particular users to specific URIs using regular expressions.

When configuring Users authorized to use the REST API, on the Add User Panel, the last item on the screen is Scope with a default value of **/***, which permits the user to enter any URI. Scopes consist of one or more regular expressions separated by spaces. For each URI that a particular user enters, his URI is compared to the Scopes that were entered when the user was defined. If no regular expression matches the URI presented, the URI will be rejected. Each scope regular expression that is entered by the administrator will be prefixed by the uparrow character. So the default scope of **/*** is parsed as the following regular expression:

```
^/*
```

which will permit all URIs that begin with a forward slash. The scope:

```
/cat/*
```

will permit only catalog commands.

The scope:

```
/cat/* /cmd/list/*
```

will permit all catalog commands and all Bacula **list** commands.

If a URI is given that has no matching scope, you will get the following error.

```
{
  "error": "invalid scope"
}


\smallskip
Note, if you specify a scope that you want to match a question mark, because
a scope is a regular expression, you must explicitly escape the
scope.  For example the following scope:

\begin{verbatim}
/cmd/list/[?]{1}[mp]+.*
```

will match all URI's beginning with /cmd/list? and followed by a m or a p. For example

```
/cmd/list?media
```

or

```
/cmd/list?pools
```

Disabling OAuth2 Authorization

If you wish to disable OAuth2 authorization to Bacula REST API Server, then you can edit file located in:

```
/opt/bacula/rest-api/www/protected/Data/settings.conf
```

In the file you may find the option:

```
enable_oauth2 = "1"
```

For disabling OAuth2 authorization, please set above value to 0 (zero), eg:

```
enable_oauth2 = "0"
```

From now you may get all requests to REST API directly and without any authorization, eg.:

```
https://localhost/cat/Media/

https://localhost/cmd/list?jobs
```

Please be careful disabling this option. We strongly recommend that in a production environment that this option should always be turned on (set to "1").

OAuth2 Token Expiration

The OAuth2 protocol works by acquiring a token from the authentication server, and the REST-API then uses that token for a certain amount of time and then refreshes the token. The token refresh takes a bit more time than a normal transaction. Currently the value is set to 12 seconds. If you wish to adjust it to a larger value (slightly less secure) such as 30 seconds, you can simply edit the file:

```
/opt/bacula/rest-api/www/porotected/Class/OAuth2/OAuth2.php
```

and change the line:

```
const ACCESS_TOKEN_EXPIRES_TIME = 12
```

to the value you want.

Architecture

The basic components in the Bacula Enterprise REST API as well as the flow of information and authorization is shown in the next figure.

The diagram in figure *ACL Protection* shows what is protected by Bacula Restricted Consoles and Bacula ACLs.
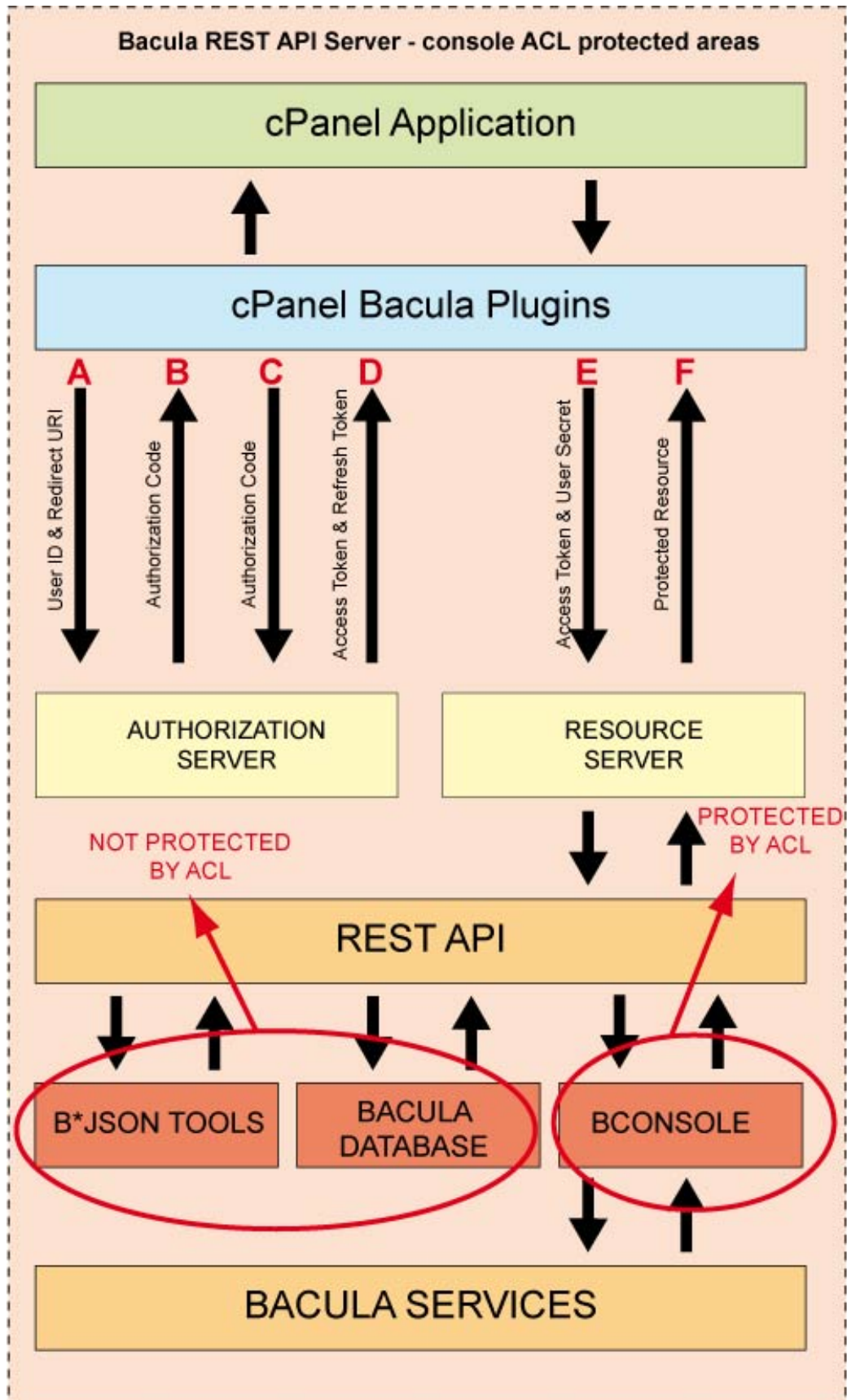
Fig. 16: ACL Protection

Working with cPanel

The REST API interface can also be used with various vendor panels such as cPanel.

The diagram in figure *Example cPanel Plugin* shows where the Bacula cPanel Plugin is placed and how it works.

Files for Bacula example cPanel Plugin are located in directory:

```
/opt/bacula/rest-api/www/protected/tools/cPanel_example_module/
```

with the front end files for the cPanel plugin visable part of the functionality in the following subdirectories of the cPanel_example_module:

```
./bbackup
./bbackup/runrestore.html
./bbackup/runbackup.html
./bbackup/index.html
```

The backend is found in:

```
/opt/bacula/rest-api/www/protected/tools//cPanel_example_module/Bbackup.pm
```

and the installation pack that contains icon and link to functionalities in the above files:

```
/opt/bacula/rest-api/www/protected/tools/cPanel_example_module/bbackup.
↪cpanelplugin
```

The ISP Accounts plugin visible on diagram is integreted with Bacula REST API code as plugin located in:

```
/opt/bacula/rest-api/www/protected/Pages/Plugins/Accounts.php
```

It makes available the following interface:

- /accounts/username123?action=list - lists backups

- /accounts/username123?action=backup - performing backup home directory of username123

- /accounts/username123?action=restore&jobid=XXX - performing restore from backup with jobid XXX to directory /home/restore/current date/

For using example Bacula REST API plugin one must force users to use ISP Accounts plugin by defining scopes. For example the user **user321** must have the following scope definition:

```
/accounts/user321/*
```

The Bacula jobs associated with **user321** would then be prefixed with the user name as follows:

```
user321-Full-Backup
```

The Filesets much also be prefixed as follows:

```
user321-Fileset
```
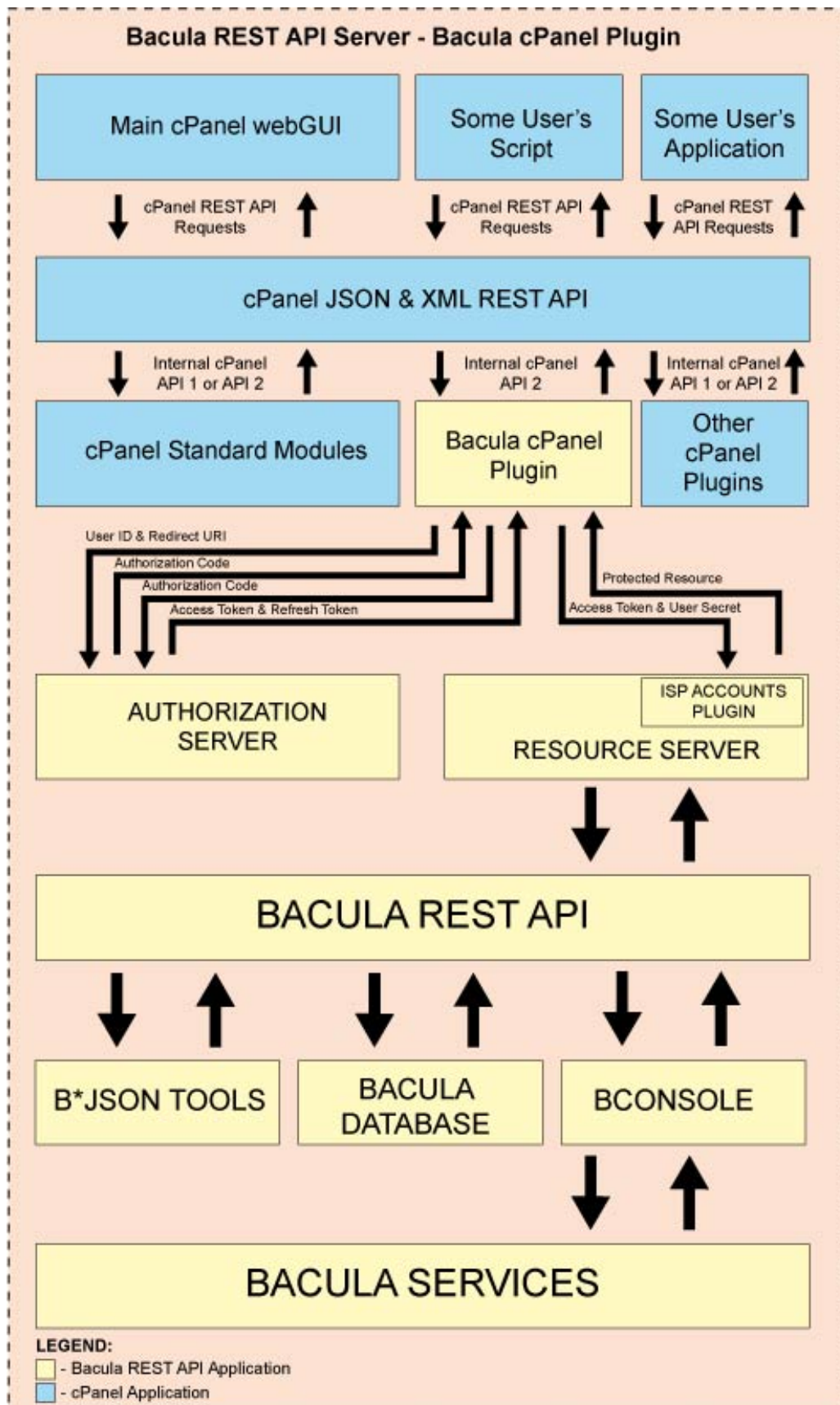
and the clients likewise:

Copyright © 2025 Bacula Systems. All trademarks are the property of their respective owners.

Fig. 17: Example cPanel Plugin

```
user321-fd
```

These parameters can be changed in the file:

```
/opt/bacula/rest-api/www/protected/Pages/Plugins/Accounts.php
```

and in

```
/opt/bacula/rest-api/www/protected/tools//cPanel_example_module/Bbackup.pm
```

10.13 Manual Installation

Normally, the Bacula Enterprise REST API is installed using rpms or debs, in which case other than fine tuning the installation, everything is already done.

Below, we give all the steps necessary to install the REST API manually.

Requirements

- Web server (recommended: lighttpd)

- PHP 5.3 or newer

- PHP modules:

    - PDO PHP support (for PostgreSQL, MySQL or other)

    - cURL PHP module

    - BCMath PHP module (php-bcmath package on RHEL)

The Web server Installation with PHP

Bacula REST API Server uses a web server environment for work. The REST API is available via http(s) protocol. The recommended web server for using with Bacula REST API Server is Lighttpd.

The whole package is installed in the directory /opt/bacula/rest-api. Under the rest-api directory, you will find the following additional directories:

- **bin**
  where any binaries or start scripts are placed.

- **etc**
  where configuration files are placed (e.g. lighttpd.conf)

- **log**
  where you can find lighttpd log files

- **scripts**
  where you can find various useful scripts

- **uploads**
  where your app can upload files if necessary (not recommended)

- **www**
  where the rest-api web files reside

Lighttpd Installation (Ubuntu/Debian)

The following command will install lighttpd:

```
# apt-get install lighttpd
```

PHP Installation (Ubuntu/Debian)

There is also need to have PHP support for web server. Command for install all PHP dependencies looks like:

```
# apt-get install php5-cgi php5-curl php5-pgsql
```

The above example shows installation for PostgreSQL database (php5-pgsql). If you use another than PostgreSQL database as Bacula catalog, then you need to install your database driver for PHP PDO (PHP Data Objects).

SSL Certificate Preparation

For using SSL encryption (https) in web server there is need to have SSL certificate.

Certificate for Lighttpd

Below is example generation of SSL certificate for Lighttpd server. Each 'yourhost' name you should replace to your own host name.

```
# mkdir -p /etc/lighttpd/ssl/yourhost
# cd /etc/lighttpd/ssl/yourhost
# openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout yourhost.key -
↪out yourhost.crt
```

After executing the openssl command you will be asked about few parameters for generating yourhost.crt file. Most of them you may skip, but in question "Common Name (e.g. server FQDN or YOUR name) []:" you need to write yourhost name (eg. api.myhost.com) for which SSL certificate will be work.

Last step for SSL certificate preparation is:

```
# cat yourhost.key yourhost.crt > yourhost.pem
```

Web Server Configuration

Below is an example lighttpd configuration file that can be used with the Bacula Enterprise REST-API. Unless you are testing, you should replace the `server.name` with your actual hostname. This configuration file is normally located in */opt/bacula/rest-api/etc/lighttpd.conf*.

```
#
# Bacula REST-API lighttpd configuration file
#

server.port            = 443
```

```
server.username       = "bacula"
server.groupname      = "bacula"
server.name           = "localhost"
server.document-root  = "/opt/bacula/rest-api/www"
server.follow-symlink = "enable"
#server.upload-dirs    = ( "/opt/bacula/rest-api/uploads" )
server.errorlog       = "/opt/bacula/rest-api/log/error.log"
accesslog.filename    = "/opt/bacula/rest-api/log/access.log"
server.breakagelog    = "/opt/bacula/rest-api/log/debug.log"
server.pid-file       = "/opt/bacula/working/lighttpd.pid"

ssl.engine            = "enable"
ssl.pemfile           = "/opt/bacula/rest-api/etc/lighttpd/server.pem"
ssl.ca-file           = "/opt/bacula/rest-api/etc/lighttpd/server.crt"

server.modules = (
        "mod_access",
        "mod_alias",
        "mod_compress",
        "mod_redirect",
        "mod_rewrite",
        "mod_fastcgi",
#       "mod_accesslog",
)


#
# Turn on the following for debug and restart lighttpd
#
#debug.log-request-handling = "enable"
#debug.log-request-header = "enable"
#debug.log-request-header-on-error = "enable"



                        "index.htm", "default.htm",
                       " index.lighttpd.html" )

url.access-deny       = ( "~", ".inc" )

static-file.exclude-extensions = ( ".php", ".pl", ".fcgi" )

## Use ipv6 if available
#include_shell "/usr/share/lighttpd/use-ipv6.pl"

dir-listing.encoding  = "utf-8"

compress.cache-dir    = "/var/cache/lighttpd/compress/"
compress.filetype     = ( "application/x-javascript", "text/css", "text/html",
→ "text/plain" )

#include_shell "/usr/share/lighttpd/include-conf-enabled.pl"

#include_shell "/usr/share/lighttpd/create-mime.assign.pl"
```

```
#
# If the above /usr/share/lighttpd/create-mime.assign.pl
#   does not exist, enable the following lines
#
mimetype.assign = (
  ".html" => "text/html",
  ".gif" => "image/gif",
  ".jpeg" => "image/jpeg",
  ".jpg" => "image/jpeg",
  ".png" => "image/png",
  ".ico" => "image/x-icon",
  ".css" => "text/css",
  ".json" => "text/plain",
  ".js" => "application/javascript",
)

fastcgi.server = (".php"=>((
  "bin-path"=>"/usr/bin/php5-cgi",
  "socket"=>"/tmp/php.socket"
)))

url.rewrite-once = (
   "^/themes/(.+)$" => "/themes/$1",
   "^/assets/(.+)$" => "/assets/$1",
   "^/(.+)$" => "/index.php/$1"
)
```

PHP configuration

For the Lighttpd server to work correctly with PHP, there is need to adjust PHP configuration file for CGI scripts (usually located in /etc/php5/cgi/php.ini). You must add the following line:

```
cgi.fix_pathinfo=1
```

10.14 Manual Configuration

Settings File

The Settings file for the REST API Administration Panel is:

```
/opt/bacula/rest-api/www/protected/Data/settings.conf
```

This file is writen by the configuration wizard available in the REST API Administration Panel available as WebGUI.

Please note that after first install Bacula Enterprise REST API this settings file will not exist. If this file is created manually you must ensure that your Web server has read access to it or even better has read and write access.

DB1 Section within the Setting File

Section "db1" allows defining the primary database inside is stored "oauth2_* tables for OAuth2 authorization. Data from other Catalog databases if any will be available by "?catalog=XXX" requests, where XXX is name of Catalog resource name from configuration file of the Bacula Director service. There is no need to define other Catalog databases.

OPTIONS

- Catalog database type.

    Possible values:

    mysql
    for MySQL catalog database type,

    pgsql
    for PostgreSQL catalog database type.

- Catalog database name.
- Catalog database login/username.
- Catalog database password as plain text.
- IP address or hostname on which the Catalog database server listens.

Note:

If database access is available via Unix socket then:

- for MySQL Catalog database please input "localhost" value,

- for PostgreSQL Catalog database please leave this field empty. I.e. "".

- port on which the Catalog database server listens.

Example configuration of "db1" section

```
[db1]
type = "pgsql"
name = "bacula"
login = "bacula"
password = "My$tR0n6Pa$$"
ip_addr = "172.16.77.1"
port = "5432"
```

Bconsole Section

The "bconsole" section is used to input locations for access to the Bacula Console (bconsole) program.

OPTIONS

- the full path and filename of the Bacula Console program.
- the full path to the Bacula Console configuration file.
- enable/disable access by the web server to using "sudo".

  Possible values:

  0

  > disable access by "sudo" and are using direct Bacula Console executions.

  1

  > enable access by "sudo" and are using Bacula Console executions with "sudo".

Example configuration of "bconsole" section

```
[bconsole]
bin_path = "/opt/bacula/bin/bconsole"
cfg_path = "/opt/bacula/etc/bconsole.conf"
use_sudo = "1"
```

Tools section

The "tools" section defines access to the bdirjson, bsdjson, bfdjson programs. These programs are used for getting Bacula configuration settings directly from configuration files. Subsequently these settings are available via REST API interface with "/res/" requests.

OPTIONS

- full path and filename to the bdirjson program.
- full path and filename to the bsdjson program.
- full path and filename to the bfdjson program.
- enable/disable access by the web server to using "sudo".

  Possible values:

  0

  > disable using "sudo" to execute the Bacula JSON tools. I.e. execute them directly.

  1

  > enable using "sudo" to execute the Bacula JSON tools executions with "sudo"

Example configuration of "tools" section

```
[tools]
bdirjson = "/opt/bacula/bin/bdirjson"
bsdjson = "/opt/bacula/bin/bsdjson"
bfdjson = "/opt/bacula/bin/bfdjson"
use_sudo = "1"
```

API section

API section defines access parameters to REST API Administration Panel available as WebGUI address:

```
http(s)://yourhost/panel/
```

OPTIONS

- username administration access to the REST API Panel.

- the password hashed by md5 to get administration access to REST API Panel.

    Possible values:

    0

    > NOT RECOMMENDED, disable OAuth2 authorization framework. When set to 0 then access to REST API resources will be available directly by HTTP GET requests.

    1

    > DEFAULT, enable OAuth2 authorization framework. When set "1" value then the REST API access is protected by the OAuth2 framework.

- determine log level for messages, warnings and errors that are saving in log file located on:

    ```
    /opt/bacula/rest-api/www/protected/Data/bacula-rest-api.log
    ```

    Possible values:

    0

    > no logs are saved.

    1

    > DEFAULT, only error messages are saved in the log file.

    2

    > all available messages including messages "info" type are saved in the log file.

Example configuration of "api" section

```
username = "gani"
password = "e42e3b59f26e17b02bee3967f78782ef"
enable_oauth2 = "1"
log_level = "1"
```

Example configuration whole settings.conf file

Please note that for the REST API to work correctly one must define all possible sections. Here is example of a configuration file:

```
[db1]
type = "pgsql"
name = "bacula"
login = "bacula"
password = "MyEasyPassword"
ip_addr = "192.168.100.1"
port = "5432"

[bconsole]
bin_path = "/opt/bacula/bin/bconsole"
cfg_path = "/opt/bacula/etc/bconsole.conf"
use_sudo = "1"

[tools]
bdirjson = "/opt/bacula/bin/bdirjson"
bsdjson = "/opt/bacula/bin/bsdjson"
bfdjson = "/opt/bacula/bin/bfdjson"
use_sudo = "1"

[api]
username = "Joe"
password = "e42e3b59f26e17b02bee3967f78782ef"
enable_oauth2 = "1"
log_level = "1"
```

11 AS/400 Backup

This article presents various techniques and strategies to integrate IBM System i® AS/400 with **Bacula Enterprise**. Its aim is to present solutions for **Bacula Enterprise** 6.0 and IBM System i5/OS v5r4 and later, which are not applicable to prior versions.

The AS/400 machine architecture is different from that of most other machines in the industry, especially the usual systems at the server level.

Unlike the "everything is a file" feature of Unix and its derivatives, on AS/400 everything is an object (with built-in persistence and garbage collection). AS/400 also offers Unix-like file directories using the Integrated File System.

Bacula Enterprise doesn't provide a native AS/400 File Daemon program. However, as the AS/400 system provides its own set of commands to back up objects from disk to tape, to virtual tape or to a file (savefile), the administrator can use those certified commands to create backup, and transfer savefiles or virtual tapes to the Bacula Enterprise server.

A word of caution is that you must still maintain proper tape based saves of the system (SAVE option 22) because you'll need that to restore a system sufficiently to be able to return the virtual tape images to the system in order to restore from them.

11.1 AS/400 Integration into Bacula Enterprise Environment

An AS/400 system can quite easily be integrated into your Bacula Enterprise as client. Basically one uses a backup script containing AS/400 `SAV` commands, centralize the log, then transfer the backup data to a Bacula Enterprise host using NFS or FTP.
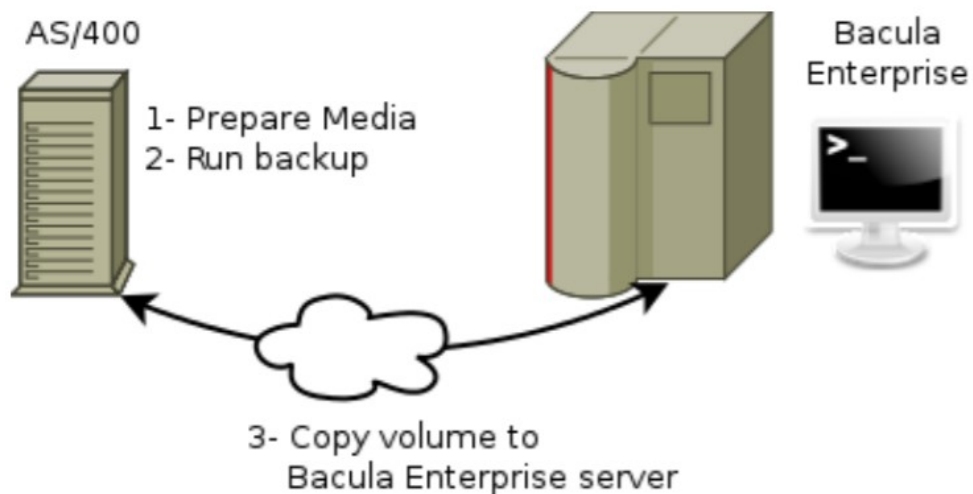

Fig. 18: AS/400 Integration with Bacula Enterprise

The backup script can be run from the AS/400 server or from the Bacula Enterprise server using FTP.

11.2 SAVE Commands Overview

11.3 Backup Media

On the AS/400, you have a choice of several kinds of devices for your backup:

- Tapes
- Files (SAVEF)
- Virtual Tapes
- Optical Drives
- Virtual Optical Drives

As our main goal is to copy backup media over the network using NFS or FTP, you will have to choose between Files and Virtual Tapes. Both can be copied onto any platform, however, as you can save multiple libraries to a Virtual Tape using one save command, but only one library to a Save File per save command.
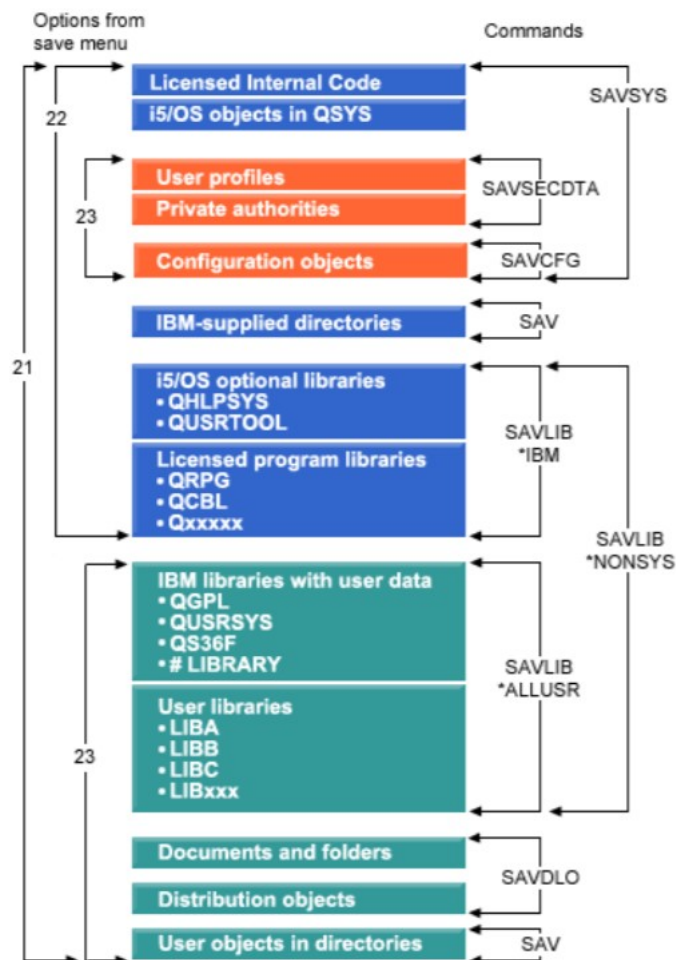
Fig. 19: SAVE Commands Overview

Table 1: Media Used with the Save Commands

Command	Virtual Tape	Save File
SAVSYS	Yes	No
SAVCFG	Yes	Yes
SAVSECDTA	Yes	Yes
SAVLIB	Yes	Yes
SAVOBJ	Yes	Yes
SAVCHGOBJ	Yes	Yes
SAVDLO	Yes	Yes
SAVSAVFDTA	Yes	No
SAVLICPGM	Yes	Yes
SAVSTG	No	No
SAV	Yes	Yes
RUNBCKUP	Yes	No
SAVSYSINF	Yes	Yes

The above table shows that you cannot use Save Files files with all SAVE commands. Given these limitations on File type, We advise you to use Virtual Tapes (Available since v5r4).

http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Frzaiu%2Frzaiuintro.htm

When a backup to a virtual tape volume is complete, you can duplicate the data to physical media at any time and not interfere with system operations.

Virtual tape is beneficial for unattended saves because it eliminates media errors that could halt an unattended save. If you do not allocate sufficient space in the virtual volumes within the image catalog to save the intended information, the virtual tape will use the auto-generate feature to create additional virtual tape volumes.

http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=%2Frzam4%2Frzam4virtualtapeusevt.htm

11.4 Implementation Details

Creating Virtual Tape System

We recommend that you read the following RedBook on Virtual Tape system http://www.redbooks.ibm.com/abstracts/sg247164.html

In this section, we will present commands to create a Virtual Tape `DEV1` and a Catalog `CAT1`.

The first step is to create the Virtual Tape device and set the various parameters.

```
CRTDEVTAP DEVD(DEV1) RSRCNAME(*VRT) ONLINE(*YES)
VRYCFG CFGOBJ(DEV1) CFGTYPE(*DEV) STATUS(*ON)
```

Then through FTP, you can create the Catalog `CAT1` that will store volumes into `/CAT1` directory. It is important to exclude the `/CAT1` directory from backup.

```
CRTIMGCLG IMGCLG(CAT1) DIR('/CAT1') CRTDIR(*YES) TYPE(*TAP)
CHGATR OBJ('/CAT1') ATR(*ALWSAV) VALUE(*NO)
```

Once done, you can create volumes in the catalog, the initial size will be minimal, and you can set the maximum size `IMGSIZ` to something coresponding to your real tape device in order to allow copy from

Virtual Tapes to Real Tapes. However, note that it will be more convenient if your daily backups fit on a single volume.

```
ADDIMGCLGE IMGCLG(CAT1) FROMFILE(*NEW) TOFILE(VOL001) IMGSIZ(10000)
     ALCSTG(*MIN) VOLNAM(VOL001) VOLTYP(*SL) DENSITY(*VRT256K)
ADDIMGCLGE IMGCLG(CAT1) FROMFILE(*NEW) TOFILE(VOL002) IMGSIZ(10000)
     ALCSTG(*MIN) VOLNAM(VOL002) VOLTYP(*SL) DENSITY(*VRT256K)
ADDIMGCLGE IMGCLG(CAT1) FROMFILE(*NEW) TOFILE(VOL003) IMGSIZ(10000)
     ALCSTG(*MIN) VOLNAM(VOL003) VOLTYP(*SL) DENSITY(*VRT256K)
```

Running a Backup

Now you are ready to run your first backup.

```
SAVLIB QUSRSYS DEV(DEV1) VOL(VOL001) OUTPUT(*PRINT) INFTYPE(*ERR)
```

Getting Logs

To retrieve the backup output using FTP, you can store logs to a library LOGS

```
CRTLIB LIB(LOGS)
```

To transfer the backup log from the Printer to the LOGS library, you can use the following command:

```
DLTF LOGS/BACKUP
CRTPF FILE(LOGS/BACKUP) RCDLEN(133)
CPYSPLF FILE(QPSAVOBJ) JOB(*) MBROPT(*ADD)  SPLNBR(*LAST)  TOFILE(LOGS/BACKUP)
```

To clear a volume (after a successful FTP transfer), use the following command:

```
INZTAP CHECK(*NO) DEV(DEV1) NEWVOL(VOL001) VOL(VOL001) CLEAR(*YES)
```

Running Script Remotely

To run a script remotely, use the RCMD FTP command as in the following example, the script will create a backup of the library YOURLIB in a SAVF file, and retrieve it.

```
cat << EOF > ftp.txt
user YOURPROFILE YOURPASSWORD
bin
quote rcmd CRTSAVF QTEMP/YOURLIB
quote rcmd SAVLIB YOURLIB dev(*savf) SAVF(QTEMP/YOURLIB) DTACPR(*YES)␣
↪OUTPUT(*PRINT)
get QTEMP/YOURLIB YOURLIB.savf
quote DLTF QTEMP/YOURLIB
quit
EOF
ftp -n -i -s:ftp.txt as400host
```

Remote Backup

The following script is an example to backup two libraries (YOURLIB, OTHERLIB), retrieve the volume and backup status.

```bash
#!/bin/bash
mkdir -p /opt/bacula/working/as400
cd /opt/bacula/working/as400
rm -f VOL001 BACKUP-001.log

cat << EOF > ftp.txt
user YOURPROFILE YOURPASSWORD
quote rcmd CRTPF FILE(LOGS/BACKUP) RCDLEN(133)
quote rcmd SAVLIB YOURLIB DEV(DEV1) VOL(VOL001) DTACPR(*YES) OUTPUT(*PRINT)
quote rcmd CPYSPLF FILE(QPSAVOBJ) JOB(YOURPROFILE/QPRTJOB) MBROPT(*ADD)⌑
↪SPLNBR(*LAST)  TOFILE(LOGS/BACKUP)
quote rcmd SAVLIB OTHERLIB DEV(DEV1) VOL(VOL001) DTACPR(*YES) OUTPUT(*PRINT)
quote rcmd CPYSPLF FILE(QPSAVOBJ) JOB(YOURPROFILE/QPRTJOB) MBROPT(*ADD)⌑
↪SPLNBR(*LAST)  TOFILE(LOGS/BACKUP)
ascii
get LOGS/BACKUP BACKUP-001.log
quote rcmd DLTF LOGS/BACKUP
quit
EOF

cat << EOF > ftp-get-volume.txt
user YOURPROFILE YOURPASSWORD
bin
cd /CAT1
get VOL001
quit
EOF

ftp as400host < ftp.txt
cat BACKUP-001.log
ftp as400host < ftp-get-volume.txt
```

Integration with Bacula

You can use the RunBeforeJob command to execute the backup and retrieve the result. Note that you can schedule the backup on your AS/400 server, and just get the result in the FTP script.

```
Job {
  Name = AS400_QUSRSYS
  Client = director-fd
  RunBeforeJob = "/opt/bacula/scripts/backup_as400.sh"
  RunBeforeJob = "cat /opt/bacula/working/as400/BACKUP-001.log"
  Fileset = FS_AS400
  ...
}

Fileset {
```

```
  Name = FS_AS400
  Include {
    File = /opt/bacula/working/as400
  }
}
```

The `cat` command will display the AS/400 backup log in the Bacula script as in this example.

```
5722SS1 V5R4M0  060210          Save Library - Object Information
Device . . . . . . . . :   DEV1
Target release . . . . :   V5R4M0         Storage  . . . . . . . :   *KEEP
Save access paths  . . :   Yes            Save file data . . . . :   Yes
Data compressed  . . . :   No             Data compacted . . . . :   No
Expiration date  . . . :   *PERM
Library  . . . . . . . :   TEST
Save date/time . . . . :   09/24/22    10:12:18
 Object              Type      Attribute    Saved           Size  Owner          ␣
↪                    Text
 TEST                *LIB      PROD         YES            86016  QSECOFR
                     Library ------Objects------         File              ␣
↪File
 Library     ASP        Saved    Saved   Not saved  Volume Label            ␣
↪Sequence    Owner     Size Text
 TEST        1                   YES         1              0  VOL001 TEST         ␣
↪  8  QSECOFR     86016
                        * * * * *   E N D   O F   L I S T I N G   * * * * *
```