Storage event reporting and monitoring - PoC

In the previous blog post we have presented a proposal for reporting and monitoring storage-related events using *journald* and structured logging. To test if the proposal is viable we need some proof of concept. Such a PoC should demonstrate the complexity of the proposed solution as well as the sufficiency of the proposed set of stored (logged) items and the catalog entry.

Out of all the storage subsystems like LVM, Btrfs,... and MD RAID, only the last one mentioned provides a monitoring tool that can run any executable when an event appears, passing it the information about the event as arguments. It is thus a logical choice for the implementation of a PoC tool that reports such events using structured logging.

The monitoring tools for MD RAID is actually the well-known `mdadm` tool, just run with the `--monitor` option. There is also the systemd service `mdmonitor.service` [1] that makes sure the monitoring is started as part of the booting process. But only when the configuration file `/etc/mdadm.conf` exists, plus it has to specify which tool to run or where to send emails with the event reports. This is described in the mdadm(8) man page (in the *Monitor mode* section) together with specifications of all events that can be reported.

The program specified by the `PROGRAM` directive in `/etc/mdadm.conf` is run with two or three arguments for every event. The first one is the name of the event, the second one is the path of the MD RAID array the event is reported for, and in some cases the third argument specifies a member device the event is related to (e.g. when a member device is marked as failed).

Here is the trivial implementation of the necessary code to handle the events:

```
if (argc < 3)
    error (1, 0, "Not enough arguments given!");

event = argv[1];
md_dev = argv[2];

if (argc > 3)
    member = argv[3];
```

[1] at least on Fedora/RHEL systems, but surely in others too

Once the program has the data it can get from `mdadm`, it needs to derive some more to have everything it needs to report the event using structured logging with all the fields specified by the proposal. Every report is supposed to describe a change in device's state. Thus the program has to derive in which state a given device (MD RAID array or a member) is together with the severity of such change/state and a description of the change/state to make it easier to understand for users.

`mdadm` only gives the program a name of the event with no details. What every event means is described in the mdadm(8) man page so the easiest way for the PoC is to just have the information stored in itself and do matching based on the event name. So an array of structures like these:

```
{"DeviceDisappeared", "deactivated", "MD array was deactivated", LOG_WARNING},
{"RebuildStarted", "rebuilding", "MD array is rebuilding", LOG_INFO},
{"RebuildFinished", "rebuilt", "MD array is now rebuilt", LOG_INFO},
{"Fail", "failed", "Device was marked as failed", LOG_WARNING},
...
```

where the first item is the event name as defined by `mdadm`, which allows the program to do the matching, and then the additional/derived information follows.

With all the data it needs, the program can use the *journal* API to log the message (IOW, save the data):

```
ret = sd_journal_send ("MESSAGE_ID=3183267b90074a4595e91daef0e01462",
                       "MESSAGE=mdadm reported %s on the MD device %s", event, md_dev,
                       "SOURCE=MD RAID", "SOURCE_MAN=mdadm(8)",
                       "DEVICE=%s", md_dev, "STATE=%s", info->state,
                       "PRIORITY=%i", info->priority,
                       "PRIORITY_DESC=%s", log_lvl_desc[info->priority],
                       "DETAILS=%s", info->details,
                       md_fields (event, md_dev, ""),
                       NULL);
```

It should be obvious what's going on in the above function call. `info` is the matched structure with extra information, `log_lvl_desc` provides a string description for the given log level (an `enum` and thus `int`). Last but not least, there's the `md_fields` macro that provides extra fields specific to an MD RAID event so that no information is lost when reporting the event:

```
#define md_fields(event, md_dev, member) "MD_EVENT=%s", event, \
                                         "MD_ARRAY=%s", md_dev, \
                                         "MD_MEMBER=%s", member
```