Management Agent API (MAAPI)

Greg Olin
ConfD Solution Architect
2019-10-22

Management Agent API (MAAPI)

- Public API that provides application support for the internal ConfD API used by northbound interfaces
- Intended for developing northbound interfaces not supported by ConfD and other programmatic uses
  - We have a simple TL1 example
  - Plenty of utilities that ConfD ships use MAAPI as well
- As we will see, it is a bit of Swiss army knife

Boilerplate code

- Need to call `confd_init()` to initialize libconfd
  - Unlike DP API, daemon context is not needed, so don't need to call `confd_init_daemon()`
- Create a streaming socket
- Call `maapi_connect()` to connect to IPC port

Authentication

- If you are providing a northbound interface or an application, and you need to authenticate, there is `maapi_authenticate()`.
- `maapi_authenticate()` takes a username and password.
- Uses normal ConfD authentication mechanisms as configured (local, PAM, external)
- If successful, ConfD returns an array of groups
- Authentication not enforced by ConfD, up to application to do this
- There is also `maapi_authenticate2()` which allows additional info to be passed in that will in turn be provided to external authentication.

Session management

- After `maapi_authenticate()`, if used, then we must start a user session
- There is limit to the number of overall sessions allowed at one time.
- `maapi_start_user_session()` takes a username, groups, context, and src IP address
  - context is any string, but it is used in NACM rules
  - context string of ‘system’ has special meaning
    - Can be useful for applications not related to any northbound interface
    - Bypasses authorization checking
    - Bypasses maxSessions
    - Session not logged in audit log
    - Session doesn’t appear in ‘show users’

Session management (cont'd)

- `maapi_start_user_session2()` in addition takes the src port number
- `maapi_start_user_session3()` takes additional information such as vendor, product, version and client id
- `maapi_end_user_session()` ends the user session. Closing the maapi socket implicitly ends the session as well
- `maapi_kill_user_session()` can be used to end another user session.
- `maapi_get_my_user_session_id()` returns the session id.
  - Implies there is at most one session per maapi socket.
- `maapi_get_user_session()` returns a confd_user_info struct
  - Can be useful in CDB subscribers to see who triggered configuration change

Session management (cont'd)

- `maapi_set_user_session()` can be used to use an existing user session rather than start a new one.
- Several more rarely used API calls, but these are all documented in the manpage

Transactions

- MAAPI main purpose is to support the ability to write configuration data into ConfD
  - typically this is CDB, put MAAPI is agnostic whether it goes to CDB or DP
  - CDB API, with the exception of writing operational data stored in CDB, is used to only read data from CDB.
- Configuration changes always occur within a transaction in ConfD
- So we need a way to start, validate, and commit transactions.

Transactions

- MAAPI main purpose is to support the ability to write configuration data into ConfD
  - typically this is CDB, put MAAPI is agnostic whether it goes to CDB or DP
  - CDB API, with the exception of writing operational data stored in CDB, is used to only read data from CDB.
- Configuration changes always occur within a transaction in ConfD
- So we need a way to start, validate, and commit transactions.

Transactions (cont'd)

- `maapi_start_trans()` is used to start a transaction
  - Takes the datastore for the transaction
    - CONFD_CANDIDATE
    - CONFD_STARTUP
    - CONFD_RUNNING
    - CONFD_OPERATIONAL
    - Others once NMDA is implemented
  - Also the type of transaction, read-only or read-write
  - Returns a transaction handle, often referred to as th, used in many MAAPI calls

Transactions (cont'd)

- `maapi_start_trans2()` is used to start a transaction
  - Takes the datastore for the transaction
    - CONFD_CANDIDATE
    - CONFD_STARTUP
    - CONFD_RUNNING
    - CONFD_OPERATIONAL
    - Others once NMDA is implemented
  - Also the type of transaction, read-only or read-write
  - This also takes a user session id.
  - Often used in actions

Transactions (cont'd)

- `maapi_start_trans_flags()` can be used to set flags in the transaction
- `maapi_start_trans_flags2()` is like the above call but allows additional information to be supplied
- `maapi_validate_trans()` can be used to validate the existing transaction
- `maapi_prepare_trans()` can be used to initiate the first phase of a two phase commit
- `maapi_commit_trans()` or `maapi_abort_trans()` must now be called
- `maapi_apply_trans()` can be called in place of the 3 previous calls if fine-grained control over 2-phase commit is not needed
- `maapi_finish_trans()` is the last call in the sequence that needs to be called

Transactions – reading and modifying data

- There are many `maapi_get_*` calls, much like the CDB API
- Also calls to create, delete, and set items within the transaction
- See manpage for the many details ☺

Datastore locking

- Since MAAPI needs to give you access to everything that other NB interfaces have, this includes locking
- `maapi_lock()` will lock the datastore you requested if not already locked
- `maapi_unlock()` will unlock the datastore
  - Datastore is implicitly unlocked when the session ends.
- `maapi_is_lock_set()` will tell you if the lock is set or not
- `maapi_lock_partial()` will perform a partial lock, on a specified list of leafs or subtrees
  - This is supported by our NETCONF server, so accessible through MAAPI
- `maapi_unlock_partial()` will unlock

Datastore manipulation

- ConfD supports <startup>, <running> and <candidate> datastores
  - Will add support for <intended> and <operational> from NMDA in ConfD 7.3
  - Which datastores you have is configurable in confd.conf
- Main datastore you might have to work with is <candidate>, if you enable this.
- Modifying candidate involves starting a transaction towards <candidate>
- Once the changes have been made, the transaction can be applied/committed
- But the changes are still in <candidate>, not in <running>
- `maapi_candidate_commit()` will attempt to move changes from <candidate> to <running>
  - However, changes from other users may be in candidate, so the groups associated with this transaction need to have permission to make the changes made by other users.
  - Often the reason that <candidate> might be locked while making changes

Datastore manipulation (cont'd)

- Now instead of doing a straightforward commit of <candidate> to <running>, you can do a 'provisional' commit, called a confirmed commit
  - If the commit is not confirmed within a period of time, then it is reverted.
- `maapi_candidate_confirmed_commit()` can be called to start this confirmed commit process
  - API call takes the timeout in seconds
- `maapi_candidate_commit()` is then called to make the confirmed commit permanent.
- `maapi_candidate_abort_commit()` is used to revert a confirmed commit before the timeout is reached.
- Other variants of confirmed commit, see manpage for details.

Datastore manipulation (cont'd)

- `maapi_candidate_reset()` removes earlier changes made in <candidate>
- `maapi_is_candidate_modified()` tells you whether changes have already been made to <candidate>
- `maapi_copy_running_to_startup()` copies <running> to <startup>
- `Maapi_is_running_modified()` tells you if <running> and <startup> are different

Using an existing transaction

- In some situations, such as a transform, set hook or transaction hook, or validation code, you may want to attach to the existing transaction.
- This allows the application code to see the changes that are part of the transaction as well as what is in CDB.
- `maapi_attach()` allows you to attach to an existing transaction. Transaction context, tctx, is passed to your application in callback function.

Using an existing transaction

- There is a special situation during model upgrade.
- For example, you may have added a new mandatory leaf which does not have a default value.
- Normally this upgrade will fail because ConfD can't supply a value for this leaf node
- You can write a special application which can attach to the upgrade transaction and manipulate it by adding the mandatory value
- Steps for this:
  - Start ConfD at phase 0. This creates the upgrade transaction
  - Start your application, which calls `maapi_attach_init()` Note this doesn't take tctx, and ConfD returns th to you to use in other MAAPI calls.
  - Once your application finishes, have ConfD go to phase 1, which commits the upgrade transaction, then to phase 2 to turn NB interfaces on

Saving and loading configuration

- MAAPI can also be used to save configuration and load configuration/operational data
  - ConfD provides a utility called confd_load which can perform both of these operations
  - Many flags/options that can be specified. Reflects that flags available on the C API
  - Source is provided in $CONFD_DIR/src/confd/tools/confd_load.c
- Saving configuration is a bit convoluted (IMO)
  - `fopen()` the file in write mode
  - `maapi_save_config()` returns a id
  - Create a stream socket.
  - `maapi_stream_connect()` to connect the id with the socket
  - `read()` from the socket and `write()` to the file until ConfD closes the socket
  - `maapi_save_config_result()` to find out whether this succeeded

Saving and loading configuration (cont'd)

- Loading configuration from a file is much simpler.
- `maapi_load_config()` takes the filename as a parameter.
- A caveat
  - By default, when you load a configuration you are replacing the existing the existing configuration
    - Specify `MAAPI_LOAD_MERGE` if you want to merge the configuration with the existing configuration
- You can also load a configuration using a stream, if you want
  - `maapi_load_config_stream()` returns a id
  - Create a stream socket.
  - `maapi_stream_connect()` to connect the id with the socket
  - `write()` to the socket from whatever source you get the configuration
  - `maapi_load_config_stream_result()` to find out whether this succeeded

Starting and stopping ConfD

- Your platform may have a process manager that will want to be able to stop and start applications, or you may have init scripts
- `maapi_stop()` will stop ConfD
- `maapi_start_phase()` allows you to move ConfD to phase 1 or phase 2
- `maapi_wait_start()` Allows the application to wait until ConfD reaches a certain phase

Query data

- MAAPI also provides you a way to query data using an Xpath expression
  - Other NB interfaces have this, so it is provided in MAAPI
- `maapi_query_start()`
  - XPath expression
  - Chunk size (how many results to return at a time)
  - Initial offset
  - Items you want returned, for example, the leafs in a list
- `maapi_query_result()` returns a chunk of results.
  - When the number of results reaches 0, then query is finished.
- `maapi_query_free_result()` MAAPI allocates the results on the heap, so this call must be done to free the memory

Other resources

- As we already mentioned, there is the `confd_load` command and the source for that command
  - Feel free to take and modify
- `confd_cmd` is a wrapper for many common CDB and MAAPI API calls
  - Source is also in $CONFD_DIR/src/confd/tools
  - Allows you to create scripts and invoke the APIs
- `maapi` is another wrapper for use in custom CLI commands or actions
  - Makes use of the current transaction
  - Source is in $CONFD_DIR/src/confd/cli

Examples which make heavy use of MAAPI

- $CONFD_DIR/examples.confd/intro/12-c_maapi
- intro/10-c_transform
- intro/11-c_hooks
- validate/c and validate/c_dependency
- cli/c_cli, cli/climods, cli/command, and cli/completions
- misc/shell_tools – illustrates confd_load and confd_cmd utilities

Conclusion

- There is a lot to MAAPI
  - And a lot that we haven't covered
- Skim the MAAPI manpage periodically, just to see if there is some obscure API that might help with something you are working on

Thank you for listening

www.tail-f.com