NAME

qemu–img – QEMU disk image utility

SYNOPSIS

qemu-img [*standard options*] *command* [*command options*]

DESCRIPTION

qemu-img allows you to create, convert and modify images offline. It can handle all image formats supported by QEMU.

Warning: Never use qemu-img to modify images in use by a running virtual machine or any other process; this may destroy the image. Also, be aware that querying an image that is being modified by another process may encounter inconsistent state.

OPTIONS

Standard options:

–h, ––help
Display this help and exit

–V, ––version
Display version information and exit

–T, ––trace [[enable=]*pattern***][,events=***file***][,file=***file***]**
Specify tracing options.

[enable=]*pattern*
Immediately enable events matching *pattern* (either event name or a globbing pattern). This option is only available if QEMU has been compiled with the *simple*, *log* or *ftrace* tracing backend. To specify multiple events or patterns, specify the **–trace** option multiple times.

Use `-trace help` to print a list of names of trace points.

events=*file*
Immediately enable events listed in *file*. The file must contain one event name (as listed in the *trace-events-all* file) per line; globbing patterns are accepted too. This option is only available if QEMU has been compiled with the *simple*, *log* or *ftrace* tracing backend.

file=*file*
Log output traces to *file*. This option is only available if QEMU has been compiled with the *simple* tracing backend.

The following commands are supported:

amend [––object *objectdef***] [––image–opts] [–p] [–q] [–f** *fmt***] [–t** *cache***] –o** *options filename*
bench [–c *count***] [–d** *depth***] [–f** *fmt***] [––flush–interval=***flush_interval***] [–n] [––no–drain] [–o** *offset***] [––pattern=***pattern***] [–q] [–s** *buffer_size***] [–S** *step_size***] [–t** *cache***] [–w] [–U]** *filename*
check [––object *objectdef***] [––image–opts] [–q] [–f** *fmt***] [––output=***ofmt***] [–r [leaks | all]] [–T** *src_cache***] [–U]** *filename*
commit [––object *objectdef***] [––image–opts] [–q] [–f** *fmt***] [–t** *cache***] [–b** *base***] [–d] [–p]** *filename*
compare [––object *objectdef***] [––image–opts] [–f** *fmt***] [–F** *fmt***] [–T** *src_cache***] [–p] [–q] [–s] [–U]** *filename1 filename2*
convert [––object *objectdef***] [––image–opts] [––target–image–opts] [–U] [–C] [–c] [–p] [–q] [–n] [–f** *fmt***] [–t** *cache***] [–T** *src_cache***] [–O** *output_fmt***] [–B** *backing_file***] [–o** *options***] [–l** *snapshot_param***] [–S** *sparse_size***] [–m** *num_coroutines***] [–W] [––salvage]** *filename* **[***filename2* **[...]]** *output_filename*
create [––object *objectdef***] [–q] [–f** *fmt***] [–b** *backing_file***] [–F** *backing_fmt***] [–u] [–o** *options***]** *filename* **[***size***]**
dd [––image–opts] [–U] [–f *fmt***] [–O** *output_fmt***] [bs=***block_size***] [count=***blocks***] [skip=***blocks***] if=***input* **of=***output*
info [––object *objectdef***] [––image–opts] [–f** *fmt***] [––output=***ofmt***] [––backing–chain] [–U]** *filename*

map [−−object *objectdef***] [−−image−opts] [−f** *fmt***] [−−output=***ofmt***] [−U]** *filename*
measure [−−output=*ofmt***] [−O** *output_fmt***] [−o** *options***] [−−size** *N* **| [−−object** *objectdef***] [−−image−opts] [−f** *fmt***] [−l** *snapshot_param***]** *filename***]**
snapshot [−−object *objectdef***] [−−image−opts] [−U] [−q] [−l | −a** *snapshot* **| −c** *snapshot* **| −d** *snapshot***]** *filename*
rebase [−−object *objectdef***] [−−image−opts] [−U] [−q] [−f** *fmt***] [−t** *cache***] [−T** *src_cache***] [−p] [−u] −b** *backing_file* **[−F** *backing_fmt***]** *filename*
resize [−−object *objectdef***] [−−image−opts] [−f** *fmt***] [−−preallocation=***prealloc***] [−q] [−−shrink]** *filename* **[+ | −]***size*

Command parameters:

filename
is a disk image filename

fmt is the disk image format. It is guessed automatically in most cases. See below for a description of the supported disk formats.

size is the disk image size in bytes. Optional suffixes `k` or `K` (kilobyte, 1024) `M` (megabyte, 1024k) and `G` (gigabyte, 1024M) and T (terabyte, 1024G) are supported. `b` is ignored.

output_filename
is the destination disk image filename

output_fmt
is the destination format

options
is a comma separated list of format specific options in a name=value format. Use `-o ?` for an overview of the options supported by the used format or see the format descriptions below for details.

snapshot_param
is param used for internal snapshot, format is 'snapshot.id=[ID],snapshot.name=[NAME]' or '[ID_OR_NAME]'

−−object *objectdef*
is a QEMU user creatable object definition. See the `qemu(1)` manual page for a description of the object properties. The most common object type is a `secret`, which is used to supply passwords and/or encryption keys.

−−image−opts
Indicates that the source *filename* parameter is to be interpreted as a full option string, not a plain filename. This parameter is mutually exclusive with the *−f* parameter.

−−target−image−opts
Indicates that the *output_filename* parameter(s) are to be interpreted as a full option string, not a plain filename. This parameter is mutually exclusive with the *−O* parameters. It is currently required to also use the *−n* parameter to skip image creation. This restriction may be relaxed in a future release.

−−force−share (−U)
If specified, `qemu-img` will open the image in shared mode, allowing other QEMU processes to open it in write mode. For example, this can be used to get the image information (with 'info' subcommand) when the image is used by a running guest. Note that this could produce inconsistent results because of concurrent metadata changes, etc. This option is only allowed when opening images in read-only mode.

−−backing−chain
will enumerate information about backing files in a disk image chain. Refer below for further description.

−c indicates that target image must be compressed (qcow format only)

−h with or without a command shows help and lists the supported formats

−p display progress bar (compare, convert and rebase commands only). If the *−p* option is not used for a command that supports it, the progress is reported when the process receives a `SIGUSR1` or `SIGINFO` signal.

−q Quiet mode − do not print any output (except errors). There's no progress bar in case both *−q* and *−p* options are used.

−S *size*
indicates the consecutive number of bytes that must contain only zeros for qemu-img to create a sparse image during conversion. This value is rounded down to the nearest 512 bytes. You may use the common size suffixes like `k` for kilobytes.

−t *cache*
specifies the cache mode that should be used with the (destination) file. See the documentation of the emulator's `−drive cache=...` option for allowed values.

−T *src_cache*
specifies the cache mode that should be used with the source file(s). See the documentation of the emulator's `−drive cache=...` option for allowed values.

Parameters to snapshot subcommand:

snapshot
is the name of the snapshot to create, apply or delete

−a applies a snapshot (revert disk to saved state)

−c creates a snapshot

−d deletes a snapshot

−l lists all snapshots in the given image

Parameters to compare subcommand:

−f First image format

−F Second image format

−s Strict mode − fail on different image size or sector allocation

Parameters to convert subcommand:

−n Skip the creation of the target volume

−m Number of parallel coroutines for the convert process

−W
Allow out-of-order writes to the destination. This option improves performance, but is only recommended for preallocated devices like host devices or other raw block devices.

−C Try to use copy offloading to move data from source image to target. This may improve performance if the data is remote, such as with NFS or iSCSI backends, but will not automatically sparsify zero sectors, and may result in a fully allocated target image depending on the host support for getting allocation information.

−−salvage
Try to ignore I/O errors when reading. Unless in quiet mode (`−q`), errors will still be printed. Areas that cannot be read from the source will be treated as containing only zeroes.

Parameters to dd subcommand:

bs=*block_size*
defines the block size

count=*blocks*
sets the number of input blocks to copy

if=*input*
sets the input file

of=*output*
sets the output file

skip=*blocks*
sets the number of input blocks to skip

Command description:

amend [−−object *objectdef***] [−−image−opts] [−p] [−q] [−f** *fmt***] [−t** *cache***] −o** *options filename*
Amends the image format specific *options* for the image file *filename*. Not all file formats support this operation.

bench [−c *count***] [−d** *depth***] [−f** *fmt***] [−−flush−interval=***flush_interval***] [−n] [−−no−drain] [−o** *offset***] [−−pattern=***pattern***] [−q] [−s** *buffer_size***] [−S** *step_size***] [−t** *cache***] [−w] [−U]** *filename*
Run a simple sequential I/O benchmark on the specified image. If `-w` is specified, a write test is performed, otherwise a read test is performed.

A total number of *count* I/O requests is performed, each *buffer_size* bytes in size, and with *depth* requests in parallel. The first request starts at the position given by *offset*, each following request increases the current position by *step_size*. If *step_size* is not given, *buffer_size* is used for its value.

If *flush_interval* is specified for a write test, the request queue is drained and a flush is issued before new writes are made whenever the number of remaining requests is a multiple of *flush_interval*. If additionally `--no-drain` is specified, a flush is issued without draining the request queue first.

If `-n` is specified, the native AIO backend is used if possible. On Linux, this option only works if `-t none` or `-t directsync` is specified as well.

For write tests, by default a buffer filled with zeros is written. This can be overridden with a pattern byte specified by *pattern*.

check [−−object *objectdef***] [−−image−opts] [−q] [−f** *fmt***] [−−output=***ofmt***] [−r [leaks | all]] [−T** *src_cache***] [−U]** *filename*
Perform a consistency check on the disk image *filename*. The command can output in the format *ofmt* which is either `human` or `json`. The JSON output is an object of QAPI type `ImageCheck`.

If `-r` is specified, qemu-img tries to repair any inconsistencies found during the check. `-r leaks` repairs only cluster leaks, whereas `-r all` fixes all kinds of errors, with a higher risk of choosing the wrong fix or hiding corruption that has already occurred.

Only the formats `qcow2`, `qed` and `vdi` support consistency checks.

In case the image does not have any inconsistencies, check exits with `0`. Other exit codes indicate the kind of inconsistency found or if another error occurred. The following table summarizes all exit codes of the check subcommand:

0	Check completed, the image is (now) consistent
1	Check not completed because of internal errors
2	Check completed, image is corrupted
3	Check completed, image has leaked clusters, but is not corrupted
63	Checks are not supported by the image format

If `-r` is specified, exit codes representing the image state refer to the state after (the attempt at) repairing it. That is, a successful `-r all` will yield the exit code 0, independently of the image state before.

commit [−−object *objectdef***] [−−image−opts] [−q] [−f** *fmt***] [−t** *cache***] [−b** *base***] [−d] [−p]** *filename*

Commit the changes recorded in *filename* in its base image or backing file. If the backing file is smaller than the snapshot, then the backing file will be resized to be the same size as the snapshot. If the snapshot is smaller than the backing file, the backing file will not be truncated. If you want the backing file to match the size of the smaller snapshot, you can safely truncate it yourself once the commit operation successfully completes.

The image *filename* is emptied after the operation has succeeded. If you do not need *filename* afterwards and intend to drop it, you may skip emptying *filename* by specifying the `-d` flag.

If the backing chain of the given image file *filename* has more than one layer, the backing file into which the changes will be committed may be specified as *base* (which has to be part of *filename*'s backing chain). If *base* is not specified, the immediate backing file of the top image (which is *filename*) will be used. Note that after a commit operation all images between *base* and the top image will be invalid and may return garbage data when read. For this reason, `-b` implies `-d` (so that the top image stays valid).

compare [−−object *objectdef***] [−−image−opts] [−f** *fmt***] [−F** *fmt***] [−T** *src_cache***] [−p] [−q] [−s] [−U]** *filename1 filename2*

Check if two images have the same content. You can compare images with different format or settings.

The format is probed unless you specify it by *−f* (used for *filename1*) and/or *−F* (used for *filename2*) option.

By default, images with different size are considered identical if the larger image contains only unallocated and/or zeroed sectors in the area after the end of the other image. In addition, if any sector is not allocated in one image and contains only zero bytes in the second one, it is evaluated as equal. You can use Strict mode by specifying the *−s* option. When compare runs in Strict mode, it fails in case image size differs or a sector is allocated in one image and is not allocated in the second one.

By default, compare prints out a result message. This message displays information that both images are same or the position of the first different byte. In addition, result message can report different image size in case Strict mode is used.

Compare exits with `0` in case the images are equal and with `1` in case the images differ. Other exit codes mean an error occurred during execution and standard error output should contain an error message. The following table sumarizes all exit codes of the compare subcommand:

0	Images are identical
1	Images differ
2	Error on opening an image
3	Error on checking a sector allocation
4	Error on reading data

convert [−−object *objectdef***] [−−image−opts] [−−target−image−opts] [−U] [−C] [−c] [−p] [−q] [−n] [−f** *fmt***] [−t** *cache***] [−T** *src_cache***] [−O** *output_fmt***] [−B** *backing_file***] [−o** *options***] [−l** *snapshot_param***] [−S** *sparse_size***] [−m** *num_coroutines***] [−W]** *filename* **[***filename2* **[...]]** *output_filename*

Convert the disk image *filename* or a snapshot *snapshot_param* to disk image *output_filename* using format *output_fmt*. It can be optionally compressed (`-c` option) or use any format specific options like encryption (`-o` option).

Only the formats `qcow` and `qcow2` support compression. The compression is read-only. It means that if a compressed sector is rewritten, then it is rewritten as uncompressed data.

Image conversion is also useful to get smaller image when using a growable format such as `qcow`: the empty sectors are detected and suppressed from the destination image.

sparse_size indicates the consecutive number of bytes (defaults to 4k) that must contain only zeros for qemu-img to create a sparse image during conversion. If *sparse_size* is 0, the source will not be scanned for unallocated or zero sectors, and the destination image will always be fully allocated.

You can use the *backing_file* option to force the output image to be created as a copy on write image of the specified base image; the *backing_file* should have the same content as the input's base image, however the path, image format, etc may differ.

If a relative path name is given, the backing file is looked up relative to the directory containing *output_filename*.

If the `-n` option is specified, the target volume creation will be skipped. This is useful for formats such as `rbd` if the target volume has already been created with site specific options that cannot be supplied through qemu-img.

Out of order writes can be enabled with `-W` to improve performance. This is only recommended for preallocated devices like host devices or other raw block devices. Out of order write does not work in combination with creating compressed images.

num_coroutines specifies how many coroutines work in parallel during the convert process (defaults to 8).

create [--object *objectdef***] [-q] [-f** *fmt***] [-b** *backing_file***] [-F** *backing_fmt***] [-u] [-o** *options***]** *filename* **[***size***]**

Create the new disk image *filename* of size *size* and format *fmt*. Depending on the file format, you can add one or more *options* that enable additional features of this format.

If the option *backing_file* is specified, then the image will record only the differences from *backing_file*. No size needs to be specified in this case. *backing_file* will never be modified unless you use the `commit` monitor command (or qemu-img commit).

If a relative path name is given, the backing file is looked up relative to the directory containing *filename*.

Note that a given backing file will be opened to check that it is valid. Use the `-u` option to enable unsafe backing file mode, which means that the image will be created even if the associated backing file cannot be opened. A matching backing file must be created or additional options be used to make the backing file specification valid when you want to use an image created this way.

The size can also be specified using the *size* option with `-o`, it doesn't need to be specified separately in this case.

dd [--image-opts] [-U] [-f *fmt***] [-O** *output_fmt***] [bs=***block_size***] [count=***blocks***] [skip=***blocks***] if=***input* **of=***output*

Dd copies from *input* file to *output* file converting it from *fmt* format to *output_fmt* format.

The data is by default read and written using blocks of 512 bytes but can be modified by specifying *block_size*. If count=*blocks* is specified dd will stop reading input after reading *blocks* input blocks.

The size syntax is similar to **dd**(1)'s size syntax.

info [--object *objectdef***] [--image-opts] [-f** *fmt***] [--output=***ofmt***] [--backing-chain] [-U]** *filename*

Give information about the disk image *filename*. Use it in particular to know the size reserved on disk which can be different from the displayed size. If VM snapshots are stored in the disk image, they are displayed too.

If a disk image has a backing file chain, information about each disk image in the chain can be recursively enumerated by using the option `--backing-chain`.

For instance, if you have an image chain like:

```
base.qcow2 <- snap1.qcow2 <- snap2.qcow2
```

To enumerate information about each disk image in the above chain, starting from top to base, do:

```
qemu-img info --backing-chain snap2.qcow2
```

The command can output in the format *ofmt* which is either `human` or `json`. The JSON output is an object of QAPI type `ImageInfo`; with `--backing-chain`, it is an array of `ImageInfo` objects.

`--output=human` reports the following information (for every image in the chain):

image
: The image file name

file format
: The image format

virtual size
: The size of the guest disk

disk size
: How much space the image file occupies on the host file system (may be shown as 0 if this information is unavailable, e.g. because there is no file system)

cluster_size
: Cluster size of the image format, if applicable

encrypted
: Whether the image is encrypted (only present if so)

cleanly shut down
: This is shown as `no` if the image is dirty and will have to be auto-repaired the next time it is opened in qemu.

backing file
: The backing file name, if present

backing file format
: The format of the backing file, if the image enforces it

Snapshot list
: A list of all internal snapshots

Format specific information
: Further information whose structure depends on the image format. This section is a textual representation of the respective `ImageInfoSpecific*` QAPI object (e.g. `ImageInfoSpecificQCow2` for qcow2 images).

map [−−object *objectdef***] [−−image−opts] [−f** *fmt***] [−−output=***ofmt***] [−U]** *filename*

Dump the metadata of image *filename* and its backing file chain. In particular, this commands dumps the allocation state of every sector of *filename*, together with the topmost file that allocates it in the backing file chain.

Two option formats are possible. The default format (`human`) only dumps known-nonzero areas of the file. Known-zero parts of the file are omitted altogether, and likewise for parts that are not allocated throughout the chain. **qemu-img** output will identify a file from where the data can be read, and the offset in the file. Each line will include four fields, the first three of which are hexadecimal numbers. For example the first line of:

```
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/overlay.qcow2
0x100000        0x10000         0x95380000      /tmp/backing.qcow2
```

means that 0x20000 (131072) bytes starting at offset 0 in the image are available in /tmp/overlay.qcow2 (opened in `raw` format) starting at offset 0x50000 (327680). Data that is compressed, encrypted, or otherwise not available in raw format will cause an error if `human` format is in use. Note that file names can include newlines, thus it is not safe to parse this output format in scripts.

The alternative format `json` will return an array of dictionaries in JSON format. It will include similar information in the `start`, `length`, `offset` fields; it will also include other more specific information:

– whether the sectors contain actual data or not (boolean field `data`; if false, the sectors are either unallocated or stored as optimized all-zero clusters);

– whether the data is known to read as zero (boolean field `zero`);

– in order to make the output shorter, the target file is expressed as a `depth`; for example, a depth of 2 refers to the backing file of the backing file of *filename*.

In JSON format, the `offset` field is optional; it is absent in cases where `human` format would omit the entry or exit with an error. If `data` is false and the `offset` field is present, the corresponding sectors in the file are not yet in use, but they are preallocated.

For more information, consult *include/block/block.h* in QEMU's source code.

measure [−−output=*ofmt***] [−O** *output_fmt***] [−o** *options***] [−−size** *N* **| [−−object** *objectdef***] [−−image−opts] [−f** *fmt***] [−l** *snapshot_param***]** *filename***]**

Calculate the file size required for a new image. This information can be used to size logical volumes or SAN LUNs appropriately for the image that will be placed in them. The values reported are guaranteed to be large enough to fit the image. The command can output in the format *ofmt* which is either `human` or `json`. The JSON output is an object of QAPI type `BlockMeasureInfo`.

If the size *N* is given then act as if creating a new empty image file using **qemu-img create**. If *filename* is given then act as if converting an existing image file using **qemu-img convert**. The format of the new file is given by *output_fmt* while the format of an existing file is given by *fmt*.

A snapshot in an existing image can be specified using *snapshot_param*.

The following fields are reported:

```
required size: 524288
fully allocated size: 1074069504
```

The `required size` is the file size of the new image. It may be smaller than the virtual disk size if the image format supports compact representation.

The `fully allocated size` is the file size of the new image once data has been written to all sectors. This is the maximum size that the image file can occupy with the exception of internal snapshots, dirty bitmaps, vmstate data, and other advanced image format features.

snapshot [−−object *objectdef***] [−−image−opts] [−U] [−q] [−l | −a** *snapshot* **| −c** *snapshot* **| −d** *snapshot***]** *filename*

List, apply, create or delete snapshots in image *filename*.

rebase [−−object *objectdef***] [−−image−opts] [−U] [−q] [−f** *fmt***] [−t** *cache***] [−T** *src_cache***] [−p] [−u] −b** *backing_file* **[−F** *backing_fmt***]** *filename*

Changes the backing file of an image. Only the formats `qcow2` and `qed` support changing the backing file.

The backing file is changed to *backing_file* and (if the image format of *filename* supports this) the backing file format is changed to *backing_fmt*. If *backing_file* is specified as "" (the empty string), then the image is rebased onto no backing file (i.e. it will exist independently of any backing file).

If a relative path name is given, the backing file is looked up relative to the directory containing *filename*.

cache specifies the cache mode to be used for *filename*, whereas *src_cache* specifies the cache mode for reading backing files.

There are two different modes in which `rebase` can operate:

Safe mode

This is the default mode and performs a real rebase operation. The new backing file may differ from the old one and qemu-img rebase will take care of keeping the guest-visible content of *filename* unchanged.

In order to achieve this, any clusters that differ between *backing_file* and the old backing file of *filename* are merged into *filename* before actually changing the backing file.

Note that the safe mode is an expensive operation, comparable to converting an image. It only works if the old backing file still exists.

Unsafe mode

qemu-img uses the unsafe mode if `-u` is specified. In this mode, only the backing file name and format of *filename* is changed without any checks on the file contents. The user must take care of specifying the correct new backing file, or the guest-visible content of the image will be corrupted.

This mode is useful for renaming or moving the backing file to somewhere else. It can be used without an accessible old backing file, i.e. you can use it to fix an image whose backing file has already been moved/renamed.

You can use `rebase` to perform a "diff" operation on two disk images. This can be useful when you have copied or cloned a guest, and you want to get back to a thin image on top of a template or base image.

Say that `base.img` has been cloned as `modified.img` by copying it, and that the `modified.img` guest has run so there are now some changes compared to `base.img`. To construct a thin image called `diff.qcow2` that contains just the differences, do:

```
qemu-img create -f qcow2 -b modified.img diff.qcow2
qemu-img rebase -b base.img diff.qcow2
```

At this point, `modified.img` can be discarded, since `base.img + diff.qcow2` contains the same information.

resize [−−object *objectdef*] **[−−image−opts] [−f** *fmt*] **[−−preallocation=***prealloc*] **[−q] [−−shrink]** *filename* **[+ | −]***size*

Change the disk image as if it had been created with *size*.

Before using this command to shrink a disk image, you MUST use file system and partitioning tools inside the VM to reduce allocated file systems and partition sizes accordingly. Failure to do so will result in data loss!

When shrinking images, the `--shrink` option must be given. This informs qemu-img that the user acknowledges all loss of data beyond the truncated image's end.

After using this command to grow a disk image, you must use file system and partitioning tools inside the VM to actually begin using the new space on the device.

When growing an image, the `--preallocation` option may be used to specify how the additional image area should be allocated on the host. See the format description in the `NOTES` section which values are allowed. Using this option may result in slightly more data being allocated than necessary.

NOTES

Supported image file formats:

raw

Raw disk image format (default). This format has the advantage of being simple and easily exportable to all other emulators. If your file system supports *holes* (for example in ext2 or ext3 on Linux or NTFS on Windows), then only the written sectors will reserve space. Use `qemu-img info` to know the real size used by the image or `ls -ls` on Unix/Linux.

Supported options:

`preallocation`
: Preallocation mode (allowed values: `off`, `falloc`, `full`). `falloc` mode preallocates space for image by calling **posix_fallocate()**. `full` mode preallocates space for image by writing data to underlying storage. This data may or may not be zero, depending on the storage location.

qcow2

QEMU image format, the most versatile format. Use it to have smaller images (useful if your filesystem does not supports holes, for example on Windows), optional AES encryption, zlib based compression and support of multiple VM snapshots.

Supported options:

`compat`
: Determines the qcow2 version to use. `compat=0.10` uses the traditional image format that can be read by any QEMU since 0.10. `compat=1.1` enables image format extensions that only QEMU 1.1 and newer understand (this is the default). Amongst others, this includes zero clusters, which allow efficient copy-on-read for sparse images.

`backing_file`
: File name of a base image (see **create** subcommand)

`backing_fmt`
: Image format of the base image

`encryption`
: If this option is set to `on`, the image is encrypted with 128−bit AES-CBC.

The use of encryption in qcow and qcow2 images is considered to be flawed by modern cryptography standards, suffering from a number of design problems:

- The AES-CBC cipher is used with predictable initialization vectors based on the sector number. This makes it vulnerable to chosen plaintext attacks which can reveal the existence of encrypted data.
- The user passphrase is directly used as the encryption key. A poorly chosen or short passphrase will compromise the security of the encryption.
- In the event of the passphrase being compromised there is no way to change the passphrase to protect data in any qcow images. The files must be cloned, using a different encryption passphrase in the new file. The original file must then be securely erased using a program like shred, though even this is ineffective with many modern storage technologies.
- Initialization vectors used to encrypt sectors are based on the guest virtual sector number, instead of the host physical sector. When a disk image has multiple internal snapshots this means that data in multiple physical sectors is encrypted with the same initialization vector. With the CBC mode, this opens the possibility of watermarking attacks if the attack can collect multiple sectors encrypted with the same IV and some predictable data. Having multiple qcow2 images with the same passphrase also exposes this weakness since the passphrase is directly used as the key.

Use of qcow / qcow2 encryption is thus strongly discouraged. Users are recommended to use an alternative encryption technology such as the Linux dm-crypt / LUKS system.

`cluster_size`
: Changes the qcow2 cluster size (must be between 512 and 2M). Smaller cluster sizes can improve the image file size whereas larger cluster sizes generally provide better performance.

`preallocation`
: Preallocation mode (allowed values: `off`, `metadata`, `falloc`, `full`). An image with preallocated metadata is initially larger but can improve performance when the image needs to grow. `falloc` and `full` preallocations are like the same options of `raw` format, but sets up metadata also.

`lazy_refcounts`

If this option is set to `on`, reference count updates are postponed with the goal of avoiding metadata I/O and improving performance. This is particularly interesting with **cache=writethrough** which doesn't batch metadata updates. The tradeoff is that after a host crash, the reference count tables must be rebuilt, i.e. on the next open an (automatic) `qemu-img check -r all` is required, which may take some time.

This option can only be enabled if `compat=1.1` is specified.

`nocow`

If this option is set to `on`, it will turn off COW of the file. It's only valid on btrfs, no effect on other file systems.

Btrfs has low performance when hosting a VM image file, even more when the guest on the VM also using btrfs as file system. Turning off COW is a way to mitigate this bad performance. Generally there are two ways to turn off COW on btrfs: a) Disable it by mounting with nodatacow, then all newly created files will be NOCOW. b) For an empty file, add the NOCOW file attribute. That's what this option does.

Note: this option is only valid to new or empty files. If there is an existing file which is COW and has data blocks already, it couldn't be changed to NOCOW by setting `nocow=on`. One can issue `lsattr filename` to check if the NOCOW flag is set or not (Capital 'C' is NOCOW flag).

Other

QEMU also supports various other image file formats for compatibility with older QEMU versions or other hypervisors, including VMDK, VDI, VHD (vpc), VHDX, qcow1 and QED. For a full list of supported formats see `qemu-img --help`. For a more detailed description of these formats, see the QEMU Emulation User Documentation.

The main purpose of the block drivers for these formats is image conversion. For running VMs, it is recommended to convert the disk images to either raw or qcow2 in order to achieve good performance.

SEE ALSO

The HTML documentation of QEMU for more precise information and Linux user mode emulator invocation.

AUTHOR

Fabrice Bellard