netconf_client Documentation

ADTRAN

Aug 03, 2021

Contents

CHAPTER 1

Quick Start

In order to connect to a NETCONF server we can use one of the `connect` methods from the *`netconf_client.connect`* module. In our example we will connect to a NETCONF server running over SSH, so we will use the *`connect_ssh`* function.:

```
from netconf_client.connect import connect_ssh

with connect_ssh(host='192.0.2.1',
                 port=830,
                 username='admin',
                 password='password') as sesssion:
    # TODO: Do things with the session object
    pass
```

The object returned from any of the `connect` functions is a *`Session`* object. It acts as a context manager, and as such it should generally be used alongside a `with` statement. It is importat to either use a `with` statement with the object, or to manually call *`Session.close`* in order to free the sockets associated with the connection.

The *`Session`* object can be used to send and receive raw messages. However, a higher-level API is desireable in most circumstances. For this we can use the *`Manager`* class.

The *`Manager`* class is from the *`netconf_client.ncclient`* module. The module overall attempts to mimick the most common uses of the ncclient API (another, Open Source, Python NETCONF client). Most of the common NETCONF operations such as performing an `<edit-config>` or a `<get>` are implemented as functions of this class.

In this example we will perform an `<edit-config>` for a single node, and then run a `<get-config>` to see the change.:

```
from netconf_client.connect import connect_ssh
from netconf_client.ncclient import Manager

with connect_ssh(host='192.0.2.1',
                 port=830,
                 username='admin',
```

(continues on next page)

(continued from previous page)

```
                password='password') as session:
    mgr = Manager(session, timeout=120)
    mgr.edit_config(target='running', '''
        <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
          <service-alpha xmlns="http://example.com">
            <simple-string>Foo</simple-string>
          </service-alpha>
        </config>''')
    print(mgr.get_config(source='running').data_xml)
```

An instance of the *Manager* class should be a drop-in replacement for a manager object from `ncclient`.

CHAPTER 2

Migrating

To migrate from the `ncclient` API to the `netconf_client` API you can generally follow these two steps.

First change your imports. For example, convert from:

```
from ncclient.operations import RPCError
from ncclient.xml_ import to_ele
```

To this:

```
from netconf_client.ncclient import RPCError, to_ele
```

Then you will need to migrate your connection code. For example, if your old connection method looked like this:

```
def mgr():
    from ncclient import manager, operations
    m = manager.connect_ssh(host='localhost', port=830,
                            username='root', password='password',
                            hostkey_verify=False,
                            timeout=120,
                            )
    m.raise_mode = operations.RaiseMode.ALL
    return m
```

Then your new connection code should look like this:

```
def mgr():
    from netconf_client.connect import connect_ssh
    from netconf_client.ncclient import Manager

    s = connect_ssh(host='localhost', port=830,
                    username='root',  password='password')
    return Manager(s, timeout=120)
```

As long as the existing code isn't doing anything too crazy, these should be the only changes needed.

CHAPTER 3

netconf_client API

3.1 netconf_client.connect

class netconf_client.connect.**CallhomeManager**(*bind_to=''*, *port=4334*, *backlog=1*)

Listener object for accepting callhome connections (**RFC 8071**)

Options on the listener socket (e.g. timeout) can be set on the `server_socket` member.

This object is a context manager, and should generally be used within *with* statements to ensure the listening socket is properly closed. Note that sessions started from one of the *accept* functions may outlive the scope of this object and will not be closed automatically.

Example of accepting a call-home connection with TLS:

```
with CallhomeManager(port=4335) as call_home_mgr:
    session = call_home_mgr.accept_one_tls(keyfile=client_key,
                                           certfile=client_cert,
                                           ca_certs=ca_cert)
with Manager(session) as mgr:
    mgr.get_config(source='running')
```

accept_one(*timeout=120*)

Accept a single TCP client and returns it

Parameters **timeout** (*int*) – Seconds to wait for an incoming connection

accept_one_ssh(**args*, ***kwds*)

Accept a single TCP client and start an SSH session on it

This function takes the same arguments as *connect_ssh()*

accept_one_tls(**args*, ***kwds*)

Accept a single TCP client and start a TLS session on it

This function takes the same arguments as *connect_tls()*

netconf_client.connect.**connect_ssh**(*host=None*, *port=830*, *username='netconf'*, *password=None*, *key_filename=None*, *sock=None*, *timeout=120*)

Connect to a NETCONF server over SSH.

Parameters

- **host** (*str*) – Hostname or IP address; unused if an already-open socket is provided
- **port** (*int*) – TCP port to initiate the connection; unused if an already-open socket is provided
- **username** (*str*) – Username to login with; always required
- **password** (*str*) – Password to login with; not required if a private key is provided instead
- **key_filename** (*str*) – Path to an SSH private key; not required if a password is provided instead
- **sock** – An already-open TCP socket; SSH will be setup on top of it
- **timeout** (*int*) – Seconds to wait when connecting the socket

Return type *netconf_client.session.Session*

netconf_client.connect.**connect_tls**(*host=None*, *port=6513*, *keyfile=None*, *certfile=None*, *ca_certs=None*, *sock=None*, *timeout=120*)

Connect to a NETCONF server over TLS.

Parameters

- **host** (*str*) – Hostname or IP address; unused if an already-open socket is provided
- **port** (*int*) – TCP port to initiate the connection; unused if an already-open socket is provided
- **keyfile** – Path to the key file used to identify the client
- **certfile** – Path to the certificate used to identify the client
- **ca_certs** – Path to a file containing the certificate autority chains for verifying the server identity
- **sock** – An already-open TCP socket; TLS will be setup on top of it
- **timeout** (*int*) – Seconds to wait when connecting the socket

Return type *netconf_client.session.Session*

3.2 netconf_client.ncclient

class netconf_client.ncclient.**DataReply**(*raw*, *ele*)

A response containing a <data> element

Variables

- **data_xml** (*str*) – The data element in string form (note that this value was handled by lxml)
- **data_ele** – The lxml parsed representation of the data
- **raw_reply** (*bytes*) – The raw reply from the server

class netconf_client.ncclient.**Manager**(*session*, *timeout=120*, *log_id=None*)

A helper class for performing common NETCONF operations with pretty logging.

This class attempts to be API compatible with the manager object from the original ncclient.

This class is also a context manager and can be used with *with* statements to automatically close the underlying session.

NETCONF requests and responses are logged using the `netconf_client.manager` scope. The log level is logger.DEBUG.

Each log entry shows a log ID (the peers' IP addresses as default). Additionally, the round-trip delay between request and its response is computed and displayed.

The Python logger receives a dictionary via *extra* parameter, whose key is `ncclient.Manager.funcname` and which contains the name of the API function being logged. This information can be used for user-specific filtering.

Variables

- **timeout** (*float*) – Duration in seconds to wait for a reply
- ***session*** – The underlying *netconf_client.session.Session* connected to the server
- **log_id** (*str*) – application-specific log ID (None as default)

close_session()

Send a <close-session> request

commit(*confirmed=False*, *confirm_timeout=None*, *persist=None*, *persist_id=None*)

Send a <commit> request

Parameters

- **confirmed** (*bool*) – Set to `True` if this is a confirmed-commit
- **confirm_timeout** (*int*) – When *confirmed* is `True`, the number of seconds until the commit will be automatically rolled back if no confirmation or extension is received
- **persist** (*str*) – When *confirmed* is `True`, sets the persist-id token for the commit and makes the commit a persistent commit
- **persist_id** (*str*) – When *confirmed* is `True` and a previous <persist> was given for the current commit, this field must match the corresponding persist-id for the commit

copy_config(*target*, *source*, *with_defaults=None*)

Send a <copy-config> request

Parameters

- **source** (*str*) – The source datastore or the <config> element containing the complete configuration to copy.
- **target** (*str*) – The destination datastore
- **with_defaults** (*str*) – Specify the mode of default reporting. See **RFC 6243**. Can be `None` (i.e., omit the with-defaults tag in the request), 'report-all', 'report-all-tagged', 'trim', or 'explicit'.

create_subscription(*stream=None*, *filter=None*, *start_time=None*, *stop_time=None*)

Send a <create-subscription> request

Received <notification> elements can be retrieved with *take_notification()*

Parameters

- **stream** (*str*) – The event stream to subscribe to
- **filter** (*str*) – The filter for notifications to select
- **start_time** (*str*) – When replaying notifications, the earliest notifications to replay
- **stop_time** (*str*) – When replaying notifications, the latest notifications to replay

delete_config(*target*)
Send a <delete-config> request

Parameters **target** (*str*) – The datastore to delete

discard_changes()
Send a <discard-changes> request

dispatch(*rpc*)
Send an <rpc> request

Parameters **rpc** (*str*) – The RPC to send; it should not include an <rpc> tag (one will be generated for you)

Return type *RPCReply*

edit_config(*config*, *target='running'*, *default_operation=None*, *test_option=None*, *error_option=None*, *format='xml'*)
Send an <edit-config> request

Parameters

- **config** (*str*) – The <config> node to use in the request
- **target** (*str*) – The datastore to edit
- **default_operation** (*str*) – The default-operation to perform; can be None, 'merge', 'replace', or 'none'.
- **test_option** (*str*) – The test-option to use; can be None, 'test-then-set', 'set', or 'test-only'
- **error_option** (*str*) – The error-option to use; can be None, 'stop-on-error', 'continue-on-error', or 'rollback-on-error'

get(*filter=None*, *with_defaults=None*)
Send a <get> request

Parameters

- **filter** (*str*) – The <filter> node to use in the request
- **with_defaults** (*str*) – Specify the mode of default reporting. See **RFC 6243**. Can be None (i.e., omit the with-defaults tag in the request), 'report-all', 'report-all-tagged', 'trim', or 'explicit'.

Return type *DataReply*

get_config(*source='running'*, *filter=None*, *with_defaults=None*)
Send a <get-config> request

Parameters

- **source** (*str*) – The datastore to retrieve the configuration from
- **filter** (*str*) – The <filter> node to use in the request

- **with_defaults** (*str*) – Specify the mode of default reporting. See RFC 6243. Can be None (i.e., omit the with-defaults tag in the request), 'report-all', 'report-all-tagged', 'trim', or 'explicit'.

Return type *DataReply*

kill_session(*session_id*)
Send a <kill-session> request

Parameters **session_id** (*int*) – The session to be killed

lock(*target*)
Send a <lock> request

Parameters **target** (*str*) – The datastore to be locked

static logger()
Returns the internally used logger instance (same for all sessions)

session_id
The session ID given in the <hello> from the server

take_notification(*block=True*, *timeout=None*)
Retrieve a notification from the incoming notification queue.

Parameters

- **block** (*bool*) – If True, the call will block the current thread until a notification is received or until *timeout* is exceeded
- **timeout** (*float*) – The number of seconds to wait when *block* is True; when None, the call can block indefinitely

Return type *Notification*

unlock(*target*)
Send an <unlock> request

Parameters **target** (*str*) – The datastore to be unlocked

validate(*source*)
Send a <validate> request

Parameters **source** (*str*) – The datastore to validate

class netconf_client.ncclient.**Notification**(*raw*, *ele*)
A <notification> received from the server

Variables

- **notification_xml** (*bytes*) – The raw notification as received from the server
- **notification_ele** – The lxml parsed representation of the notification

class netconf_client.ncclient.**RPCReply**(*xml*)
A non-error response to an <rpc>

Variables **xml** (*str*) – The raw reply from the server

netconf_client.ncclient.**to_ele**(*maybe_ele*)
Convert the given string to an lxml element

Parameters **maybe_ele** – If this is a string, it will be parsed by lxml. If it is already an lxml element the parameter is returned unchanged

3.3 netconf_client.session

class netconf_client.session.**Session**(*sock*)
A session with a NETCONF server

This class is a context manager, and should always be either used with a `with` statement or the *close()* method should be called manually when the object is no longer required.

Variables

- **server_capabilities** – The list of capabilities parsed from the server's <hello>
- **client_capabilities** – The list of capabilities parsed from the client's <hello>

close()
Closes any associated sockets and frees any other associated resources

send_msg(*msg*)
Sends a raw byte string to the server

Parameters **msg** (*bytes*) – The byte string to send

send_rpc(*rpc*)
Sends a raw RPC to the server

Parameters **rpc** (*bytes*) – The RPC to send

Return type concurrent.futures.Future with a result type of tuple(bytes, lxml.Element)

3.4 netconf_client.error

exception netconf_client.error.**NetconfClientException**
Base class for all netconf_client exceptions

exception netconf_client.error.**NetconfProtocolError**
This exception is raised on any NETCONF protocol error

exception netconf_client.error.**RpcError**(*raw*, *ele*)
This exception is raised on a future from an <rpc> call that returns a corresponding <rpc-error>

Variables

- **reply_raw** – The raw text that was returned by the server
- **reply_ele** – The lxml parsed representation of the reply
- **message** – If present, the contents of the <error-message> tag
- **tag** – If present, the contents of the <error-tag> tag
- **info** – If present, the contents of the <error-info> tag

exception netconf_client.error.**SessionClosedException**
This exception is raised on any futures when the NETCONF connection is closed

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

T

U

V