==一般==

A qcow2 image file is organized in units of constant size, which are called

(host) clusters. A cluster is the unit in which all allocations are done,

both for actual guest data and for image metadata.

Likewise, the virtual disk as seen by the guest is divided into (guest)

clusters of the same size.

All numbers in qcow2 are stored in Big Endian byte order.

== Header ==

The first cluster of a qcow2 image contains the file header:

Byte 0 - 3: magic

QCOW magic string ("QFI\xfb")

4 - 7: version

Version number (valid values are 2 and 3)

8 - 15: backing\_file\_offset

Offset into the image file at which the backing file name

is stored (NB: The string is not null terminated). 0 if the

image doesn't have a backing file.

16 - 19: backing\_file\_size

Length of the backing file name in bytes. Must not be

longer than 1023 bytes. Undefined if the image doesn't have

a backing file.

20 - 23: cluster\_bits

Number of bits that are used for addressing an offset

within a cluster (1 << cluster\_bits is the cluster size).

Must not be less than 9 (i.e. 512 byte clusters).

Note: qemu as of today has an implementation limit of 2 MB

as the maximum cluster size and won't be able to open images

with larger cluster sizes.

24 - 31: size

Virtual disk size in bytes

32 - 35: crypt\_method

0 for no encryption

1 for AES encryption

36 - 39: l1\_size

Number of entries in the active L1 table

40 - 47: l1\_table\_offset

Offset into the image file at which the active L1 table

starts. Must be aligned to a cluster boundary.

48 - 55: refcount\_table\_offset

Offset into the image file at which the refcount table

starts. Must be aligned to a cluster boundary.

56 - 59: refcount\_table\_clusters

Number of clusters that the refcount table occupies

60 - 63: nb\_snapshots

Number of snapshots contained in the image

64 - 71: snapshots\_offset

Offset into the image file at which the snapshot table

starts. Must be aligned to a cluster boundary.

If the version is 3 or higher, the header has the following additional fields.

For version 2, the values are assumed to be zero, unless specified otherwise

in the description of a field.

72 - 79: incompatible\_features

Bitmask of incompatible features. An implementation must

fail to open an image if an unknown bit is set.

Bit 0: Dirty bit. If this bit is set then refcounts

may be inconsistent, make sure to scan L1/L2

tables to repair refcounts before accessing the

image.

Bit 1: Corrupt bit. If this bit is set then any data

structure may be corrupt and the image must not

be written to (unless for regaining

consistency).

Bits 2-63: Reserved (set to 0)

80 - 87: compatible\_features

Bitmask of compatible features. An implementation can

safely ignore any unknown bits that are set.

Bit 0: Lazy refcounts bit. If this bit is set then

lazy refcount updates can be used. This means

marking the image file dirty and postponing

refcount metadata updates.

Bits 1-63: Reserved (set to 0)

88 - 95: autoclear\_features

Bitmask of auto-clear features. An implementation may only

write to an image with unknown auto-clear features if it

clears the respective bits from this field first.

Bit 0: Bitmaps extension bit

This bit indicates consistency for the bitmaps

extension data.

It is an error if this bit is set without the

bitmaps extension present.

If the bitmaps extension is present but this

bit is unset, the bitmaps extension data must be

considered inconsistent.

Bits 1-63: Reserved (set to 0)

96 - 99: refcount\_order

Describes the width of a reference count block entry (width

in bits: refcount\_bits = 1 << refcount\_order). For version 2

images, the order is always assumed to be 4

(i.e. refcount\_bits = 16).

This value may not exceed 6 (i.e. refcount\_bits = 64).

100 - 103: header\_length

Length of the header structure in bytes. For version 2

images, the length is always assumed to be 72 bytes.

Directly after the image header, optional sections called header extensions can

be stored. Each extension has a structure like the following:

Byte 0 - 3: Header extension type:

0x00000000 - End of the header extension area

0xE2792ACA - Backing file format name

0x6803f857 - Feature name table

0x23852875 - Bitmaps extension

other - Unknown header extension, can be safely

ignored

4 - 7: Length of the header extension data

8 - n: Header extension data

n - m: Padding to round up the header extension size to the next

multiple of 8.

Unless stated otherwise, each header extension type shall appear at most once

in the same image.

If the image has a backing file then the backing file name should be stored in

the remaining space between the end of the header extension area and the end of

the first cluster. It is not allowed to store other data here, so that an

implementation can safely modify the header and add extensions without harming

data of compatible features that it doesn't support. Compatible features that

need space for additional data can use a header extension.

== Feature name table ==

The feature name table is an optional header extension that contains the name

for features used by the image. It can be used by applications that don't know

the respective feature (e.g. because the feature was introduced only later) to

display a useful error message.

The number of entries in the feature name table is determined by the length of

the header extension data. Each entry look like this:

Byte 0: Type of feature (select feature bitmap)

0: Incompatible feature

1: Compatible feature

2: Autoclear feature

1: Bit number within the selected feature bitmap (valid

values: 0-63)

2 - 47: Feature name (padded with zeros, but not necessarily null

terminated if it has full length)

== Bitmaps extension ==

The bitmaps extension is an optional header extension. It provides the ability

to store bitmaps related to a virtual disk. For now, there is only one bitmap

type: the dirty tracking bitmap, which tracks virtual disk changes from some

point in time.

The data of the extension should be considered consistent only if the

corresponding auto-clear feature bit is set, see autoclear\_features above.

The fields of the bitmaps extension are:

Byte 0 - 3: nb\_bitmaps

The number of bitmaps contained in the image. Must be

greater than or equal to 1.

Note: Qemu currently only supports up to 65535 bitmaps per

image.

4 - 7: Reserved, must be zero.

8 - 15: bitmap\_directory\_size

Size of the bitmap directory in bytes. It is the cumulative

size of all (nb\_bitmaps) bitmap headers.

16 - 23: bitmap\_directory\_offset

Offset into the image file at which the bitmap directory

starts. Must be aligned to a cluster boundary.

== Host cluster management ==

qcow2 manages the allocation of host clusters by maintaining a reference count

for each host cluster. A refcount of 0 means that the cluster is free, 1 means

that it is used, and >= 2 means that it is used and any write access must

perform a COW (copy on write) operation.

The refcounts are managed in a two-level table. The first level is called

refcount table and has a variable size (which is stored in the header). The

refcount table can cover multiple clusters, however it needs to be contiguous

in the image file.

It contains pointers to the second level structures which are called refcount

blocks and are exactly one cluster in size.

Given a offset into the image file, the refcount of its cluster can be obtained

as follows:

refcount\_block\_entries = (cluster\_size \* 8 / refcount\_bits)

refcount\_block\_index = (offset / cluster\_size) % refcount\_block\_entries

refcount\_table\_index = (offset / cluster\_size) / refcount\_block\_entries

refcount\_block = load\_cluster(refcount\_table[refcount\_table\_index]);

return refcount\_block[refcount\_block\_index];

Refcount table entry:

Bit 0 - 8: Reserved (set to 0)

9 - 63: Bits 9-63 of the offset into the image file at which the

refcount block starts. Must be aligned to a cluster

boundary.

If this is 0, the corresponding refcount block has not yet

been allocated. All refcounts managed by this refcount block

are 0.

Refcount block entry (x = refcount\_bits - 1):

Bit 0 - x: Reference count of the cluster. If refcount\_bits implies a

sub-byte width, note that bit 0 means the least significant

bit in this context.

== Cluster mapping ==

Just as for refcounts, qcow2 uses a two-level structure for the mapping of

guest clusters to host clusters. They are called L1 and L2 table.

The L1 table has a variable size (stored in the header) and may use multiple

clusters, however it must be contiguous in the image file. L2 tables are

exactly one cluster in size.

Given a offset into the virtual disk, the offset into the image file can be

obtained as follows:

l2\_entries = (cluster\_size / sizeof(uint64\_t))

l2\_index = (offset / cluster\_size) % l2\_entries

l1\_index = (offset / cluster\_size) / l2\_entries

l2\_table = load\_cluster(l1\_table[l1\_index]);

cluster\_offset = l2\_table[l2\_index];

return cluster\_offset + (offset % cluster\_size)

L1 table entry:

Bit 0 - 8: Reserved (set to 0)

9 - 55: Bits 9-55 of the offset into the image file at which the L2

table starts. Must be aligned to a cluster boundary. If the

offset is 0, the L2 table and all clusters described by this

L2 table are unallocated.

56 - 62: Reserved (set to 0)

63: 0 for an L2 table that is unused or requires COW, 1 if its

refcount is exactly one. This information is only accurate

in the active L1 table.

L2 table entry:

Bit 0 - 61: Cluster descriptor

62: 0 for standard clusters

1 for compressed clusters

63: 0 for a cluster that is unused or requires COW, 1 if its

refcount is exactly one. This information is only accurate

in L2 tables that are reachable from the active L1

table.

Standard Cluster Descriptor:

Bit 0: If set to 1, the cluster reads as all zeros. The host

cluster offset can be used to describe a preallocation,

but it won't be used for reading data from this cluster,

nor is data read from the backing file if the cluster is

unallocated.

With version 2, this is always 0.

1 - 8: Reserved (set to 0)

9 - 55: Bits 9-55 of host cluster offset. Must be aligned to a

cluster boundary. If the offset is 0, the cluster is

unallocated.

56 - 61: Reserved (set to 0)

Compressed Clusters Descriptor (x = 62 - (cluster\_bits - 8)):

Bit 0 - x: Host cluster offset. This is usually \_not\_ aligned to a

cluster boundary!

x+1 - 61: Compressed size of the images in sectors of 512 bytes

If a cluster is unallocated, read requests shall read the data from the backing

file (except if bit 0 in the Standard Cluster Descriptor is set). If there is

no backing file or the backing file is smaller than the image, they shall read

zeros for all parts that are not covered by the backing file.

== Snapshots ==

qcow2 supports internal snapshots. Their basic principle of operation is to

switch the active L1 table, so that a different set of host clusters are

exposed to the guest.

When creating a snapshot, the L1 table should be copied and the refcount of all

L2 tables and clusters reachable from this L1 table must be increased, so that

a write causes a COW and isn't visible in other snapshots.

When loading a snapshot, bit 63 of all entries in the new active L1 table and

all L2 tables referenced by it must be reconstructed from the refcount table

as it doesn't need to be accurate in inactive L1 tables.

A directory of all snapshots is stored in the snapshot table, a contiguous area

in the image file, whose starting offset and length are given by the header

fields snapshots\_offset and nb\_snapshots. The entries of the snapshot table

have variable length, depending on the length of ID, name and extra data.

Snapshot table entry:

Byte 0 - 7: Offset into the image file at which the L1 table for the

snapshot starts. Must be aligned to a cluster boundary.

8 - 11: Number of entries in the L1 table of the snapshots

12 - 13: Length of the unique ID string describing the snapshot

14 - 15: Length of the name of the snapshot

16 - 19: Time at which the snapshot was taken in seconds since the

Epoch

20 - 23: Subsecond part of the time at which the snapshot was taken

in nanoseconds

24 - 31: Time that the guest was running until the snapshot was

taken in nanoseconds

32 - 35: Size of the VM state in bytes. 0 if no VM state is saved.

If there is VM state, it starts at the first cluster

described by first L1 table entry that doesn't describe a

regular guest cluster (i.e. VM state is stored like guest

disk content, except that it is stored at offsets that are

larger than the virtual disk presented to the guest)

36 - 39: Size of extra data in the table entry (used for future

extensions of the format)

variable: Extra data for future extensions. Unknown fields must be

ignored. Currently defined are (offset relative to snapshot

table entry):

Byte 40 - 47: Size of the VM state in bytes. 0 if no VM

state is saved. If this field is present,

the 32-bit value in bytes 32-35 is ignored.

Byte 48 - 55: Virtual disk size of the snapshot in bytes

Version 3 images must include extra data at least up to

byte 55.

variable: Unique ID string for the snapshot (not null terminated)

variable: Name of the snapshot (not null terminated)

variable: Padding to round up the snapshot table entry size to the

next multiple of 8.

== Bitmaps ==

As mentioned above, the bitmaps extension provides the ability to store bitmaps

related to a virtual disk. This section describes how these bitmaps are stored.

All stored bitmaps are related to the virtual disk stored in the same image, so

each bitmap size is equal to the virtual disk size.

Each bit of the bitmap is responsible for strictly defined range of the virtual

disk. For bit number bit\_nr the corresponding range (in bytes) will be:

[bit\_nr \* bitmap\_granularity .. (bit\_nr + 1) \* bitmap\_granularity - 1]

Granularity is a property of the concrete bitmap, see below.

=== Bitmap directory ===

Each bitmap saved in the image is described in a bitmap directory entry. The

bitmap directory is a contiguous area in the image file, whose starting offset

and length are given by the header extension fields bitmap\_directory\_offset and

bitmap\_directory\_size. The entries of the bitmap directory have variable

length, depending on the lengths of the bitmap name and extra data. These

entries are also called bitmap headers.

Structure of a bitmap directory entry:

Byte 0 - 7: bitmap\_table\_offset

Offset into the image file at which the bitmap table

(described below) for the bitmap starts. Must be aligned to

a cluster boundary.

8 - 11: bitmap\_table\_size

Number of entries in the bitmap table of the bitmap.

12 - 15: flags

Bit

0: in\_use

The bitmap was not saved correctly and may be

inconsistent.

1: auto

The bitmap must reflect all changes of the virtual

disk by any application that would write to this qcow2

file (including writes, snapshot switching, etc.). The

type of this bitmap must be 'dirty tracking bitmap'.

2: extra\_data\_compatible

This flags is meaningful when the extra data is

unknown to the software (currently any extra data is

unknown to Qemu).

If it is set, the bitmap may be used as expected, extra

data must be left as is.

If it is not set, the bitmap must not be used, but

both it and its extra data be left as is.

Bits 3 - 31 are reserved and must be 0.

16: type

This field describes the sort of the bitmap.

Values:

1: Dirty tracking bitmap

Values 0, 2 - 255 are reserved.

17: granularity\_bits

Granularity bits. Valid values: 0 - 63.

Note: Qemu currently doesn't support granularity\_bits

greater than 31.

Granularity is calculated as

granularity = 1 << granularity\_bits

A bitmap's granularity is how many bytes of the image

accounts for one bit of the bitmap.

18 - 19: name\_size

Size of the bitmap name. Must be non-zero.

Note: Qemu currently doesn't support values greater than

1023.

20 - 23: extra\_data\_size

Size of type-specific extra data.

For now, as no extra data is defined, extra\_data\_size is

reserved and should be zero. If it is non-zero the

behavior is defined by extra\_data\_compatible flag.

variable: extra\_data

Extra data for the bitmap, occupying extra\_data\_size bytes.

Extra data must never contain references to clusters or in

some other way allocate additional clusters.

variable: name

The name of the bitmap (not null terminated), occupying

name\_size bytes. Must be unique among all bitmap names

within the bitmaps extension.

variable: Padding to round up the bitmap directory entry size to the

next multiple of 8. All bytes of the padding must be zero.

=== Bitmap table ===

Each bitmap is stored using a one-level structure (as opposed to two-level

structures like for refcounts and guest clusters mapping) for the mapping of

bitmap data to host clusters. This structure is called the bitmap table.

Each bitmap table has a variable size (stored in the bitmap directory entry)

and may use multiple clusters, however, it must be contiguous in the image

file.

Structure of a bitmap table entry:

Bit 0: Reserved and must be zero if bits 9 - 55 are non-zero.

If bits 9 - 55 are zero:

0: Cluster should be read as all zeros.

1: Cluster should be read as all ones.

1 - 8: Reserved and must be zero.

9 - 55: Bits 9 - 55 of the host cluster offset. Must be aligned to

a cluster boundary. If the offset is 0, the cluster is

unallocated; in that case, bit 0 determines how this

cluster should be treated during reads.

56 - 63: Reserved and must be zero.

=== Bitmap data ===

As noted above, bitmap data is stored in separate clusters, described by the

bitmap table. Given an offset (in bytes) into the bitmap data, the offset into

the image file can be obtained as follows:

image\_offset(bitmap\_data\_offset) =

bitmap\_table[bitmap\_data\_offset / cluster\_size] +

(bitmap\_data\_offset % cluster\_size)

This offset is not defined if bits 9 - 55 of bitmap table entry are zero (see

above).

Given an offset byte\_nr into the virtual disk and the bitmap's granularity, the

bit offset into the image file to the corresponding bit of the bitmap can be

calculated like this:

bit\_offset(byte\_nr) =

image\_offset(byte\_nr / granularity / 8) \* 8 +

(byte\_nr / granularity) % 8

If the size of the bitmap data is not a multiple of the cluster size then the

last cluster of the bitmap data contains some unused tail bits. These bits must

be zero.

=== Dirty tracking bitmaps ===

Bitmaps with 'type' field equal to one are dirty tracking bitmaps.

When the virtual disk is in use dirty tracking bitmap may be 'enabled' or

'disabled'. While the bitmap is 'enabled', all writes to the virtual disk

should be reflected in the bitmap. A set bit in the bitmap means that the

corresponding range of the virtual disk (see above) was written to while the

bitmap was 'enabled'. An unset bit means that this range was not written to.

The software doesn't have to sync the bitmap in the image file with its

representation in RAM after each write. Flag 'in\_use' should be set while the

bitmap is not synced.

In the image file the 'enabled' state is reflected by the 'auto' flag. If this

flag is set, the software must consider the bitmap as 'enabled' and start

tracking virtual disk changes to this bitmap from the first write to the

virtual disk. If this flag is not set then the bitmap is disabled.