Whatever happened to cman?

Christine “Chrissie” Caulfield, Red Hat

ccaulfie@redhat.com

Version history

0.1	30th January 2009	First draft
0.2	3rd February 2009	Add a chapter on migrating from libcman
0.3	27th February 2009	Add some commas and fix some typos
0.4	19th February 2013	Update for Corosync 2, including removing all the RHEl5 to RHEL6 comparisons

1. Introduction

CMAN was the mainstay of Red Hat's cluster suite in Red Hat Enterprise Linux 4, 5 and 6. Actually CMAN in RHEL4 was a totally different beast to that found in RHEL5 and 6 but the external interfaces looked very similar. Before I go on to say what we've done to get rid of cman, perhaps I ought to explain what it was (or still is if you're running older software).

2. So what was CMAN?

I'm glad you asked that. CMAN stood for “Cluster MANager” (boring eh?) and provided the low-level infrastructure for the cluster; node communications, votes, quorum etc. Most of the useful bits of the cluster stack were built on top of cman so few people got to see it unless things went horribly wrong.

Starting with RHEL5 we dispensed with the custom-written kernel CMAN and based the cluster manager on OpenAIS. For more details about this see my document on the subject: http://people.redhat.com/ccaulfie/docs/aiscman.pdf.

OK, a short summary then. CMAN in RHEL 5/6 is really just a quorum module that loads into openais (these days known as corosync). It provides other API services such as messaging but those are really just wrappers around other openais services.

At the cluster summit in 2008 it was decided that keeping companies' own clustering modules was a waste of effort and almost always fails to build a community. Even though CMAN (all versions of it) were released as free software, they lived in their own source repositories and were seldom modified by the original author (mostly me). So efforts were made to make the software even more open and included in the components of their parent projects. Also part of the collaboration efforts were the pooling of resource and fence agents and several others. But this is just about CMAN OK? Come on now, concentrate.

3. Where did it go?

Now we get to the nitty-gritty. The CMAN module in RHEL5 and RHEL6 does all the cman-y things that cman does such as quorum and messaging. It also services the libcman library using an IPC mechanism all of its own. It does *not* use corosync's IPC (library to daemon communications) system for reasons that are unlikely to become clear any more.

The new code dispenses with anything called cman at all. The main work of managing quorum is done by a built-in corosync module called votequorum. This code is actually largely lifted from the original cman module, with all the non-quorum bits removed, and converted to use the corosync IPC layer. There is also a lot of really lovely new code added to handle requirements that people have wanted for ages (such as last-man-standing) and can make life simpler for smaller installations. That's smaller as in number of nodes, we don't really mind how physically large your computers are, so no boasting please or I'll have a hot flush.

This new module also includes code for managing quorum devices (eg qdiskd). Though this is inactive in a default corosync system and at the time I write this (just before tea time) qdiskd has not been ported to the new API

Currently, no libcman compatibility library exists or is planned. If you want to use cluster features in a program it's necessary to code to the new APIs which I'll get onto shortly. In practice few people actually did this so we're not really expecting huge uproar and a storming of Red Hat Towers over it. If you are offended then please have a chat with us on IRC or the linux-cluster mailing list. We're really friendly. Well, I am anyway.

4. corosync quorum

4.1. quorum plugins

A brief word about corosync's quorum system might he helpful here. Since RHEL6 Corosync has a basic notion of quorum built in to the executive. Services cannot be synchronised without it and several IPC calls will be blocked if the cluster is inquorate. Inquorate is a posh name meaning “does not have quorum” or “no, you can't do any work. Why? Because I said so, that's why”

There are two modules involved in quorum. The first is what I call the quorum module itself. This is always loaded and simply provides a binary yes/no answer to the question “do we have quorum?”, you can call into this module using the libquorum API. If you were thinking ahead you should already have been doing this, because cman talks to it.

However, as I mentioned above (or on the previous page if you were ungreen enough to print this document out), on its own this will not give you much that is useful, it will just always answer “yes” to the “do we have quorum?” question. To make it do useful work you need a *quorum provider*.

A quorum provider is a corosync module that tells the corosync quorum plugin when the quorate state changes, based on some internal algorithm. There are many ways of calculating quorum so there is the possibility of all sorts of modules being written, and I hope people will write them to suit their own needs and submit them upstream. The quorum provider is loaded by putting its name in corosync.conf as follows:

```
quorum {
     provider: testquorum
}
```

This example, as I sincerely hoped you have guessed, loads the 'testquorum' module. testquorum is an extremely simple module which I recommend as a template to other quorum provider writers. It simply changes the corosync quorate status based on the quorum.quorate variable in the cmap database. Using corosync-cmapctl the quorate state can be switched on and off at will. If you have some form of external quorum system this might actually be useful to you.

Another quorum provider shipped with corosync is YKD. This is the YKD system that has been shipped with openais for some time, it has merely been re-designated a quorum plugin now. Sadly the developers did not have time to give it a shiny new logo too so it looks pretty much like it always did. To be quite frank, I know almost nothing about YKD and it scares me a little, so unless a knight in shining armour (ideally looking like David Tennant) comes along to rescue me I'll leave it there.

Now, where was I, sorry I got distracted by the David Tennant reference. Oh yes, votequorum is the quorum module based on cman. This is the one I know most about, is probably the most interest to those of you coming from a cman background and the only one that is actually supported at the moment. So I'll devote an entire subsection it. Favouritism? Pah!

4.2. votequorum

Votequorum is the name of the corosync module that replaces most of what cman used to do. Only it does it in a more 'corosync-y' way. It uses the corosync IPC system for communication with users, it provides quorum to the central core of the corosync daemon and provides familiar tracking callbacks. It is loaded by adding this stanza to corosync.conf:

```
quorum {
    provider: corosync_votequorum
}
```

I'll outline the API calls for votequorum here. If you are familiar with libcman then a lot of this will be familiar but maybe strangely different, so it's worth paying attention, just in case I surprise you (boo!)

cs_error_t votequorum_initialize(votequorum_handle_t *handle, votequorum_callbacks_t *callbacks)

cs_error_t votequorum_finalize(votequorum_handle_t handle)

These two are the familiar corosync initialise & finalise calls that corosync/openais developers should already be familiar with. libcman users will note that the callbacks are specified here rather than in later API calls. The callbacks are for quorum or expected votes change only – remember votequorum does not have a messaging API, you need to be intimate with CPG (he's nice, honest) for that.

cs_error_t votequorum_fd_get(votequorum_handle_t handle, int *fd);

cs_error_t votequorum_dispatch(votequorum_handle_t handle,cs_dispatch_flags_t dispatch_types)

Again, these should be familiar sort of call to both corosync and libcman users. You call **votequorum_dispatch** when the FD returned from **votequrum_fd_get** becomes active from select or poll. Unlike libcman, the fd doesn't change unexpectedly while running, so feel free to store it somewhere rather than calling the API before every poll(). Yes, we implemented it properly this time.

cs_error_t votequorum_getinfo (votequorum_handle_t handle, unsigned int nodeid, struct votequorum_info *info)

This call fills in the info structure with information about the quorum subsystem and the node for which you requested information. Members starting *node_* are specific to the node in question, the others are for the cluster as a whole.

cs_error_t votequorum_setexpected(votequorum_handle_t handle, unsigned int expected_votes)

This call changes the expected_votes value for the cluster. This will persist until it is changed again. NOTE: if you are using a configuration file then a forced reload of that file will overwrite a manually set votes value, as will adding a new node to the cluster.

cs_error_t votequorum_setvotes (votequorum_handle_t handle, unsigned int nodeid, unsigned int votes)

This call changes the number of votes assigned to a existing node in the cluster. This will persist until it is changed again. NOTE: if you are using a configuration file then a forced reload of that file will overwrite a manually set votes value.

cs_error_t votequorum_qdevice_register(votequorum_handle_t handle, char *name, unsigned int votes)

cs_error_t votequorum_qdevice_unregister(votequorum_handle_t handle)

cs_error_t votequorum_qdevice_poll(votequorum_handle_t handle, unsigned int state, unsigned int cast_vote)

cs_error_t votequorum_qdevice_getinfo(votequorum_handle_t handle, struct votequorum_qdisk_info *info)

cs_error_t votequorum_qdevice_update (votequorum_handle_t handle, const char *oldname, const char *newname);

cs_error_t votequorum_qdevice_master_wins (votequorum_handle_t handle, const char *name, unsigned int allow)

This group of calls manages an optional external quorum device, often a quorum disk. There can only be one quorum device per cluster and it is up to the quorum device software to ensure that its state is consistent across the cluster or to remove it from quorum.

cs_error_t votequorum_trackstart(votequorum_handle_t handle, uint64_t context, unsigned int flags)

cs_error_t votequorum_trackstop(votequorum_handle_t handle)

These are the usual tracking start/stop calls for a corosync plugin. Trackstart enables quorum or expected_votes change messages to be sent to the notification callback. To avoid race conditions it is recommended that the flags CS_TRACKSTART | CS_TRACKCHANGES are used so the program will get one immediate callback containing the current quorum state, then further callbacks as nodes join or leave the cluster.

cs_error_t votequorum_context_get(votequorum_handle_t handle, void **context)

cs_error_t votequorum_context_set(votequorum_handle_t handle, void *context)

These two calls simply allow the context variable returned to the callbacks to be changed.

4.3 configuring votequorum

The votequorum module has a small number of variables that you can set in cmap. They are all held under the main quorum{} section. I'm not going to go into detail about them here, the man page is kept up-to-date so is a better place to read up on them

votes	The number of votes this node has. The default is 1.
expected_votes	The number of votes that this cluster expects. votequorum will use the largest value of all the nodes in the cluster. The default is 1024. ie you will *not* have quorum unless you change it!
two_node	Enables a special two-node mode where a cluster with only two nodes and no quorum device can continue running with only one node. The default is 0.
allow_downscale	Allows votequorum to decrease the number of votes under certain circumstances.
wait_for_all	Votequorum will wait for all nodes to join the cluster before allowing first quorum. After that it's business as usual.
auto_tie_breaker	use the lowest nodeid as a tie breaker should the cluster be split down the middle
last_man_standing	Allows the cluster to operate with a single node remaining, under certain circumstances

5. Migrating code from libcman

This chapter is just a list of libcman calls showing you where in corosync to find equivalents. Most of them have very similar names so it shouldn't be too hard I hope. The compatibility libcman provides most of these calls so you might like to look at the source code for that library if you are unsure about how to achieve a particular result.

cman_admin_init
cman_init
cman_finish
cman_setprivdata
cman_getprivdata
cman_start_notification
cman_stop_notification
cman_start_confchg
cman_stop_confchg
cman_get_fd
cman_dispatch

These are not really cman-specific, so just use the equivalent calls for the subsystem(s) you are using.

cman_set_votes
cman_set_expected_votes
cman_set_dirty
cman_register_quorum_device
cman_unregister_quorum_device
cman_poll_quorum_device
cman_get_quorum_device
cman_get_node_count
cman_get_nodes
cman_get_disallowed_nodes
cman_get_node

Use libvotequorum. Note though that this does not provide you with node names for the cman_get_ calls.

cman_is_quorate

Use libquorum, or libvotequorum. You should only use libvotequorum if you are also using other features of that module. If all you need to know is the quorum status then you should use libquorum as that will work regardless of quorum provider.

cman_get_version
cman_set_version
cman_get_cluster
cman_set_debuglog

Use libconfdb or libccs to read/write to the cmap database

cman_kill_node
cman_shutdown
cman_replyto_shutdown
cman_get_node_addrs

Use libcfg

cman_leave_cluster

If you just need to leave the cluster then libcfg will work. Votequorum has no equivalent of the LEAVE flag, you should configure your cluster for this behaviour in corosync.conf if you need it.

cman_send_data
cman_start_recv_data
cman_end_recv_data

Use libcpg

cman_get_node_extra
cman_get_subsys_count
cman_is_active
cman_is_listening
cman_barrier_register
cman_barrier_change
cman_barrier_wait
cman_barrier_delete
cman_get_extra_info
cman_get_fenceinfo
cman_node_fenced

These calls are not implemented at all, sorry. You're on your own.