NETCONF CALL HOME AND CONFD

tail-f a Cisco company

Table of Contents

1. Introduction 3

2. Basics 3

3. NETCONF Server (TCP Client) in Call Home 4

4. NETCONF Client (TCP server) in Call Home 4

5. Summary 10

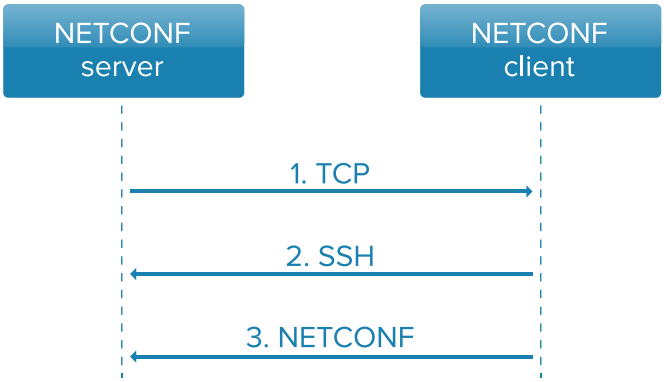
For More Information 10

1. Introduction

As described in RFC 8071, NETCONF Call Home functionality enables a NETCONF server in a network element to initiate a secure connection to a NETCONF client in a controller or orchestrator. As as discussed in RFC 8071, NETCONF Call Home is generally useful for both the initial deployment and ongoing management of networking elements. Some of the use cases include:

- Allowing access which is normally prevented by NAT/firewall boundaries between peers
- Proactively registering a networking element with a management system
- Enabling an ad-hoc management interface without permanently opened networking ports
- etc.

Normally, a NETCONF client (management station) initiates a connection to a NETCONF server (network element). The NETCONF client goes through the steps of initiating a TCP connection, starting a SSH session over the TCP socket, and then a NETCONF session over the SSH connection. NETCONF Call Home reverses the client and server roles at the TCP layer; that is, which peer is the TCP client and which is the TCP server. All the other client andserver roles in their respective protocol stacks are preserved as in regular NETCONF client initiated connections. i.e. When using NETCONF Call Home, the NETCONF server initiates the TCP connection to the NETCONF client and then the NETCONF client starts the SSH and NETCONF sessions as normal. By using this method, the network element can establish a secure connection with a network management system.



This application note describes and discusses the NETCONF Call Home functionality introduced with ConfD 6.7.1.

2. Basics

The ConfD User Guide includes the core information needed to start using NETCONF Call Home; see section 15.4 “NETCONF Call Home”. The ConfD examples package includes two examples under the directory “examples.confd/netconf_call_home” which demonstrate the new functionality. The first example uses ConfD’s built-in SSH server and the second examples shows how to use it with an external SSH server (OpenSSH). Both examples come with a simplistic setup - utilizing the netconf-console utility program

as a NETCONF client/management engine to be contacted by ConfD's NETCONF Call Home feature. The remainder of this application note briefly describes how to use the Call Home feature, and, indirectly, how the ConfD Call Home examples work. This application note is a supplement to the ConfD User Guide and referenced examples. Reading the User Guide documentation and README files for these examples is recommended.

3. NETCONF Server (TCP Client) in Call Home

For NETCONF Call Home, the role of the network element is to initiate the TCP connection towards the NETCONF client, announcing a presence and allowing the contacted NETCONF client to then initiate the SSH and NETCONF connections as normal. To trigger the Call Home connection/process, the ConfD library introduces a new API function maapi_netconf_ssh_call_home(). It is a blocking call until either the TCP connection is established or an error occurs.

This step is a rather thin layer in regards to the amount of work: The TCP connection is initiated towards specified the NETCONF client, allowing further bi-directional communication. For details regarding the maapi_netconf_ssh_call_home() function call, see the confd_lib_maapi(3) man page.

To easily trigger NETCONF Call Home without implementing an application using the ConfD library, you can use the confd_cmd utility program (see the confd_cmd(3) man page):

```
bash# confd _ cmd -dd -c 'netconf _ ssh _ call _ home 127.0.0.1 4334'
```

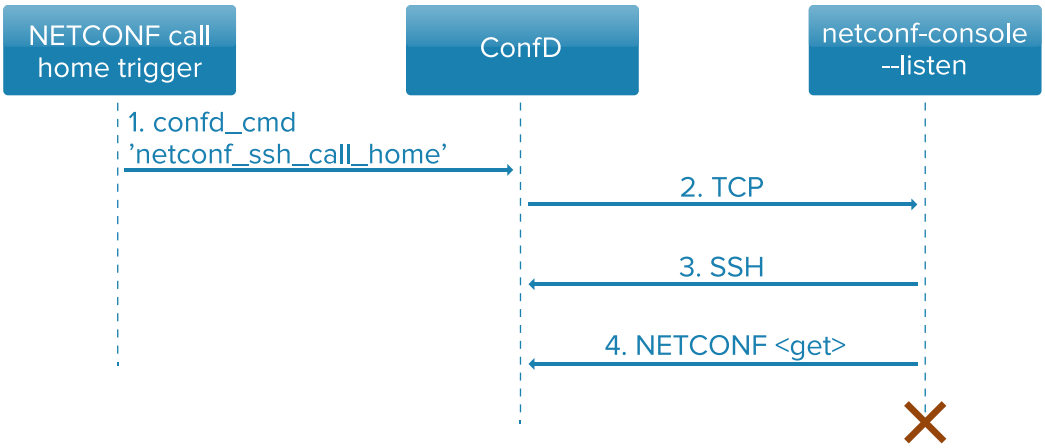
An example of usage is further in the following section.

4. NETCONF Client (TCP server) in Call Home

The role of the NETCONF client in Call Home usually belongs to a network management plane - managing the underlying network elements calling home. When testing the Call Home functionality, we probably don't need the full framework with configurable use case scenarios etc. For the purposes of this application note, we show two different simple scenarios using simple utility programs as the NETCONF client that is receiving the Call Home connection request.

Handle One Time NETCONF Call Home Connection

One approach is to use the netconf-console utility program in listen mode to wait for a Call Home connection and then automatically execute a defined action.



This is done as a part of the ConfD example referenced in previous section, i.e. running:

```
bash# netconf-console --listen --get
** listening on  0.0.0.0 4334
```

The utility program netconf-console is started in "tcp server mode" in which it waits for an incoming NETCONF Call Home connection. When a Call Home TCP connection occurs, netconf-console then executes the NETCONF <get> operation to retrieve the running configuration and device state information. In another terminal window, we can tell ConfD to initiate a NETCONF Call Home connection without writing code by using previously mentioned confd_cmd utility program:

```
bash# confd_cmd -dd -c 'netconf_ssh_call_home 127.0.0.1 4334'
netconf_ssh_call_home "127.0.0.1" "4334"
TRACE Connected (maapi) to ConfD
TRACE MAAPI_NETCONF_SSH_CALL_HOME  --> CONFD_OK
TRACE MAAPI_END_USER_SESSION  --> CONFD_OK
```

This leads to a response output from the listening netconf-console:

```
** listening on  0.0.0.0 4334
** call home from ('127.0.0.1', 37007)
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
      <authentication>
        <users>
          <user>
            <name>admin</name>
            <uid>9000</uid>
            <gid>100</gid>
>>>> LONG PRINTOUT OF <GET> RESPONSE HERE <<<<
```

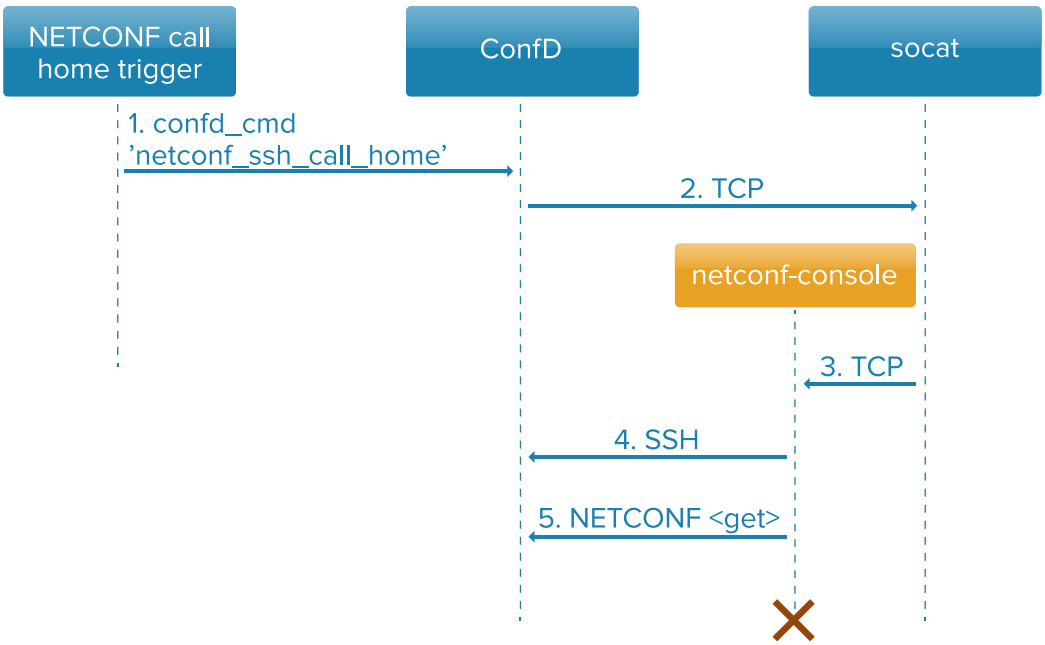
The ConfD developer log (when enabled in confd.conf) shows information regarding the inbound NETCONF client request for the <get> operation from netconf-console:

```
<INFO> 21-Nov-2018::16:08:57.283 dev confd[32016]: netconf id=14 new ssh session
for user "admin" from 127.0.0.1
<INFO> 21-Nov-2018::16:08:57.292 dev confd[32016]: netconf id=14 got rpc: {urn:iet
f:params:xml:ns:netconf:base:1.0}get attrs: message-id="1"
<INFO> 21-Nov-2018::16:08:57.292 dev confd[32016]: netconf id=14 get attrs: mes-
sage-id="1"
<INFO> 21-Nov-2018::16:08:57.293 dev confd[32016]: netconf id=14 sending rpc-re-
ply, attrs: message-id="1"
<INFO> 21-Nov-2018::16:08:57.343 dev confd[32016]: netconf id=14 close-session at-
trs: message-id="0"
<INFO> 21-Nov-2018::16:08:57.343 dev confd[32016]: netconf id=14 sending rpc-re-
ply, attrs: message-id="0"
```

In a real world use case, the <get> operation would be replaced by any necessary process, sequence of operations, or behavior required by the specific scenario.

Handle NETCONF Call Home Connections Repeatedly

As a variant of the above demo process, we can use a “SOcket CAT” (socat) Linux utility program to allow repetitive servicing of Call Home requests without the need to re-execute the netconf-console in listen mode. “socat” is run on the NETCONF client platform, registers for a specific NETCONF Call Home port, and then redirects all communication towards a defined system process.



To establish a NETCONF Call Home listener in our example, we can execute:

```
bash# socat -d -d TCP4-LISTEN:4334,bind=0.0.0.0,fork,reuseaddr,max-children=100
EXEC:./netconf-console-get.sh,fdin=4,fdout=5
```

The socat program takes a few parameters to run (for details or more options, see the socat man page):

```
-d -d          # print extra debugging info
TCP4-LISTEN:4334       # listen on 4334 for IPv4 connections
fork                   # service incoming request in child processes (keep
server ready to server more clients)
bind=0.0.0.0    # listen on any local interface
reuseaddr              # not to block / allow other sockets to bind
max-children=100       # limit maximum # of parallel workers to spawn
EXEC:./netconf-console-get.sh  # run the servicing script (see further)
fdin=4,fdout=5  # redirecting the NCH stream in/out descriptors to a custom
numbers (see further)
```

The child process netconf-console-get.sh is a simple shell script which processes the incoming connection redirected to it via file descriptors (FDs). It executes our specific command, in this case, a simple NETCONF <get> :

```
#!/bin/bash

# use a new port for a new connection towards home-caller;
# file /tmp/port.txt is a counter of ports used by child NETCONF server threads
# with each execution, raise port by 1, looping between values 12020 - 12120;
PORT_FILE=/tmp/port.txt
VAR=`cat ${PORT_FILE}` || `echo 0`
echo $(($VAR + 1)) > ${PORT_FILE}
PORT=$(eval echo $VAR)
if [ $PORT -gt 12120 ]; then
   echo "12020" > ${PORT_FILE}
fi

#
socat -d -d -d FD:4!!5 TCP4-LISTEN:$PORT,bind=127.0.0.1,su=nobody,range=127.0.0.0/8
&
date
eval echo '***NETCONF get from localhost:$PORT***'

# and finally, issue our server command - NETCONF <get>
netconf-console --get --port=$PORT
eval echo '***NETCONF get from localhost:$PORT done***'
exit 0
```

With the above commands, we can run a scenario similar to first example in this application note:

1. Run a ConfD instance

2. Start the NETCONF server, a socat command that has the shell script *netconf-console-get.sh* available in the current working directory:

```
bash# socat -d -d TCP4-LISTEN:4334,bind=0.0.0.0,fork,reuseaddr,max-children=100
EXEC:./netconf-console-get.sh,fdin=4,fdout=5
2018/11/30 14:23:26 socat[29324] N listening on AF=2 0.0.0.0:4334
```

3. In another terminal window, run the confd_cmd utility program to initiate the NETCONF Call Home process:

```
bash$ confd_cmd -dd -c 'netconf_ssh_call_home 127.0.0.1 4334'
netconf_ssh_call_home "127.0.0.1" "4334"
TRACE Connected (maapi) to ConfD
TRACE MAAPI_NETCONF_SSH_CALL_HOME  --> CONFD_OK
TRACE MAAPI_END_USER_SESSION  --> CONFD_OK
```

The window running the socat console will print debugging information regarding the executed operations:

```
2018/11/30 14:23:54 socat[29324] N accepting connection from AF=2 127.0.0.1:35613
on AF=2 127.0.0.1:4334
2018/11/30 14:23:54 socat[29324] N forked off child process 29327
2018/11/30 14:23:54 socat[29324] N listening on AF=2 0.0.0.0:4334
2018/11/30 14:23:54 socat[29327] N forking off child, using socket for reading
and writing
2018/11/30 14:23:54 socat[29327] N forked off child process 29328
2018/11/30 14:23:54 socat[29327] N forked off child process 29328
2018/11/30 14:23:54 socat[29327] N starting data transfer loop with FDs [6,6] and
[5,5]
2018/11/30 14:23:54 socat[29328] N execvp'ing "./netconf-console-get.sh"
Fri 30 Nov 14:23:54 CET 2018
***NETCONF get from localhost:4***
2018/11/30 14:23:54 socat[29331] N using file descriptor 4 for reading
2018/11/30 14:23:54 socat[29331] N using file descriptor 5 for writing
2018/11/30 14:23:54 socat[29331] N listening on AF=2 127.0.0.1:4
2018/11/30 14:23:54 socat[29331] N accepting connection from AF=2 127.0.0.1:50710
on AF=2 127.0.0.1:4
2018/11/30 14:23:54 socat[29331] N starting data transfer loop with FDs [4,5] and
[9,9]
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
      <authentication>
        <users>
            <name>admin</name>
            <uid>9000</uid>
>>>> LONG PRINTOUT OF <GET> RESPONSE HERE <<<<
…
2018/11/30 14:23:54 socat[29331] N socket 2 (fd 9) is at EOF
***NETCONF get from localhost:4 done***
2018/11/30 14:23:54 socat[29327] N childdied(): handling signal 17
2018/11/30 14:23:55 socat[29331] N exiting with status 0
2018/11/30 14:23:55 socat[29327] N exiting with status 0
2018/11/30 14:23:55 socat[29324] N childdied(): handling signal 17
```

This example shows how the socat command services one incoming NETCONF Call Home "request". The main "server loop" keeps running and allows servicing multiple parallel or subsequent clients calling home, spawning the child worker scripts repeatedly.

5. Summary

This application note has taken a look at what NETCONF Call Home is, how it is implemented in ConfD, and how it can be exercised using some simple utility programs and shell scripts.

For More Information

For more information about ConfD, visit https://www.tail-f.com

RFC 8071, “NETCONF Call Home and RESTCONF Call Home”

ConfD User Guide, Section 15.4 “NETCONF Call Home”

tail-f a Cisco company

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

Corporate Headquarters

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