THE NETCONF TRANSACTIONS CAPABILITY IN CONFD

tail-f Tail-f is now part of Cisco. cisco

Table of Contents

INTRODUCTION

Transactions have proven to be an important part of configuration management and programmability. In a transaction, a set of changes either all succeed or they all fail and the system remains at its original configuration. In configuration management, support for transactions provides many benefits chief of which are reducing the complexity of network management and increasing the robustness of the network.

The NETCONF protocol as defined in **RFC 6241 (NETCONF 1.1)** and **RFC 4741 (NETCONF 1.0)** provides many standard features which provide transactional support. Some examples are that an edit-config RPC is a transaction in ConfD and that multiple edit-confg RPCs can be performed against the candidate datastore and then the commit or confirmed commit operations can be used to transactionally commit the built up set of changes from the candidate datastore to the running datastore.

However, sometimes the standard transactional features of NETCONF are not enough and there is a need for finer grained control of transactions. ConfD takes advantage of the extensible nature of NETCONF to define the NETCONF Transactions extension capability in order provide this additional level of transaction support.

This application notes describes what ConfD's NETCONF Transactions extension capability is and how to use it.

1. Why a new capability?

Transactions in the NETCONF protocol are insured by the correct implementation of the candidate datastore and a set of RPC calls in order to manage the transaction.

In NETCONF, only one candidate datastore is available in a system. This results in the need for explicit locking/unlocking of the candidate in order to make predictable changes. Since the candidate datastore is global, any user can be writing to the candidate if unlocked and a commit from one user will result in committing other users' changes.

Some systems, don't even support a candidate datastore, and rely on a writable running for their configuration changes, thus, losing the advantage of full transactions. In this case as well, the locking and unlocking of the datastore has to be explicit.

For these systems, ConfD's NETCONF Transactions capability can be used to provide a transactional way to obtain predictable configuration of a system.

2. How does it work?

The Transactions extension capability in ConfD introduces four new NETCONF RPC calls which are used to control a two-phase commit transaction towards a specified target datastore.

A two-phase commit transaction consists of:

Prepare Phase: Data is populated into the transaction
Commit Phase: Data is committed and configuration changes take effect

The four new RPC calls introduced with this capability are:

2.1 start-transaction

Starts a transaction towards a specified target configuration datastore. There is only one possible transaction per session at any time. The configuration data is only committed to the target datastore after a commit-transaction is issued. Data stays private to that session in temporary storage as opposed to using the standard candidate capability where the candidate datastore is persistent and shared.

This RPC is similar to the **maapi_start_transaction()** MAAPI function call.

2.2 prepare-transaction

Prepares the transaction state for commit. In this phase, the NETCONF server can reject the configuration based on pre-defined logic such as semantic constraints or lack of resources. It is important to not fail the commit-transaction if this RPC call succeeds. Doing so might lead to a bad network wide configuration state, if multiple devices are being configured for the same functionality or service. Thus, it is important to only return success for this call if the resources needed for a commit are guaranteed.

This RPC is similar to the **maapi_prepare_trans()** MAAPI function call.

The applications that register a "prepare" callback in a two-phase commit transaction, will be invoked at this time to help "prepare" for the commit. An error that occurs in this phase will result in the transaction being aborted.

2.3 commit-transaction

Applies the changes made in the transaction to the target datastore which was specified in start-transaction. After this RPC call is issued, the transaction will terminate.

This RPC is similar to the **maapi_commit_trans()** MAAPI function call.

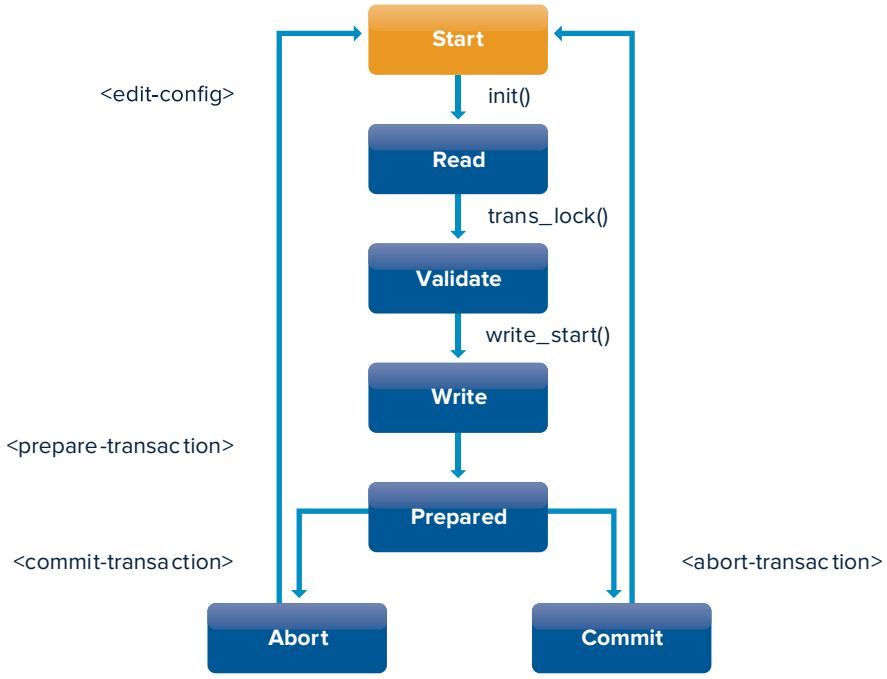
After this call is issued, data will be committed in the target datastore.

2.4 abort-transaction

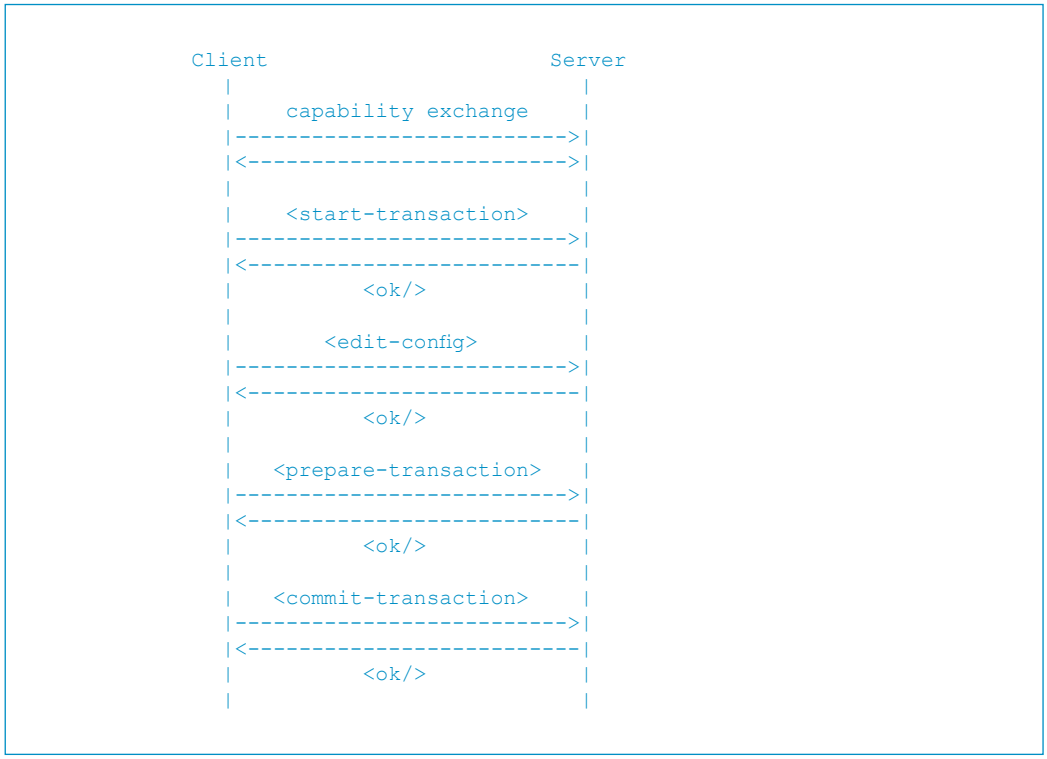
This RPC allows a user to discard their changes in a transaction. This call can be used anytime during the life time of the transaction.

Similar to **maapi_abort_trans()**, this RPC can be used anytime to abort a transaction.

Below is a minimalistic state machine showing the different states a transaction goes through when issuing the various RPC calls described above:



2. Sequence of operations



Notice the use of the standard NETCONF RPC edit-config. The target of the edit-config has to match the target used when start-transaction was issued. Note that multiple edit-config RPCs can be issued within a transaction.

If the transaction was started with the target set to running, the edit-config will not write directly to the running datastore. The edit-config will instead be writing into the transaction and the data will only be written permanently to the target datastore once the commit-transaction RPC succeeds.

3. How to enable the transactions capability

The NETCONF Transactions extension capability is identified by the hello reply message containing the string:

http://tail-f.com/ns/netconf/transactions/1.0

This string corresponds to the namespace used in the module:
tailf-netconf-transactions.yang

By default, this capability is not enabled. To enable it, one must do so in the confd.conf file.

Example: NETCONF server configuration in **confd.conf:**

```
<netconf>
    <enabled>true</enabled>
    <transport>
      <ssh>
        <enabled>true</enabled>
        <ip>0.0.0.0</ip>
        <port>2022</port>
      </ssh>

      <!-- NETCONF over TCP is not standardized, but it can be useful
           during development in order to use e.g. netcat for scripting.
      -->
      <tcp>
        <enabled>true</enabled>
        <ip>127.0.0.1</ip>
        <port>2023</port>
      </tcp>
    </transport>

    <capabilities>

      <!-- enable only if /confdConfig/datastores/candidate is enabled -->
      <candidate>
        <enabled>false</enabled>
      </candidate>

      <transactions>
        <enabled>true</enabled>
      </transactions>

    </capabilities>
  </netconf>
```

Once a NETCONF client is aware of the server's support of this capability, it can start using the four new RPCs discussed above to start and manage a transaction.

4. Example of use

For this section, the example: **$CONFD_DIR/examples.confd/intro/1-2-3-start-query-model** has been modified by enabling the NETCONF Transactions capability in confd.conf.

Note that this example doesn't support a candidate datastore and <writable-running> is also set to true. This means the running datastore would need to be locked explicitly before one can make a change. This is not ideal if there are multiple users trying to modify the configuration data.

For this case, the Transactions capability solves the problem by providing a private "store", the transaction, per user session.

Below is what a configuration operation looks like when using the NETCONF Transactions capability in this example.

Hello message:

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </capabilities>
</hello>
```

Hello reply:

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</ca-
pability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</
capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-
mode=explicit&amp;also-supported=report-all-tagged</capability>
    <capability>urn:ietf:params:netconf:capability:yang-
library:1.0?revision=2016-06-21&amp;module-set-id=5d61613c8653987798f271b
2a9876561</capability>
    <capability>http://tail-f.com/ns/netconf/transactions/1.0</capability>
    <capability>http://tail-f.com/ns/netconf/actions/1.0</capability>
    <capability>http://tail-f.com/ns/netconf/extensions</capability>
    <capability>http://tail-f.com/ns/aaa/1.1?module=tailf-
aaa&amp;revision=2015-06-16</capability>
    <capability>http://tail-f.com/ns/example/dhcpd?module=dhcpd</capability>
    <capability>http://tail-f.com/ns/kicker?module=tailf-
kicker&amp;revision=2017-03-16</capability>
    <capability>http://tail-f.com/ns/webui?module=tailf-
webui&amp;revision=2013-03-07</capability>
```

continued on next page

continued from previous page

```
    <capability>http://tail-f.com/yang/acm?module=tailf-
acm&amp;revision=2013-03-07</capability>
    <capability>http://tail-f.com/yang/common-monitoring?module=tailf-common-
monitoring&amp;revision=2013-06-14</capability>
    <capability>http://tail-f.com/yang/confd-monitoring?module=tailf-confd-mo
nitoring&amp;revision=2013-06-14</capability>
    <capability>http://tail-f.com/yang/netconf-monitoring?module=tailf-net-
conf-monitoring&amp;revision=2016-11-24</capability>
    <capability>urn:ietf:params:xml:ns:yang:iana-crypt-hash?module=iana-
crypt-hash&amp;revision=2014-08-06&amp;features=crypt-hash-sha-512,crypt-
hash-sha-256,crypt-hash-md5</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-inet-types?module=ietf-inet-
types&amp;revision=2013-07-15</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-acm?module=ietf-net-
conf-acm&amp;revision=2012-02-22</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-monito-
ring?module=ietf-netconf-monitoring&amp;revision=2010-10-04</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-notifi-
cations?module=ietf-netconf-notifications&amp;revision=2012-02-06</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-restconf-moni-
toring?module=ietf-restconf-monitoring&amp;revision=2016-08-15</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-yang-library?module=ietf-
yang-library&amp;revision=2016-06-21</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-yang-types?module=ietf-yang-
types&amp;revision=2013-07-15</capability>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&a
mp;revision=2011-06-01</capability>
    <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-with-
defaults?module=ietf-netconf-with-defaults&amp;revision=2011-06-01</capabil-
ity>
  </capabilities>
  <session-id>29</session-id>
</hello>
```

Note: The NETCONF Transactions extension capability is supported on this device. The Hello message reply contains the capability advertisement:

<capability>http://tail-f.com/ns/netconf/transactions/1.0</capability>

The NETCONF client used for this demo is “netconf-console” in interactive mode.

Note: You can also use netconf-console in a non-interactive mode by providing a file containing the RPCs.

Step 1: Start netconf-console:

To start the “netconf-console” program in interactive mode from a shell prompt:

netconf-console –i

This program takes care of encapsulating the payload in “**<rpc>**” tags. So, there is no need to specify the **<rpc>** tags.

Step 2: **Start a transaction using the Transactions capability:**

```
<start-transaction xmlns="http://tail-f.com/ns/netconf/transactions/1.0">
<target>
<running/>
</target>
</start-transaction>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>
```

Step 3: **Edit configuration:**

```
<edit-config>
    <target>
      <running/>
    </target>
    <config>
      <dhcp xmlns="http://tail-f.com/ns/example/dhcpd"
            xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <defaultLeaseTime nc:operation="merge">
          PT1H
        </defaultLeaseTime>
      </dhcp>
    </config>
  </edit-config>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>
```

Step 4: Prepare the transaction for commit:

```
<prepare-transaction xmlns="http://tail-f.com/ns/netconf/transactions/1.0"/>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>
```

Step 5: Commit the change:

```
<commit-transaction xmlns="http://tail-f.com/ns/netconf/transactions/1.0"/>
TRACE("get_elem value found val=%i", val);
    confd_data_reply_value(tctx, &v);

    TRACE_EXIT("ret %i", ret);
    return ret;
}

static int get_next(struct confd_trans_ctx *tctx, confd_hkeypath_t *keypath,
        long next)
{
    TRACE_ENTER("");
    int ret = CONFD_ERR;
    FATAL("As data model has no list, the 'get_next' should not be called!");
    TRACE_EXIT("ret %i", ret);
    return ret;
}
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <ok/>
</rpc-reply>
```

After this reply from the server, we can go ahead and verify that the change was committed successfully either by using a get RPC in NETCONF or other NB interfaces like CLI.

Notice that the server was responding with an **<ok/>** message.

In a failure case, the server would return an <rpc-error> reply. Example of a syntax type of error:

```
<edit-config>
    <target>
      <running/>
    </target>
    <config>
      <dhcp xmlns="http://tail-f.com/ns/example/dhcpd"
            xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <defaultLeaseTime nc:operation="merge">
          iuroiur
        </defaultLeaseTime>
      </dhcp>
    </config>
  </edit-config>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path xmlns:dhcpd="http://tail-f.com/ns/example/dhcpd">
    /rpc/edit-config/config/dhcpd:dhcp/dhcpd:defaultLeaseTime
  </error-path>
    <error-message xml:lang="en">"oiuroiur" is not a valid value.</error-mes-
sage>
    <error-info>
      <bad-element>defaultLeaseTime</bad-element>
    </error-info>ß
  </rpc-error>
</rpc-reply>
```

Note that an attempt to create a second transaction in the same session while the first transaction is ongoing will result in the following error from the Server:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>bad-state</error-app-tag>
    <error-info>
      <bad-element>start-transaction</bad-element>
    </error-info>
  </rpc-error>
</rpc-reply>
```

5. Summary

This application note has shown what ConfD's NETCONF Transactions extension capability is and how it can be used.

More information regarding ConfD's NETCONF Transactions extension capability can be found in the "NETCONF Server" chapter of the ConfD User Guide. The formal YANG definitions of the new RPCs can be found in the YANG module:

$CONFD_DIR/src/confd/yang/tailf-netconf-transactions.yang

For more information regarding ConfD, visit: http://www.tail-f.com

www.tail-f.com
info@tail-f.com

Corporate Headquarters

Sveavagen 25
111 34 Stockholm
Sweden
+46 8 21 37 40

©2017 Tail-f systems All rights reserved.