Docker Plugin

Bacula Systems Documentation

Contents

Contents

1 Features Summary

- Docker container image-based backup
- Docker system images backup
- Docker volumes files data backup
- Ability to restore single Docker container, image and volume
- Ability to restore Docker container as a new Docker system image
- Ability to create or run restored Docker container
- Ability to restore image, container or volume archives to local directory

2 Backup with Docker Plugin

Containers are very light system level virtualization with less overhead because programs in virtual partitions use the operating system's normal system call interface and do not need to be subjected to emulation or be run in an intermediate virtual machine. Especially Docker containers rely on sophisticated fs-level data abstraction with a number of read-only images which creates a template used for container initialization. When initialized a container include a writable layer where all file modifications are stored.

With Docker Plugin, the Bacula Enterprise will save the full container image including all read-only and writable layers into a single image archive. It is possible to backup a defined Docker images only which will be used to create new containers when required.

It is not needed to install a Bacula File daemon in each container, so you can backup containers based on common image repository. The Bacula Docker Plugin will contact the Docker service to read and save the contents of any system image or container image using snapshots (default behavior) and dump them using the Docker API.

Bacula does not need to walk through the container file-system to open, read, close and `stat` files, so it consumes less resources on the Docker infrastructure than a standard file level backup.

3 Docker Persistent Volume Backup

All Containers as seen in Docker are ephemeral. You can execute a container with auto destroy option (**—-rm**) which will remove all changes made in docker images during runtime by default. This feature is very useful when you want to execute your computations in never changing environment but is useless when computation result should be preserved instead of removed. In this case you will use a Docker Persistent Volumes.

This brings a new challenge to data backup solution. Fortunate most of the challenges we can meet here are almost the same as we have in standard bare metal or virtualized environments. All data stored in different kind of databases should be saved with dedicated Bacula Enterprise Plugins exploiting network backup data dump.

You should refer to appropriate Bacula Enterprise Plugin Whitepapers for more details.

On the other hand most non-database applications stores its data as a simple flat files we can backup as-is without forcing complicated transactions or data consistency procedures. This case is handled directly with Docker Plugin using a dedicated Bacula Archive container executed from a prepared image.

4 Installation

The Bacula File Daemon and its Docker Plugin need to be installed on the host for the Docker service. Docker could be installed on different operating systems and distributions, so the Bacula Enterprise File Daemon for this operating system and platform has to be used. You can backup or restore Docker objects on remote host using **docker_host=...** plugin parameter, check **genericparameters**.

Installation of the Bacula Enterprise Docker Plugin is most easily done by adding the repository file suitable for the existing subscription and the distributions package manager configuration. An example would be `/etc/apt/sources.list.d/bacula.list` for deb based Linux distributions with the following content:

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

After that, a run of `apt-get update` is needed. Then, the Plugin can be installed using `apt-get install bacula-enterprise-docker-plugin`

On Redhat/CentOS 7 extend the repository file for your package manager to contain a section for the plugin - `/etc/yum.repos.d/bacula.repo`:

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

(continues on next page)

(continued from previous page)

```
[Bacula EnterpriseDockerPlugin]
name=Bacula Enterprise Docker Plugin
baseurl=https://www.baculasystems.com/dl/@customer@/rpms/docker/@version@/rhel7-64/
enabled=1
protect=0
gpgcheck=0
```

Then perform a `yum update` and after that the package `bacula-enterprise-docker-plugin` can be installed with `yum install`.

The package `bacula-enterprise-docker-plugin` must be installed for Docker volume backup. Please see *Bacula Archive docker image* section for more details.

Manual installation of the packages, can be done after downloading the right files from the Bacula Systems provided download area, and then using the low-level package manager (`rpm` or `dpkg`) to do the plugin installation.

5 File Daemon Configuration

The **Plugin Directory** directive of the resource in `/opt/bacula/etc/bacula-fd.conf` must point to where the `docker-fd.so` plugin file is installed. The standard Bacula plugin directory is `/opt/bacula/plugins`

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

6 Bacula Archive docker image

For proper Docker volume backup Bacula Enterprise Docker Plugin requires a dedicated Bacula Archive Container image that is available in the bacula-enterprise-docker-tools package.

The docker image should be installed automatically, but if not, it is possible to load manually the `baculatar` to local Docker installation with the following command:

```
# cd /opt/bacula/lib
# docker load -i baculatar-docker-07Dec22.tar.gz
```

7 Backup and Restore Operations

8 Backup

Plugin can backup a three distinct Docker objects: *Docker Container*, *Image* and *Persistent Volume* files data.

The backup of a single container consists of the following steps:

1. Save current container state to new image (container commit - snapshot).
2. Execute `docker` utility and save data.

3. Remove saved snapshot to free not needed resources.

The backup of a docker system images does not make a snapshot as every system image is a read-only template used for container creation.

The Persistent Volume files data backup is performed with a dedicated helper container and consist of the following steps:

1. Prepare a local log archive Docker Volume.
2. Execute a dedicated Bacula Archive container which will access *Docker Persistent Volume* data files for backup.
3. Save Volume data files stream from container.
4. Terminate Bacula Archive container when done.

Backups can be performed for container in any state (created, running or stopped).

The Docker Plugin will inform you about every container or image backup start and finish:

```
JobId 127: docker: Start Backup Docker Container: myubuntu (4d0a4fadb50d)
JobId 127: dkcommctx: Commit created: myubuntu/4d0a4fadb50d/127:backup
JobId 127: dkcommctx: Commit removed: myubuntu/4d0a4fadb50d/127:backup
JobId 127: docker: Backup of Docker Container: myubuntu (4d0a4fadb50d) OK.
...
```

The backup will create a single (`.tar`) file for any container, image or Docker volume which is saved. Inside Bacula, those are represented as follows.

- `/@docker/container/<name-label>/<id>.tar` for container backup
- `/@docker/container/<name-label>/volume: <name> -> <mountpoint>` during volume backup for this container
- `/@docker/image/<repository:tag>/<id>.tar` for image backup
- `/@docker/volume/<name>.tar` for volume backup

Multiple files will be created during a backup if multiple containers images or volumes are selected for backed up with one job. The distinct file names as shown above allow to locate the proper container, system image or volume archive for restore. You can list a special support file (symbolic link) generated at container backup directory tree which represent a volume name and mount point for this volume for easy data recovery.

To list available Docker containers, system images or defined volumes, a listing mode is available, described in chapter **listing**.

9 Restore

The Docker Plugin provides two targets for restore operations:

- Restore to Docker service
- Restore to a local directory as archive files

9.1 Restore to Docker

To use this restore method, the **where=** or parameter of a Bacula `restore` command is used.

The Docker container archive will be sent to the Docker service and restored as a new image and then created as container if the `container_create` restore parameter is set (this is a default). If the restore parameter `container_create` is not set then any container will be restored to image level only and user has to create or run the container manually. If restore parameter `container_run` is set (default is no) then restored container will be started immediate after successful restore.

The Docker image archive will be loaded into the Docker service as the original docker image overwriting the one already exist. This is a default behavior and cannot be changed. You can skip docker image restore of already existent image with **Replace** option of `restore` command.

Docker volume restore will extract all files into the same Docker volume. If volume is not defined in Docker then a default local volume will be created. Volume data files restore will always overwrite restored data. This is a current limitation of the plugin and will be removed in the future.

You can change default container name or container image label during restore with `container_defaultnames` or `container_imageid` restore parameters (see `container_defaultnames: <Yes|No>` or `container_imageid: <Yes|No>`).

9.2 Restore To Local Directory

To use this mode, the **where=/some/path** Bacula `restore` parameter is set to a full path on the server where the Docker Plugin is installed. If the path does not exist, it will be created by the Bacula Docker Plugin. With this restore mode you can restore any saved Docker object including containers, images and volumes.

10 Plugin Configuration

The plugin is configured using **Plugin Parameters** defined in a FileSets **Include** section of the Bacula Enterprise Director configuration.

11 Generic Plugin Parameters

The following Docker plugin parameter affects any type of Job (Backup, Estimation, or Restore).

abort_on_error[=<0 or 1>] specifies whether or not the plugin should abort execution (and the Bacula Job) if a fatal error occurs during a Backup, Estimation, or Restore operation. The default value is 0.

This parameter is optional.

docker_host=tcp://<host:port> if specified then all operations will be executed using **<host:port>** endpoint. The default is to use a local default socket for local `dockerd`. This parameter can be set or overloaded for restore operations using restore plugin options. Single **docker_host=...** option is supported. If multiple defined the first one will be used.

When this parameter is set then plugin will disable volume backup or restore functionality as this feature is not supported with remote Docker. This is a current plugin limitation and will be removed in the future.

This parameter is optional.

12 Estimation and Backup Plugin Parameters

These plugin parameters are relevant only for Backup and Estimation jobs:

container=<name-label>|<id> specifies a Docker container to backup. Multiple `container=...` parameters are allowed. If container with `name-label` can not be found, then a single job error will be generated and the backup will proceed to the next container unless `abort_on_error` is set which will cause the backup job to be aborted. You can use Docker container names (`name-label`) or container id () to select required container to backup.

This parameter is optional.

image=<repository:tag>|<id> specifies a Docker image to backup. Multiple `image=...` parameters are allowed. If image with `repository:tag` can not be found, then a single job error will be generated and the backup will proceed to the next image unless `abort_on_error` is set which will cause the backup job to be aborted. You can use Docker image repository and tag names (`repository:tag`) or image id () to select required image to backup. This parameter is optional.

include_container=<name-label-regex> specifies a list of Docker container names to backup using regular expression syntax. All containers with names matching the regular expression provided will be selected for backup. Multiple **include_container=...** parameters may be provided.

If no containers are matching the name expression provided, the backup will proceed to the next parameter or finish successfully without backing up any containers. The **abort_on_error** parameter will not abort the job when no containers are found using name matching. This parameter is optional.

include_image=<repository-regex> specifies a list of Docker image repository names to backup using regular expression syntax. All images with repository name matching the regular expression provided will be selected for backup. Multiple **include_images=...** parameters may be provided.

If no images are matching the repository name expression provided, the backup will proceed to the next parameter or finish successfully without backing up any images. The **abort_on_error** parameter will not abort the job when no images are found using repository name matching. This parameter is optional.

exclude_container=<name-label-regex> specifies a list of a Docker container names which will be excluded from backup using regular expression matching. All containers with names matching the provided regular expression, and selected for backup using the **include_container=...** parameter will be excluded. This parameter does not affect any containers selected to be backed up using **container=...** parameter. Multiple **exclude_container=...** parameters may be provided. This parameter is optional.

exclude_image=<repository-regex> specifies a list of a Docker image repository names which will be excluded from backup using regular expression matching. All images with repository names matching the provided regular expression, and selected for backup using the **include_image=...** parameter will be excluded. This parameter does not affect any images selected to be backed up using **image=...** parameter. Multiple **exclude_image=...** parameters may be provided. This parameter is optional.

volume=<name> specifies the Docker volume to backup. Multiple `volume=...` parameters are allowed. If volume with `name` can not be found, then a single job error will be generated and the backup will proceed to the next volume unless `abort_on_error` is set which will cause the backup job to be aborted.

This parameter is optional.

allvolumes specifies the automatic Docker volumes selection for backup for every container selected. You should configure a single **allvolumes** parameter if required.

This parameter is optional.

mode=<pause|nopause> specifies the default backup mode for containers to use. The **pause** mode will generate backup using container pause commit which quiesce I/O operations during backup and allow to achieve a consistent backup with some possible downtime. The container will remain suspended during backup commit phase

and will resume once the commit operation finishes. The **nopause** mode will commit container without quiesce I/O operations. This parameter is optional. If not set then **pause** mode will be used by default.

timeout=nn specifies the default timeout for connecting to Bacula Archive container during volume backup. The default is 30 seconds if parameter is not specified. The value **<nn>** specifies a number of seconds how long plugin will wait. The value zero (0) is invalid.

This parameter is optional.

If none of the parameters **container=...**, **image=...**, **include_container=...** , **include_image=...**, and **exclude=...** are specified, all available containers and images on Docker will be backed up.

13 Plugin Restore Parameters

During restore, the Docker plugin will use the same parameters which were set for the backup job and saved in the catalog. Some of them may be changed during the restore process if required.

container_create: <Yes|No> specifies if Docker container restore should automatically create container. The default option is to create container on restore. If you want to restore container as an image only then you should set this parameter to `No`.

This parameter is optional.

container_run: <Yes|No> specifies if Docker container restore should automatically create and run container. The default option is to not run container on restore. If you want to automatically run container on restore then you should set this parameter to `Yes`. If you set this parameter to `Yes` then **container_create** parameter will be ignored.

This parameter is optional.

container_imageid: <Yes|No> specifies if Docker Plugin should use image id value to create or run restored container. The default is to use image repository and tag values for that. This parameter will be ignored when both **container_create** and **container_run** will be set to `No` as no container create or run operation will be performed.

This parameter is optional.

container_defaultnames: <Yes|No> specifies if Docker Plugin should setup a container names based on original container name and JobId values which is a default. If parameter is set to `Yes` then Docker service will setup default names for created or run container.

This parameter is optional.

timeout: nn specifies the default timeout for connecting to Bacula Archive container during volume restore. The default is 30 seconds. The value zero (0) is invalid.

This parameter is optional.

docker_host: tcp://<host:port> check **genericparameters** for more info.

14 FileSet Examples

In the example below, all Docker containers, images and volumes will be backed up.

```
FileSet {
  Name = FS_DockerAll
  Include {
    Plugin = "docker:"
  }
}
```

In this example, a single Docker container with name-label of “mcache1” will be backed up.

```
FileSet {
  Name = FS_Docker_mcache1
  Include {
    Plugin = "docker: container=mcache1"
  }
}
```

The same example as above, but using container id instead:

```
FileSet {
  Name = FS_Docker_mcache1
  Include {
    Plugin = "docker: container=cd77eb89e59a"
  }
}
```

In the following example, all Docker containers which contain “ngnix” in their names will be backed up.

```
FileSet {
  Name = FS_Docker_nginixAll
  Include {
    Plugin = "docker: include_container=ngnix"
  }
}
```

In this final example, all Docker containers except whose names begins with “test” will be backed up.

```
FileSet {
  Name = FS_Docker_AllbutTest
  Include {
    Plugin = "docker: include_container=.* exclude_container=^test"
  }
}
```

In this final example, a single Docker volume will be backed up.

```
FileSet {
  Name = FS_Docker_Volume
  Include {
    Plugin = "docker: volume=myvolume"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

In this final example, a single Docker container with all mounted volumes will be backed up.

```
FileSet {
  Name = FS_Docker_Container_VolumeAll
  Include {
    Plugin = "docker: container=mycontainer allvolumes"
  }
}
```

In the example below, all Docker objects will be backed up using remote docker host.

```
FileSet {
  Name = FS_RemoteDockerAll
  Include {
   Plugin = "docker: docker_host=tcp://10.0.0.1:2376"
  }
}
```

15 Restore examples

16 Restore to a Docker service

To restore a container or image to a Docker service, the administrator should execute the restore command and specify the **where** parameter as in this example:

```
* restore where=/
```

and then set any other required restore plugin parameters for the restore.

In the following restore session example, the **container_run** plugin restore option is set to “Yes”:

```
* restore where=/
...
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/docker-test-dir.restore.1.bsr
Where:           /
Replace:         Always
FileSet:         Full Set
Backup Client:   docker-test-fd
Restore Client:  docker-test-fd
Storage:         File1
When:            2018-09-28 14:09:30
Catalog:         MyCatalog
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): mod
```

(continues on next page)

(continued from previous page)

```
Parameters to modify:
    1: Level
    2: Storage
    3: Job
    4: FileSet
    5: Restore Client
    6: When
    7: Priority
    8: Bootstrap
    9: Where
    10: File Relocation
    11: Replace
    12: JobId
    13: Plugin Options
Select parameter to modify (1-13): 13
Automatically selected : docker: container=mcache1 abort_on_error
Plugin Restore Options
container_create:    *None*              (*Yes*)
container_run:       *None*              (*No*)
container_imageid:   *None*              (*No*)
container_defaultnames: *None*              (*No*)
docker_host:         *None*              (*local*)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
    1: container_create (Create container on restore)
    2: container_run (Run container on restore)
    3: container_imageid (Use Image Id for container creation/start)
    4: container_defaultnames (Use default docker Names on container creation)
    5: docker_host (Use defined docker host to restore)
Select parameter to modify (1-4): 2
Please enter a value for container_run: yes
Plugin Restore Options
container_create:    *None*              (*Yes*)
container_run:       yes                 (*No*)
container_imageid:   *None*              (*No*)
container_defaultnames: *None*              (*No*)
docker_host:         *None*              (*local*)
Use above plugin configuration? (yes/mod/no): yes
```

The restore job log will indicate which container is restored and which new container id was created:

```
JobId 139: Start Restore Job RestoreFiles.2018-09-28_14.13.31_03
JobId 139: Using Device "FileChgr1-Dev1" to read.
JobId 139: Ready to read from volume "vol001" on File device "FileChgr1-Dev1" (/opt/
↪bacula/archive).
JobId 139: Forward spacing Volume "vol001" to addr=225
JobId 139: docker: Docker Container restore: mcache1/b97d4dd88063
JobId 139: End of Volume "vol001" at addr=62177197 on device "FileChgr1-Dev1" (/opt/
↪bacula/archive).
JobId 139: dkcommctx: Successfully run container as: ef48c6b5b867
```

The new container created during the restore will get a new container id. You can check it running with a following command:


```
# docker ps -a
CONTAINER ID        IMAGE                                 CREATED             STATUS
ef48c6b5b867        mcache1/b97d4dd88063/139:restore   4 minutes ago       Up 4 minutes
```

17 Restore to Local Directory

It is possible to restore the container images, Docker images and volume archives to a file without loading them into Docker service. To do so, the **where** restore option should point to the local directory:

```
* restore where=/tmp/bacula/restores
```

Please check the following example for the message "Docker local restore":

```
JobId 141: Start Restore Job RestoreFiles.2018-09-28_14.26.34_03
JobId 141: Using Device "FileChgr1-Dev1" to read.
JobId 141: Ready to read from volume "vol001" on File device "FileChgr1-Dev1" (/opt/
↪bacula/archive).
JobId 141: docker: Docker local restore: container/mcache1/b97d4dd8806(...)
```

The restore job log will show that the restore was done to a local directory. The log above was truncated for a clear view.

18 Other

19 Resource listing

The Bacula Enterprise Docker Plugin supports the new Plugin Listing feature of Bacula Enterprise 8.x or newer. This mode allows a Plugin to display some useful information about available Docker resources such as:

- List of Docker containers name-labels
- List of Docker images name-repository
- List of Docker volumes

The new feature uses the special **.ls** command with a new **plugin=<plugin>** parameter. The command requires the following parameters to be set:

client=<client> A Bacula Client name with the docker Plugin installed.

plugin=<plugin> A Plugin name, which would be **docker:** in this case, with optional plugin parameters as described in section **genericparameters**.

path=<path> An object path to display.

The supported values for a **path=<path>** parameter are:

/ to display object types available to list

container to display a list of containers name-labels data

image to display a list of images name-repository data

volume to display a list of volume names data

To display available object types, follow the following command example:


```
*.ls plugin=docker: client=docker-test-fd path=/
Connecting to Client docker-test-fd at docker-test:9102
drwxr-x---   1 root     root               0 2018-09-28 14:32:20  image
drwxr-x---   1 root     root               0 2018-09-28 14:32:20  container
drwxr-x---   1 root     root               0 2018-09-28 14:32:20  volume
2000 OK estimate files=3 bytes=0
```

To display the list of all available Docker containers, the following command example can be used:

```
*.ls plugin=docker: client=docker-test-fd path=container
Connecting to Client docker-test-fd at docker-test:9102
root     root        88185293 2018-10-01 09:18:00  myubuntu -> 4d0a4fadb50d
root     root        61656268 2018-10-01 09:18:00  mcache1_98 -> cb0c2e54dd00
root     root        88185240 2018-10-01 09:18:00  my_docker -> 4eefcf7d61ee
root     root        88185240 2018-10-01 09:18:00  amazing_hamilton -> 73ce08ad3d59
root     root        88185240 2018-10-01 09:18:00  with.label -> 1f476fd3c1b1
root     root        61656268 2018-10-01 09:18:00  mcache1 -> b97d4dd88063
root     root       239075328 2018-10-01 09:18:00  brave_edison -> 66f45d8601ba
root     root       239075391 2018-10-01 09:18:00  some-postgres -> 28e4c3a3cd27
root     root       580911104 2018-10-01 09:18:00  my-ubuntu -> a6ba1cb597d5
root     root        88185245 2018-10-01 09:18:00  sharp_visvesvaraya -> ce5844df6842
root     root       239075391 2018-10-01 09:18:00  my-postgres -> a7c9518405e8
root     root        88185240 2018-10-01 09:18:00  frosty_kowalevski -> 0f601bcb1ef5
root     root        88185302 2018-10-01 09:18:00  infallible_bose -> 00571da76da6
root     root            1894 2018-10-01 09:18:00  romantic_hermann -> 37285d94347a
2000 OK estimate files=14 bytes=2,038,748,444
```

To display the list of all available Docker images, use the following command example:

```
*.ls plugin="docker:" client=docker-test-fd path=image
Connecting to Client docker-test-fd at docker-test:9102
root     root        61656268 2018-10-01 09:19:34  memcached:latest -> 80256dbd25ae
root     root       239075328 2018-10-01 09:19:34  postgres:latest -> ac25c2bac3c4
root     root        88185240 2018-10-01 09:19:34  ubuntu:latest -> 16508e5c265d
root     root            1894 2018-10-01 09:19:34  hello-world:latest -> 2cb0d9787c4d
2000 OK estimate files=4 bytes=0
```

The **container** and lists display an estimated size of Docker objects. The final size of backup could be different.

All above Docker objects lists use a short (truncated) object ids during display. It is technically plausible that you'll get the same truncated id for two or more objects. To properly distinguish them you can use `notrunc` plugin parameter. In this case all Docker objects ids will be displayed in full sha256 form, i.e.

```
*.ls plugin="docker: notrunc" client=docker-test-fd path=image
Connecting to Client docker-test-fd at docker-test:9102
(...) memcached:latest ->␣
↪80256dbd25aedb4c025d495f18e0e513dd011005726326f529c296e4141811f6
(...) postgres:latest -> ac25c2bac3c4e56f949c60ca343e1c4cd95f493db28bbf29b8ce466b4171cc8f
(...) ubuntu:latest -> 16508e5c265dcb5c05017a2a8a8228ae12b7b56b2cda0197ed5411bda200a961
(...) hello-world:latest ->␣
↪2cb0d9787c4dd17ef9eb03e512923bc4db10add190d3f84af63b744e353a9b34
2000 OK estimate files=4 bytes=0
```

This plugin parameter works only in plugin listing mode and does not affect any backup, estimate or restore jobs. In this case you can get *Invalid parameter* job error.

To display the list of all available Docker volumes, the following command example can be used:

```
*.ls plugin=docker: client=docker-test-fd path=volume
Connecting to Client docker-test-fd at docker-test:9102
root  root 2019-07-19 18:13:41 ␣
↪a563ae0edf55a01a0cdb3165d854a7326f13793119a708c44a5e49bd72a0286d
root  root 2019-07-19 18:13:41 ␣
↪aa9d3074f8c65a5afafddc6eaaf9827e99bb51f676aafaacc05cfca0188e65bf
root  root 2019-07-19 18:13:41 ␣
↪b49a3607eb04a4d3d00ed9dc0910e12201ad85f6ca38cb8fc7b43333207203a9
root  root 2019-07-19 18:13:41 ␣
↪bd80f277bd000d493dde4de763b0e82f1c0d5fd760acc07cb5c971f74a314471
root  root 2019-07-19 18:13:41 ␣
↪c0a478d317195ba27dda1370b73e5cb94a7773f2a611142d7dff690abdcfdcbf
root  root 2019-07-19 18:13:41 ␣
↪e3f25066e7957a12f084de87686c37fdb954ab2c15068369d887d3718b860a4c
root  root 2019-07-19 18:13:41  my-vol
root  root 2019-07-19 18:13:41  testvolume
2000 OK estimate files=8 bytes=0
```

The **volume** list display a volume real size if this information is available from Docker else it display simple 0. The final size of backup could be different.

20 Limitations

- Granular restore (*Single Item Restore*) is not supported. This feature could be implemented in the future.
- Only Full level backups are possible. This is a Docker limitation for containers and images. For volume backup this limitation could be removed in the future.
- You can restore volume files data into the same volume name only. This limitation could be removed in the future.
- You will always overwrite restored volume files data regardless of **Replace** parameter value. This limitation will be removed in the future.
- Backup or restore of the Volume data files for remote Docker service is unsupported. In this case you should get a following warning when want to explicitly backup a such data:

  ```
  Warning: dkcommctx: Docker Volume backup with docker_host is unsupported!
  ```

  and a following warning when want to restore:

  ```
  Warning: docker: Docker Volume restore with docker_host is unsupported! \
  All volumes restore skipped.
  ```

  If you access the remote Docker service outside plugin configuration then your job might hung. This limitation will be removed in the future.
- The `restart` command has limitations with plugins, as it initiates the Job from scratch rather than continuing it. Bacula determines whether a Job is restarted or continued, but using the `restart` command will result in a new Job.

