NAME

qemu-storage-daemon − QEMU storage daemon

SYNOPSIS

qemu−storage−daemon [options]

DESCRIPTION

qemu−storage−daemon provides disk image functionality from QEMU, **qemu−img**, and **qemu−nbd** in a long−running process controlled via QMP commands without running a virtual machine. It can export disk images, run block job operations, and perform other disk−related operations. The daemon is controlled via a QMP monitor and initial configuration from the command−line.

The daemon offers the following subset of QEMU features:

- Block nodes
- Block jobs
- Block exports
- Throttle groups
- Character devices
- Crypto and secrets
- QMP
- IOThreads

Commands can be sent over a QEMU Monitor Protocol (QMP) connection. See the **qemu−storage−daemon−qmp−ref(7)** manual page for a description of the commands.

The daemon runs until it is stopped using the **quit** QMP command or SIGINT/SIGHUP/SIGTERM.

Warning: Never 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 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.

−−blockdev BLOCKDEVDEF
is a block node definition. See the **qemu(1)** manual page for a description of block node properties and the **qemu−block−drivers(7)** manual page for a description of driver−specific parameters.

−−chardev CHARDEVDEF
is a character device definition. See the **qemu(1)** manual page for a description of character device properties. A common character device definition configures a UNIX domain socket:

```
--chardev socket,id=char1,path=/var/run/qsd-qmp.sock,server=on,wait=off
```

−−export [type=]nbd,id=<id>,node−name=<node−name>[,name=<export−name>][,writable=on|off][,bitmap=<name>]

−−export [type=]vhost−user−blk,id=<id>,node−name=<node−name>,addr.type=unix,addr.path=<socket−path>[,writable=o ical−block−size=<block−size>][,num−queues=<num−queues>]

−−export [type=]vhost−user−blk,id=<id>,node−name=<node−name>,addr.type=fd,addr.str=<fd>[,writable=on|off][,logical−block−size=<block−size>][,num−queues=<num−queues>]

−−export [type=]fuse,id=<id>,node−name=<node−name>,mountpoint=<file>[,growable=on|off][,writable=on|off]
is a block export definition. **node−name** is the block node that should be exported. **writable** determines whether or not the export allows write requests for modifying data (the default is off).

The **nbd** export type requires **−−nbd−server** (see below). **name** is the NBD export name (if not specified, it defaults to the given **node−name**). **bitmap** is the name of a dirty bitmap reachable from the block node, so the NBD client can use NBD_OPT_SET_META_CONTEXT with the metadata context name "qemu:dirty−bitmap:BITMAP" to inspect the bitmap.

The **vhost−user−blk** export type takes a vhost−user socket address on which it accept incoming connections. Both **addr.type=unix,addr.path=<socket−path>** for UNIX domain sockets and **addr.type=fd,addr.str=<fd>** for file descriptor passing are supported. **logical−block−size** sets the logical block size in bytes (the default is 512). **num−queues** sets the number of virtqueues (the default is 1).

The **fuse** export type takes a mount point, which must be a regular file, on which to export the given block node. That file will not be changed, it will just appear to have the block node's content while the export is active (very much like mounting a filesystem on a directory does not change what the directory contains, it only shows a different content while the filesystem is mounted). Consequently, applications that have opened the given file before the export became active will continue to see its original content. If **growable** is set, writes after the end of the exported file will grow the block node to fit.

−−monitor MONITORDEF
is a QMP monitor definition. See the **qemu(1)** manual page for a description of QMP monitor properties. A common QMP monitor definition configures a monitor on character device **char1**:

```
--monitor chardev=char1
```

−−nbd−server addr.type=inet,addr.host=<host>,addr.port=<port>[,tls−creds=<id>][,tls−authz=<id>][,max−connections=<n>]

−−nbd−server addr.type=unix,addr.path=<path>[,tls−creds=<id>][,tls−authz=<id>][,max−connections=<n>]

−−nbd−server addr.type=fd,addr.str=<fd>[,tls−creds=<id>][,tls−authz=<id>][,max−connections=<n>]

is a server for NBD exports. Both TCP and UNIX domain sockets are supported. A listen socket can be provided via file descriptor passing (see Examples below). TLS encryption can be configured using **−−object** tls−creds−* and authz−* secrets (see below).

To configure an NBD server on UNIX domain socket path **/var/run/qsd−nbd.sock**:

```
--nbd-server addr.type=unix,addr.path=/var/run/qsd-nbd.sock
```

−−object help

−−object <type>,help

−−object <type>[,<property>=<value>...]

is a QEMU user creatable object definition. List object types with **help**. List object properties with **<type>,help**. See the **qemu(1)** manual page for a description of the object properties.

−−pidfile PATH

is the path to a file where the daemon writes its pid. This allows scripts to stop the daemon by sending a signal:

```
$ kill -SIGTERM $(<path/to/qsd.pid)
```

A file lock is applied to the file so only one instance of the daemon can run with a given pid file path. The daemon unlinks its pid file when terminating.

The pid file is written after chardevs, exports, and NBD servers have been created but before accepting connections. The daemon has started successfully when the pid file is written and clients may begin connecting.

EXAMPLES

Launch the daemon with QMP monitor socket **qmp.sock** so clients can execute QMP commands:

```
$ qemu-storage-daemon \
    --chardev socket,path=qmp.sock,server=on,wait=off,id=char1 \
    --monitor chardev=char1
```

Launch the daemon from Python with a QMP monitor socket using file descriptor passing so there is no need to busy wait for the QMP monitor to become available:

```
#!/usr/bin/env python3
import subprocess
import socket

sock_path = '/var/run/qmp.sock'

with socket.socket(socket.AF_UNIX, socket.SOCK_STREAM) as listen_sock:
    listen_sock.bind(sock_path)
    listen_sock.listen()

    fd = listen_sock.fileno()

    subprocess.Popen(
        ['qemu-storage-daemon',
         '--chardev', f'socket,fd={fd},server=on,id=char1',
         '--monitor', 'chardev=char1'],
        pass_fds=[fd],
    )

# listen_sock was automatically closed when leaving the 'with' statement
# body. If the daemon process terminated early then the following connect()
# will fail with "Connection refused" because no process has the listen
# socket open anymore. Launch errors can be detected this way.

qmp_sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
qmp_sock.connect(sock_path)
...QMP interaction...
```

The same socket spawning approach also works with the **−−nbd−server addr.type=fd,addr.str=<fd>** and **−−export type=vhost−user−blk,addr.type=fd,addr.str=<fd>** options.

Export raw image file **disk.img** over NBD UNIX domain socket **nbd.sock**:

```
    $ qemu−storage−daemon \
        −−blockdev driver=file,node−name=disk,filename=disk.img \
        −−nbd−server addr.type=unix,addr.path=nbd.sock \
        −−export type=nbd,id=export,node−name=disk,writable=on
```

Export a qcow2 image file **disk.qcow2** as a vhosts−user−blk device over UNIX domain socket **vhost−user−blk.sock**:

```
    $ qemu−storage−daemon \
        −−blockdev driver=file,node−name=file,filename=disk.qcow2 \
        −−blockdev driver=qcow2,node−name=qcow2,file=file \
        −−export type=vhost−user−blk,id=export,addr.type=unix,addr.path=vhost−user−
```

Export a qcow2 image file **disk.qcow2** via FUSE on itself, so the disk image file will then appear as a raw image:

```
    $ qemu−storage−daemon \
        −−blockdev driver=file,node−name=file,filename=disk.qcow2 \
        −−blockdev driver=qcow2,node−name=qcow2,file=file \
        −−export type=fuse,id=export,node−name=qcow2,mountpoint=disk.qcow2,writabl
```

SEE ALSO

qemu(1), **qemu−block−drivers(7)**, **qemu−storage−daemon−qmp−ref(7)**

COPYRIGHT