qcow2 镜像格式是 QEMU 模拟器支持的一种磁盘镜像。它也是可以用一个文件的形式来表示一块固定大小的块设备磁盘。与普通的 raw 格式的镜像相比，有以下特性：

更小的空间占用，即使文件系统不支持空洞(holes)；

支持写时拷贝（COW, copy-on-write），镜像文件只反映底层磁盘的变化；

支持快照（snapshot），镜像文件能够包含多个快照的历史；

可选择基于 zlib 的压缩方式

可以选择 AES 加密

qemu-img命令可以用来创建qcow2镜像，或者将qcow2文件转换成raw格式文件，等其它功能：

$> qemu-img create -f qcow2 test.qcow2 4G

Formating 'test.qcow2', fmt=qcow2, size=4194304 kB

$> qemu-img convert test.qcow2 -O raw test.img

qcow2 镜像文件格式

头部信息

每一个 qcow2 文件都以一个大端（big-endian）格式的头开始，结构如下:

typedef struct QCowHeader {

uint32\_t magic;

uint32\_t version;

uint64\_t backing\_file\_offset;

uint32\_t backing\_file\_size;

uint32\_t cluster\_bits;

uint64\_t size; /\* in bytes \*/

uint32\_t crypt\_method;

uint32\_t l1\_size;

uint64\_t l1\_table\_offset;

uint64\_t refcount\_table\_offset;

uint32\_t refcount\_table\_clusters;

uint32\_t nb\_snapshots;

uint64\_t snapshots\_offset;

} QcowHeader;

前 4 个比特包含了字符 Q，F，I，然后是 0xfb，实例中的 5146 49fb 是 magic 字段。

接下来的 4 个比特包含了该镜像文件的版本号，实例中的 0000 0002 是 version 字段，代表使用的是 qcow2 版本。

当前存在两种版本的格式，版本1和版本2。

backing\_file\_offset 占用 8 个字节，实例中 0000 0000 0000 0000，给出相对于qcow2文件起始位置的偏移，指出一个字符串的位置，该字符串为backing file文件的绝对路径。该字符串为backing file文件的绝对路径。由于该字符串不是以'\0'结束，所以backing\_file\_size指出字符串的长度。如果当前镜像是一个copy-on-write镜像，则存在backing file文件，否则没有；

backing\_file\_size 给出了一个不以 null 结尾的字符串的长度，实例中为 0000 0000。如果这个镜像文件是一个写时拷贝的，那么它是原始文件的路径。

cluster\_bits，32 位（0000 0010），描述了如何映射一个镜像的地址到一个本地文件，它决定了在一个 cluster 中，偏移地址的低位是如何作为索引的。因为 L2 表占用了一个单独的 cluster 并且包含 8 字节的表项（entry），所以 cluster\_bits 只有不足 3 个位，作为 L2 表的索引。

接下来的 size ，8 字节代表了该镜像文件所表示的块设备的大小，单位字节；实例中为 0000 0002 8000 0000 字节，也就是 10G 的空间。

crypt\_method 如果为 1 代表使用 AES 加密，0表示没有加密。

l1\_size（0000 0014）和 l1\_table\_offset（0000 0000 0003 0000::）分别给出了 L1 表大小和偏移量。

refcount\_table\_offset 给出 refcount 表的偏移量（0000 0000 0001 0000）而 refcount\_table\_clusters 描述了以 cluster 为单位的 refcount 表的大小（0000 0001）。

nb\_snapshots 给出了该镜像包含的快照数量（0000 0000）， snapshots\_offset 给出每个快照到 QCowSnapshotHeader 的偏移量（0000 0000 0000 0000），每个快照都会有这样一个header。

一个典型的 qcow2 镜像文件包含一下几部分：

上文中提到的头部信息

L1 表

refcount 表，仍然是簇对齐的

一个或者多个 refcount 块

快照头，第一个header要求簇对齐，之后的header要求8字节对齐；

L2 表，每一个table占据一个单独的cluster；

数据 cluster

L2 表，每一个table占据一个单独的cluster；

数据 cluster

2 级查找

在 qcow2 中，磁盘的内容是保存在 cluster 中（每个 cluster 包含一些大小为 512 字节的扇区）。为了找到给定地址所在的 cluster，我们需要查找两张表，L1->L2。L1 表保存一组到 L2 表的偏移量，L2 表保存一组到 cluster 的偏移量；

所以一个地址要根据 cluster\_bits（64 位）的设置分成 3 部分，比如说 cluster\_bits=12；

低 12 位是一个 4Kb cluster 的偏移（2 的 12 次方=4Kb）；

接下来 9 位是包含 512 个表项目的 L2 表；

剩下 43 位的代表 L1 表偏移量。

注意，L1 table的最小值，可以通过给定磁盘镜像的大小来计算，公式如下：

l1\_size = round\_up(disk\_size / (cluster\_size \* l2\_size), cluster\_size)

为了获取一个给定地址（64 位）的偏移位置：

从 Head 域中的 l1\_table\_offset 取得 L1 表的地址

用前（64-l2\_bits-cluster\_bits）位地址去索引 L1 表

在 L1 表中的偏移量获得 L2 表的地址

用地址中的接下来的 l2\_bits 去索引 L2 表，获得一个 64 位的表项

用 L2 表中的偏移量获得 cluster 的地址

用地址中剩下的 cluster\_bits 位去索引该 cluster，获得该数据块

如果 L1 表和 L2 表中的偏移量都是空，这块区域就尚未被镜像文件分配。

注意 L1 表和 L2 表中的偏移量的前两位被保留，用做表示'copied' 或'compressed'。

引用计数

每一个cluster都有一个引用计数，cluster可以被删除，但前提条件是没有任何快照再使用这个cluster。

针对每一个cluster的2个字节的引用计数，存放在cluster sized blocks。通过refcount\_table\_offset字段可以获取到refcount table的位置，refcount\_table\_clusters字段给出refcount table的大小（单位为cluster），refcount table给出了这些refcount blocks在镜像文件中的偏移地址。

为了获取一个给定的cluster的引用计数，你需要将cluster offset划分成refcount table offset和refcount block offset。一个refcount block是一个单独的cluster，这个cluster里包含了若干个2字节的项，低（cluster\_size -1)位作为block offset，剩余的位作为table offset。

qcow2有一个优化处理，任何一个L1或L2表项指向的cluster的引用计数为1，则L1/L2表项的最高有效位被置上“copied”标记。这表明没有快照在使用这个cluster，所以这个cluster可以马上写入数据，而不需要复制一份给快照使用。

Copy-on-Write 镜像文件

qcow2 镜像可以用来保存另一个镜像文件的变化，它并不去修改原始镜像文件，只记录与原始镜像文件的不同即可，这种镜像文件就叫做 copy-on-write 镜像。虽然是一个单独的文件，但它的大部分的数据都来自原始镜像，只有跟原始镜像文件相比有变化的 cluster 才会被记录下来。

这很容易去实现，在头部信息中记录原始文件路径即可。当需要从一个 copy-on-write 镜像文件中读取一个 cluster 的时候，首先检查这块区域是否已经在该镜像文件中被分配，如果没有就从原始文件读取。

快照

快照有些类似 Copy-On-Write 文件，但区别是快照是一个可写的。快照就是原始文件本身（内部快照）。它既包含做快照之前的原始文件部分，它本身也包含可写的部分。

每一个快照都包含如下的头部结构：

typedef struct QCowSnapshotHeader {

/\* header is 8 byte aligned \*/

uint64\_t l1\_table\_offset;

uint32\_t l1\_size;

uint16\_t id\_str\_size;

uint16\_t name\_size;

uint32\_t date\_sec;

uint32\_t date\_nsec;

uint64\_t vm\_clock\_nsec;

uint32\_t vm\_state\_size;

uint32\_t extra\_data\_size; /\* for extension \*/

/\* extra data follows \*/

/\* id\_str follows \*/

/\* name follows \*/

} QcowSnapshotHeader;

各字段介绍如下：

快照有名字和ID，都是字符串，id\_str\_size，name\_size给出字符串长度，字符串紧接在QCowSnapshotHeader后面；

快照至少有原来的L1 table的副本，其通过l1\_table\_offset和l1\_size来定位；

在快照被创建的时候，qemu会调用gettimeofday()，快照时间被保存在date\_sec和date\_nsec字段中；

vm\_clock\_nsec给出VM clock当前的状态；

vm\_state\_size表示作为快照的一部分被保存的虚拟机状态的大小。这个状态被保存在原来L1 table的位置，直接在镜像header的后面；

extra\_data\_size表示在QCowSnapshotHeader之后的扩展数据的长度，不包括id和name字符串。这一段扩展数据是留给以后用的。

创建一个快照，就会添加一个QCowSnapshotHeader，然后复制一份L1 table，同时会增加所有L2 table和数据clusters的被L1 table引用的引用计数。打完快照之后，如果任何在这个镜像中的L2 table或者data clusters被修改了——也就是，如果一个cluster的引用计数大于1，且"copied"标记被置上了——qemu则会先复制一份这个cluster，然后再写入数据。就这样，所有的快照都不会被修改。

qcow2 的其他特性

qcow2 支持压缩，它允许每个簇（cluster）单独使用 zlib 压缩。它也支持使用 128 位的 AES 密钥进行加密。

qcow2镜像格式支持压缩特性，其允许每一个cluster独立的通过zlib进行压缩。

/\*cluster offset表示一个簇在qcow2文件中的偏移，其最高的2位是标记位\*/

从L2 table中获取cluster offset的流程如下：

如果cluster offset的第二最高有效位是1，则这是一个被压缩的cluster；

cluster offset中之后的cluster\_bits - 8 位是这个压缩过的cluster的大小，单位是sectors；

cluster offset剩余的位是压缩的cluster在文件中的实际偏移地址。

加密

qcow2格式，也支持针对cluster的加密。

如果QCowHeader中的crypt\_method字段被置为1，则会采用一个16个字符的密码作为128位AES key。

每一个Cluster中的每一个sector都是通过AES密码块链接模式来单独加密，采用sector的偏移地址（小端模式）来作为128位初始化向量的头64位。

qcow镜像——上一代镜像

qcow2格式相对于qcow格式的不同点有：

支持快照的概念，qcow只支持增量镜像；

在qcow2中，引入了cluster的引用计数的概念；引用计数也被用来支持快照；

在qcow2中，L2 table将一直占一个单独的cluster； 之前，是通过QCowHeader中的l2\_bits来确定的；

压缩的cluster的大小，现在单位为sector，之前是字节。