



# Global Endpoint Deduplication

**Bacula Systems Documentation**

---

## Contents

1	Global Endpoint Deduplication	2
2	Global Endpoint Deduplication 2	27
3	Moving from Dedup 1 to Dedup 2	43

## Contents

---

**Bacula Enterprise** offers two versions of the Global Endpoint Deduplication plugin:

### 1 Global Endpoint Deduplication

- *Executive Summary*
- *Deduplication*
- *Dedupengine*
- *Hardware Requirements*
- *Installation*
- *Restrictions and Limitations*
- *Best Practices*

#### 1.1 Executive Summary

IT organizations are constantly being challenged to deliver high quality solutions with reduced total cost of ownership. One of those challenges is the growing amount of data to be backed up, together with limited time to run backup jobs (backup window). Bacula Enterprise offers several ways to tackle these challenges, one of them being *Global Endpoint Deduplication*, which minimizes network transfer and Bacula Volume size using deduplication technology.

This document is intended to provide insight into the considerations and processes required to successfully implement this backup technique.

#### 1.2 Deduplication

Deduplication is a complex subject. Generally speaking, it detects that data being backed up (usually chunks) has already been stored and rather than making an additional backup copy of the same data, the deduplication software keeps a pointer referencing the previously stored data (chunk). Detecting that a chunk has already been stored is done by computing a hash code (also known as signature or fingerprint) of the chunk, and comparing the hash code with those of chunks already stored.

The picture becomes much more complicated when one considers where the deduplication is done. It can either be done on the server and/or on the client machines. In addition, most deduplication is done on a block by block basis, with

some deduplication systems permitting variable length blocks and/or blocks that start at arbitrary boundaries (sliding blocks), rather than on specific alignments.

## Advantages of Deduplication

- Deduplication can significantly reduce the disk space needed to store your data. In good cases, it may reduce disk space needed by half, and in the best cases, it may reduce disk space needed by a factor of 10 or 20.
- Deduplication can be combined with compression to further reduce the storage space needed. Compression depends on data type and deduplication depends on the data usage (on the need or the will of the user to keep multiple copies or versions of the same or similar data). Bacula takes advantage that both techniques work perfectly together and combines them in its Dedupengine.
- Deduplication can significantly reduce the network bandwidth required because both ends can exchange references instead of the actual data itself. It works when the destination already has a copy of the original chunks.
- Handling references instead of the data can speed up most of the processing inside the Storage Daemon. For example, Bacula features like copy/migrate and Virtual Full can be up to 1,000 times faster. See the following article for more information on this subject.

## Cautions About Using Deduplication

Here are a few of the things that you should be aware of before using deduplication techniques.

- To do efficient and fast deduplication, the Storage Daemon will need additional CPU power (to compute hash codes and do compression), as well as additional RAM (for fast hash code lookups). Bacula Systems can help you to calculate memory needs.
- For effective performance, the deduplication Index should be stored on SSDs as the index will have many random accesses and many updates.
- Due the extra complexity of deduplication, performance tuning is more complicated.
- We recommend Index and Containers are stored in xfs or ext4 file systems. But we are also compatible with zfs file system.
- Deduplication collisions can cause data corruption. This is more likely to happen if the deduplicating system uses a weak hash code such as MD5 or Fletcher. The problem of hash code collisions is mitigated in Bacula by using a strong hash code (SHA512/256).
- Deduplication is not implemented for tape devices. It works only with disk-based backups.
- The immutable flag is not compatible or does not apply to the dedup index or dedup containers.

## Aligned Volumes

Bacula Systems' first step in deduplication technology was to take advantage of underlying deduplicating filesystems by offering an alternative (additional) Volume format that is aligned on specific chunk boundaries. This permits an underlying file system that does deduplication to efficiently deduplicate the data. This new Bacula Enterprise Deduplication Optimized Volume format is often called "Aligned" Volume format. Another way of describing this is that we have filtered out all the metadata and record headers and put them in the Metadata Volume (same as existing Volume format) and put only file data that can be easily deduplicated into the Aligned Volume.

Since there are a number of deduplicating file systems available on Linux or Unix systems (ZFS, lessfs, ddumbfs, SDFS (OpenDedup), LiveDFS, ScaleDFS, NetApp (via NFS), Epitome (OpenBSD), Quantum (in their appliance), ..., this Bacula Aligned Volume implementation allows users to choose the deduplication engine they want to use.

More information about Deduplication Optimized Volume Format can be found in **Bacula Systems'** **DedupVolumes** whitepaper.

## Global Endpoint Deduplication

Bacula Systems' first data source agnostic deduplication technology is the *Global Endpoint Deduplication* feature. With Global Endpoint Deduplication, Bacula will analyze data at the block level, then Bacula will store only new chunks in the deduplication engine, and use references in standard Bacula volumes to chunks stored in the deduplication engine. The deduplication can take place at the File Daemon side (saving network and storage resources), and/or at the Storage Daemon side (saving storage resources).

The remainder of this white paper will discuss only Global Endpoint Deduplication.

## How Bacula Global Endpoint Deduplication Works

- First, please be aware that you need the **dedup-sd.so** or the **bacula-sd-dedup-driver-x.y.z.so** Storage Daemon plugin for Global Endpoint Deduplication to work. Please do not forget to define the `Plugin Directory` in the Storage Daemon configuration file `bacula-sd.conf`.
- Dedup devices are enabled by specifying the `dedup` keyword as a `DeviceType` directive in each disk `Device` resource in the `bacula-sd.conf` where you want to use deduplicated Volumes.

```
DeviceType = Dedup
```

- You must pay particular attention to define a unique `Media Type` for devices that are `Dedup` as well as for each `Virtual Autochanger` that uses a different `Archive Device` directory. If you use the same `Media Type` for a `Dedup` device type as for a normal disk `Volume`, you run the risk that you will have data corruption on disk `Volumes` that are used on `Dedup` and non-`Dedup` devices.
- When Global Endpoint Deduplication is enabled, the `Device` will fill in disk volumes with chunk references instead of the chunks. Bacula encrypted data, and very small files will be stored in the `Volumes` as usual. The deduplicated chunks are stored in the "Containers" of the `Dedupengine`, and are shared by all other dedup-aware devices in the same `Storage Daemon`.
- We advise to set a limit on the number of `Jobs` or the usage duration when working with dedup `Volumes`. In case you prefer to use `Maximum Volume Bytes`, please consider that two `Catalog` fields are considered when computing the volume size. `VolBytes` represents the volume size on disk and `VolaBytes` considers the amount of non-dedup data stored in the volumes, i.e., the rehydrated data. If the directive `Maximum Volume Bytes` is used for a dedup `Volume`, Bacula will consider both `VolBytes` and `VolaBytes` values to check the limits.

## Global Endpoint Deduplication During Backup Jobs

- When starting a `Backup Job`, the `Storage Daemon` will inform the `File Daemon` that the `Device` used for the `Job` can accept dedup data.
- If the `FileSet` uses the `dedup = bothsides` option, the `File Daemon` will compute a strong hash code for each chunk and send references to the `Storage Daemon` which will request the original chunk from the `File Daemon` if the `Dedupengine` is unable to resolve the reference.
- If the `FileSet` uses the `dedup = storage` option, the `File Daemon` will send data as usual to the `Storage Daemon`, and the `Storage Daemon` will compute hash codes and store chunks in the `Dedupengine` and the references in the disk volume.
- If the `FileSet` uses the `dedup = none` option, the `File Daemon` will send data as usual to the `Storage Daemon`, and the `Storage Daemon` will store the chunks in the `Volume` without performing any deduplication functions.
- If the `File Daemon` doesn't support Global Endpoint Deduplication, the deduplication will be done on the `Storage` side if the `Device` is configured with `DeviceType = dedup`.

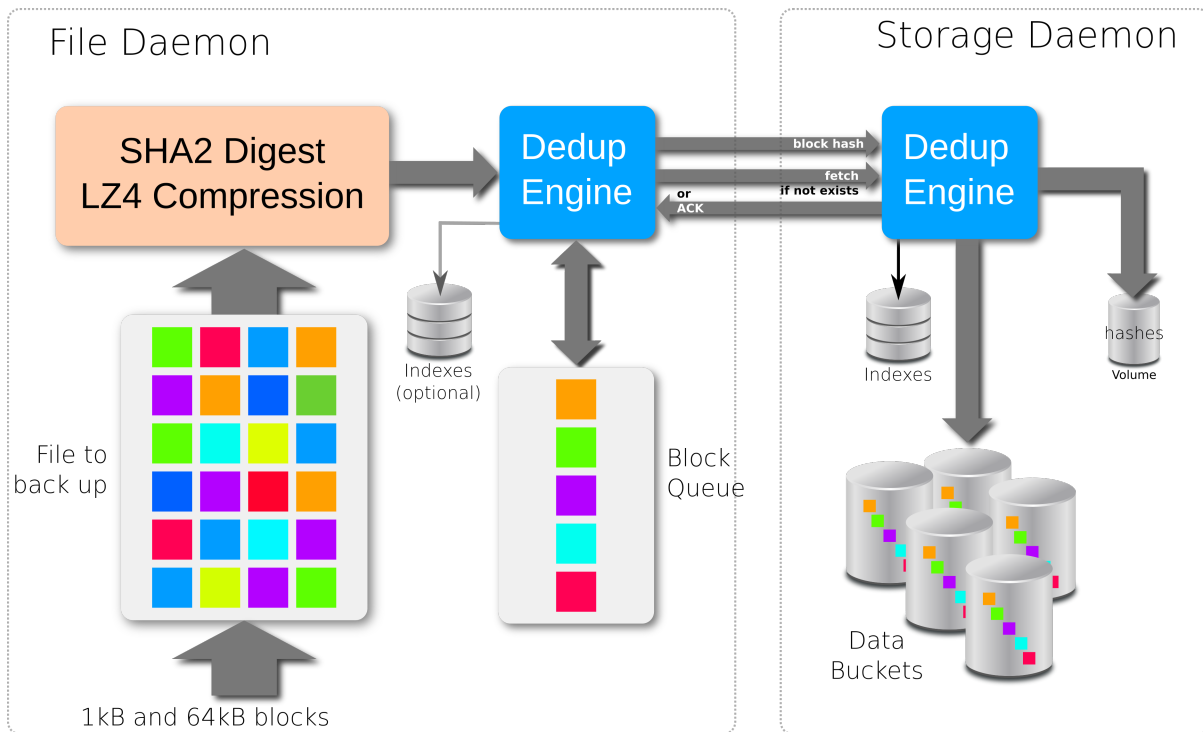


Fig. 1: Backup Scenario with bothsides deduplication

### Global Endpoint Deduplication During Restore Jobs

- If the directive `Enable Client Rehydration` is set to “yes” in the File Daemon configuration file, the Storage Daemon will send references to the File Daemon during a restore. If the directive is set to “no”, the Storage Daemon will rehydrate all the references and send the chunks to the File Daemon.
- When the File Daemon receives a reference, it will try to rehydrate the data using local data, see section `clientsiderehydration` below.

### Client Side Rehydration

**Attention: Client Side Rehydration is deprecated**

Client side rehydration is deprecated and should not be used with Bacula versions greater than 12.0.0. If you run into one of the specific cases described below for which this feature could be very useful, please contact the Support Team.

The File Daemon can try to do some rehydration on its own using local data. This feature can increase restore speeds for systems connected through a slow network and doesn't consume any resources at backup time.

This feature is activated with a FileDaemon resource directive called **Enable Client Rehydration** in `bacula-fd.conf`.

We recommend against using this feature on a client connected through a fast network, because the extra disk accesses and computation can slow down the speed of the restore jobs.

To take advantage of this feature you must understand how it works. At restore time, the client receives the original location, the offset and the hash of every chunk to restore. It then looks to see if the original file still exists, opens it

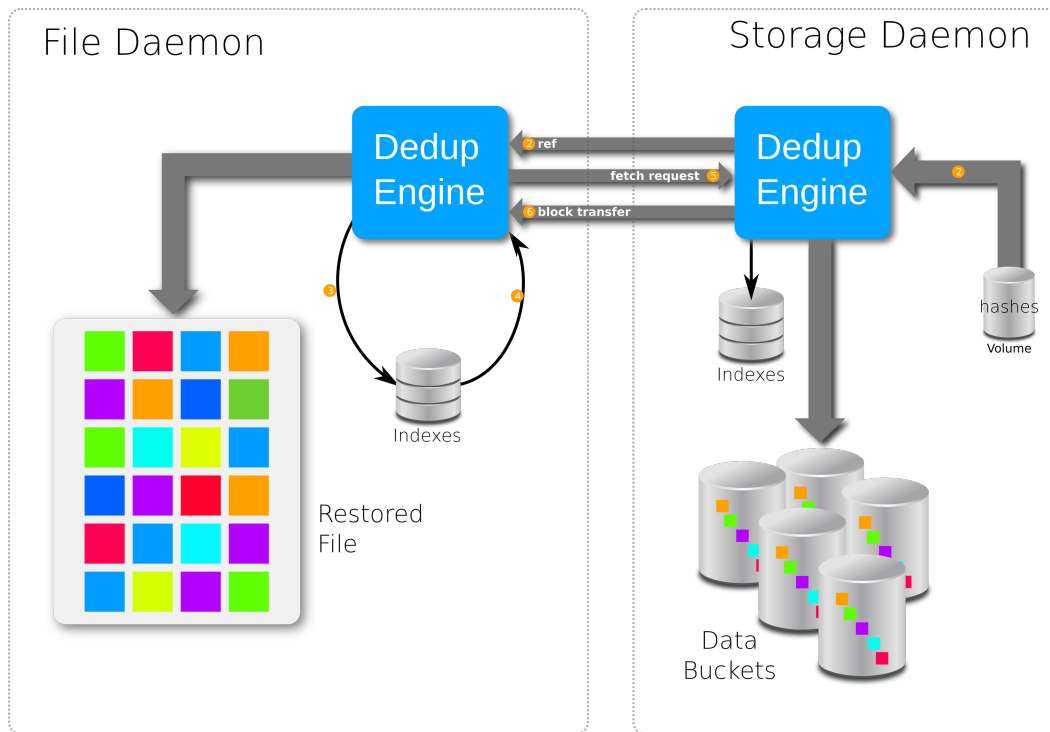


Fig. 2: Restore Scenario when using the directive 'Enable Client Rehydration'

and checks if the chunk at the given offset matches the given hash. If it matches, the File Daemon uses it and does not download the chunk from the Storage Daemon.

It is obvious that to take advantage of this feature, you must:

- Restore the data to another location.
- Have some piece of the original data in the original location.

This feature can be very helpful to retrieve an old version of the current data.

Notice that this feature doesn't work for files that are not going into the Deduplication Engine like small files or when data encryption is used. This also doesn't work when the data is transformed by Bacula before reaching the Deduplication Engine. For example, when compression is used or when backing up Windows systems without the `portable = yes` option in the FileSet.

Unfortunately there is no evidence of the efficiency of the algorithm in the Job report yet. The only evidence is the `read chunk` counter shown by the `dedup usage` command that is not incremented for chunks found on the Client.

### Storage Daemon Deduplication Related Directives

- `Plugin Directory = <directory-path>`  
This directive tells the Storage Daemon where to find plugins. The file `dedup-sd.so` or the `bacula-sd-dedup-driver-x.y.z.so` must be present in this directory before starting the Storage Daemon.
- `Dedup Directory = <directory-path>`  
Deduped chunks will be stored in the **Dedup Directory**. This directory is common for all **Dedup** devices configured on a Storage Daemon and should have a large amount of free space. We advise you to use LVM on Linux Systems to ensure that you can extend the space in this directory. The **Dedup Directory** directive is mandatory. We recommend that you do not change this directory afterward, because if you make a mistake, it

would invalidate all of your backups. If you do change the Dedup Directory directive, the following files must be moved to the new directory:

- \*.blk

- Dedup Index Directory = <directory-path>

Indexes will be stored in the **Dedup Index Directory**. Indexes will have a lot of random update accesses, and will benefit from fast drives such as SSD drives. By default, the **Dedup Index Directory** is set to the **Dedup Directory**.

As with the **Dedup Directory**, we recommend against changing the Dedup Index Directory directive. If you do, the following files and directories must be moved to the new directory:

- \*.idx
- \*.tch
- recovery
- recovery.new

The file `bee_dde.tch.new` is a temporary file used by the `optimize` part of the `vacuum` that remain when the process is interrupted. This file don't need to be moved.

- Maximum Container Size = <size>

No container will be allowed to grow to more than <size> bytes. When this size is reached, a new container file will be created. The default value is zero, meaning there is no limit. This limit is useful when you store your containers on a filesystem that limits the size of the file to a pretty low value.

The number of containers is limited to 511, so we recommend to keep this value unlimited or pretty high, at least 1TB. This value may be modified after the initialization of the `DedupEngine`. If a container is already bigger than the new limit, then no new data will be written to it, but its size will not be reduced. Other containers will comply with the new limit.

- Device Type = Dedup

This directive is required to make the Device write Dedup volumes. Once turned on, Bacula will use references in Volumes and will store data chunks into specific container files.

Once a Device has been defined with a certain Type (such as Dedup, Aligned, File or Tape), it cannot be changed to another Type. If you do so, the Bacula Storage Daemon will not be able to properly mount volumes that were created before the change.

```
# From bacula-sd.conf
Storage {
  Name = my-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Subsys Directory = /opt/bacula/working

  Plugin Directory = /opt/bacula/plugins
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index # Recommended to be on fast local SSD
  ↪storage
  Maximum Container Size = 4TB # Maximum 511 containers can be created, please adapt to
  ↪your need
}

Device {
  Name = "DedupDisk"
```

(continues on next page)

```

Archive Device = /mnt/bacula/dedup/volumes
Media Type = DedupVolume
Device Type = Dedup                # Required
LabelMedia = yes
Random Access = Yes
AutomaticMount = yes
RemovableMedia = no
AlwaysOpen = no
}

```

## Deduplication Related Director Daemon FileSet Directive

Within the Director, the Global Endpoint Deduplication system is enabled with a FileSet Option directive called **Dedup**. Each Include section can have a different behavior depending on your needs.

```

# Use the default dedup option of 'storage' side deduplication
FileSet {
  Name = FS_BASE
  Include {
    Options {
      Dedup = storage
    }
    File = /opt/bacula/etc
  }

  # Do not dedup my encrypted data
  Include {
    Options {
      Dedup = none
    }
    File = /encrypted
  }

  # Minimize the network transfer by using 'bothsides' dedup option
  Include {
    Options {
      Dedup = bothsides
    }
    File = /bigdirectory
  }
}

```

The **Dedup** FileSet option can have the following values:

- **storage** - All the deduplication work is done on the Storage Daemon side if the device type is **dedup**. The File Daemon will send all data to the SD just as it normally would. (Default value)
- **none** - Disable deduplication on both the File Daemon and Storage Daemon.
- **bothsides** - The deduplication work is done on the File Daemon and the Storage Daemon.

### About FileSet Compression

The data stored by the Global Endpoint Deduplication Engine is automatically compressed using the LZ4 algorithm.



Using the `FileSet Compression = LZ0|GZIP` option might reduce the deduplication efficiency, and compressing the data twice consumes extra CPU cycles on the client side. Thus we advise that you do not use client-side GZIP or LZ0 compression when using a Dedup Device. To prevent such an inefficient configuration, we recommend setting the `Allow Compression` directive in a Director Storage resource to `No`:

```
# cat bacula-dir.conf
...
Storage {
    Name = Dedup
    Allow Compression = No    # Disable FileSet Compression
                             # option automatically

    Address = baculasd.lan
    Password = xxx
    Media Type = DedupMedia
    ...
}
```

### Deduplication Related File Daemon Directive

The `Enable Client Rehydration FileDaemon` directive is optional and allows Bacula to try to do rehydration using existing local data, see section `clientsiderehydration`. The valid values are `Yes` or `No`. The default is `No`.

Starting with 8.2.0, the `FileDaemon Dedup Index Directory` in `bacula-fd.conf` directive is deprecated and replaced by `Enable Client Rehydration` directive.

```
# cat /opt/bacula/etc/bacula-fd.conf
FileDaemon {
...
    Enable Client Rehydration = yes
}
```

### Things to Know About Bacula

- You must pay particular attention to define a unique `Media Type` for devices that are `Dedup` as well as for each `Virtual Autochanger` that uses a different `Archive Device` directory. If you use the same `Media Type` for a `Dedup` device type as for a normal disk `Volume`, you run the risk that you will have data corruption on disk `Volumes` that are used on `Dedup` and non-`Dedup` devices.
- `Dedup` devices are compatible with Bacula's `Virtual Disk Changers`
- We strongly recommend that you not use the Bacula `disk-changer` script, because it was written only for testing purposes. Instead of using `disk-changer`, we recommend using the `Virtual Disk Changer` feature of Bacula, for which there is a specific white paper.
- We strongly recommend that you update all `File Daemons` that are used to write data into `Dedup Volumes`. It is not required, but old `File Daemons` do not support the newer `FD to SD` protocol, and consequently the `Global Endpoint Deduplication` cannot not be done on the `FD` side.
- The `immutable` flag is compatible with `dedup volumes`, see more details in `Volume Protection Enhancements` and `Volume Protection`.

## Deduplication Engine Vacuum

Over time, you will normally delete files from your system, and in doing so, it may happen that there will be chunks that are stored in dedup containers that are no longer referenced.

In order to reclaim these unused chunks in containers, the administrator needs to schedule a `vacuum` option of the `dedup` command. The `vacuum` option will analyze dedup volumes and mark any chunks that are not referenced as free, thus allowing the disk space to be reused. The `vacuum` command can run while other jobs are running.

```
* dedup
Dedup Engine choice:
  1: Vacuum data files
  2: Cancel running vacuum
  3: Display data files usage
Select action to perform on Dedup Engine (1-3): 1
The defined Storage resources are:
  1: File1
  2: Dedup
Select Storage resource (1-2): 2
Connecting to Storage daemon Dedup at localhost:9103 ...
3000 Found 1 volumes to scan for MediaType=DedupMedia
Ready to read from volume "Vol1" on dedup data device "Dedup-Dev1" (/mnt/bacula/dedup/
↪volumes).
End of Volume at file 0 on device "Dedup-Dev1" (/mnt/bacula/dedup/volumes), Volume "Vol1"
Ready to read from volume "Vol2" on dedup data device "Dedup-Dev1" (/mnt/bacula/dedup/
↪volumes).
End of Volume at file 0 on device "Dedup-Dev1" (/mnt/bacula/dedup/volumes), Volume "Vol2"
Ready to read from volume "Vol3" on dedup data device "Dedup-Dev1" (/mnt/bacula/dedup/
↪volumes).
End of Volume at file 0 on device "Dedup-Dev1" (/mnt/bacula/dedup/volumes), Volume "Vol3"
End of all volumes.
Vacuum cleaning up index.
Vacuum done.
```

## Deduplication Engine Status

Is it possible to query the Deduplication Engine to get some information and statistics. Note that the current interface is oriented toward developers and is subject to change. For example, the `Stats` counters can be reset to estimate the work done by the engine for one job or for one period of time. Here is an example output of the `dedup usage` command, followed by an explanation of each section in the output:

```
* dedup storage=Dedup usage
Dedupengine status:
DDE: hash_count=1275 ref_count=1276 ref_size=78.09 MB
    ref_ratio=1.00 size_ratio=1.13 dde_errors=0
Config: bnum=1179641 bmin=33554393 bmax=335544320 mlock_strategy=1
    mlocked=9MB mlock_max=0MB
Containers: chunk_allocated=3469 chunk_used=1275
    disk_space_allocated=101.2 MB disk_space_used=68.87 MB
    containers_errors=0
Vacuum: last_run="06-Nov-14 13:28" duration=1s ref_count=1276
    ref_size=78.09 MB vacuum_errors=0 orphan_addr=16
Stats: read_chunk=4285 query_hash=7591 new_hash=3469 calc_hash=3470
```

(continues on next page)

(continued from previous page)

[1]	filesize=40.88KB/499.6KB	usage=36/484/524288	7%	***.....
[2]	filesize=40.13KB/589.0KB	usage=18/286/524288	6%	**5.....
[3]	filesize=25.47KB/655.2KB	usage=7/212/524288	3%	*4.....
...				
[64]	filesize=4.096KB/4.096KB	usage=0/0/524288	0%	.....
[65]	filesize=53.25MB/63.90MB	usage=800/960/524288	83%	.....3*****

**DDE:**

- **hash\_count** Number of hashes in the Index.
- **ref\_count** Number of references in all the Volumes.
- **ref\_size** The total of all rehydrated references in all the volumes. This is the size that would be needed if deduplication was not in use.
- **ref\_ratio** The ratio between **ref\_count** and **hash\_count**.
- **size\_ratio** The ratio between **ref\_size** and **disk\_space\_used**.
- **dde\_error** The number of invalid data found in the Index.

**Config:**

- **bnum** The capacity of the hash table in the Index. This is the number of buckets in the Tokyo Cabinet hash database.
- **bmin** The minimum size of the hash table in the Index. Bacula will not go below this value when resizing the Index.
- **bmax** The maximum size of the hash table in the Index. Bacula will not go above this value when resizing the Index. Zero means no limit.
- **mlock\_strategy** This is the strategy to apply to lock only the hash table or the hash table and Index into memory.
  - 0 Do not lock any memory.
  - 1 Use at most **mlock\_max** bytes to lock only the hash table of the Index.
  - 2 Use at most **mlock\_max** bytes to lock all the Index.
- **mlocked** The current number of bytes locked by the Index.
- **mlock\_max** The maximum number of bytes that the Index can lock.

**Containers:**

- **chunk\_allocated** The number of chunks allocated in all containers.
- **chunk\_used** The number of chunks that are really in use.
- **disk\_space\_allocated** The space allocated for all containers.
- **disk\_space\_used** The space that is really used inside all containers.
- **containers\_error** The number of errors related to the containers.

**Vacuum:**

- **last\_run** The date of the last vacuum.
- **duration** The time the vacuum took to complete.
- **ref\_count** Number of references handled by last vacuum.



## Disaster Recovery

### Catalog

The Catalog doesn't contain any reference to the deduplication engine. However, the dedup volumes' records in the Catalog are used during the vacuum process. For this reason, you must make sure to have the Catalog properly restored before starting a dedup vacuum process.

### Volumes

If a dedup Volume is in the Catalog but not on disk, a dedup vacuum process will stop and report an error.

### Index

The Index is essential for the deduplication engine. In the case of a disaster, contact Bacula Systems Support Team.

### Free Space Map (FSM)

The deduplication engine creates a copy (during a commit) of the FSM after every important operation in the recovery sub-directory. When the deduplication engine is not shut down properly, the last copy is used as a reference by the recovery procedure to remove any operations that started after the time of the last commit and that could be incomplete. When the original and the copy of the FSM are lost, it is still possible to rebuild the FSM using references found in volumes. See section `dderecoverytools`.

### Containers

Containers hold chunks of data. When a container (or part of a container) file is lost, the data is lost and it is not recoverable by Bacula. Use the deduplication engine recovery tools (`dderecoverytools`) to identify chunks of data that are lost and restore the deduplication engine consistency.

## 1.3 Dedupengine

The deduplication engine is the heart of Bacula's Global Endpoint Deduplication. It has to store, to index and to retrieve the chunks. All chunks are stored in the `Container Area` and registered in the `Index`. The `Index` is the bottleneck of the deduplication process because all operations need to access it randomly, and very quickly. Memory caching and storing the `Index` on SSD drives will help to maintain good performance.

The Deduplication Index maintains all the hashes of all chunks stored in the Dedup Containers. To get effective performance very fast low latency storage is critical. For large back up data it is best to have the Containers and Deduplication Index on the same hardware server with the Deduplication Index on solid-state drives (SSDs). The faster the disk performance, the faster and more efficient the deduplication process and the data protection will be. In production environments it is best to avoid configurations which introduce latency and delays in the storage infrastructure for the Deduplication Index. It is therefore best to avoid spinning disks, remote access configurations like NFS or CIFS and virtualized SDs. These can be acceptable for small containers (1-2TB) or to perform tests but will normally not provide acceptable performance in larger production environments.

### Sizing the Index

The size of the index depends on the number of chunks that you want to store in the deduplication engine. An upper limit would be 1 chunk per file plus 1 chunk per 64K block of data.

$$\text{number\_of\_chunks} = \text{number\_of\_files} + \frac{\text{data\_amount}}{64\text{K}}$$

If all you have is the storage capacity of your Storage Daemon and want to maximize it, you must know the average compressed size of the chunks you expect to store in Containers. If you don't know the size, you may use 16K.

$$\text{number\_of\_chunks} = \frac{\text{storage\_capacity}}{16\text{K}}$$

When you know the number of chunks, you can calculate the size of your index.

$$\text{index\_size} = 1.3 * \text{number\_of\_chunk} * (8 + 70)$$

The index can be split into two parts: the table and the records.

$$\text{index\_size} = \text{table\_size} + \text{record\_size}$$

$$\text{table\_size} = 1.3 * \text{number\_of\_chunk} * 8$$

$$\text{record\_size} = 1.3 * \text{number\_of\_chunk} * 70$$

The table part is small and is accessed by all operations. The record part is bigger and is sometimes not used for read operations.

Table 1: Samples of Index size for chosen Storage sizes

Storage size	Index size	Table part	Record part
<b>1 TB</b>	6.3 GB	0.65 GB	5.7 GB
<b>10 TB</b>	63.3 GB	6.5 GB	56.9 GB

For good performance, you must try to lock the entire Index into memory, if this is not possible due to lack of memory resources, keeping at least the hash table in memory is highly recommended.

But these are not the only requirements. Bacula needs some extra space on disk and in memory to optimize and resize the Index. We recommend the following:

- Be sure to have 3 times the `index_size` on an SSD drive for the Index.
- Try to have `index_size+table_size` of RAM for the Index.
- At least be sure to have 2 times the “`table_size`” of RAM for the Index.

### Setting up the Index size

The Index is based on a hash table that by design has a fixed size. A B-Tree structure is used to handle collisions in the hash table. The size of the table is important. If too small, the table will have to handle overflow that will slowdown the Index. If too big, the table will consume space and memory uselessly. The table can be resized online and Bacula takes advantage of the vacuum procedure to optimize the table size when needed. Creating the table at the right size from the start will ensure good performance from the beginning and avoid a reduction in performance. The user can define the minimum and maximum sizes of the table. At the end of the vacuum, if the amount of data to delete is large, or if the size of the table is unbalanced regarding the amount of remaining data, Bacula resizes the table to a size equal to 1.3 time the number of hashes remaining in the Index. This new size will be adjusted to match the minimum and maximum values chosen by the user.

$$bnum\_min < \text{table\_size} * 1.3 < bnum\_max$$

The default values for `bnum_min` is 33,554,432 and 0 for `bnum_max`, meaning that their is no limit. These numbers are the number of chunks that the Index can handle efficiently. A chunk can have a size between 1K to 64K. 16K is a good mean value. This means that the default index range is well suited for a storage space between 1TB and 10TB.

Keep in mind that the size of the index affects the amount of memory required to lock the index in memory.

### Locking the index into memory

The operating system caches data that is used often in memory. Unfortunately the huge amount of data going in and out of the Storage Daemon usually wipes out the Index data from the system cache. The alternative is to force the system to map and lock some parts of the Index into memory.

The user has a choice between 3 strategies:

- 0 nothing is locked into memory
- 1 try to lock the table part of the Index into memory
- 2 try to lock the entire Index into memory

Bacula will not allocate more than the maximum value defined by the user (`mlock_max`) and will check the amount of memory available to not overload the system.

See how to change these variables in section `sectiontuningtheindex`.

## Commands to Tune the Index

Bacula Enterprise 8.2 added 4 new parameters to tune the Index. These parameters are initialized with default values when the Dedupengine is created or when Bacula upgrades the Dedupengine from an older version. These parameters may be modified at any time. They will be saved inside the Dedupengine and will be used during the next vacuum.

The Dedupengine can be tuned by changing some internal variables. To have a good understanding of how the deduplication engine works, be sure to read sections `sectionsizingtheindex` and `sectiontuningtheindex`.

- `bnum_min` The minimum capacity of the hash table in the Index. Bacula will not go below this value when resizing the Index.
- `bnum_max` The maximum capacity of the hash table in the Index. Bacula will not go above this value when resizing the Index. Zero means no limit.
- `mlock_strategy` This is the strategy to lock the Index into memory. You have the choice between 3 strategies:
  - 0 Do not lock any memory.
  - 1 Use at most `mlock_max` bytes to lock only the hash table of the Index into memory. (the default)
  - 2 Use at most `mlock_max` bytes to lock all the Index into memory.
- `mlock_max` The maximum amount of bytes that may be used to lock the Index into memory. Zero means no limit. (the default)

Each of these variables may be modified using the `dedup` command together with the name of the variable. The previous value is displayed for reference.

```
*dedup storage=Dedup bnum_min=33554393
3000 dedupsetvar bnum_min previous value was 33554393
*dedup storage=Dedup bnum_max=33554393
3000 dedupsetvar bnum_max previous value was 0
*dedup storage=Dedup mlock_strategy=1
3000 dedupsetvar mlock_strategy previous value was 1ff
*dedup storage=Dedup mlock_max=4096MB
3000 dedupsetvar mlock_max previous value was 0
```

You can review all of these values at once using the `dedup usage` command. At the top of the output you have the section `Config::`:

```
* dedup storage=Dedup usage
Dedupengine status:
...
Config: bnum=1179641 bmin=33554393 bmax=33554393 mlock_strategy=1
        mlocked=9MB mlock_max=0MB
...
```

See the section `sectiondedupenginestatus` for an explanation of the other variables.

These values will be used during the next vacuum if the Index needs to be optimized. You can force an optimize by adding the option `forceoptimize` to the the `dedup vacuum` command.

```
* dedup storage=Dedup vacuum forceoptimize
```

To force the Dedupengine to use a new `mlock` value without running a `dedup vacuum`, you may use the `dedup tune indexmemory` command.

```
* dedup storage=Dedup tune indexmemory
```

## Punching holes in containers

Some Linux filesystems like XFS and EXT4 have the ability to punch a hole into files. A portion of the file can be marked as unwanted and the associated storage released. Of course when a process writes into such a hole, the filesystem allocates space to this area.

Because the use of this technique can increase fragmentation of the filesystem and contribute to slower performance, it is recommended to avoid it when not needed, even though Bacula does its best to use it in a way that will not significantly impact performance.

The hole punching can happen in two places in the DDE:

1. detect and release large unused areas in containers,
2. prevent the allocation of chunks in these holes and prefer areas that are too small to be converted into holes.

Both of these processes are independent. As soon as you set up a `hole_size`, the DDE tries to allocate space outside of areas that are good candidates for hole creation, even if no holes have been created before.

### Theory: creating holes

Because these holes can be reused by any container or file on the filesystem, this approach contributes to its fragmentation. That is why you must keep the size of these holes large enough to not reduce the performance of the filesystem. It as been shown that reading or writing random blocks of 4MB is done at a speed similar to sequential reads or writes. That is why we recommend setting the `hole_size` to 4MB. Smaller values can increase the work for the filesystem to manage all these small holes, reduce the performance, and make filesystem recovery processes (`fsck`) take longer. Using a higher value would reduce the probability to find such an unused amount of space inside the containers.

The DDE doesn't store the holes that it has created and doesn't use the information stored in the filesystem itself. The DDE creates the holes on top of the previous ones, and the filesystem ignores the requests for areas that are already holes.

The holes are aligned on the `hole_size` boundaries that we call extents. Remember that containers handle chunks of different sizes, and their sizes are not necessarily powers of 2, so they can span extents. Spanning chunks have weird consequences on the holes:

1. A single used chunk spanning two extents will prevent the conversion of these 2 extents into holes.
2. A hole that has free spanning chunks at one or both ends holds more space than the space that has been given back to the filesystem.

### Theory: smart allocating in between holes

As previously stated, if an unused area is big enough, only the part that is aligned on the `hole_size` boundaries will be converted into a hole. This allows some space around these holes that is still allocated by the filesystem and can be used without "consuming" any new space. The DDE will chose to allocate new chunks in these spaces first, even if these areas have not been converted into holes yet because the DDE relies on existing free space and not on holes that have been created in the past.



When all space between holes has been allocated, the system goes back to the sequential allocation strategy and uses space in existing holes and finally allocates space at the end of the file.

### Commands to create and manage holes

Add the `holepunching` option to the `vacuum` command to create the holes at the end of the vacuum procedure. The command in `bconsole` is:

```
* dedup vacuum holepunching storage=<DeviceName>
```

The first time you use the `holepunching` option, the DDE sets the hole size to 4194304 (4MB). The size is stored in the `hole_size` variable and can be modified or initialized before the first use. The option `forceoptimize` can be used together with the `holepunching` option without restriction. The time required to identify and create holes should not require more than 10s per TB.

You can change the `hole_size` to any value that is a power of 2 bigger than 1 MB. There is no upper limit, but values above 32MB are probably excessive. To change the `hole_size`, use the command:

```
* dedup storage=<DeviceName> hole_size=<Size_in_Byte>
```

For example you can chose a smaller value with the aim of releasing more space.

```
*dedup storage=Dedup hole_size=1048576
3000 dedupsetvar hole_size previous value was 4194304
```

This new value will be used the next time the `vacuum` is run with the `holepunching` option. However, this value will be immediately used by the allocation process to avoid using free space that could be released by the next `holepunching` procedure.

You can disable smart allocation by setting the value to zero. Notice that this value will set the default value to 4MB the next time you use the `holepunching` option in the `vacuum` command.

```
*dedup storage=Dedup hole_size=0
3000 dedupsetvar hole_size previous value was 4194304
```

You can review this value using the `dedup usage` command. At the top of the output you have the section `Hole::`:

```
* dedup storage=Dedup usage
Dedupengine status:
...
HolePunching: hole_size=1024 KB
...
```

### Quiesce and Unquiesce

It is possible to quiesce the dedupengine to safely copy its data without shutting down the DDE. The commands `quiesce` and `unquiesce` allow to freeze and unfreeze the DDE.

```
* dedup storage=Dedup quiesce
3900 quiesce successful
* dedup storage=Dedup unquiesce
3900 unquiesce successful
```

---

**Note:** This functionality is available as of version 10.2.

---

When the `quiesce` command is run, all running backups and restores are suspended. If a `scrub` is running, it is paused. If a `vacuum` is running, the `quiesce` waits for the end of the `vacuum` before returning. When the DDE is frozen, you can backup or copy all the data related to the DDE. The data are in a crash-consistent state, this means that after a recovery, the data will be consistent. When the `unquiesce` command is run, all the backups and restores resume from the point where they had previously stopped. A `scrub` continues from where it was interrupted.

## Detect, Report and Repair Dedupengine Inconsistencies

The `dedup vacuum` command provides three options: `checkindex`, `checkmiss` and `checkvolume` to detect, report and possibly repair inconsistencies in the DDE. `checkindex` can be used with the two others. When `checkmiss` and `checkvolume` are used together, `checkmiss` is ignored.

The `checkindex` and `checkvolume` options use a temporary file `chunkdb.tch` that stores the hash for every *suspicious* chunk to save multiple computations.

The three options will log information to the trace file.

### checkindex option of the vacuum command

The option `checkindex` checks the consistency of the Index with itself and the coherence between the Index and the FSM. When multiple entries in the Index address the same chunk in one container (an address collision), the procedure reads the chunk, calculates the hash and deletes all invalid entries from the index. This procedure is executed before reading the volumes, and it iterates through the index twice: Once to detect collisions, and one more time to delete all invalid entries.

The `checkindex` option displays some statistics in the trace file:

```
cleanup_index_addr_duplicate unset2miss=0
cleanup index Phase 1 cnt=703783 badaddr=0 suspect=0 unset=0 2miss=0 miss=0
  (count=703784 err=0 2miss_err_cnt=0)
cleanup index Phase 2 cnt=703783 2miss=0 (count=703784 err=0 2miss_err=0)
```

- `cnt`: the number of data entries in the Index.
- `badaddr`: the number of entries in the Index with a fanciful address that don't match any container or any chunk inside a container.
- `suspect`: the number of colliding addresses that must be checked.
- `unset`: the number of addresses that were unexpectedly marked as free in the FSM and that have been temporary marked as used until the `vacuum` determines if the entry is needed or not.
- `2miss`: the number of new missing entries.
- `miss`: the number of entries that are missing, meaning that there is no matching chunk in the containers. This includes the newly created entries.
- `count`: the number of entries including the meta data ( `cnt + 1` )
- `err`: a counter for uncommon errors.
- `2miss_err`: the number of errors when creating or converting an erroneous entry into a missing one.

A `Scrub` process always starts a `checkindex` as its final action.

### checkmiss and checkvolume options of the vacuum command

The option `checkvolume` is deprecated since the availability of the `Scrub` process. In future releases the `checkvolume` option will be silently replaced by the option `checkmiss`.

These options search the Index for every reference found in the volumes. This can significantly increase the time of the vacuum if the Index doesn't fit into memory. Be sure to check that using the command `dedup storage=Dedup tune indexmemory`.

The option `checkmiss` simply creates dummy entries when a reference is not found in the Index. This entry indicates that the chunk is missing and could be resolved by future backups or by a Scrub. This option is less resource intensive than the `checkvolume` because it doesn't access the containers.

The option `checkvolume` checks the consistency of the Index with every reference found in the volumes. If the hash of the reference is not found in the Index or doesn't match the address, then the chunks at the given addresses are read, the hashes are calculated and the Index is fixed when appropriate. Incorrect entries are converted into `missing` to indicate that some chunks are missing. This option no longer uses the file `orphanaddr.bin`. This file is now deleted after a successful vacuum.

Every mismatch is logged in the `checkvolume` trace file with the coordinate of the file that holds the reference. Only one line is logged per file and per type of mismatch, others are counted in the statistics. Tools that can use this information to exclude the faulty file during a restore (for example) will come later.

The lines in the trace file look like this:

```
bacula-sd: dedupengine.c:4151 VacuumBadRef FixedIndex FI=1 SessId=1
  SessTime=1479138666 : ref(#55fd99e7 addr=0x0016000000000001 size=22254)
  idx_addr=0x0038000000000001
```

Every related line holds the keyword "VacuumBadRef" followed by one second keyword, see below for the details:

- `RefTooSmall`: The record in the volume that holds the reference is too small to hold a reference and is then unusable and not processed further.
- `BadAddr`: The address in the reference looks fanciful and is ignored. The record in the volume may be corrupted.
- `FixedIndex`: One reference has been verified and used to fix the Index. Maybe the Index had no entry for the hash of this reference or had a different address.
- `OrphanRef`: The hash related to this reference doesn't match the related chunk or the one given by the Index if any. This reference is an orphan. The file that holds this reference cannot be fully recovered.
- `RecoverableRef`: The hash related to this reference doesn't match the related chunk, but the Index has a different address that does match the chunk. Then the file can be restored using "`dedup storage=XXXXX rehydra_check_hash=1`" during the time of the restore. The address is written in file `orphanaddr.bin`

The other fields on the line depend on the type:

- `FI`, `SessId` and `SessTime` are the coordinates of the file as written in the Catalog.
- fields inside `ref()` are related to the reference.

At the end, Bacula displays some statistics in the trace file:

```
Vacuum: idxfix=0 2miss=0 orphan=0 recoverable=0
Vacuum: idxupd_err=0 chunk_read=0 chunk_read_err=0 chunkdb_err=0
```

- `idxfix`: The number of entries fixed in the Index. See `FixedIndex` above.
- `orphan`: The number of orphan chunks. See `OrphanRef` above.
- `recoverable`: The number of recoverable chunks. See `RecoverableRef` above.
- `idxfix_err`: The number of errors while trying to fix the entries.
- `chunk_read`: The number of chunks that have been read from disk to verify the hash.
- `chunk_read_err`: The number of errors while reading the chunks.

- `chunkdb_err`: The number of errors while updating the cache that stores the hashes of the block that have been read.

The last three counters are also updated by the “checkindex” option.

### Self Healing

It is possible to enable an option to store all chunks of data to the Deduplication Engine even if the chunks are already stored.

```
dedup storage=Dedup self_healing=1
```

### Container Scrubbing

The Scrub process reads every chunk in every container and compares them with the Index. If an inconsistency is found, the Index is corrected automatically.

Since the container files can be very large, the Scrub process can take days to read everything within them. Bacula jobs (backup, restore, verify, migration, copy, ...) can run while the Scrub process is running. A vacuum process automatically pauses the Scrub process for the duration of the vacuum.

It is recommended to run the Scrub on a regular basis. To minimize the impact of the Scrub process during your backup window, it is possible to control the speed or suspend and resume the process with a `bconsole` command. This may be done manually, or scripted as part of a cron job:

```
$ cat /etc/cron.d/bacula-scrub
BCONS=/opt/bacula/bin/bconsole
LOG=/opt/bacula/working/scrub.log

#M H  DOM M DOW USER  CMD
01 18 * * * bacula echo "dedup scrub suspend storage=Dedup" | $BCONS > $LOG
01  8 * * * bacula echo "dedup scrub resume  storage=Dedup" | $BCONS > $LOG

# a softer solution limiting then bandwidth
#01 18 * * * bacula echo "dedup scrub_bwlimit=10mb/s storage=Dedup" | $BCONS > $LOG
#01  8 * * * bacula echo "dedup scrub_bwlimit=0 storage=Dedup" | $BCONS > $LOG
```

To limit the speed of the Scrub process, you can set the `DedupScrubMaximumBandwidth` directive on the `Storage` resource in the `bacula-sd.conf` file. The default maximum bandwidth value is 50MB/s. This is the total amount that the scrub can use, all the workers, and this amount will not be available for backup jobs.

```
Storage {
    Name
    ...
    DedupScrubMaximumBandwidth = 20MB/s
}
```

This value may be adjusted manually with a `bconsole` command:

```
* dedup scrub_bwlimit=10mb/s
```

Scrubbing is more effective after a “`dedup vacuum checkmiss`”. The `checkmiss` option forces the vacuum to create dummy entries in the Index for every orphan reference found in the volumes. The Scrub process will resolve these dummy entries when it finds a matching chunk. When the Scrub doesn’t find any related entry in the Index, the chunk is marked as free. The `checkvolume` option of the vacuum command also creates dummy entries. See the differences in the vacuum section.

```

$ cat /opt/bacula/scripts/dedup-scrub
#!/bin/sh

SD=Dedup1
LOG=/opt/bacula/working/scrub.log
PATH=$PATH:/opt/bacula/bin

exec 1>> $LOG
exec 2>> $LOG

date
echo "dedup vacuum checkmiss storage=$SD" | bconsole
echo "dedup scrub run storage=$SD" | bconsole
date

```

Scrubbing can be done by multiple threads, each of them handling one container at a time and should be able to reach a throughput up to 400MB/s per CPU core (limited by the SHA512/256 calculation). The Scrub saves its state at regular intervals and can restart from where it has been interrupted. The Scrub process doesn't restart automatically after a restart or a reboot.

The Scrub starts handling the containers that are largest to efficiently balance the work between the threads.

At the end, the Scrub process does a `checkindex` to check the coherence between the Index and the FSM and to detect if an address is used twice or if an entry refers to an empty chunk.

The Scrub process can be controlled from `bconsole` via the `dedup` command:

```

*dedup
Dedup Engine choice:
  1: Vacuum data files
  2: Cancel running vacuum
  3: Display data files usage
  4: Scrub data files options
Select action to perform on Dedup Engine (1-4): 4
Dedup Engine Scrub Process choice:
  1: Run Scrub
  2: Stop Scrub
  3: Suspend Scrub
  4: Resume Scrub
  5: Status Scrub
Select Scrub action to perform on Dedup Engine (1-5):

```

It is possible to run every Scrub sub-command from the command line:

```

* dedup scrub run storage=Dedup
* dedup scrub run worker=3 storage=Dedup
* dedup scrub run reset storage=Dedup

```

The `"dedup scrub run"` command starts the Scrub process. If the Scrub has been interrupted by a crash or a restart of the daemon, the Scrub process will continue from its last saved point. Notice that the Scrub process doesn't continue automatically after a restart or a reboot. The `"worker"` option controls the number of threads, the default is one. The `"reset"` option forces the Scrub to ignore its last saved point and restart from the beginning.

Other Scrub commands available:

```
* dedup scrub wait storage=Dedup
* dedup scrub stop storage=Dedup
* dedup scrub suspend storage=Dedup
* dedup scrub resume storage=Dedup
```

- **wait.** Wait until the end of the Scrub process.
- **stop.** Stop any running threads of the Scrub process.
- **suspend.** Suspend all the threads of the Scrub process.
- **resume.** Resume all the threads of the Scrub process.

“suspend” and “resume” do not modify the options given to the “run” command. “stop” stops the threads and allows a restart of the Scrub process using the “run” command and a different number of threads for example.

Finally you can get the status of last Scrub that has been started.

```
* dedup scrub status storage=Dedup
Scrubber: last_run="10-Aug-2017 12:05:38" started=1 suspended=0 paused=0 quit=0
pos=1225639597 pos=8% bw=44957018/500000000
* dedup scrub wait storage=Dedup
* dedup scrub status storage=Dedup
Scrubber: last_run="10-Aug-2017 12:05:38" started=0 suspended=0 paused=0 quit=1
pos=14453933383 pos=100% bw=3269270/500000000
```

The “status” sub-command tells you if the Scrub process is running, if it has been suspended by the user or paused by the vacuum, the absolute position that is the total of all the bytes that have been read for all the containers, the relative position in percent and also the disk bandwidth in bytes/s compared with what has been allowed by the variable “scrub\_bwlimit”. The position is updated every 10 seconds.

Some useful information is logged to the trace file. The “status” sub-command, displays the status of the Scrub process for each container:

```
...
Scrub status [66] size=327677663 scrub_pos=-1 scrub_start=1502366453
Scrub status [67] size=327677780 scrub_pos=43670 scrub_start=1502367337
Scrub status [68] size=327679152 scrub_pos=0 scrub_start=0
...
Scrub status read_err=0 fix_err=0 feed=0
Scrub status fix miss=0 wrong=0 false_set=0 false_unset=0
```

Here, the Scrub process for container [66] is finished, container [67] is being processed and the Scrub process for container [68] is still pending.

The position is in bytes, and must be compared to the size of the container on the left - also in bytes. The “scrub\_start” is the epoch when the Scrub started handling the container. The counters at the end of the output show the current general statistics:

- **read\_err** is the number of chunks that were unreadable from the disk, or corrupted and finally ignored. As Holes are not yet skipped by the Scrub process, chunks in these areas will increment this counter.
- **fix\_err** is the number of errors encountered when trying to fix an existing error
- **feed** is used internally and only relevant to our developers. It should always be 0.
- **miss** is the number of entries in the Index that were missing and have been resolved by the Scrub process.
- **wrong** is the number of entries in the Index that were pointing to the wrong chunk and that have been fixed by the Scrub process.

- `false_set` is the number of chunks that were incorrectly marked as used, but were not required by the Index and were then marked as free.
- `false_unset` is the number of chunks that were incorrectly marked as free, but required by the Index and were then marked as used.

When the Scrub process finishes a container, it logs the statistics for this container in the trace file:

```
ScrubContainer [106] end pos=-1 fatal=0 read_err=0 fix_err=0 feed=0
ScrubContainer [106] fix miss=0 wrong=0 false_set=0 false_unset=0
```

At the end, the Scrub process performs a cleanup identical to the cleanup done by the vacuum “`checkindex`” command and finally displays consolidated statistics for all of the containers.

```
cleanup_index_addr_duplicate unset2miss=1
cleanup index Phase 1 cnt=2362298 badaddr=0 suspect=0 unset=0 2miss=0 miss=0
↳(count=2362299 err=0 2miss_err=0)
cleanup index Phase 2 cnt=2362298 2miss=0 (count=2362299 err=0 2miss_err=0)
Scrubber index cleanup chunk_read=0 chunk_read_err=0 chunkdb_err=0
ScrubContainer END read_err=0 fix_err=0 feed=0 cleanup=OK
ScrubContainer FIX miss=0 wrong=0 false_set=0 false_unset=0
```

## 1.4 Hardware Requirements

### CPU

Bacula’s Global Endpoint Deduplication consumes CPU resources on both File Daemon and Storage Daemon. The table *below* shows operations done by both daemons depending on the deduplication mode.

Note that the Storage Daemon has to re-calculate hashes of the chunks sent by the File Daemon to ensure the validity of the data added to the Dedupengine.

Table 2: Operations done by each daemon

	<b>Dedup=none</b>	<b>Dedup=storage</b>	<b>Dedup=bothside</b>
<b>Client</b>	–	–	hash + compress
<b>Storage</b>	–	hash + compress + DDE	decompress + hash + DDE

On recent Intel processors, compression and hash calculations each require about 1GHz of CPU power per 100MB/sec (1Gbps). Decompression requires only 0.25GHz for the same throughput. The Dedupengine depends more on IOPs rather than on CPU power (about 0.1GHz per 100MB/sec). Both daemons must also handle network and disks (around 1GHz per 100MB/sec).

The rules of thumb might be to dedicate 3GHz per 100MB/s for the File Daemon or the Storage Daemon when doing deduplication.

Table 3: CPU requirements (Intel based)

	<b>100MB/sec (Gbps)</b>	<b>400MB/sec (4Gbps)</b>	<b>1000MB/s (10Gbps)</b>
<b>Client or storage</b>	3GHz	12GHz	30GHz

Add about 50% more GHz for latest generation of AMD processors.

## Memory

The File Daemon requires additional RAM to do bothsides deduplication because it has to keep the chunks in memory until the Storage Daemon sends a command to send or to drop a chunk of data. The extra memory required is about 4MB per running job.

The Storage Daemon also requires about 4MB of memory for every running job. The Dedupengine also needs more of memory for efficient lookups in the index file, see section sectiondedupengine

## Disks

On the File Daemon, the directive `Enable Client Rehydration = yes` can generate some extra reads during the restore process and increase the disk load and possibly slow down the job.

On the Storage Daemon, chunks are stored randomly in Containers, and the disk systems might have to do significantly more random I/O during backup and restore. Note that migration/copy and virtual full Jobs do not need to rehydrate data if the destination device supports deduplication. Chunks are stored in 65 or more container files in the `Dedup Directory`. All Volumes use references to those container files. This means that your system must be configured to manage disk space and extend disk space if necessary. We advise you to use LVM, ZFS, or BTRFS.

For effective performance, it is strongly recommended to store the deduplication engine Index on dedicated solid state storage, NVMe or SSDs. Please see section sectiondedupengine. It is not recommended to store deduplication engine containers on the same file systems the Catalog database resides on.

The index file used to associate SHA512/256 digests with data chunk addresses will be constantly accessed and updated for each chunk backed up. Using solid state storage for the index file will give better performance. The number of I/O operations per second that can be achieved will limit the global performance of the deduplication engine. For example, if your disk system can do 10,000 operations per second, it means that your Storage Daemon will be able to write between 5,000 and 10,000 blocks per second to your data disks. (310 MB/sec to 620 MB/sec with 64 KB block size, 5 MB/sec to 10 MB/sec with 1 KB block size). The index is shared between all concurrent Jobs.

To ensure thatfile systems required for containers, index, and volumes are mounted before the Storage Daemon starts, you can edit the `bacula-sd.service` unit file

```
# systemctl edit bacula-sd.service
```

This will create the file `/etc/systemd/system/bacula-sd.service.d/override.conf` to add `bacula-sd.service` customized settings. Please add the following line to the file and save it:

```
RequiresMountsFor=/bacula/dedup/index /bacula/dedup/containers /bacula/dedup/volumes
```

Of course, paths may need to be adjusted per your actual configuration.

## 1.5 Installation

The recommended way to install deduplication plugin is using BIM, where the deduplication plugin installation can happen alongside the installation of Storage Daemon, at the point of choosing the plugin.



## Linux

To install deduplication plugin for Linux, visit [Linux: Install Storage Daemon](#) and, in step 5, choose the dedup plugin. If you have already installed SD, run the installation again and choose the dedup plugin.

---

**Important:** While going through the installation steps again, your configuration file will not be overwritten.

---

## 1.6 Restrictions and Limitations

- You must take care to define unique Media Types for Dedup Volumes that are different from Media Types for non-Dedup Volumes.
- The “hole punching” feature is available on Linux systems with kernel 2.6.37 and later. The function was also backported by Redhat to their 2.6.32 kernel (on Redhat 6.7). The feature is not available on FreeBSD or Solaris OSes.
- Some files are not good candidates for deduplication. For example, a mail server using maildir format will have a lot of small files, and even if one email was sent to multiple users, SMTP headers will probably interfere with the deduplication process. Small files will tend to enlarge your chunk index file resulting in a poor dedup ratio. A good dedup ratio of 20 for a file of 1024 bytes will save only 19 KB of storage, so much less gain than with a file of 64 KB with a poor dedup ratio of 2.
- Dedup Volumes cannot just be copied for offsite storage. Dedup Volumes should stay where the deduplication engine is running. In order to do offsite copies, it is recommended to run a Copy Job using a second Dedup Storage Daemon for example, or to create rehydrated volumes with a Copy/Migration/Virtual Full job using a regular disk Device. The VirtualFull support was added in version 8.0.7. The Storage Daemon to Storage Daemon Copy/Migration process with deduplication protocol was added in version 8.2.0.
- A Virtual Full between two Storage Daemons is currently not supported.
- Data spooling cannot be used with deduplication. Since versions 8.2.12 and 8.4.10, data spooling is automatically disabled whenever a device resource is configured with **Device Type = Dedup**.
- All Bacula Enterprise File Daemons (including Linux, FreeBSD, Solaris, Windows, ...) support the Global Endpoint Deduplication. The Community Bacula File Daemon supports only the Global Endpoint Deduplication in `dedup=storage` mode. The list of the platforms supported by Bacula Enterprise is available on [www.baculasystems.com](http://www.baculasystems.com).
- We strongly recommend that you update all File Daemons that are used to write data into Dedup Volumes. It is not required, but old File Daemons do not support the newer FD to SD protocol, and consequently the Global Endpoint Deduplication will be done only on the Storage daemon side.

## 1.7 Best Practices

### RAID

If you plan to use RAID for performance and redundancy, please note that read and write performances are essential for the deduplication engine. The Index is highly accessed for reading and for writing during backup jobs run and during the maintenance tasks required by the deduplication plugin. Also, the `forceoptimize` process that rebuilds the Index strongly depends on the read and write performance of the disk infrastructure.

Some RAID solutions don't fit the deduplication engine read and write performance requirements. RAID 10 is recommended for both the dedup index and dedup containers as it provides both redundancy of performance better than RAID\_1. Please note that many RAID solutions have better performance than RAID\_5, thus RAID\_5 should be avoided if possible.

## ZFS

If you plan to use ZFS file system to store the dedup index, it is important to guarantee that you have enough memory and CPU resources in the Storage Daemon server to have both the deduplication plugin and ZFS in good condition.

The Global Endpoint Deduplication plugin does both deduplication and chunk compression. This means there is no need to enable deduplication or compression in the zfs pool that will be used to store containers. In fact, it is not recommended to have them enabled as it may cause slow performance and there will be no gain in space savings.

Aligned disk access is a key factor when using ZFS. ZFS is able to detect the sector size of the drive, but disks can report the emulated value instead. As we recommend solid state storage for the dedup Index, performance can be improved if the `ashift` property is explicitly set to match the sector size of the underlying storage, which will often be `4096_bytes`.

The disk block size is defined by the `ashift` value, but as ZFS stores data in records, there is another parameter that determines the individual dataset to be written in the ZFS pool, this is the `recordsize` ZFS pool parameter.

Thus another important ZFS pool setting to consider is the `recordsize` value. It defaults to 128K in recent ZFS versions. This value should be adjusted to match the typical I/O operations. For the ZFS pool used by the dedup Index, it was reported that setting the `recordsize` to 8K increased the vacuum performance. In addition, setting the ZFS kernel `zfs_vdev_aggregation_limit_non_rotating` parameter to the same value as `recordsize` highly improved performance.

Please note these values are recommended for most SSD disks, but they may vary depending on the SSD model. For example, 16K could fit some SSD models and give a better performance. We recommend to perform I/O benchmark using different settings before the deduplication engine is setup in production.

Regarding the use of ZFS to store dedup containers, as it is not possible to preview the typical dataset because some containers can be much more used than others, it is more probable that a `recordsize` value of 64K or even the 128K default value are sufficient to have a good performance. However, it is important to not allow container files to grow too much and limit the size of containers files to 3TB or 4TB.

## Maximum Container Size

For better performance, it is strongly recommended to not allow container files to grow indefinitely even if the underlying file system supports very big files. This can be accomplished by setting the “Maximum Container Size” directive in the Storage resource in the Storage Daemon configuration file. It is recommended to set this directive to a value between 1 TB and 4 TB.

This setting is also recommended because container files cannot be shrunk. Once they grow, it is not possible to have these files reduced in size. The holepunching process will not reduce the container file sizes by shrinking them.

## Vacuum and Scrub

It is strongly recommended to keep the deduplication engine healthy by regularly performing the maintenance tasks triggered by the vacuum and scrub processes.

Please make sure to regularly run, in all deduplication engines in your environment, the following tasks: daily pruning and truncation of volumes, a simple vacuum daily (it can be run weekly when not too many new chunks are added to the deduplication engine), a dedup vacuum checkindex checkmisss monthly and the scrub process preferably outside of the backup time windows.

These maintenance tasks can be scheduled in an Admin Job and they will help to clean both the deduplication engine index and containers, marking unused entries in the index and chunks in containers, thus allowing capacity reuse. They will also contribute to avoid invalid index entries to be used by backup jobs in case of any problems with the server hosting the deduplication engine.

## Holepunching

The holepunching process allows you to mark unused chunks in container files as free after a successful vacuum is run. This process will punch a hole in the file and it is required that the underlying file system support holes. We do recommend to use file systems that support hole punching, such as xfs and ext4.

It is important to note that the released amount of space will depend on the I/O block size of the underlying file system. This means that, if you have a file system configured with block size of 4 MB, only entire blocks of 4 MB will be released to the system to be used by either container files or other files in the file system.

This is the version available to our customers for a few years.

## 2 Global Endpoint Deduplication 2

- *Executive Summary*
- *Deduplication*
- *Dedupengine*
- *Hardware Requirements*
- *Installation*
- *Restrictions and Limitations*
- *Best Practices*

### 2.1 Executive Summary

IT organizations are constantly being challenged to deliver high quality solutions with reduced total cost of ownership. One of those challenges is the growing amount of data to be backed up, together with limited time to run backup jobs (backup window). Bacula Enterprise offers several ways to tackle these challenges, one of them being *Global Endpoint Deduplication 2*, which minimizes the network transfer and Bacula Volume size using deduplication technology.

This document is intended to provide insight into the considerations and processes required to successfully implement this innovative backup technique.

### 2.2 Deduplication

Deduplication is a complex subject. Generally speaking, it detects that data being backed up (usually chunks) has already been stored and rather than making an additional backup copy of the same data, the deduplication software keeps a pointer referencing the previously stored data (chunk). Detecting that a chunk has already been stored is done by computing a hash code (also known as signature or fingerprint) of the chunk, and comparing the hash code with those of chunks already stored.

The picture becomes much more complicated when one considers where the deduplication is done. It can either be done on the server and/or on the client machines. In addition, most deduplication is done on a block by block basis, with some deduplication systems permitting variable length blocks and/or blocks that start at arbitrary boundaries (sliding blocks), rather than on specific alignments.

## Advantages of Deduplication

- Deduplication can significantly reduce the disk space needed to store your data. In good cases, it may reduce disk space needed by half, and in the best cases, it may reduce disk space needed by a factor of 10 or 20.
- Deduplication is combined with compression to further reduce the storage space needed. Compression depends on data type and deduplication depends on the data usage (on the need or the will of the user to keep multiple copies or versions of the same or similar data). Bacula takes advantage that both techniques work perfectly together and combines them in its original Dedup Engine.
- Deduplication can significantly reduce the network bandwidth required because both ends can exchange references instead of the actual data itself. It works when the destination already has a copy of the original chunks.
- Handling references instead of the data can speed up most of the processing inside the Storage Daemon. For example, Bacula features like copy/migrate and Virtual Full can be up to 1,000 times faster. See the following article for more information on this subject.

## Cautions About Using Deduplication

Here are a few of the things that you should be aware of before using deduplication techniques.

- To do efficient and fast deduplication, the Storage Daemon will need additional CPU power (to compute hash codes and do compression), as well as additional RAM (for fast hash code lookups). Bacula Systems can help you to calculate memory needs.
- For effective performance, the deduplication Index should be stored on SSDs as the index will have many random accesses and many updates.
- Due the extra complexity of deduplication, performance tuning is more complicated.
- We recommend Index and Containers are stored in xfs or ext4 file systems. But we are also familiar with the zfs file system.
- Deduplication collisions can cause data corruption. This is more likely to happen if the deduplicating system uses a weak hash code such as MD5 or Fletcher. The problem of hash code collisions is mitigated in Bacula by using a strong hash code (SHA512/256).
- Deduplication is not implemented on tape devices. It works only with disk-based backups.
- Deduplication 2 is compatible with cloud based infrastructure taking into consideration the virtual hardware specification required.
- The immutable flag is not compatible or does not apply to the dedup index or dedup containers.

## Global Endpoint Deduplication

Bacula Systems' latest step in deduplication technology is to offer the *Global Endpoint Deduplication* feature. With Global Endpoint Deduplication, Bacula will analyze data at the block level, then Bacula will store only new chunks in the deduplication engine, and use references in standard Bacula volumes to chunks stored in the deduplication engine. The deduplication can take place at the File Daemon side (saving network and storage resources), and/or at the Storage Daemon side (saving storage resources).

The remainder of this white paper will discuss only Global Endpoint Deduplication.

## How Bacula Global Endpoint Deduplication Works

- First, be aware that you need the Dedup bacula-sd-dedup-driver-x.y.z.so Storage Daemon plugin for Global Endpoint Deduplication to work. This plugin holds both *Legacy* and *Dedup2* drivers.
- Dedup2 includes new directives that allows to have both Legacy and Dedup2 engine types in the same Storage Daemon. The goal of these new directives is to support multiple Dedup Engine in the same Storage Daemon. This will allow to handle smooth migration between two Dedup of different generations. To accomplish that, you must modify your current dedup engine to use the new directives and setting “Driver=Legacy”.
- Dedup devices are enabled by specifying the Dedup keyword as a DeviceType directive in each disk Device resource in the bacula-sd.conf where you want to use deduplicated Volumes. If you have multiple Dedud Engine you must also specify the name that you have defined in the Dedup Engine.

```
Device {
  Name = DiskDedup_Dev0
  . . .
  Device Type = Dedup
  Dedupengine = Dedupengine_name_with_dedup2
}
```

- You must pay particular attention to define a unique Media Type for devices that are Dedup as well as for each Virtual Autochanger that uses a different Archive Device directory. If you use the same Media Type for a Dedup device type as for a normal disk Volume, you run the risk that you will have data corruption on disk Volumes that are used on Dedup and non-Dedup devices.
- When Global Endpoint Deduplication is enabled, the Device will fill in disk volumes with chunk references instead of the chunks. Bacula encrypted data, and very small files will be stored in the Volumes as usual. The deduplicated chunks are stored in the “Containers” of the Dedupengine, and are shared by all other dedup-aware devices (related to the same Dedup LegacyDedupEngine) in the same Storage Daemon.
- We advise to set a limit on the number of Jobs or the usage duration when working with dedup Volumes. In case you prefer to use Maximum Volume Bytes, consider that two Catalog fields are considered when computing the volume size. VolBytes represents the volume size on disk and VolaBytes considers the amount of non-dedup data stored in the volumes, i.e., the rehydrated data. If the directive Maximum Volume Bytes is used for a dedup Volume, Bacula will consider both VolBytes and VolaBytes values to check the limits.

## Global Endpoint Deduplication During Backup Jobs

- When starting a Backup Job, the Storage Daemon will inform the File Daemon that the Device used for the Job can accept dedup data.
- If the FileSet uses the dedup = *bothsides* option, the File Daemon will compute a strong hash code for each chunk and send *references* including these hashes to the Storage Daemon which will request the original chunks from the File Daemon for any of them that the Dedupengine is unable to resolve.
- If the FileSet uses the dedup = *storage* option, the File Daemon will send data as usual to the Storage Daemon, and the Storage Daemon will compute hash codes and store chunks in the Dedupengine and the references in the disk volume.
- If the FileSet uses the dedup = *none* option, the File Daemon will send data as usual to the Storage Daemon, and the Storage Daemon will store the chunks in the Volume without performing any deduplication functions.
- If the File Daemon doesn’t support Global Endpoint Deduplication, the deduplication will be done on the Storage side if the Device is configured with DeviceType = dedup.

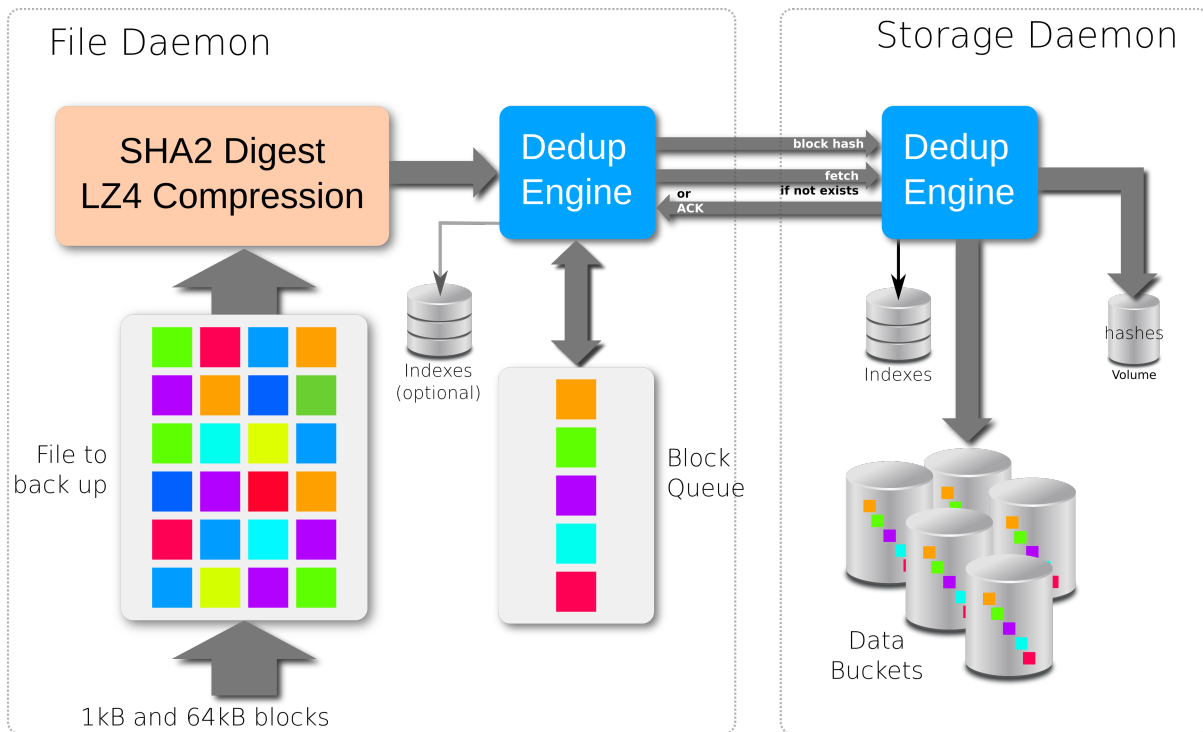


Fig. 3: Backup Scenario with bothsides deduplication

### New Storage Daemon Directives

Remember to specify the directive Plugin Directory.

- Plugin Directory = <directory-path>  
This directive tells the Storage Daemon where to find plugins. The file bacula-sd-dedup-driver-x.y.z.so must be present in this directory before starting the Storage Daemon.

### New Dedupengine Resource

As said earlier, it is possible to have both the legacy and the new *Dedup2* engines in the same Storage Daemon. A new format has been introduced to allow it. The new resource *Dedupengine* will allow to have your current legacy dedup engine and a new *Dedup2* engine in the same Storage Daemon. All Dedup Devices that refer to the same *DedupEngine* shares the same storage space.

```
Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index
  Driver = Dedup2
  MaximumContainerSize = 2TB
}
```

- Name = <name>  
The name of the DedupEngine that will be used in the Device resource. The Dedup name is required.
- Dedup Directory = <directory-path>

Deduped chunks will be stored in the Dedup Directory. This directory is shared by all Dedup devices configured on a Storage Daemon and should have a large amount of free space. We advise you to use LVM on Linux Systems to ensure that you can extend the space in this directory. The Dedup Dedup Directory directive is mandatory. We recommend that you do not change this directory afterward, because if you make a mistake, it would invalidate all of your backups. If you do change the Dedup Directory directive, the following files must be moved to the new directory:

- beed2-XXXXXXXX.ctn
- beed2.esm0
- beed2.esm1

The .esm0 and .esm1 files hold the state of every extent in each container.

- Dedup Index Directory = <directory-path>

Indexes will be stored in the Dedup Index Directory. Indexes will have a lot of random update accesses, and will benefit from fast drives such as SSD drives. By default, the Dedup Index Directory is set to the Dedup Directory.

As with the Dedup Directory, we recommend against changing the Dedup Index Directory directive. If you do, the following files and directories must be moved to the new directory:

- beeindex.tch
- beeindex.tch.new

The file `beeindex.tch.new` is a temporary file created by the `optimize` part of the vacuum that remain when the process is interrupted. This file holds a new version of the index. At startup the DedupEngine rename this file into `beeindex.tch` and use this new file. If this `beeindex.tch.new` file exists you can just copy it as is and ignore `beeindex.tch`

- Maximum Container Size = <size>

No container will be allowed to grow to more than <size> bytes. When this size is reached, a new container file will be created. The default value is 100GiB. This value is used at initialization time and cannot be modified later, any change would be ignored. This limit is useful when you store your containers on a filesystem that limits the size of the file to a pretty low value. For now, the number of containers is limited to 8192 but we recommend to keep the number below 1024. This directive should be used to limit the size of container files to any limit from the file system where containers are stored. Dedup2 allows up to 100 million container files.

- Maximum Container Open = <count>

This is the maximum number of containers that can be simultaneously open. When *Maximum Container Size* has been setup to a small size, this directive force Bacula to close unused containers before to open a new one and avoid to have 1000th of simultaneously open files. The default value is 100 GB. The recommended value is 20 + the number of simultaneous backup + the number of simultaneous restore.

- Driver = <driver>

You have the choice between the old Dedup **Legacy** or the new one **Dedup2**.

- MaximumIndexMemorySize = <size>

This is the maximum memory the DedupEngine will allocate for the Index. If unspecified, Bacula uses the available RAM as a starting point, saving enough RAM for the system. When the Index is created the first time, the half of this value or the half of the RAM is used to size the hash table. The initial hash table should not use more than 8GiB, this is about 1 billion entries.

To modify your current legacy Dedup engine to the new configuration format:

- Stop the Storage Daemon
- Modify the current dedup engine settings by creating a DedupEngine resource with Dedup=Legacy and the same index and containers directories of your current dedup engine:

```
Dedupengine {
  Name = Dedupengine_legacy
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index
  Driver = Legacy
  MaximumContainerSize = 2TB
}
```

- Remove the index and containers directories from the Storage resource configuration
- Start the Storage Daemon

## New Device Directives

```
Device {
  Name = FileDedup1
  ...
  Device Type = Dedup
  Dedupengine = Dedupengine_Name_1
}
```

- Device Type = Dedup  
This directive is required to make the Device write Dedup volumes. Once turned on, Bacula will use references in Volumes and will store data chunks into the specified DedupEngine.  
  
Once a Device has been defined with a certain Type (such as Dedup, Aligned, File or Tape), it cannot be changed to another Type. If you do so, the Bacula Storage Daemon will not be able to properly mount volumes that were created before the change.
- Dedupengine = <dedupengine-name>  
This directive defines the DedupEngine where this device will store its chunks.

```
# From bacula-sd.conf
Storage {
  Name = my-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Subsys Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
}

Dedupengine {
  Name = Dedupengine_Name_1
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index # Recommended to be on fast local SSD
↪ storage
  Driver = Dedup2
  Maximum Container Size = 2TB # Try to get a maximum of 1000 containers at most
}

Device {
  Name = "DedupDisk"
  Archive Device = /mnt/bacula/dedup/volumes
```

(continues on next page)



```

Media Type = DedupVolume
Device Type = Dedup                # Required
dedupengine = Dedupengine_Name_1
LabelMedia = yes
Random Access = Yes
AutomaticMount = yes
RemovableMedia = no
AlwaysOpen = no
}

```

## Director Daemon FileSet Directive

Within the Director, the Global Endpoint Deduplication system is enabled with a FileSet Option directive called Dedup. Each Include section can have a different behavior depending on your needs.

```

# Use the default dedup option of 'storage' side deduplication
FileSet {
  Name = FS_BASE
  Include {
    Options {
      Dedup = storage
    }
    File = /opt/bacula/etc
  }

  # Do not dedup my encrypted data
  Include {
    Options {
      Dedup = none
    }
    File = /encrypted
  }

  # Minimize the network transfer by using 'bothsides' dedup option
  Include {
    Options {
      Dedup = bothsides
    }
    File = /bigdirectory
  }
}

```

The Dedup FileSet option can have the following values:

- **Dedup = storage** - All the deduplication work is done on the Storage Daemon side if the device type is Dedup. The File Daemon will send all data to the SD just as it normally would. (Default value)
- **Dedup = none** - Disable deduplication on both the File Daemon and Storage Daemon.
- **Dedup = bothsides** - The deduplication work is done on the File Daemon and the Storage Daemon. This reduce the network traffic between both Daemon as soon as the Storage Daemon already have the same data.

## About FileSet Compression

The data stored by the Global Endpoint Deduplication Engine is automatically compressed using the LZ4 algorithm. Using the `FileSet Compression = LZ0|GZIP` option might reduce the deduplication efficiency, and compressing the data twice consumes extra CPU cycles on the client side. Thus we advise that you do not use client-side GZIP or LZO compression when using a Dedup Device. To prevent such an inefficient configuration, we recommend setting the `Allow Compression` directive in a Director Storage resource “No”:

```
# cat bacula-dir.conf
...
Storage {
  Name = Dedup
  Allow Compression = No      # Disable FileSet Compression
                              # option automatically

  Address = baculasd.lan
  Password = xxx
  Media Type = DedupMedia
  ...
}
```

## Things to Know About Bacula

- You must pay particular attention to define a unique Media Type for devices that are Dedup as well as for each Virtual Autochanger that uses a different Archive Device directory. If you use the same Media Type for a Dedup device type as for a normal disk Volume, you run the risk that you will have data corruption on disk Volumes that are used on Dedup and non-Dedup devices.
- Dedup devices are compatible with Bacula’s Virtual Disk Changers
- We strongly recommend that you not use the `Bacula disk-changer` script, because it was written only for testing purposes. Instead of using `disk-changer`, we recommend using the Virtual Disk Changer feature of Bacula, for which there is a specific white paper.
- We strongly recommend that you update all File Daemons that are used to write data into Dedup Volumes. It is not required, but old File Daemons do not support the newer FD to SD protocol, and consequently the Global Endpoint Deduplication cannot not be done on the FD side.
- The immutable flag is compatible with dedup volumes, see more details in Volume Protection Enhancements and Volume Protection.

## Deduplication Engine Vacuum

Over time, you will normally delete files from your system, and in doing so, it may happen that there will be chunks that are stored in dedup containers that are no longer referenced.

In order to reclaim these unused chunks in containers, the administrator needs to schedule a `vacuum` option of the `dedup` command. The `vacuum` option will analyze dedup volumes and mark as free any chunks that are not referenced, thus allowing the disk space to be reused. The `vacuum` command can run while other jobs are running. During the index optimization backups are temporary suspended and resume just after the optimization.

```
* dedup
Dedup Engine choice:
  1: Vacuum data files
```

(continues on next page)

```

2: Cancel running vacuum
3: Display data files usage
Select action to perform on Dedup Engine (1-3): 1
The defined Storage resources are:
1: File1
2: Dedup
Select Storage resource (1-2): 2
Connecting to Storage daemon File at localhost:9103 ...
dedupvacuum start inventory
Found 1 volumes (4.547 MB) to scan for MediaType=File
Ready to read from volume "TestVolume001" on Dedup device "FileStorage" (/home/bac/
↪workspace2/bee/regress/tmp).
End of Volume "TestVolume001" at addr=4547374 on device "FileStorage" (/home/bac/
↪workspace2/bee/regress/tmp).
Found 0 volumes (0 B) to scan for MediaType=File1
dedupvacuum start index cleanup and container defrag
Vacuum OK.

```

## Deduplication Engine Status

Is it possible to query the Deduplication Engine to get some information and statistics. Here is an example output of the dedup usage command, followed by an explanation of each section in the output:

```

Dedupengine status: name="dedup2-Dedupengine" now="11-Nov-2022 10:07:26"
Index: usage=0% hash_count=928859 index_capacity=1006632947
Storage: usage=5% used=62.80GB free=1.002TB unused=54.30GB
References: count=1881851 size=123.3GB count_ratio=2.03 size_ratio=1.96
Last Vacuum: start="04-Nov-2022 20:16:20" end="04-Nov-2022 20:18:48"
total=0:02:28 inventory=0:00:06 index_cleanup=0:00:12 defrag=0:00:48
volume=5 ref=479198 ref_size=31.40G
suspect=0 orphan=0 2miss=0 miss=0 idx_err=0

```

Dedupengine: General information about the *Dedupengine*

- name This is the name of the Dedupengine
- now This is when the status has been generated in local time

Index: Information about the *Index*

- usage This is the usage of the index in %. A value above 100% means that the hash table is overloaded and that a vacuum should be run to optimize the index and increase the size of the hash table.
- hash\_count This is the number of entries in the index. This is the number of chunk that are indexed.
- index\_capacity This is the size of the hash table of the index

Storage: Information about the storage and the containers

- usage This is usage in % of space used by the *containers* of the Dedupengine. This is the ratio of the used space vs the total amount of space that could be used.
- used This is the amount of space used inside the containers.
- free This is the amount of space that is available for new data. This include the space unused inside the container and the free space in the filesystem.

- **unused** This is amount of space that is unused inside the *containers*. This space will be used first by the Dedupengine before to grow the last *container* or create new ones.

References: Information about the references inside the *Volume* from the last *Vacuum*

- **count** This is the number of references inside all the volumes.
- **size** This is the total amount of data that the volumes refer to.
- **count\_ratio** This is ratio between the number of reference and the number of chunks (*hash\_count*)
- **size\_ratio** This is ratio between the size without deduplication and the size with deduplication. The difference with the **count\_ratio** give and idea of the efficiency of the LZ4 compression.

**Last Vacuum: These information have been generated during the last Vacuum**

- **start** This is the time of the start of the last *vacuum*.
- **end** This is when the last *vacuum* ended.
- **total** This is the total amount of time used by the vacuum in H:m:s
- **inventory** This is the time used read the volumes and make the *inventory* of the chunk that must be kept.
- **index\_cleanup** This is the time to cleanup and optimize the index.
- **defrag** This is the time to reorganize the data inside the containers and release unused extends.
- **volume** This is the number of volume that have been read to create the inventory.
- **ref** This is the number of references found in the volumes.
- **ref\_size** This is the total of all rehydrated references in all the volumes. This is the size that would be needed if deduplication was not in use.
- **suspect** This is the number of references that are missformed. This can come from an unexpected shut-down or a volume corruption. When the *checkmiss* option is used this also count the *orphan* references.
- **orphan** When the *checkmiss* option is used, this is the number of orphan references. Orphan references are reference that don't have a matching entry in the *index* and a matching chunk in the *containers*.
- **zmiss** This is the number of new *missing* entries created in the index by the the last vacuum when the *checkmiss* option is used.
- **miss** This is the number of *missing* entries in the index. This is entries that are known to be needed but that don't have a matching chunk in the container. One missing entry can have multiple related *orphan* references.
- **idx\_err** This is number of errors when creating *missing* entries in the index. If the this counter is greater than zero this can confirm that the old index was suffering of some problem. The value should goes to zero at the next use of the *checkmiss* option

To see more details on the containers usage, you can use the following command:

```
*dedup qcontainer storage=dedup2-autochanger
Connecting to Storage daemon bacula-sd at bacula-sd:9103...
...
dedup device=dedup2-Chgr drive=-1 cmd=qcontainer
container[00] [0-476836] 00% free=476827 used=8 unavailable=1 other=0 file=/mnt/dedup2/
↔containers/beed2-00000000.ctn
container[00] 0.....
```

## Disaster Recovery

### Catalog

The Catalog doesn't contain any reference to the deduplication engine. However, the dedup volumes' records in the Catalog are used during the vacuum process. For this reason, you must make sure to have the Catalog properly restored before starting a dedup vacuum process.

### Volumes

If a dedup Volume is in the Catalog but not on disk, a dedup vacuum process will stop and report an error.

### Index

The Index is essential for the deduplication engine. In the case of a disaster, contact Bacula Systems Support Team.

### Extent Space Map (ESM)

The DedupEngine use two ESM file. One is active the other is ready to receive a new version of the ESM. The active version can be guessed with a field that works like the Rock, Paper Scissor logic. There is no reconstruction procedure for ESM yet.

### Containers

Containers are cut into extents that hold the chunks of data. When a container (or part of a container) file is lost, the data is lost and it is not recoverable by Bacula. There is no specific tool able to identify and update the index with the data that are missing.

## 2.3 Dedupengine

The deduplication engine is the heart of Bacula's Global Endpoint Deduplication. It has to store, to index and to retrieve the chunks. All chunks are stored in the `Container Area` and registered in the `Index`. The `Index` is the bottleneck of the deduplication process because all operations need to access it randomly, and very quickly. Memory caching and storing the `Index` on SSD drives will help to maintain good performance.

The Deduplication Index maintains all the hashes of all chunks stored in the Dedup Containers. To get effective performance very fast low latency storage is critical. For large back up data it is best to have the Containers and Deduplication Index on the same hardware server with the Deduplication Index on solid-state drives (SSDs). The faster the disk performance, the faster and more efficient the deduplication process and the data protection will be. In production environments it is best to avoid configurations which introduce latency and delays in the storage infrastructure for the Deduplication Index. It is therefore best to avoid spinning disks, remote access configurations like NFS or CIFS and virtualized SDs. These can be acceptable for a small amount of data (1-2TB) or to perform tests but will normally not provide acceptable performance in larger production environments.

## Sizing the Index

The size of the index depends on the number of chunks that you want to store in the deduplication engine. An upper limit would be 1 chunk per file plus 1 chunk per 64K block of data.

$$\text{number\_of\_chunks} = \text{number\_of\_files} + \frac{\text{data\_amount}}{64\text{K}}$$

If all you have is the storage capacity of your Storage Daemon and want to maximize it, you must know the average compressed size of the chunks you expect to store in Containers. If you don't know the size, you may use 16K.

$$\text{number\_of\_chunks} = \frac{\text{storage\_capacity}}{16\text{K}}$$

When you know the number of chunks, you can calculate the size of your index.

$$\text{index\_size} = 1.3 * \text{number\_of\_chunk} * (8 + 70)$$

The index can be split into two parts: the hash table and the records.

$$\text{index\_size} = \text{table\_size} + \text{record\_size}$$

$$\text{table\_size} = 1.3 * \text{number\_of\_chunk} * 8$$

$$\text{record\_size} = 1.3 * \text{number\_of\_chunk} * 70$$

The hash table part is small and is accessed for all operations. The record part is bigger.

For good performance, Bacula try to lock the hash table into memory. The OS will keep in memory a part or the whole part of the records in memory depending the the amount of RAM and the size of the Index.

But these are not the only requirements. Bacula needs some extra space on disk and in memory to optimize and resize the Index. We recommend the following:

- Be sure to have 3 times the `index_size` on an SSD drive for the Index.
- Try to have `index_size+table_size` of RAM for the Index.
- At least be sure to have 3 times the “`table_size`” of RAM for the Index.

## Punching holes in containers

New Dedup2 does not need and does not provide any kind of hole punching. The `vacuum` moves the chunks inside the containers to optimize the space. This also improve the backup speed and the restore speed. The `vacuum holepunching` option is then ignored.

## checkmiss and checkvolume options of the vacuum command

In Dedup2 there is no more difference between options `checkvolume` and `checkmiss`.

These options search the Index for every reference found in the volumes. This can significantly increase the time of the vacuum if the Index doesn't fit into memory.

The option `checkmiss` simply creates dummy entries when a reference is not found in the Index. This entry indicates that the chunk is missing and could be resolved by future backups.

Every mismatch is logged in the trace file with the coordinate of the file that holds the reference. Only one line is logged per file and per type of mismatch, others are counted in the statistics.

The lines in the trace file look like this:

```

bacula-sd: dedupengine2.cc:872 VacuumBadRef OrphanRef FI=1729 SessId=1
  SessTime=1654854105 : ref(#1f41101f addr=0x0000000000000003 size=10769)
  idx_addr=0x0000000000000002

```

Every related line holds the keyword “VacuumBadRef” followed by one second keyword, see below for the details:

- **RefTooSmall**: The record in the volume that holds the reference is too small to hold a reference and is then unusable and not processed further.
- **RefWrongSize**: The record in the volume that holds the reference has an unexpected size and is then unusable and not processed further.
- **OrphanRef**: The hash related to this reference was not found in the index or is know by the Index to be missing. This reference is an orphan one. The file that holds this reference cannot be fully recovered.

The other fields on the line depend on the type:

- **FI**, **SessId** and **SessTime** are the coordinates of the file as written in the Catalog.
- **fields inside ref()** are related to the reference.
- **idx\_addr** this is the address in the index. This should always be 0x2

At the end, Bacula displays some statistics in the trace file. these values can be see via the *dedup usage* command and are documented there.

```

Vacuum: volumes=1 ref_count=1975 ref_size=97382548 suspect_ref=1
Vacuum: 2miss=0 orphan=1 idxupd_err=0

```

## 2.4 Hardware Requirements

### CPU

Bacula’s Global Endpoint Deduplication consumes CPU resources on both File Daemon and Storage Daemon. The table 1 shows operations done by both daemons depending on the deduplication mode.

Note that the Storage Daemon has to re-calculate hashes of the chunks sent by the File Daemon to ensure the validity of the data added to the Dedupengine.

Table 4: Operations done by each daemon

	<b>Dedup=none</b>	<b>Dedup=storage</b>	<b>Dedup=bothside</b>
<b>Client</b>	N/A	N/A	hash + compress
<b>Storage</b>	N/A	hash + compress + DDE	decompress + hash + DDE

On recent Intel processors, compression and hash calculations each require about 1GHz of CPU power per 100MB/sec (1Gbps). Decompression requires only 0.25GHz for the same throughput. The Dedupengine depends more on IOPs rather than on CPU power (about 0.1GHz per 100MB/sec). Both daemons must also handle network and disks (around 1GHz per 100MB/sec).

The rules of thumb might be to dedicate 3GHz per 100MB/s for the File Daemon or the Storage Daemon when doing deduplication.

Table 5: CPU requirements (Intel & AMD Xen based)

	<b>100MB/sec (Gbps)</b>	<b>400MB/sec (4Gbps)</b>	<b>1000MB/s (10Gbps)</b>
<b>Client or storage</b>	3GHz	4 cores at 3GHz	10 cores at 3GHz

## Memory

The File Daemon requires additional RAM to do *bothsides* deduplication because it has to keep the chunks in memory until the Storage Daemon sends a command to send or to drop a chunk of data. The extra memory required is about 4MB per running job.

The Storage Daemon also requires about 4MB of memory for every running job. The Dedupengine also needs more of memory for efficient lookups in the index file, see section *#section:dedupengine*

## Disks

On the Storage Daemon, chunks are stored together into *Extent* that are *randomly* stored into Containers free space. The disk systems might have to do a mix of sequential and random I/O during backup and restore. Note that migration/copy and virtual full Jobs do not need to rehydrate data if the destination device is connected to the same dedupengine.

Container files are stored in the Dedup Directory. All Volumes use references to those container files.

For effective performance, it is recommended to store the deduplication engine Index on dedicated SSDs, see section *#section:dedupengine*. It is not recommended to store deduplication engine containers with the Catalog.

The index file used to associate SHA512/256 digests with data chunk addresses will be constantly accessed and updated for each chunk backed up. Using SSD disks to store the index file will give better performance. The number of I/O operations per second that SSD devices can achieve will drive the global performance of the deduplication engine. For example, if your disk system can do 10,000 operations per second, it means that your Storage Daemon will be able to write between 5,000 and 10,000 blocks per second to your data disks. (310 MB/sec to 620 MB/sec with 64 KB block size, 5 MB/sec to 10 MB/sec with 1 KB block size). The index is shared between all concurrent Jobs.

To ensure that a file system required for container, disk, volumes is mounted before the Storage Daemon starts, you can edit the `bacula-sd.service` unit file

```
# systemctl edit bacula-sd.service
```

This will create the file `/etc/systemd/system/bacula-sd.service.d/override.conf` to add `bacula-sd.service` customized settings. Add the following line line to the file and save it:

```
RequiresMountsFor=/bacula/dedup/index /bacula/dedup/containers /bacula/dedup/volumes
```

## 2.5 Installation

The recommended way to install deduplication plugin is using BIM, where the deduplication plugin installation can happen alongside the installation of Storage Daemon, at the point of choosing the plugin.

### Linux

To install deduplication plugin for Linux, visit [Linux: Install Storage Daemon](#) and, in step 5, choose the dedup plugin. If you have already installed SD, run the installation again and choose the dedup plugin.

---

**Important:** While going through the installation steps again, your configuration file will not be overwritten.

---



## 2.6 Restrictions and Limitations

- You must take care to define unique Media Types for Dedup Volumes that are different from Media Types for non-Dedup Volumes.
- Some files are not good candidates for deduplication. For example, a mail server using maildir format will have a lot of small files, and even if one email was sent to multiple users, SMTP headers will probably interfere with the deduplication process. Small files will tend to enlarge your chunk index file resulting in a poor dedup ratio. A good dedup ratio of 20 for a file of 1024 bytes will save only 19 KB of storage, so much less gain than with a file of 64 KB with a poor dedup ratio of 2.
- Dedup Volumes cannot just be copied for offsite storage. Dedup Volumes should stay where the deduplication engine is running. In order to do offsite copies, it is recommended to run a Copy Job using a second Dedup Storage Daemon for example, or to create rehydrated volumes with a Copy/Migration/Virtual Full job using a regular disk Device. The VirtualFull support was added in version 8.0.7. The Storage Daemon to Storage Daemon Copy/Migration process with deduplication protocol was added in version 8.2.0.
- A Virtual Full between two Storage Daemons is currently not supported.
- Data spooling cannot be used with deduplication. Since versions 8.2.12 and 8.4.10, data spooling is automatically disabled whenever a device resource is configured with Type = Dedup.
- All Bacula Enterprise File Daemons (including Linux, FreeBSD, Solaris, Windows, ...) support the Global Endpoint Deduplication. The Community Bacula File Daemon supports only the Global Endpoint Deduplication in dedup=storage mode. The list of the platforms supported by Bacula Enterprise is available on [www.baculasystems.com](http://www.baculasystems.com).
- We strongly recommend that you update all File Daemons that are used to write data into Dedup Volumes. It is not required, but old File Daemons do not support the newer FD to SD protocol, and consequently the Global Endpoint Deduplication will be done only on the Storage daemon side.

## 2.7 Best Practices

### RAID

If you plan to use RAID for performance and redundancy, note that read and write performances are essential for the deduplication engine. The Index is highly accessed for reading and for writing during backup jobs run and during the maintenance tasks required by the deduplication plugin. Also, the optimize process that rebuilds the Index strongly depend on the read and write performance of the disk infrastructure.

Some RAID solutions don't fit the deduplication engine read and write performance requirements. RAID 1 and RAID 10 with a small strip size are recommended for the dedup index. RAID 5, 6, 10, 50, 60 can be used for the containers. The strip size should be compatible with the size of the *extent* that is 4MiB. The goal is for a *stripe* to never be shared by two *extent*. If the extents are well aligned and are a multiple in size with the size of the stripe, then writing a single 4MiB extent should not require to read and rewrite the stripes that are nearby. The *extent* are aligned on a 4MiB boundary inside the *containers*, up to you to configure your raid array, your partitions, your LVM layers and the filesystem parameters (like the sunit and swidth of XFS) to get the container aligned too.

## ZFS

If you plan to use ZFS file system to store the dedup index, it is important to guarantee that you have enough memory and CPU resources in the Storage Daemon server to have both the deduplication plugin and ZFS in good condition.

The Global Endpoint Deduplication plugin does both deduplication and chunk compression. This means there is no need to have enabled deduplication or compression in the zfs pool that will be used to store containers. In fact, it is not recommended to have them enabled as it may cause slow performance and there will be no gain in space savings.

Aligned disk access is a key factor when using ZFS. ZFS is able to detect the sector size of the drive, but disks can report the emulated value instead. As we recommend SSD disks for the dedup Index, performance can be improved if the `ashift` property is explicitly set to match the 4096 byte sector size of SSD disks.

The disk block size is defined by the `ashift` value, but as ZFS stores data in records, there is another parameter that determines the individual dataset to be written in the ZFS pool, this is the `recordsize` ZFS pool parameter.

Thus another important ZFS pool setting to consider is the `recordsize` value. It defaults to 128K in recent ZFS versions. This value should be adjusted to match the typical I/O operations. For the ZFS pool used by the dedup Index, it was reported that setting the `recordsize` to 8K increased the vacuum performance. In addition, setting the ZFS kernel `zfs_vdev_aggregation_limit_non_rotating` parameter to the same value as `recordsize` highly improved performance.

Note these values are recommended for most SSD disks, but they may vary depending on the SSD model. For example, 16K could fit some SSD models and give a better performance. We recommend to perform I/O benchmark using different settings before the deduplication engine is setup in production.

Regarding the use of ZFS to store dedup containers, as it is not possible to preview the typical dataset because some containers can be much more used than others, it is more probably that a `recordsize` value of 64K or even the 128K default value are sufficient to have a good performance. However, it is strongly important to not allow container files to grow too much and limit the size of containers files to 3TB or 4TB.

## Maximum Container Size

For better performance, it is strongly recommended to not allow container files to grow indefinitely even if the underlying file system support very big files. This can be accomplished by setting the “Maximum Container Size” directive in the Storage resource in the Storage Daemon configuration file. It is recommended to set this directive to a value between 1 TB and 4 TB.

## Vacuum

It is strongly recommended, to keep the deduplication engine healthy, to regularly perform the maintenance tasks triggered by the vacuum.

Make sure to run regularly, in all deduplication engines in your environment, the following tasks: daily prune and truncation of volumes, a simple vacuum daily (it can be run weekly when not too much new chunks are added to the deduplication engine), a dedup vacuum checkindex checkmiss monthly

These maintenance tasks can be scheduled in a job of type Admin and they will help to clean both the deduplication engine index and containers, marking unused entries in the index and chunks in containers, thus allowing the reuse of space. It will contribute to avoid invalid entries in the index to be used by backup jobs in the case of any problems with the server hosting the deduplication engine.

The scrub process is not implemented in Dedup2.

## Holepunching

Holepunching doesn't exist anymore in Dedup2. The last part of the *Vacuum* release the *extents* that are left empty but also merge the data that are in *extents* that are nearly empty or half used into new containers. This allows new backup to always write into fully recycled extents, reduce the fragmentation and increase the efficiency of the storage.

**The Global Endpoint Deduplication version 2 offers the following advantages:**

- Maintenance tools are eased and faster to process. There is only a need to run *vacuum* and *optimize* with GED 2
- All new data are grouped per job in the containers, which makes backup and restore jobs faster. The performance gain is particularly important on restore jobs. This is due to the data being aligned in the containers.
- The Containers structure brings more flexibility in regards to containers files size and hole creation, thus the storage space management is optimized. The new technology should create no fragmentation.
- GED 2 is available for production use from Bacula Enterprise version 16.0

**See also:**

For information about migration from Dedup 1 to Dedup 2, look here.

## 3 Moving from Dedup 1 to Dedup 2

Starting with Bacula Enterprise Edition 16.0, the new dedup2 engine brings a new engine design and a new configuration format. Find more details in the Global Endpoint Deduplication 2 article.

It is not possible to simply reuse the current dedup1 index, containers and volumes data in the new dedup2 engine. It means you cannot configure the dedup2 engine in your environment, and make it use the current dedup1 index, containers and volumes.

However, it is possible to migrate data from the current dedup1 engine (now called dedup legacy; the terms are used interchangeably) to the new dedup2 engine. Also, you may keep the current dedup1 engine for restore purposes, whilst using the new dedup2 engine to store new backups.

In this document, a few scenarios will be presented to help adopting the new dedup2 engine, assuming you are already using the deduplication plugin with the old dedup engine layout and configuration.

---

**Important:** Before implementing any strategy in production, read carefully the Important Considerations section before implementing any strategy in production.

---

### 3.1 Important Considerations

- Running two dedup engines in parallel may be quite intensive in regards to the I/O on the SSD disk, where the dedup index is stored, and also in regards to the I/O where containers are stored, especially if the containers are stored in NFS mounts.
- If the same NFS mount is used to store the containers of both the dedup1 and the dedup2 engines in a migration process, you may have a high load on the NFS mount.
- It is recommended to schedule migration jobs when backup jobs are not running.

- In scenarios where the dedup1 engine is kept for restore purposes, note that the storage occupancy in dedup1 will be the same until this storage is fully decommissioned. This is because the containers cannot be shrunk, even if data is pruned.
- In scenarios where the data is not migrated to dedup2, the new dedup2 engine will start from scratch without a high dedup ratio which also means that you can potentially have the very same data (chunks) in both dedup engines, and as a consequence, the storage occupancy will be high.

We present the scenarios that show how to move on from dedup1 to dedup2.

Scenarios for migration of data from dedup1 to dedup2:

## Scenario 1. Dedup Legacy and Dedup2 Running on Different Hosts

**Goal of the scenario: Migrating data from your current dedup1 engine to a new Dedup2 engine that use different Storage Daemon hosts.**

---

**Note:** In this scenario, there are two Storage Daemons running.

---

It is required to have both dedup engines running at the same time to migrate the data.

The new dedup2 engine will be implemented in a different Storage Daemon host, so only its configuration, using the new DedupEngine resource, is needed. In the bacula-sd.conf file of the new Storage Daemon host, add a DedupEngine resource configured to use `Driver = Dedup2`:

```
Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}
```

And the dedup devices should point to the dedup2 engine, for example:

```
Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

Device {
  Name = FileDedup2-dev1
  Drive Index = 1
  ...
```

(continues on next page)

```

Device Type = Dedup
DedupEngine = Dedupengine_name_with_dedup2
}

```

After you have the new Dedup2 engine correctly configured and running, you may use Migration Jobs to migrate jobs from the dedup1 storage to the new dedup2 storage. Once all jobs have been migrated, the current dedup1 engine can be decommissioned.

More details about Migration Jobs: Migration and Copy.

## Scenario 2. Dedup Legacy and Dedup2 Running on the Same Storage Daemon

**Goal of the scenario: Migrating data from your current dedup1 engine to a new dedup2 engine that use the same Storage Daemon host. The new dedup2 engine uses the same Storage Daemon port, SDPort 9103.**

---

**Note:** In this scenario, there is only one Storage Daemon running.

---

It is required to have both dedup engines running at the same time to migrate the data.

If you decide to have both dedup engines in the same host and use the same Storage Daemon port, **it is required to modify the current dedup1 engine configuration to use the new DedupEngine resource.**

The new dedup2 configuration format uses the new DedupEngine resource, and it is not compatible with the old dedup1 configuration format. It means that it is not possible to add the dedup2 engine using the new Dedupengine Resource in the current Storage Daemon configuration if there is already a Dedup1 engine configured using the old configuration format.

For example, **it is not possible** to have both, the old and new formats, configured in the same Storage Daemon configuration file:

```

# From bacula-sd.conf
Storage {
  Name = Dedup1-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  Dedup Directory = /mnt/bacula/dedup1/containers
  Dedup Index Directory = /mnt/SSD/dedup1/index
  Maximum Container Size = 2TB
}

```

and

```

Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}

```

However, **it is possible** to have one dedup legacy engine, and one dedup2 engine in the same Storage Daemon by modifying the current dedup1 configuration to the new format as documented in the New Dedupengine Resource section in the Global Endpoint Deduplication 2 article.

To modify your current legacy Dedup engine to the new configuration format:

1. Stop the Storage Daemon
2. Modify the current dedup1 engine settings by creating a DedupEngine resource with `Driver = Legacy`, and the same index and containers directories of your current dedup engine:

```
Dedupengine {
  Name = Dedupengine_legacy
  Dedup Directory = /mnt/bacula/dedup1/containers
  Dedup Index Directory = /mnt/SSD/dedup1/index
  Driver = Legacy
  Maximum Container Size = 2TB
}
```

3. Add the `Dedupengine = Dedupengine_legacy` directive to all the dedup1 devices:

```
Autochanger {
  Name = FileDedup1-Chgr1
  Device = FileDedup1-Dev0, FileDedup1-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup1-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_legacy <--- @aga, should we highlight this?
}

Device {
  Name = FileDedup1-dev1
  Drive Index = 1
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_legacy <--- @aga, should we highlight this?
}
```

4. Remove or comment the index and containers directories from the Storage resource configuration:

```
# From bacula-sd.conf
Storage {
  Name = Dedup1-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  # Dedup Directory = /mnt/bacula/dedup1/containers <--- you may remove this line
  # Dedup Index Directory = /mnt/SSD/dedup1/index <--- you may remove this line
  Maximum Container Size = 2TB
}
```

5. Add the new dedup2 engine resource and autochanger/devices to the same bacula-sd.conf file:

```

Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}

Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

Device {
  Name = FileDedup2-dev1
  Drive Index = 1
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

```

#### 6. Start the Storage Daemon.

After you have the new Dedup2 engine correctly configured and running, you may use Migration Jobs to migrate jobs from the dedup1 storage to the new dedup2 storage. Once all jobs have been migrated, the current dedup1 engine can be decommissioned.

More details about Migration Jobs: Migration and Copy.

### Scenario 3. Dedup Legacy and Dedup2 Running on the Same Host on Two Different Storage Daemons

**Goal of the scenario: Migrating data from your current dedup1 engine to a new dedup2 engine that use the same Storage Daemon host. The new Dedup2 engine uses a different Storage Daemon port, for example, SDPort 9104.**

---

**Note:** In this scenario, there are two Storage Daemons running: one using 9103 SDPort, the other using 9104 SDPort.

---

It is required to have both dedup engines running at the same time to migrate the data.

In this case, **it is not required** to modify the current dedup1 configuration to use the new format, unless you wish to use the new configuration format to have a standard configuration for all the dedup engines in your environment.

As the new dedup2 engine is to be configured in a new bacula-sd.conf file, there is no need to modify the current bacula-sd.conf file used by Dedup1.

Thus, you should keep the current Dedup1 configuration in the bacula-sd.conf file (if the SDPort = 9103 is not explicitly configured, it is used by default by the Storage Daemon).

Example of the configuration of your dedup1 engine, running on port 9103:

```
# From bacula-sd.conf
Storage {
  Name = Dedup1-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  Dedup Directory = /mnt/bacula/dedup1/containers
  Dedup Index Directory = /mnt/SSD/dedup1/index # Recommended to be on fast local SSD
  ↪storage
  Maximum Container Size = 2TB # Maximum 511 containers can be created, please adapt to
  ↪your need
}
```

Then, create a new **bacula-sd-dedup2.conf** file for the new dedup2 engine, using port 9104:

```
Storage {
  Name = Dedup2-sd
  SDPort = 9104
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  Maximum Concurrent Jobs = 20
}

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

Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}

Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
}
```

(continues on next page)



```

...
Device Type = Dedup
DedupEngine = Dedupengine_name_with_dedup2
}

Device {
Name = FileDedup2-dev1
Drive Index = 1
...
Device Type = Dedup
DedupEngine = Dedupengine_name_with_dedup2
}

Messages {
Name = Default
Director = bacula-dir = all
Append = "/opt/bacula/log/bacula-sd.log" = All, !Skipped
}

```

To start the new dedup2 Storage Daemon, you can use this command: `sudo -u bacula /opt/bacula/bin/bacula-sd -dt -c /opt/bacula/etc/bacula-sd-dedup2.conf`, but it is recommended to create a systemd service for the new Dedup2 Storage Daemon, for example, the `bacula-sd-dedup2.service` file:

```

# Description:
#   Used to start the bacula Dedup2 storage daemon service (bacula-sd-dedup2)
#   enable : systemctl enable bacula-sd-dedup2
#   start  : systemctl start bacula-sd-dedup2
#
#
# from http://www.freedesktop.org/software/systemd/man/systemd.unit.html
[Unit]
Description=Bacula Enterprise Storage Daemon service
Requires=network.target
After=network.target
RequiresMountsFor=/opt/bacula/working /opt/bacula/etc /opt/bacula/bin
/mnt/bacula/dedup2/containers /mnt/SSD/dedup2/index /mnt/bacula/dedup2/volumes

# from http://www.freedesktop.org/software/systemd/man/systemd.service.html
[Service]
Type=forking
User=bacula
Group=disk
Environment="LD_LIBRARY_PATH=/opt/bacula/lib"
ExecStart=/opt/bacula/bin/bacula-sd -dt -c /opt/bacula/etc/bacula-sd-dedup2.conf
StandardError=syslog
TimeoutStopSec=3min
LimitMEMLOCK=infinity
LimitNOFILE=10000
LimitNPROC=10000
OOMScoreAdjust=-1000

```

```
[Install]
WantedBy=multi-user.target
```

Then, you can enable and start the new dedup2 engine Storage Daemon using systemd commands:

```
$ sudo systemctl enable bacula-sd-dedup2.service
$ sudo systemctl start bacula-sd-dedup2.service
```

After you have the new Dedup2 engine correctly configured and running, you may use Migration Jobs to migrate jobs from the dedup1 storage to the new dedup2 storage. Once all jobs have been migrated, the current dedup1 engine can be decommissioned.

More details about Migration Jobs: Migration and Copy.

Scenarios for keeping dedup1 for restore purposes only:

#### Scenario 4. Setup Dedup2 on a New Host and Redirect All Backup Jobs There

**Goal of the scenario: Keeping the dedup1 engine for restore purposes, and setting up a new dedup2 engine for new backups in a different Storage Daemon host.**

---

**Note:** In this scenario, there are two Storage Daemons running.

---

In this scenario, you will need to setup the new dedup2 engine in the new Storage Daemon host as described in Scenario 1.

The new Dedup2 engine is to be implemented in a different Storage Daemon host, so only its configuration, using the new DedupEngine resource, is needed. In the bacula-sd.conf file of the new Storage Daemon host, add a DedupEngine resource configured to use **Driver = Dedup2**:

```
Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}
```

And the dedup devices should point to this dedup2 engine, for example:

```
Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
```

(continues on next page)

```

DedupEngine = Dedupengine_name_with_dedup2
}

Device {
  Name = FileDedup2-dev1
  Drive Index = 1
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

```

Then, you should redirect all your backup jobs (this can be done gradually if you have a huge number of Jobs and/or Pools that need to be modified) to the new Dedup2 engine storage.

The dedup1 engine can be decommissioned when the data retention has expired, meaning the retention period for all the jobs previously stored in the dedup1 engine expired, and you will not need to restore any data stored in this dedup1 engine.

If you want to use this scenario to keep the dedup1 engine for restore purposes only, consider to have the dedup1 engine Storage Daemon shut down, and you may start it only when restore jobs are needed.

### Scenario 5. Setup Dedup2 on the Same Storage Daemon Host and Using the Same SDPort and Redirect All Backup Jobs There

**Goal of the scenario: Keeping the dedup1 engine for restore purposes, and setting up a new dedup2 engine for new backups in the same Storage Daemon host. The new Dedup2 engine uses the same Storage Daemon port, SDPort 9103.**

---

**Note:** In this scenario, there is only one Storage Daemon running.

---

This scenario is similar to Scenario 2.

If you decide to have both dedup engines in the same host and using the same Storage Daemon port, **it is required to modify the current dedup1 engine configuration to use the new DedupEngine resource.**

Then, **you must modify** the current dedup1 configuration to use the new configuration format as documented in the New Dedupengine Resource section in the Global Endpoint Deduplication 2 article.

1. Stop the Storage Daemon
2. Modify the current dedup1 engine settings by creating a DedupEngine resource with Driver=Legacy, and the same index and containers directories of your current dedup engine:

```

Dedupengine {
  Name = Dedupengine_legacy
  Dedup Directory = /mnt/bacula/dedup1/containers
  Dedup Index Directory = /mnt/SSD/dedup1/index
  Driver = Legacy
  Maximum Container Size = 2TB
}

```

3. Add the Dedupengine = Dedupengine\_legacy directive to all the dedup1 devices:

```

Autochanger {
  Name = FileDedup1-Chgr1
  Device = FileDedup1-Dev0, FileDedup1-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup1-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_legacy <--- @aga, should we highlight this?
}

Device {
  Name = FileDedup1-dev1
  Drive Index = 1
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_legacy <--- @aga, should we highlight this?
}

```

4. Remove or comment the index and containers directories from the Storage resource configuration:

```

# From bacula-sd.conf
Storage {
  Name = Dedup1-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  # Dedup Directory = /mnt/bacula/dedup1/containers <--- you may remove this line
  # Dedup Index Directory = /mnt/SSD/dedup1/index <--- you may remove this line
  Maximum Container Size = 2TB
}

```

5. Add the new dedup2 engine resource and autochanger/devices to the same bacula-sd.conf file:

```

Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}

Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

```

(continues on next page)

```

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

Device {
  Name = FileDedup2-dev1
  Drive Index = 1
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

```

#### 6. Start the Storage Daemon.

Then, you should redirect all your backup jobs (this can be done gradually if you have a huge number of Jobs and/or Pools that need to be modified) to the new dedup2 engine storage.

The dedup1 engine can be decommissioned when the data retention has expired, meaning the retention period for all the jobs previously stored in the dedup1 engine expired, and you will not need to restore any data stored in this dedup1 engine.

If you want to use this scenario to keep the dedup1 engine for restore purposes only, consider to have the dedup1 engine Storage Daemon shut down, and you may start it only when restore jobs are needed.

### Scenario 6. Setup Dedup2 on the Same Storage Daemon Host and Using Different SDPorts and Redirect All Backup Job There

**Goal of the scenario: Keeping the dedup1 engine for restore purposes, and setting up a new dedup2 engine for new backups that use the same Storage Daemon host. The new Dedup2 engine uses the a different Storage Daemon port, for example, SDPort 9104.**

---

**Note:** In this scenario, there are two Storage Daemons running: one using 9103 SDPort, the other using 9104 SDPort.

---

This scenario is similar to Scenario 3.

In this scenario, **you must setup** the new Dedup2 engine in a new bacula-sd.conf file, for example, bacula-sd-dedup2.conf.

Also, **it is not required** to modify the current dedup1 configuration to use the new format, unless you wish to use the new configuration format to have a standard configuration for all the dedup engines in your environment.

As the new Dedup2 engine is to be configured in a new bacula-sd.conf file, there is no need to modify the current bacula-sd.conf file used by Dedup1.

Thus, you should keep the current Dedup1 configuration in the bacula-sd.conf file (if the SDPort = 9103 is not explicitly configured, it is used by default by the Storage Daemon).

Example of the configuration of your dedup1 engine, running on port 9103:

```

# From bacula-sd.conf
Storage {
  Name = Dedup1-sd
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  Dedup Directory = /mnt/bacula/dedup1/containers
  Dedup Index Directory = /mnt/SSD/dedup1/index # Recommended to be on fast local SSD
  ↳storage
  Maximum Container Size = 2TB # Maximum 511 containers can be created, please adapt to
  ↳your need
}

```

Then, create a new **bacula-sd-dedup2.conf** file for the new Dedup2 engine, using port 9104:

```

Storage {
  Name = Dedup2-sd
  SDPort = 9104
  Working Directory = /opt/bacula/working
  Pid Directory = /opt/bacula/working
  Plugin Directory = /opt/bacula/plugins
  Maximum Concurrent Jobs = 20
}

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

Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup2/containers
  Dedup Index Directory = /mnt/SSD/dedup2/index
  Driver = Dedup2
  Maximum Container Size = 2TB
}

Autochanger {
  Name = FileDedup2-Chgr1
  Device = FileDedup2-Dev0, FileDedup2-Dev1
  Changer Command = /dev/null
  Changer Device = /dev/null
}

Device {
  Name = FileDedup2-dev0
  Drive Index = 0
  ...
  Device Type = Dedup
  DedupEngine = Dedupengine_name_with_dedup2
}

Device {

```

(continues on next page)

```

Name = FileDedup2-dev1
Drive Index = 1
...
Device Type = Dedup
DedupEngine = Dedupengine_name_with_dedup2
}

Messages {
Name = Default
Director = bacula-dir = all
Append = "/opt/bacula/log/bacula-sd.log" = All, !Skipped
}

```

To start the new dedup2 Storage Daemon, you can use this command: `sudo -u bacula /opt/bacula/bin/bacula-sd -dt -c /opt/bacula/etc/bacula-sd-dedup2.conf`, but it is recommended to create a systemd service for the new Dedup2 Storage Daemon, for example, the `bacula-sd-dedup2.service` file:

```

# Description:
#   Used to start the bacula Dedup2 storage daemon service (bacula-sd-dedup2)
#   enable : systemctl enable bacula-sd-dedup2
#   start  : systemctl start bacula-sd-dedup2
#
#
# from http://www.freedesktop.org/software/systemd/man/systemd.unit.html
[Unit]
Description=Bacula Enterprise Storage Daemon service
Requires=network.target
After=network.target
RequiresMountsFor=/opt/bacula/working /opt/bacula/etc /opt/bacula/bin
/mnt/bacula/dedup2/containers /mnt/SSD/dedup2/index /mnt/bacula/dedup2/volumes

# from http://www.freedesktop.org/software/systemd/man/systemd.service.html
[Service]
Type=forking
User=bacula
Group=disk
Environment="LD_LIBRARY_PATH=/opt/bacula/lib"
ExecStart=/opt/bacula/bin/bacula-sd -dt -c /opt/bacula/etc/bacula-sd-dedup2.conf
StandardError=syslog
TimeoutStopSec=3min
LimitMEMLOCK=infinity
LimitNOFILE=10000
LimitNPROC=10000
OOMScoreAdjust=-1000

[Install]
WantedBy=multi-user.target

```

Then you can enable and start the new Dedup2 engine Storage Daemon using systemd commands:

```

$ sudo systemctl enable bacula-sd-dedup2.service
$ sudo systemctl start bacula-sd-dedup2.service

```

Then, you should redirect all your backup jobs (this can be done gradually if you have a huge number of Jobs and/or Pools that need to be modified) to the new Dedup2 engine storage.

The dedup1 engine can be decommissioned when the data retention has expired, meaning the retention period for all the jobs previously stored in the dedup1 engine expired, and you will not need to restore any data stored in this dedup1 engine.

If you want to use this scenario to keep the dedup1 engine for restore purposes only, consider to have the dedup1 engine Storage Daemon shut down, and you may start it only when restore jobs are needed.