QEMU Plugin

Bacula Systems Documentation

Contents

Contents

Important: Remember to read the Best Practices chapter common for all of our hypervisor plugins.

---

1 Features Overview

- Transaction based online backup of any running QEMU VM guests
- Full and Incremental virtual disk image-level backup
- VM guest configuration for easy recovery
- Ability to restore virtual disk images as QCOW2 files
- Ability to completely restore a Proxmox VM guest, including configuration
- Ability to completely restore a target disk images for Genuine QEMU
- Ability to restore VM virtual disk images to an alternate directory

2 Guest VM Backup Strategies

2.1 Installing a Bacula Client on Each Guest VM

Using this strategy, you do not use the Bacula Enterprise QEMU Plugin, but instead install a Bacula Enterprise File Daemon on every virtual machine as if they were normal physical clients. In order to optimize the I/O usage on your QEMU system, you will use Bacula's Schedules, Priorities, and Job concurrency settings to spread backup jobs throughout the backup window. Since all guest VMs could possibly reside on the same storage on the Proxmox hypervisor, running all your backup jobs at the same time can create a bottleneck on the disk/network subsystem.

Installing a Bacula Enterprise File Daemon in each virtual machine allows you to manage your virtual servers like physical servers and also to use all Bacula Enterprise's features such as:

- Quick restores of individual files
- Checksum of individual files for Virus and Spyware detection
- Verify Jobs
- File or Directory exclusion (such as swap or temporary files)
- File level compression
- Accurate backups
- Plugins

2.2 Image Backup with QEMU Plugin

With the image level backup strategy, the Bacula Enterprise QEMU Plugin will save VM disks as QCOW2 images for QEMU VMs for both Full and Incremental backups.

For this to work, a Bacula File Daemon is not needed in each guest VM. The Bacula QEMU Plugin will contact the QEMU hypervisor to read and save the contents of virtual machine disks using the QMP transaction backups feature and dump them using the QMP API.

Bacula does not need to walk through the Client filesystems to stast, open, read, and close files, so it consumes less resources on the QEMU infrastructure than a file level backup on each VM would. On the other hand, Bacula will also read and save useless data in the VMs such as swap files or temporary files.

Tip: The QEMU Plugin will save not only the disk images of the guest VMs, but also guest VM configurations which allows for very easy guest VM restores. This feature is available for Proxmox infrastructure only.

---

3 Backup and Restore Operations

3.1 Backup

The QEMU Plugin supports the QMP control protocol which is a low-level managing interface for every QEMU Hypervisor flavor. In most cases this interface is used by Hypervisor management tools to manage QEMU VM Guests.

The current Plugin version supports two QEMU Hypervisor types:

- Genuine QEMU - `mode=QEMU` (*the default*)
- Proxmox (PVE) - `mode=PVE`

The core backup and restore procedure is the same for every QEMU hypervisor type. The main difference is the way a VM is selected to be backed up and how the restore process is finished.

The core backup of a single guest VM consists of the following steps:

1. Query VM information and save it
2. For a Full backup level, during the backup transaction, create or clear Block dirty bitmap and dump VM disk image to the working area (check *Working Directory Location*)
3. For an Incremental backup level, during the backup transaction, dump the incremental VM disk image to the working area and clear the dirty bitmap
4. When a VM disk image dump is ready, the Bacula File Daemon will send the data to the Storage Daemon

Backups can be performed for guest VMs in a `running` state only. Any first Full level backup will create a backup chain used for subsequent Incremental backups. The current plugin version creates and maintains a single backup chain for each VM.

Warning: You should never mix different backup jobs for a single QEMU VM as it will break the backup chain and backups become unrecoverable!

The QEMU Plugin will log the start and end of each guest VM backup:


```
JobId 104: Start Backup JobId 104, Job=proxmox.2021-10-28_12.04.54_42
JobId 104: Recycled volume "vol01"
JobId 104: Using Device "FileChgr1-Dev2" to write.
JobId 104: qemu: Connected to Proxmox pve-manager/6.4-13/9f411e79 (running kernel: 5.4.
↪143-1-pve)
JobId 104: qemu: Start Backup vm: vm1 (100)
JobId 104: qemu: Finish backup of vm1/drive-scsi0
...
```

The backup will create a single (`.qcow2`) file for every VM disk device saved and a single VM configuration file (`config.json`) during a Full backup.

- `/@qemu/<vm-name>/<vmid>/<drive-name>.qcow2` for every VM disk image
- `/@qemu/<vm-name>/<vmid>/config.json` for VM configuration data

Tip: For *Genuine QEMU* mode <*vmid*> will generally always be set to zero and <*vm-name*> is the value set with the `-name <vm-name>` QEMU execution parameter.

Multiple files will be created during a backup if multiple guest VMs are backed up with one job (for Proxmox backup mode only). The distinct file names as shown above will help to locate the proper guest VM images for restore.

3.2 Restore

The QEMU Plugin allows the following targets for restore operations:

- Restore to a local directory as QCOW2 disk image files and single configuration file
- Restore to Genuine QEMU hypervisor to the original device file location
- Restore to Proxmox virtualization infrastructure as an original or new VMID

Restore To Local Directory

To use this mode, the **where=/some/path** parameter of a Bacula restore is set to a full path on the server where the QEMU Plugin is installed. If the path does not exist, it will be created by the Bacula QEMU Plugin. Bacula will automatically patch destination disk images with all Incremental backups. The end results will be a "ready to use" QCOW2 disk image file. For this target type you will always get disk images in QCOW2 format.

Restore to QEMU

To use this restore mode, the **where=/** parameter of a Bacula restore is used. The guest VM disk images will be restored and automatically patched with all Incremental backups in the *working directory* (check *Working Directory Location* for details). Then all patched and ready to use image files will be converted to the original destination location and image format when the **convdestination: yes** restore plugin parameter is set. This allow you to get ready to run QEMU guest VM images in their original location.

This restore mode is selected for all backups executed for *Genuine QEMU*.

Restore to Proxmox

To use this restore mode, the **where=/** parameter of a Bacula restore is used. The guest VM disk images will be restored and automatically patched with all Incremental backups in the *working directory* (check *Working Directory Location* for details). Then all patched and ready to use image files will be transferred to Proxmox storage attached to the original image.

The QEMU Plugin will try to create the same :term::*VMID* as it was during the backup. If VM guest already exist then with this :term::*VMID*, the plugin will allocate a new VMID. It will never overwrite an existing VM guest.

All other guest VM configuration parameters will be restored as they were backed up, including network MAC addresses. For this reason, it is recommended to inspect and possibly update a guest VM's configuration before it is started. Otherwise, resource conflicts may arise.

The Storage target to be used for the restored guest VM disk(s) can be set using the plugin's **storage** restore option. If this option is not set, all guest VM disks will be restored to their original Storage.

To list available Storages, a listing mode is available, described in the chapter **listing**)

This restore mode is selected for all backups executed for *Proxmox*.

Attention: The QEMU Plugin cannot be used for automatic migration of QEMU guest VM from *Genuine QEMU* into a Proxmox virtualization infrastructure, but you can perform a *Proxmox* to *Genuine QEMU* migration using backup `mode=QEMU` for both.

4 Installation

The Bacula File Daemon and its QEMU Plugin need to be installed on the host of the Proxmox hypervisor which runs the guest VMs that are to be backed up. Proxmox uses a customized Debian distribution, so the Bacula Enterprise File Daemon for that platform has to be used.

4.1 Configuration

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

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

4.2 Installation of the Plugin

Installation of the Bacula Enterprise QEMU Plugin is most easily done by adding the repository file suitable for the existing subscription to the Linux package manager for your distribution of choice.

For RHEL/CentOS distributions an example repository file would be `/etc/yum.repos.d/bacula-qemu.repo` with the following content:

```
[bacula-enterprise-qemu]
name=Bacula Enterprise QEMU Plugin for RHEL $releasever - $basearch
baseurl=https://www.baculasystems.com/dl/@customer-string@/rpms/qemu/@version@/rhel8-64
enabled=1
gpgcheck=1
gpgkey=https://www.baculasystems.com/dl/@customer-string@/BaculaSystems-Public-Signature.
↪asc
```

For Ubuntu/Debian distributions an example repository file would be `/etc/apt/sources.list.d/bacula-qemu.list` with the following content:


```
# Bacula Enterprise QEMU Plugin
deb https://www.baculasystems.com/dl/@customer-string@/debs/qemu/@version@/bullseye-64/␣
↪bullseye qemu
```

After that, a run of `apt-get update` for any Ubuntu/Debian distribution is needed. Then, the Plugin may be installed using `apt-get install bacula-enterprise-qemu-plugin` or `yum install bacula-enterprise-qemu-plugin` for RHEL/CentOS.

5 Plugin Configuration

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

5.1 Generic Plugin Parameters

The following QEMU Plugin parameters effect any type of Job (Backup, Estimation, or Restore).

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

working=</path/to/dir> specifies the directory used for any QEMU Plugin backup or restore operations. The default value is the *WorkingDir* Bacula FD configuration parameter (check *Working Directory Location*). This parameter is optional.

5.2 Genuine QEMU Estimation and Backup Plugin Parameters

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

qmpcontrol=</path/to/qmp/socket> specifies the location of the QMP control socket used for backup operations. When this parameter is set it implies `mode=QEMU`. This parameter is required for *Genuine QEMU* and is automatically set up for the other virtualization platforms. For other virtualization platforms this parameter is prohibited.

mode=[QEMU|PVE] specifies the default type of QEMU support. You should set it to `QEMU` for Genuine QEMU virtualization, and `PVE` for Proxmox virtualization. When not set, *QEMU* mode of operation is set which requires a proper *qmpcontrol=...* parameter to be configured.

timeout=<seconds> specifies a time the QEMU Plugin will wait for an image dump to finish. When this timeout is reached, the job will be failed. If not set, a default value of 3600 seconds will be used. This parameter is optional.

bitmap-persistence=[0|1] specifies if QEMU should persist a dirty block map used for Incremental backups in the device itself. The bitmap persistance is supported on the QCOW2 image files only. Setting this option on devices which do not support it will cause the backup job to fail. The bitmap persistance is set during the first Full backup job only. For Proxmox this parameter is set to '0'. This parameter is optional.

Important: Ephemeral dirty block bitmap (when `bitmap-persistence=0`) will be recreated on every VM restart forcing a Full VM backup.

---

vm=<name> specifies a guest VM name to backup. All guest VMs with a *<name>* provided will be selected for backup. Multiple `vm=...` parameters are allowed. If a guest VM with *<name>* can not be found, then a single

job error will be generated and the backup will proceed to the next VM unless `abort_on_error` is set which will cause the backup job to be failed. This parameter is optional and used when `mode=PVE` only.

vmid=<vmid> specifies a guest VM VMID to backup. Multiple `vmid=...` parameters may be provided. If a guest VM with <*vmid*> can not be found, a job error will be generated and the backup will proceed to the next VM unless `abort_on_error` is set which will cause the backup job to be failed. This parameter is optional and used when `mode=PVE` only.

include=<name-regex> specifies a list of a guest VM names to backup using regular expression syntax. All guest VMs with names matching the `name-regex` regular expression provided will be selected for backup. Multiple `include=...` parameters may be provided. The match performed is case insensitive.

If no guest VMs match the name-regex expression provided, the backup will proceed to the next parameters or finish successfully without backing up any VMs. The `abort_on_error` parameter will not fail the job when no guest VMs are found using name matching. This parameter is optional and used when `mode=PVE` only.

exclude=<name-regex> specifies a list of a guest VMs names which will be excluded from backup using regular expression matching. All guest VMs with names matching the provided regular expression, and selected for backup using the `include=...` parameter will be excluded. The match is case insensitive. This parameter does not affect any guest VM selected to be backed up using `vm=...` or `vmid=...` parameters. Multiple `exclude=...` parameters may be provided. This parameter is optional and used when `mode=PVE` only.

If none of the paramaters `vm=...`, `vmid=...`, `include=...` and `exclude=...` are specified then all available guest VMs on the Proxmox hypervisor will be backed up. For *Genuine QEMU* a single VM pointed to by the `qmpcontrol=...` parameter will be backed up.

5.3 Plugin Restore Parameters

During restore, the QEMU Plugin will use the same parameters which were set for the backup job. These settings are saved in the catalog at the time the backup job is run. Some of them may be changed during the restore process if required.

convdestination: [yes|no] specifies if a restored disk image should be converted to the original format and location as it was defined during backup. This parameter works in *Genuine QEMU* mode only. If not set or set to *no* (the default), then QEMU virtual disk images will be available in the *working* (see next parameter) location as QCOW2 format files. In this case you can manually move it to the required destination or convert it to the desired format.

This parameter is optional.

working: </path/to/dir> specifies the directory used for QEMU Plugin restore operations as described at *Working Directory Location*. The default value is the same as for backup and can be set by the plugin configuration or the *WorkingDir* Bacula FD configuration parameter.

This parameter is optional.

storage: <storage> specifies a Proxmox Storage where restored guest VMs will be restored to. If not set then a guest VM will be restored to the Proxmox Storage it was backed up from. If this parameter points to a nonexistent Storage, the original Storage of the guest VM will be used.

This parameter is optional.

5.4 Working Directory Location

To perform any of the backup or restore job operations, the QEMU Plugin requires some storage space to be available. The size space required depends on the size of the largest disk image file during backup and the sum of the VM disk images during restore. For the restore operations the space is required for proper incremental image patching.

This limitation is a direct consequence of the QEMU QMP drive backup limitation which supports disk images saved to a regular file only.

The exact location of the working area can be selected with the `working=...` plugin parameter. When this parameter is not set then a default value of *WorkingDirectory* Bacula parameter will be used.

This storage space is automatically cleaned after the operation unless you select a restore operation without disk image conversion to the destination (check **convdestination** restore parameter).

5.5 FileSet Examples

In the example below, a single QEMU VM qill be backed up.

```
FileSet {
  Name = FS_qemu
  Include {
    Plugin = "qemu: qmpcontrol=/images/vm1.qmp"
  }
}
```

Now the same QEMU VM but with custom working location.

```
FileSet {
  Name = FS_qemu_working
  Include {
    Plugin = "qemu: qmpcontrol=/images/vm1.qmp working=/tmp"
  }
}
```

In the example below, all Promox guest VMs will be backed up.

```
FileSet {
  Name = FS_ProxmoxAll
  Include {
    Plugin = "qemu: mode=PVE"
  }
}
```

In this example, a single guest VM with name of “VM1” will be backed up.

```
FileSet {
  Name = FS_Proxmox_VM1
  Include {
    Plugin = "qemu: mode=PVE vm=VM1"
  }
}
```

The same example as above, but using `vmid` instead:


```
FileSet {
  Name = FS_Proxmox_VM1
  Include {
    Plugin = "qemu: mode=PVE vmid=101"
  }
}
```

In the following example, all guest VMs which contain “Prod” in their names will be backed up.

```
FileSet {
  Name = FS_Proxmox_ProdAll
  Include {
    Plugin = "qemu: mode=PVE include=Prod"
  }
}
```

In this final example, all guest VMs except VMs whose name begins with “Test” will be backed up.

```
FileSet {
  Name = FS_Proxmox_AllbutTest
  Include {
    Plugin = "qemu: mode=PVE include=.* exclude=^Test"
  }
}
```

6 Restore

6.1 Restore with Genuine QEMU

To restore a VM or VMs using *Genuine QEMU* mode, you 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 **convdestination** plugin restore option is set to *“yes”*:

```
* restore where=/
...
Run Restore job
JobName:         RestoreFiles
Bootstrap:       /opt/bacula/working/qemu-test-dir.restore.2.bsr
Where:           /
Replace:         Always
FileSet:         Full Set
Backup Client:   qemu-test-fd
Restore Client:  qemu-test-fd
Storage:         File1
When:            2021-11-01 13:19:16
Catalog:         MyCatalog
```

(continues on next page)

(continued from previous page)

```
Priority:        10
Plugin Options:  *None*
OK to run? (yes/mod/no): mod
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 : qemu: qmpcontrol=/images/vm1.qmp abort_on_error
Plugin Restore Options
Option               Current Value        Default Value
working:             *None*               (*None*)
convdestination:     *None*               (*None*)
Use above plugin configuration? (yes/mod/no): mod
You have the following choices:
   1: working (Plugin working directory)
   2: convdestination (Convert qcow2 restore to original destination)
Select parameter to modify (1-2): 2
Please enter a value for convdestination: yes
Plugin Restore Options
Option               Current Value        Default Value
working:             *None*               (*None*)
convdestination:     yes                  (*None*)
Use above plugin configuration? (yes/mod/no): yes
...
```

During a restore job you should get information about each restore process including Incremental patching of the restored disk devices and the destination convertion process.

```
JobId 121: Start Restore Job RestoreFiles.2021-11-01_13.05.11_03
JobId 121: Restoring files from JobId(s) 115,116,120
JobId 121: Using Device "FileChgr1-Dev1" to read.
JobId 121: qemu: VM to restore: VM1 (oid:0)
JobId 121: qemu: Start Restore vm: VM1 (rid:0) devices: 2
JobId 121: qemu: Restoring device: ide0-hd0
JobId 121: qemu: Restoring device: ide0-hd1
JobId 121: qemu: Patching incremental: ide0-hd0
JobId 121: qemu: Patching incremental: ide0-hd1
JobId 121: qemu: Patching incremental: ide0-hd0
JobId 121: qemu: Patching incremental: ide0-hd1
```

The new guest VM created during the restore will get a new VMID (if the original VMID is in use) but the name / hostname will stay the same as it was with the original VM.

6.2 Restore to a Proxmox Hypervisor

To restore a VM or VMs to a Proxmox hypervisor, you should execute the same restore command as above and the basic restore process is the same. The main difference is the final configuration and device importing procedure as shown below:

```
JobId 117: Start Restore Job RestoreFiles.2021-11-02_17.26.25_07
JobId 117: Restoring files from JobId(s) 104,105,106,115,116
JobId 117: Using Device "FileChgr1-Dev2" to read.
JobId 117: qemu: VM to restore: vm1 (oid:100)
JobId 117: qemu: Start Restore vm: vm1 (rid:104) devices: 1
JobId 117: qemu: Restoring device: drive-scsi0
JobId 117: qemu: Patching incremental: drive-scsi0
JobId 117: qemu: Patching incremental: drive-scsi0
JobId 117: qemu: Patching incremental: drive-scsi0
JobId 117: qemu: Patching incremental: drive-scsi0
JobId 117: Elapsed time=00:06:42, Transfer rate=20.28 M Bytes/second
JobId 117: qemu: Successfully imported device drive-scsi0 as local-lvm:vm-104-disk-0
...
```

The new guest VM created during the restore will get a new VMID (if the original VMID is in use) but the name / hostname will stay the same as it was with the original VM.

6.3 Restore to Local Directory

It is possible to restore the guest VM disk image(s) to a local directory instead of restoring back to a hypervisor as a new VM. To do so, the **where** restore option should point to a local directory:

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

Please check the following example for the test "VM local restore":

```
JobId 118: Start Restore Job RestoreFiles.2021-11-02_17.37.34_09
JobId 118: Restoring files from JobId(s) 104,105,106,115,116
JobId 118: Using Device "FileChgr1-Dev1" to read.
JobId 118: Ready to read from volume "vol01" on File device "FileChgr1-Dev1" (/opt/
↪bacula/archive).
JobId 118: qemu: VM local restore: vm1 (oid:100)
JobId 118: Forward spacing Volume "vol01" to addr=227
JobId 118: qemu: Restoring device: drive-scsi0
JobId 118: qemu: Patching incremental: drive-scsi0
JobId 118: qemu: Patching incremental: drive-scsi0
JobId 118: qemu: Patching incremental: drive-scsi0
JobId 118: qemu: Patching incremental: drive-scsi0
JobId 118: Elapsed time=00:06:57, Transfer rate=19.55 M Bytes/second
...
```

The restore job log will show that the restore was done to a local directory.

7 Other

7.1 Resource listing

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

- List of guest VM names
- List of guest VM VMIDs
- List of Proxmox Storages

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

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

plugin=<plugin> A plugin name, which would be **qemu:** 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.

vm to display a list of guest VM name-labels.

vmid to display a list of guest VM VMIDs and name-label pointers.

storage to show the list of available Storages.

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

```
*.ls client=proxmoxtest-fd plugin="qemu: mode=PVE" path=/
Connecting to Client proxmoxtest-fd at proxmoxtest:9102
drwxr-x---   1 root     root                     0 2021-11-01 12:55:55  vm
drwxr-x---   1 root     root                     0 2021-11-01 12:55:55  vmid
drwxr-x---   1 root     root                     0 2021-11-01 12:55:55  storage
2000 OK estimate files=3 bytes=0
```

To display the list of all available guest VMs, the following command example can be used:

```
*.ls client=proxmoxtest-fd plugin="qemu: mode=PVE" path=/vm
Connecting to Client proxmoxtest-fd at proxmoxtest:9102
-rw-r-----   1 root     root             8589934592 2021-11-01 12:56:18  vm1
-rw-r-----   1 root     root             8589934592 2021-11-01 12:56:18  vm3
2000 OK estimate files=2 bytes=17,179,869,184
```

To display the list of guest VM VMIDs, use the following command example:

```
*.ls client=proxmoxtest-fd plugin="qemu: mode=PVE" path=/vmid
Connecting to Client proxmoxtest-fd at proxmoxtest:9102
-rw-r-----   1 root     root             8589934592 2021-11-01 12:56:46  100 -> vm1
-rw-r-----   1 root     root             8589934592 2021-11-01 12:56:46  102 -> vm3
2000 OK estimate files=2 bytes=17,179,869,184
```

The VM and VMID lists display an estimated size of the guest VM based on *virtual* or *actual* disk image sizes.

To display available Proxmox Storages, the following command example can be used:


```
*.ls client=proxmoxtest-fd plugin="qemu: mode=PVE" path=/storage
Connecting to Client proxmoxtest-fd at proxmoxtest:9102
brw-r-----   1 root     root                     0 2021-11-01 12:57:56  data
brw-r-----   1 root     root                     0 2021-11-01 12:57:56  local-lvm
2000 OK estimate files=2 bytes=0
```

7.2 Resource Query

The Bacula Enterprise QEMU Plugin supports the plugin query feature of Bacula Enterprise 14.0 or newer. This mode allows a plugin to display the same information about available Proxmox resources which is defined at *Resource listing*. The QEMU Plugin returns response data in JSON format.

In this example is a query about available VMs:

```
*.query client=proxmoxtest-fd plugin="qemu: mode=PVE" parameter=vm
[{"name":"vm1","vmid":100},{"name":"vm3","vmid":102}]
*.query client=proxmoxtest-fd plugin="qemu: mode=PVE" parameter=vmid
[{"name":"vm3","vmid":102},{"name":"vm1","vmid":100}]
```

In this example is a query about available Proxmox storages:

```
*.query client=proxmoxtest-fd plugin="qemu: mode=PVE" parameter=storage
[{"name":"local"},{"name":"local-lvm"}]
```

7.3 Limitations

- Granular restore (*Single Item Restore*) is not available yet. A file level backup of the VM with the Bacula FD inside the Guest VM is needed to enable single file restores of a QEMU VM guest.
- It is not possible to run the very same backup job of the same guest VM concurrently (duplicate job).
- It is not possible to have different backup jobs that backup the same guest VM as it will break the Incremental dirty bitmap backup chain. In this case, a successful recovery won't be even possible.
- VM templates available on the Proxmox system can not be backed up. This is a Proxmox limitation.
- In listing and query mode with the **vm** or **vmid** parameters, the plugin will display running VMs only.
- Any VM backup or restore operation requires sufficient storage space in the *working directory* (check *Working Directory Location*). This is a general QEMU QMP backup procedure limitation.
- Differential backup level is not yet supported. Only Full and Incremental backup levels are supported. This limitation will be removed in the future.

8 Supported QEMU Hypervisor versions

The following QEMU Hypervisor infrastructure versions are tested and supported:

- Genuine QEMU 3.1 - 5.2
- Proxmox VE 6.4 - 7.0

