Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'mdmon.8'

$ man mdmon.8

MDMON(8)   System Manager's Manual   MDMON(8)

NAME

    mdmon - monitor MD external metadata arrays

SYNOPSIS

    mdmon [--all] [--takeover] [--foreground] CONTAINER

OVERVIEW

    The 2.6.27 kernel brings the ability to support external metadata ar?

    rays. External metadata implies that user space handles all updates to

    the metadata. The kernel's responsibility is to notify user space when

    a "metadata event" occurs, like disk failures and clean-to-dirty tran?

    sitions.   The kernel, in important cases, waits for user space to take

    action on these notifications.

DESCRIPTION

  Metadata updates:

    To service metadata update requests a daemon, mdmon, is introduced.

    Mdmon is tasked with polling the sysfs namespace looking for changes in

    array_state, sync_action, and per disk state attributes. When a change

    is detected it calls a per metadata type handler to make modifications

to the metadata. The following actions are taken:

array_state - inactive

Clear the dirty bit for the volume and let the array be

stopped

array_state - write pending

Set the dirty bit for the array and then set array_state

to active. Writes are blocked until userspace writes ac?

tive.

array_state - active-idle

The safe mode timer has expired so set array state to

clean to block writes to the array

array_state - clean

Clear the dirty bit for the volume

array_state - read-only

This is the initial state that all arrays start at.   md?

mon takes one of the three actions:

1/   Transition the array to read-auto keeping the

dirty bit clear if the metadata handler determines

that the array does not need resyncing or other

modification

2/   Transition the array to active if the metadata

handler determines a resync or some other manipu?

lation is necessary

3/   Leave the array read-only if the volume is marked

to not be monitored; for example, the metadata

version has been set to "external:-dev/md127" in?

stead of "external:/dev/md127"

sync_action - resync-to-idle

Notify the metadata handler that a resync may have com?

pleted. If a resync process is idled before it completes

this event allows the metadata handler to checkpoint

resync.

sync_action - recover-to-idle

A spare may have completed rebuilding so tell the meta?
data handler about the state of each disk. This is the
metadata handler's opportunity to clear any "out-of-sync"
bits and clear the volume's degraded status. If a recov?
ery process is idled before it completes this event al?
lows the metadata handler to checkpoint recovery.

<disk>/state - faulty

A disk failure kicks off a series of events. First, no?
tify the metadata handler that a disk has failed, and
then notify the kernel that it can unblock writes that
were dependent on this disk. After unblocking the kernel
this disk is set to be removed+ from the member array.
Finally the disk is marked failed in all other member ar?
rays in the container.

+ Note This behavior differs slightly from native MD ar?
rays where removal is reserved for a mdadm --remove
event. In the external metadata case the container holds
the final reference on a block device and a mdadm --re?
move <container> <victim> call is still required.

Containers:

External metadata formats, like DDF, differ from the native MD metadata
formats in that they define a set of disks and a series of sub-arrays
within those disks. MD metadata in comparison defines a 1:1 relation?
ship between a set of block devices and a RAID array. For example to
create 2 arrays at different RAID levels on a single set of disks, MD
metadata requires the disks be partitioned and then each array can be
created with a subset of those partitions. The supported external for?
mats perform this disk carving internally.

Container devices simply hold references to all member disks and allow
tools like mdmon to determine which active arrays belong to which con?
tainer. Some array management commands like disk removal and disk add
are now only valid at the container level. Attempts to perform these
actions on member arrays are blocked with error messages like:

"mdadm: Cannot remove disks from a ?member? array, perform this

operation on the parent container"

Containers are identified in /proc/mdstat with a metadata version

string "external:<metadata name>". Member devices are identified by

"external:/<container device>/<member index>", or "external:-<container

device>/<member index>" if the array is to remain readonly.

OPTIONS

CONTAINER

The container device to monitor. It can be a full path like

/dev/md/container, or a simple md device name like md127.

--foreground

Normally, mdmon will fork and continue in the background.

Adding this option will skip that step and run mdmon in the

foreground.

--takeover

This instructs mdmon to replace any active mdmon which is cur?

rently monitoring the array. This is primarily used late in the

boot process to replace any mdmon which was started from an

initramfs before the root filesystem was mounted.   This avoids

holding a reference on that initramfs indefinitely and ensures

that the pid and sock files used to communicate with mdmon are

in a standard place.

--all This tells mdmon to find any active containers and start moni?

toring each of them if appropriate. This is normally used with

--takeover late in the boot sequence. A separate mdmon process

is started for each container as the --all argument is over-

written with the name of the container. To allow for containers

with names longer than 5 characters, this argument can be arbi?

trarily extended, e.g. to --all-active-arrays.

Note that

mdmon is automatically started by mdadm when needed and so does

not need to be considered when working with RAID arrays.   The

only times it is run other than by mdadm is when the boot

scripts need to restart it after mounting the new root filesys‐

tem.

START UP AND SHUTDOWN

As mdmon needs to be running whenever any filesystem on the monitored
device is mounted there are special considerations when the root
filesystem is mounted from an mdmon monitored device. Note that in
general mdmon is needed even if the filesystem is mounted read-only as
some filesystems can still write to the device in those circumstances,
for example to replay a journal after an unclean shutdown.

When the array is assembled by the initramfs code, mdadm will automati‐
cally start mdmon as required. This means that mdmon must be installed
on the initramfs and there must be a writable filesystem (typically
tmpfs) in which mdmon can create a .pid and .sock file. The particular
filesystem to use is given to mdmon at compile time and defaults to
/run/mdadm.

This filesystem must persist through to shutdown time.

After the final root filesystem has be instantiated (usually with
pivot_root) mdmon should be run with --all --takeover so that the mdmon
running from the initramfs can be replaced with one running in the main
root, and so the memory used by the initramfs can be released.

At shutdown time, mdmon should not be killed along with other pro‐
cesses. Also as it holds a file (socket actually) open in /dev (by de‐
fault) it will not be possible to unmount /dev if it is a separate
filesystem.

EXAMPLES

  mdmon --all-active-arrays --takeover

Any mdmon which is currently running is killed and a new instance is
started. This should be run during in the boot sequence if an
initramfs was used, so that any mdmon running from the initramfs will
not hold the initramfs active.

SEE ALSO

mdadm(8), md(4).