Virt Backup

Release 0.5.0

Jan 05, 2021

Contents:

virt-backup does hot external backups of your Libvirt guests, using the BlockCommit feature. The goal is to do an automatic backup system, with optional compression, and be able to easily restore a backup.

virt-backup is based around groups: a group contains a list of domains to backup, that can be matched by regex. Each group contains its own configuration, specifying how to store the backups (compression, directory, etc.), where to store them, the retention by period of time when a cleanup is called, etc.

CHAPTER 1

Features

- Hot backup one or multiple qemu/raw disk, snapshoting everything at the same time.
- Cold backup a qemu/raw disk.
- Multithreading: can backup multiple domains in parallel.
- Supports multiple targets for backups:
  - Directory: just copies images in a directory.
  - Tar: stores all images of a backup in a tar file (with optional xz/gz/bzip2 compression).
  - ZSTD: compresses the images using ZSTD algorithm (supports multithreading).
- Restore a backup to a folder.
- List all backups, by VM name.
- Clean backup, with configurable time retention (number of backups to keep, per domain, per hours/day/weeks/months/years)

CHAPTER 2

Limitations

- Only supports file type disks (qemu, raw, etc.). Does not support LVM or any block disk.
- Does not handle Libvirt external snapshots. BackingStores are just ignored and only the current running disk is backup.
- virt-backup has to run on each hypervisor. It has to be able to read the disks in order to backup them, and it uses the same disk path as configured in Libvirt.

2.1 Quickstart

virt-backup has 4 main functions:

- *backup*
- *list backups*
- *restore*
- *clean backups*

This page describes how to install virt-backup, create a generic configuration then how to use these 4 functions.

Table of Contents

- *Clean*

2.1.1 Installation

Run:

```
pip3 install virt-backup
```

Or by using setuptools:

```
python3 ./setup.py install
```

virt-backup is Python 3 compatible only.

2.1.2 Configuration

virt-backup is based around the definition of groups. Groups can include or exclude as many domains as needed, and define the backup properties: compression, disks to backup, where to store the backups, retention, etc..

Groups definition is the biggest part of the configuration.

The configuration is a yaml file. Here is a quite generic one:

```
---

#########################
#### Global options ####
#########################

## Be more verbose ##
debug: False

## How many threads (simultaneous backups) to run. Use 0 to use all CPU threads
## detected, 1 to disable multitheading for backups, or the number of threads
## wanted. Default: 1
threads: 1


############################
#### Libvirt connection ####
############################

## Libvirt URI ##
uri: "qemu:///system"

## Libvirt authentication, if needed ##
username:
passphrase:


########################
#### Backup groups ####
########################

## Groups are here to share the same backup options between multiple domains.
```

(continues on next page)

(continued from previous page)

```
## That way, it is possible, for example, to have a different policy retention
## for a pool of guests in testing than for the one in production.

## Define default options for all groups. ##
## Here we set the retention parameters for each VM when calling `virt-backup clean`.
default:
  hourly: 1
  daily: 4
  weekly: 2
  monthly: 5
  yearly: 1

## Groups definition ##
groups:
  ## Group name ##
  test:
    ## Backup directory ##
    target: /mnt/kvm/backups

    ## Use ZSTD compression, configured at lvl 6
    packager: zstd
    packager_opts:
      compression_lvl: 6

    ## When doing `virt-backup backup` without specifying any group, only
    ## groups with the autostart option enabled will be backup.
    autostart: True

    ## Enable the Libvirt Quiesce option when taking the external snapshots.
    ##
    ## From Libvirt documentation: libvirt will try to freeze and unfreeze the guest
    ## virtual machine's mounted file system(s), using the guest agent. However, if␣
↪the
    ## guest virtual machine does not have a guest agent, snapshot creation will fail.
    ##
    ## However, virt-backup has a fallback mechanism if the snapshot happens to fail
    ## with Quiesce enabled, and retries without it.
    quiesce: True

    ## Hosts definition ##
    hosts:
      ## Will backup everything.
      - "r:.*"

# vim: set ts=2 sw=2:
```

Adapt it and save it either as:

- `~/.config/virt-backup/config.yml`
- `/etc/virt-backup/config.yml`

2.1.3 Backup

All groups set with the *autostart* option to *True* can be started by running:

```
$ virt-backup backup
```

A specific group (`test`) can be started by running:

```
$ virt-backup backup test
```

The group has to be defined in the configuration.

Multiple groups can be ran with:

```
$ virt-backup backup group1 group2 [...]
```

2.1.4 List

To list the backups for all groups, as a summary:

```
$ virt-backup list

 generic
=========

Total backups: 2 hosts, 22 backups
Hosts:
    vm-foo-0: 11 backup(s)
    vm-bar-0: 11 backup(s)

 test
======

Total backups: 1 hosts, 11 backups
Hosts:
    vm-foo-1: 11 backup(s)
```

To have a really short summary for all groups:

```
$ virt-backup list -s

 generic
=========

Total backups: 9 hosts, 99 backups

 test
======

Total backups: 1 hosts, 11 backups
```

By default, only domains with at least one backup will be listed, but all domains matching with the group rules can be printed by using the `-a/--all` option.

To list exactly all the backups done for one domain, here `vm-foo-0`:

```
$ virt-backup list -D vm-foo-0

 generic
=========
```

(continues on next page)

(continued from previous page)

```
vm-foo-0: 11 backup(s)
    2020-09-17T01:02:53+00:00: /backups/vm-foo-0/20200917-010253_8_vm-foo-0.json
    2020-09-16T01:02:56+00:00: /backups/vm-foo-0/20200916-010256_8_vm-foo-0.json
    2020-09-15T01:02:39+00:00: /backups/vm-foo-0/20200915-010239_8_vm-foo-0.json
    2020-09-14T01:02:34+00:00: /backups/vm-foo-0/20200914-010234_8_vm-foo-0.json
    2020-09-07T01:03:07+00:00: /backups/vm-foo-0/20200907-010307_8_vm-foo-0.json
    2020-09-01T01:02:22+00:00: /backups/vm-foo-0/20200901-010222_8_vm-foo-0.json
    2020-08-01T01:02:20+00:00: /backups/vm-foo-0/20200801-010220_8_vm-foo-0.json
    2020-07-01T00:55:01+00:00: /backups/vm-foo-0/20200701-005501_3_vm-foo-0.json
    2020-06-01T00:55:02+00:00: /backups/vm-foo-0/20200601-005502_3_vm-foo-0.json
    2020-05-01T00:55:01+00:00: /backups/vm-foo-0/20200501-005501_3_vm-foo-0.json
    2020-04-01T00:55:01+00:00: /backups/vm-foo-0/20200401-005501_3_vm-foo-0.json
```

Which lists when the backup was taken, and where its definition file is stored. If the domain matches multiple groups, backups will be listed per group.

2.1.5 Restore

To restore the last backup of a domain (`vm-foo-0`) part of a given group (`generic`), and extract the result in the given target destination (`~/disks`):

```
$ virt-backup restore generic vm-foo-0 ~/disks
```

Which extracts everything backuped to `~/disks`.

To extract a specific backup, its date can be specified (`2020-09-17T01:02:53+00:00`):

```
$ virt-backup restore --date 2020-09-17T01:02:53+00:00 generic vm-foo-0 ~/disks
```

The format is for the moment non convenient and some work will be needed to facilitate it. For the moment, the exact date and format as given by `virt-backup list` has to be used.

2.1.6 Clean

It is possible to automatically clean old backups, by following the configured *rentention policy*, but also broken backups (for which the backup process was not correctly interrupted, by a crash or server shutdown for example).

To clean old and broken backups for all groups:

```
$ virt-backup clean
```

To limit the cleaning to one group only (`test`):

```
$ virt-backup clean test
```

To only clean the broken backups, but not handle the old (correct) backups:

```
$ virt-backup clean -b
```

Opposite situation, to not clean the broken backups but only handle the old (correct) backups:

```
$ virt-backup clean -B
```

A systemd service is available in example/virt-backup-clean.service to trigger a cleaning of all broken backups at start. This way, if the hypervisor crashed during a backup, the service will clean all temporary files and pivot all disks to their original images (instead of running on a temporary external snapshot).

2.2 Configuration

This page describes how to configure virt-backup and goes in detail for each section.

Table of Contents

2.2.1 Full example

The configuration is a yaml file virtually split into 3 main sections: the global options, libvirt connection and backup groups. Here is a full example:

```
---

#########################
#### Global options ####
#########################

## Be more verbose ##
debug: False

## How many threads (simultaneous backups) to run. Use 0 to use all CPU threads
## detected, 1 to disable multitheading for backups, or the number of threads
## wanted. Default: 1
threads: 1


############################
#### Libvirt connection ####
############################

## Libvirt URI ##
uri: "qemu:///system"

## Libvirt authentication, if needed ##
username:
passphrase:
```

(continues on next page)

(continued from previous page)

```
########################
#### Backup groups ####
########################

## Groups are here to share the same backup options between multiple domains.
## That way, it is possible, for example, to have a different policy retention
## for a pool of guests in testing than for the one in production.

## Define default options for all groups. ##
default:
  hourly: 1
  daily: 4
  weekly: 2
  monthly: 5
  yearly: 1

## Groups definition ##
groups:
  ## Group name ##
  test:
    ## Backup directory ##
    target: /mnt/kvm/backups

    ## Packager to use for each backup:
    ##   directory: images will be copied as they are, in a directory per domain
    ##   tar: images will be packaged in a tar file
    ##   zstd: images will be compressed with zstd. Requires python "zstandard"␣
↪package to be installed.
    packager: tar

    ## Options for the choosen packager:
    ## tar:
    ##   # Compression algorithm to use. Default to None.
    ##   compression: None | "xz" | "gz" | "bz2"
    ##   # Compression level to use for each backup.
    ##   # Generally this should be an integer between 1~9 (depends on the
    ##   # compression algorithm), where 1 will be the fastest while having
    ##   # the lowest compression ratio, and 9 gives the best compression ratio
    ##   # but takes the longest time to compress.
    ##   compression_lvl: [1-9]
    ##
    ## zstd:
    ##   # Compression level to use for each backup.
    ##   # 1 will be the fastest while having the lowest compression ratio,
    ##   # and 22 gives the best compression ratio but takes the longest time
    ##   # to compress.
    ##   compression_lvl: [1-22]
    packager_opts:
      compression: xz
      compression_lvl: 6

    ## When doing `virt-backup backup` without specifying any group, only
    ## groups with the autostart option enabled will be backup.
    autostart: True

    ## Retention policy: the first backup of the day is considered as the
    ## "daily" backup, first of the week "weekly", etc. The following options
```

(continues on next page)

(continued from previous page)

```
    ## detail how many backups of each type has to be kept. Set to "*" or None for an
    ## infinite retention.
    ## Default to 5 for everything, meaning that calling "virt-backup clean" will let␣
↪5
    ## backups for each period not specified in the config.
    hourly: 5
    daily: 5
    weekly: 5
    monthly: 5
    yearly: 1

    ## Enable the Libvirt Quiesce option when taking the external snapshots.
    ##
    ## From Libvirt documentation: libvirt will try to freeze and unfreeze the guest
    ## virtual machine's mounted file system(s), using the guest agent. However, if␣
↪the
    ## guest virtual machine does not have a guest agent, snapshot creation will fail.
    ##
    ## However, virt-backup has a fallback mechanism if the snapshot happens to fail
    ## with Quiesce enabled, and retries without it.
    quiesce: True

    ## Hosts definition ##
    hosts:
      ## This policy will match the domain "domainname" in libvirt, and will
      ## backup the disks "vba" and "vdb" only.
      - host: domainname
        disks:
          - vda
          - vdb
        ## Quiesce option can also be overriden per host definition.
        quiesce: False
      ## Will backup all disks of "domainname2" ##
      - domainname2
      ## Regex that will match for all domains starting with "prod". The regex
      ## syntax is the same as the python one
      - "r:^prod.*"
      ## Exclude the domain domainname3 (useful with regex, for example)
      - "!domainname3"
      ## Exclude all domains starting with "test"
      - "!r:^test.*"

# vim: set ts=2 sw=2:
```

It can be saved as (the order defines the priority of the import):

- `~/.config/virt-backup/config.yml`
- `/etc/virt-backup/config.yml`

2.2.2 Global options

They define the global behavior of virt-backup:

- `debug`: if `True`, virt-backup is more verbose. Enable this option (or use the global *-d* command line option) for bug reports. (Optional, default: `False`)

- `threads`: how many simultaneous backups to run. Set it to the number of threads wanted, or 1 to disable multithreading, or 0 to use all CPU threads detected. (Optional, default: `1`)

2.2.3 Libvirt connection

They define the options to connect to libvirt:

- `uri`: libvirt URI: https://libvirt.org/uri.html
- `username`: connection username. (Optional)
- `password`: connection password. (Optional)

virt-backup can technically connect to a distant Libvirt, but in order to actually backup the domain disks, it has to have access to the files. Therefore, it should run on the same hypervisor than Libvirt.

2.2.4 Backup groups

Groups domains allow to share the same backup options between multiple domains. This way, it is possible to define for example a different retention set or compression for a pool of domains in production than one in testing.

- `default`: dictionary containing all the default options for the groups. If a group redefines an option, it overrides it.
- `groups`: dictionary defining the groups. Groups are defined per names, and are themselves dictionary defining their options.

Group options

- `target`: backup directory.
- `packager`: which packager to use. Read the *Packagers section* for more info.
- `packager_opts`
- `autostart`: if `True`, this group will be automatically backup when doing `virt-backup backup` without the need of specifying it. Otherwise, if set to `False`, it needs to be specifically called (`virt-backup backup foo bar`).
- `hourly`, `daily`, `weekly`, `monthly`, `yearly`: retention policy. Read the *Retention section* for more info.
- `quiesce`: Enable the Libvirt Quiesce option when taking the external snapshots.

  From Libvirt documentation: libvirt will try to freeze and unfreeze the guest virtual machine's mounted file system(s), using the guest agent. However, if the guest virtual machine does not have a guest agent, snapshot creation will fail.

  However, virt-backup has a fallback mechanism if the snapshot happens to fail with Quiesce enabled, and retries without it.
- `hosts`: domains to include in this group. Read the *Hosts section* for more info.

Packagers

Packagers define the storage mechanism. The existing packagers are:

- `directory`: images will be copied as they are, in a directory per domain
- `tar`: images will be packed into a tar file

- `zstd`: images will be compressed with zstd. Requires python `zstandard` library to be installed.

Then, depending on the packager, some options can be set.

Tar options:

- `compression`: set the compression algorithm for the tar archive. (Valid options: `None` | `xz` | `gz` | `bz2`, default: `None`)
- `compression_lvl`: set the compression level for the given algorithm. Generally this should be an integer between 1 and 9 (depends on the compression algorithm), where 1 will be the fastest while having the lowest compression ratio, and 9 gives the best compression ratio but takes the longest time to compress.

  For more info, read https://docs.python.org/3/library/tarfile.html.

ZSTD options:

- `compression_lvl`: set the compression level, between 1 and 22. 1 will be the fastest while having the lowest compression ratio, and 22 gives the best compression ratio but takes the longest time to compress.

Hosts

The `hosts` option contain a list of domains to match for this group. Each item of this list can also limit the backup to specific disks, and override different options.

To only do host matching:

```
hosts:
  # Will backup all disks of "domainname2"
  - domainname2
  # Regex that will match for all domains starting with "prod". The regex syntax is␣
↪the same as the python one
  - "r:^prod.*"
  # Exclude the domain domainname3 (useful with regex, for example)
  - "!domainname3"
  # Exclude all domains starting with "test"
  - "!r:^test.*"
```

To do a more detailed definition, and limit the host to only a list of disks:

```
hosts:
  - host: domainname
    disks:
      - vda
      - vdb
    ## Quiesce option can also be overriden per host definition.
    quiesce: False
  # It can still also be a regex.
  - host: "r:^prod.*"
    disks:
      - vda
```

As shown in the example, exclusion is possible by adding `!`. The order of definition does not matter, and exclusion will always take precedence over the inclusion.

Retention

The available retention options define how many backups to keep per period when cleaning this group. The available time periods are:

- `hourly`
- `daily`
- `weekly`
- `monthly`
- `yearly`

The default value is `5` for everything.

The first backup of the hour is called an `hourly` backup, first of the day is `daily`, etc. Setting `daily` to `2` would mean to keep the first backups of the day of the last 2 days. `weekly` to `2` would mean to keep the first backup of the week of the last 2 weeks.

The last 2 days/weeks/etc. is here a simplification in the explanation. Please read the *backups cleaning documentation* to get a full explanation of the cleaning process.

2.3 Backup

This page describes how the backup process works.

Table of Contents

2.3.1 Principle

- A complete backup is defined by its definition. A definition is a json file, stored next to the backup, containing informations such as the domain name, the disk backups, path to the backup, etc. This is the file listed when doing `virt-backup list -D domain`.
- A pending backup is defined by its `pending_info`. The `pending_info` is a definition with some additional attributes computed when running the backup. It is stored next to the backup, and removed when the backup is complete. It is used to rebuild a temp backup if a crash happened, and clean everything.

2.3.2 How it works

When backuping multiple groups, first virt-backup will build all the groups with the given rules, then merge it into one. It allows to have a unique entry point to start everything, and deduplicate the similar backups. Read the *groups unicity section* for more details.

If multithreading is disabled, it then starts the backups one by one. However, if multithreading is enabled, a safety mechanism is followed if multiple backups target the same domain. Read the *groups multithreading section* for more details.

Then, each backups are started. For each backup, the first step is to create the backup directory, and take an external snapshot of all targeted disks (read the *domain external snapshot section* for more details). This method is used in order to freeze the disks by the time virt-backup backup them, and then merge them back with the main disks and pivot the domains like before. There is however multiple inconvenient for that: if the VM is doing a lot of "remove" (freeing blocks), it's more operations as the external snapshot needs to log it. And it obviously requires temporarily more space.

Then the pending info are dumped on disk, where the backup should be. This step allows to be able to clean the backup if virt-backup would happen to crash (by using `virt-backup clean`). It contains the snapshot names and different informations that are known only when starting the backups.

Now that the disks are frozen, they can be safely copied somewhere. This somewhere is defined by the packager (see the `virt_backup.backups.packagers` package). A packager is a way to store a backup, and expose a standard API so the backup does not have to care about it. Each disks are copied sequentially into the packager.

The definition is dumped again, with all the final info. The pending info are removed, the external snapshots are cleaned (meaning for each snapshot, a blockcommit is triggered, the external snapshot is removed, the disk is pivot).

If anything goes wrong during the backup, the external snapshot is cleaned, the pending info are removed such as everything created for the backup (only the backup directory is left).

2.3.3 Groups

Unicity

If multiple groups are backup and some share the same domains to backup, virt-backup will try to see if the backups could be compatible to avoid doing the exact same backup multiple times.

Example of a groups configuration:

```
groups:
  group1:
    target: /mnt/kvm/backups

    packager: zstd
    packager_opts:
      compression_lvl: 6

    ## Hosts definition ##
    hosts:
      - "test1"

  group2:
    target: /mnt/kvm/backups

    packager: zstd
    packager_opts:
```

(continues on next page)

(continued from previous page)

```
    compression_lvl: 6

  ## Hosts definition ##
  hosts:
    - "r:test.*"

group3:
  target: /mnt/kvm/backups_disk1_only

  packager: tar

  ## Hosts definition ##
  hosts:
    - name: "test1"
      disks:
        - disk1
```

Here *group1* and *group2* will try to backup the domain *test1* with all its disks, with the same compression parameters and to the same target directory. Therefore, *test1* can only be backup once.

However, *group3* specifies that only the disk *disk1* of *test1* has to be backup, and put it in a tarfile in a different target directory. It is not considered as compatible with what *group1* and *group2* specify, therefore it will be backup a second time.

Running a backup with this configuration will do 2 backups for *test1*: one shared between *group1* and *group2*, one for *group3*.

Multithreading

Backuping a group can be done in single thread or multithread. As a group can contain the same domain with different options, some safety have been done to avoid backuping the same domain in parallel. It is needed as the process relies on external snapshot, doing so would take an external snapshot of a snapshot (with the current implementation).

As it is considered to be a rare case, all backups targeting the same domain are scheduled in a queue. If other domains are to backup, the backups in these queues are normally handled in parallel of other backups.

2.3.4 Domain external snapshot

A custom helper is implemented to handle the external snapshots (see the `virt_backup.backups.snapshot` package). It uses libvirt to create it, then allows to remove it and pivot back to the main disk by using blockcommit (read this libvirt example for more details).

Quiesce is an option when creating the snapshot. It allows to communicate with the virt agent present on the domain to force a sync of the disk before taking the snapshot. If Quiesce is wanted, when doing the snapshot, it first tries to do it with this option. If it fails, because for example there is no virt-agent running on this domain, it fallbacks on a snapshot without Quiesce (but logs an error).

Pivoting back to the main disk depends if the domain is up or not. Libvirt does not allow a blockcommit on a shutdown domain. In this case, `qemu-img` is used directly to manually handle the blockcommit. Otherwise, libvirt API is used.

To blockcommit, libvirt uses an event mechanism. Libvirt takes a function that it will call if there is an issue with the blockcommit, or if it's done. To centralize it, a custom helper `DomExtSnapshotCallbackRegistrer` is used (see the `virt_backup.backups.snapshot` package). It stores the callback to call per snapshot path, so when libvirt calls the register as a callback, it then look for the known snapshots and call the function to trigger a pivot. This function is handled by the `DomExtSnapshot`, which aborts the blockjob and removes the snapshot.

2.3.5 Packagers

Packagers in virt-backup are a common way to deal with storage. They are defined in the `virt_backup.backup.packagers` package. A packager can provide an abstracted way to deal with a folder, archive or else.

Each packager is split in 2:

- Read packager, inherited from `virt_backup.backup.packagers._AbstractReadBackupPackager`. Provides mechanisms to list backups from a packager and restore a specific backup (by copying it to a given path).
- Write packager, inherited from `virt_backup.backup.packagers._AbstractWriteBackupPackager`. Provide mechanisms to add a new backup in a packager, delete the package and, when possible, remove a specific image from a backup. When the package is shareable between backups (for example, with a folder storing all the images of a domain), it also provide a way to remove a specific backup from the package.

Splitting in read/write allows more safety when dealing with backups: the write packager is used only when the backup mechanism absolutely needs it, otherwise the read packager is used.

Available packagers are:

- `directory`: store the images directly in a directory. Can be a directory per backup, or a directory shared for multiple backups.
- `tar`: store the backups in a tar archive. Can handle compression.
- `zstd`: store the backups in a zstd archive. Compression level is customizable. Can also handle multithreading for the compression itself.

2.4 Data maps

This page lists the custom data defined and used by virt-backup, and their schema.

Table of Contents

2.4.1 Compatibility layers

In order to ensure that virt-backup can read old backups, old configurations and pending datas, it uses compatibility layers.

Compatibility layers are defined in `virt_backup.compatibility_layers`, and each data has its own package.

Compatibility layers can use a range of version if the data allows it. A configuration doesn't define any version for example, so its compatibility layers will be executed iteratively. However, Definitions and Pending Info contained a version, therefore only the compatibility layers between its version and the last one will be ran.

Depending the data, warnings can be shown to the user to apply the migrations themselves. Configuration for example will indicate the needed steps to migrate the configuration file. Things will still run if it is not migrated, but the support of old configurations can be dropped in the future.

To ensure that old data can be migrated to a last wanted state, some tests run all the compatibility layers (`tests/test_compat_layers_*`).

2.4.2 Configuration

The configuration file is a yaml file used by virt-backup in `virt_backup.config.Config`:

```
# Be more verbose.
# Default: False
debug: bool

# How many threads (simultaneous backups) to run. Use 0 to use all CPU threads
# detected, 1 to disable multitheading for backups, or the number of threads wanted.
# Default: 1
threads: int


############################
#### Libvirt connection ####
############################

# Libvirt URI.
uri: str

# Libvirt authentication, if needed.
username: str
passphrase: str


#######################
#### Backup groups ####
#######################

# Groups are here to share the same backup options between multiple domains.
# That way, it is possible, for example, to have a different policy retention
# for a pool of guests in testing than for the one in production.

# Define default options for all groups.
default:
  target: str
  packager: str
  packager_opts: dict{packager_option: value}
  quiesce: bool
  hourly: int
  daily: int
  weekly: int
  monthly: int
  yearly: int

# Groups definition.
groups:
  # Group name
  str:
```

(continues on next page)

(continued from previous page)

```
# Backup directory.
target: str

# Packager to use for each backup:
packager: str

# Options for the choosen packager:
packager_opts: dict{packager_option: value}

# When doing `virt-backup backup` without specifying any group, only groups with
# the autostart option enabled will be backup.
# Default: False
autostart: bool

# Retention policy: the first backup of the day is considered as the
# "daily" backup, first of the week "weekly", etc. The following options
# detail how many backups of each type has to be kept. Set to "*" or None for an
# infinite retention.
# Default:
#  hourly: 5
#  daily: 5
#  weekly: 5
#  monthly: 5
#  yearly: 5
hourly: int
daily: int
weekly: int
monthly: int
yearly: int

# Enable the Libvirt Quiesce option when taking the external snapshots.
#
# From Libvirt documentation: libvirt will try to freeze and unfreeze the guest
# virtual machine's mounted file system(s), using the guest agent. However, if the
# guest virtual machine does not have a guest agent, snapshot creation will fail.
#
# However, virt-backup has a fallback mechanism if the snapshot happens to fail
# with Quiesce enabled, and retries without it.
quiesce: bool

# Hosts definition.
hosts:
  # Can either be a dictionary or a str.
  - host: str
    disks: []str
    quiesce: bool
  # If a str, can be the domain name, or a regex.
  - str
```

2.4.3 Backup definition

A backup definition is a JSON file defining a backup. It is stored next to the backup package to quickly get all the needed information about it, without the need of unpacking anything:

```
{
    name: str,
    domain_id: int,
    domain_name: str,
    // Dump of the libvirt definition of the targeted domain.
    domain_xml: str,
    disks: { disk_name <str>: backup_disk_name <str> },
    version: str,
    date: int,
    packager: {
        type: str,
        opts: {},
    },
}
```

Example:

```
{
    "name": "20191001-003401_3_test-domain",
    "domain_id": 3,
    "domain_name": "test-domain",
    "domain_xml": "<domain type='kvm' id='3'></domain>",
    "disks": {
        "vda": "20191001-003401_3_test-domain_vda.qcow2",
    },
    "version": "0.4.0",
    "date": 1569890041,
    "packager": {
        "type": "tar",
        "opts": {
            "compression": "gz",
            "compression_lvl": 6,
        },
    },
}
```

2.4.4 Pending data

Pending data is a temporary backup definition, following the same structure but with a bit more information in order to clean everything if something failed:

```
{
    name: str,
    domain_id: int,
    domain_name: str,
    // Dump of the libvirt definition of the targeted domain.
    domain_xml: str,
    disks: {
        disk_name <str>: {
            src: str,
            snapshot: str,
            target: str,
        }
    },
    version: str,
    date: int,
```

(continues on next page)

(continued from previous page)

```
    packager: {
        type: str,
        opts: {},
    },
}
```

Example:

```
{
    "name": "20191001-003401_3_test-domain",
    "domain_id": 3,
    "domain_name": "test-domain",
    "domain_xml": "<domain type='kvm' id='3'></domain>",
    "disks": {
        "vda": {
            "src": "/tmp/test/vda.qcow2",
            "snapshot": "/tmp/test/vda.qcow2.snap",
            "target": "20191001-003401_3_test-domain_vda.qcow2",
        },
    },
    "version": "0.4.0",
    "date": 1569890041,
    "packager": {
        "type": "tar",
        "opts": {
            "compression": "gz",
            "compression_lvl": 6,
        },
    },
}
```

The structure is the closest as possible from the backup definition.

2.5 Backups cleaning

This page describes how the process of cleaning the backups works.

Table of Contents

2.5.1 Remove a specific backup

Not implemented yet in virt-backup. Removing a backup needs to be done manually, by removing the files.

2.5.2 Clean outdated backups

To be documented.

For more details about the retention period, for now please read this comment in a github issue: https://github.com/aruhier/virt-backup/issues/38#issuecomment-659590425

CHAPTER 3

Indices and tables

- genindex
- modindex
- search