NSO Tailf HCC

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1

NSO Tailf HCC Guide

Introduction

The Tail-f HCC Function pack (HCC) is a High-availability framework that is used to manage the master and slave relationship for the NSO CDB HA. In other words, HCC is used to tell NSO HA which node should be master and which nodes should be slaves.

This document contains deployment information and procedures for Tail-f NSO HA (CDB replication) and the Tail-f High Availability Cluster Communications (tailf-hcc) minimal HA framework application.

HAFW In this document, a *HA group member* is a node who shares a unique id, a token, and has a master-slave relation with other group members.

A *Cluster member* is a node who is either a service node or a device node in an NSO Cluster.

CHAPTER 2

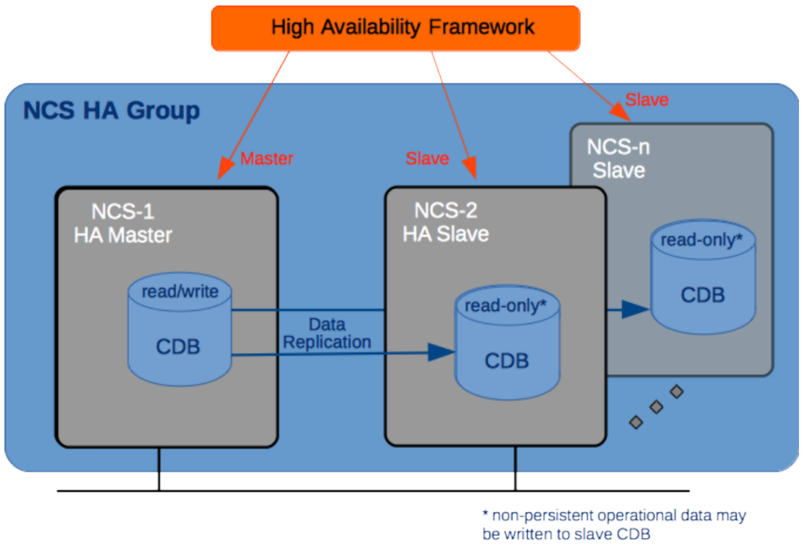
NSO Tailf HCC Usage

NSO HA

Before we can go into how the HCC function pack is deployed and used, we need to understand the NSO CDB replication - NSO High Availability (HA). The NSO HA is shipped with NSO and HCC can be added on top of that, if it is needed. HCC detects when nodes fail and instructs NSO to update the master and slave relationship. If the master node fails, the HCC elects one of the remaining slaves to be the new master. Any remaining slaves are not updated with information about the new master by HCC. Instead an alarm is raised and an operator needs to investigate the cause of the failover and resolve the situation manually.

NSO natively supports replication of the CDB configuration as well as operational data kept in CDB. The replication architecture is that of one active master and at least one passive slave.

Figure 1. NCS HA



A group of NSO hosts consisting of a master, and one or more slaves, is referred to as an HA group.

All configuration write operations must occur at the master and NSO will automatically distribute the configuration updates to the set of live slaves. All write operations for replicated operational data must also occur at the master, with the updates distributed to the live slaves, whereas non-replicated operational data can also be written on the slaves.

The only thing NSO HA does is replicate the CDB data among the members in the HA group. It does not determine which NSO nodes are members of the HA Group or which NSO node is designated as Master and which are Slaves - this is the task of a High-Availability Framework (HAFW) which must be in place. The HAFW must instruct NSO which nodes are up and down using methods from Ha class in the NSO Java library.

HAFW tailf-hcc is a High-Availability Framework (HAFW)

For more information on NSO HA and CDB replication, see *NSO Administration Guide - High Availability*.

Tail-f High Availability Cluster Communications (tailf-hcc)

As previously stated, the Tail-f High Availability Cluster Communications (tailf-hcc function pack) is a lightweight, minimal featured implementation of the high availability framework to coordinate the master-slave relationships of the HA groups and cluster member nodes via interaction with the NSO HA capabilities. The tailf-hcc package supports both NSO non-clustered deployments and NSO clustered deployments.

HCC can run in many different scenarions, in this document we describe three basic scenarios:

- **Basic**

  In this scenario you have a HA group where all nodes can reach each other.
- **Cluster**

  Separate clusters of HA-groups, where the clusters nodes can reach each other internally in the cluster and the configured remote-nodes can reach each other over.
- **Border Gateway Protocol - BGP**

  Using the **quagga-bgp** NED, HCC can react on BGP changes to update the master and slave relationships.

Configuration

NSO/OS requirements and configuration

When setting up a HA cluster using HCC both NSO and HCC configuration needs to be setup correctly in order for everything to work.

- On all nodes the OS must be configured to allow for NSO HA operations, i.e configure the firewall to allow the ports used by NSO in regards of HA.
- NSO HA must be enabled in `ncs.conf` configuration file. The default location on a system install is `/etc/ncs/ncs.conf`. Add or uncomment these lines in `ncs.conf`:

```
<ha>
  <enabled>true</enabled>
</ha>
```

- The encryption keys in `ncs.conf` must be the same on the Master and the Slave nodes. A rule of thumb is that the slave and master should have identical `ncs.conf` files:

```
<encrypted-strings>
  <DES3CBC>
    <key1>0123456789abcdef</key1>
    <key2>0123456789abcdef</key2>
    <key3>0123456789abcdef</key3>
    <initVector>0123456789abcdef</initVector>
  </DES3CBC>
  <AESCFB128>
    <key>0123456789abcdef0123456789abcdef</key>
    <initVector>0123456789abcdef0123456789abcdef</initVector>
  </AESCFB128>
</encrypted-strings>
```

Note If the user fails to enable HA a specific log message can be found in the `ncs-java-vm.log`:

```
...
<ERROR> 22-May-2014::15:30:32.908 tcmApp (tailf-hcc:tcm)-Run-0: - NCS
HA is likely not enabled
<ERROR> 22-May-2014::15:30:32.908 NcsMain (tailf-hcc:tcm)-Run-0: -
Received exception from tailf-hcc
...
```

- All Slave nodes must be reachable by the Master
- All Remote-nodes must be reachable by the Cluster-Manager

Local Install If running as non-root user, verify that the `sudo` command works without being prompted for a password for privileged commands. **Only** important for a local installation of NSO.

HA

The top container named `ha` is used for configuration and status checks of the tailf-hcc HA framework. This is a presence container.

Global configuration

The HA cluster's global configuration parameters found under `/hcc:ha/` are parameters applicable to all nodes sharing the same token. The global parameters are described in Table 2, "Global configuration".

Table 2. Global configuration

Parameters	Type	Description
token	string	This value is used as the shared secret when setting up HA between nodes. Mandatory.
local-user	string	local-user used for remote device and/or quagga device authorization

Parameters	Type	Description
failure-limit	int32	The number of failed HA state checks before declaring a failure [default: 10]
interval	int32	Interval on which HA state and BGP state are checked (seconds) [default: 4]

VIP configuration

L2 or L3 VIP is for L2 failover handling

The HA cluster configuration parameters are found in list `/ha/vips/vip` containing VIP parameters applicable for virtual IP address management. For each such VIP configuration element a label `$interface:ncsvip` will be added for the current master node with the address specified, and removed when the node is no longer master. Table 3, "VIP parameters" shows the global VIP parameters. The support for multiple VIPs aims to replace the older and now deprecated single VIP solution in the YANG model.

Table 3. VIP parameters

Parameters	Type	Description
address	inet:ip-address	The Virtual IP address to bring up on a labeled interface
broadcast-address	inet:ip-address	The Broadcast address used for the labeled interface. If not set, the broadcast address will be calculated from the address and bitmask of the interface

BGP Anycast Configuration

L2 or L3 BGP is for L3 failover handling

The HA cluster configuration parameters found under `/hcc:ha/bgp` are parameters applicable for BGP Anycast failover detection mechanism. Details for the global parameters of BGP Anycast failover are shown in Table 4, "BGP Anycast configuration".

QUAGGA-NED When BGP is enabled, tailf-hcc requires the NED **quagga-bgp** to be of version 3.3 and up.

Table 4. BGP Anycast configuration

Parameters	Type	Description
failure-limit	int32	The number of failed anycast path state checks before declaring a failure

Parameters	Type	Description
anycast-prefix	inet:ip-prefix	BGP anycast prefix to monitor
anycast-path-min	int32	The minimum number of valid anycast paths.
clear-enable	boolean	In the case of applying configuration towards a quagga device, should the node also clear BGP sessions and routes on the quagga

Member Configuration

The HA group member's configuration parameters found under `/hcc:ha/member` are parameters applicable to each configured HA group member node. Table 5, "Member parameters" details the HA group membership parameters.

Table 5. Member parameters

Parameters	Type	Description
name	string	The name of the member.
address	inet:ip-address	Member IP address
default-ha-role	cluster-role-type	The default HA role for the node
failover-master	boolean	HA slave nodes capability of serving as a failover-master.
relay-name	string	The name of the node that should act as a relay between this node and the master
cluster-manager	boolean	Cluster manager node
managed-cluster-nodes	leafref -> /cluster/remote-node/name	Which remote nodes the cluster-manager should monitor
quagga-device	leafref -> /devices/device/name	Device used for quagga device monitoring and managing
vip-interfaces/vip-interface[vip-interface address]/vip-interface	string	The interface used for each vip
vip-interfaces/vip-interface[vip-interface address]/address	leafref -> ha/vips/vip[address]/address	The address used for each vip interface

Actions

The tailf-hcc package is operated by a series of `tailf:actions`

Node actions

The actions in Table 6, "Node actions" are applicable for all types of nodes, and only effects the node itself.

Table 6. Node actions

action	input	Description
activate	none	The node will be activated and take the HA role configured under /ha/member/default-ha-role
deactivate	none	The node will be deactivated, and take the HA role `none`
role-override	ha-role	Override the current HA role with the specified role
role-revert	none	Revert the HA role to the default-ha-role value
status	none	Retrieve node HA status
force-be-slave-to	member	Set the node in slave-mode and try to connect and set master

Cluster manager node actions

The actions in Table 7, "Cluster manager specific actions" are only applicable for cluster-manager nodes, and effects the node itself as well as its configured remote nodes.

Table 7. Cluster manager specific actions

action	input	Description
cluster-activate	none	The node and its managed remote nodes will be activated and take the HA role configured under `/ha/member/default-ha-role`
cluster-deactivate	none	The node and its managed remote nodes will be deactivated and take the HA role `none`
cluster-role-override	ha-role	Override the current HA role with the specified role for the node and its managed remote nodes
cluster-role-revert	none	Revert the HA role to the default-ha-role value for the node and its managed remote nodes
cluster-status	none	Retrieve node HA status of the node and its managed remote nodes

Action Authorization

When invoking an action, the `user` and `context`, will be that of the user session doing the invocation. For `cluster`-actions, this `user` must have authorization to invoke remote actions. This is configured under `/cluster/authgroup`.

How it works

Basics

When tailf-hcc is activated, by issuing the ha action `activate`, each node will try to assume the `default-ha-role` configured under `/hcc:ha/hcc:member{name}/hcc:default-ha-role`. If it is a slave it will try to connect to the master, which is the member with default-ha-role set to `master`.

Each Slave node will periodically do a status check to see if a HA failure has occurred. The interval between each check is determined by the global ha configuration value `/hcc:ha/hcc:interval`. The default value is four (4 seconds). The number of times a status check is allowed to fail before a Slave declares a failure is determined by the global ha configuration value `/hcc:ha/hcc:failure-limit`. The default value is ten (10). This means that with default values, the Slave will declare a failure around 40 seconds after first persisting negative status check (4 times 10).

What a Slave should do when a failure is detected is determined by its ha configuration and the failure type. There are three (3) types of failures, described below.

Three types of failure

node-failure	A `node-failure` is when a Slave node loses HA connection with its master.
device-node-failure	A `device-node-failure` is when a Slave managed cluster node (remote-node) loses HA connection with its master (Can only happen in a Cluster Setup).
bgp-failure	A `bgp-failure` is when a Slave node loses its BGP prefix path to its master (Can only happen if BGP is configured).

Failover

If the Slave is a `failover-master`, `/hcc:ha/hcc:member/hcc:failover-master = true`, it will upon a failure initiate a failover; that is, transition to the 'Master' role. This holds true only for nodes configured to be slaves by default (default-ha-role = slave); nodes that are slaves due to the action `role-override` will not initiate the failover process.

In a Cluster

In a Cluster-setup, the configured cluster-manager, `/hcc:ha/hcc:member/hcc:cluster-manager = true`, will check the HA status of its managed remote nodes, `/hcc:ha/hcc:member/hcc:managed-cluster-nodes`, as well as its own status. If the cluster-manager detects that its Master node, or the Master node of one of its managed remote-nodes, is no longer available, the cluster-manager will initiate the failover as per above, with the difference that it will also instruct its managed remote-nodes to failover.

The cluster-manager will communicate with its managed remote-nodes via actions, and thus it is important that tailf-hcc has the authorization to do so. tailf-hcc will use the `user` configured under /ha/local-user for authorization. This `user` must match a local user under /cluster/authgroup/umap, used towards the remote-nodes.

With BGP Enabled

QUAGGA-NED When BGP is enabled, tailf-hcc requires the NED **quagga-bgp** to be of version 3.3 and up.

If a BGP is enabled, when `/hcc:ha/hcc:bgp` is configured, the Slave node will, upon each ha status check, also issue a *show ip bgp <prefix>*, where prefix is the value of `/hcc:ha/hcc:bgp/hcc:anycast-prefix`, query towards its configured quagga device, `/hcc:ha/hcc:member/hcc:quagga-device`. The query will result in something like:

```
....
    BGP routing table entry for 192.168.60.100/32
    Paths: (2 available, best #2, table Default-IP-Routing-Table)
      Advertised to non peer-group peers:
      192.168.31.2
      Local
        0.0.0.0 from 0.0.0.0 (192.168.31.3)
          Origin incomplete, metric 1, localpref 100, weight 32768, valid,
          sourced
          Last update: Thu Nov  5 11:19:06 2015

      Local
        0.0.0.0 from 0.0.0.0 (192.168.31.3)
          Origin IGP, metric 0, localpref 100, weight 32768, valid,
        sourced, local, best
          Last update: Thu Nov  5 11:19:06 2015
....
```

If the available 'Paths' - 'Paths: (2 available, best #2, table Default-IP-Routing-Table)' - are below the configured minimum, `/hcc:ha/hcc:bgp/hcc:anycast-path-min`, the Slave will initiate a failover as per above.

BGP configuration logic

QUAGGA-NED When BGP is enabled, tailf-hcc requires the NED **quagga-bgp** to be of version 3.3 and up.

If BGP is enabled for a node, the node will reconfigure its quagga-device depending on its current HA state. It will do this by applying pre-configured device templates. There MUST exist (at least, see below) 4 device templates, with specific names. There must be a template for when a node is in state Master, Slave, None, and finally Failover Master. The Failover Master is quite similare to Master state, but with the exception that the node became Master due to a failover. The following templates must exist:

hcc-master	The template to apply when state is Master. Will be applied to the `default-ha-role = master` node, or the node set to master via the action `role-override`
hcc-slave	The template to apply when state is Slave. Will be applied to the `default-ha-role = slave` node, or the node set to slave via the action `role-override` and `force-be-slave-to`
hcc-none	The template to apply when state is None. Will be applied to the `default-ha-role = none` node, or the node set to none via the

	action `role-override`. This template will also be applied when `/ha/bgp` is created but the node is not activated, when the node is deactivated, or when `/ha/bgp` is deleted
hcc-failover-master	The template to apply when state is Master due to a failover. Will be applied to a slave node that automatically transitioned to master due to a failover

Node specific templates

As a Master and a Slave will contain the same CDB configuration, they will share the device templates as well. As there might be a need for a Master and a Slave to have different configurations for its quagga device, each template can be suffixed with its member name, which will have precedence. For example, `hcc-master-NAME`.

The node will apply configuration to the quagga device, and thus it is important that tailf-hcc has the authorization to do so. tailf-hcc will use the `user` configured under `/ha/local-user` for authorization. This `user` must match a local user under `/devices/authgroups/umap`, used towards the quagga device.

See chapter *Deployment Example: NSO HA* for details on setup, configuration and operations.

VIP logic

If a VIP is configured in list `/ha/vips/vip`, each node will check its HA state. If it is Master, the node will bring up a virtual IP with an address set under `/ha/vips/vip[address]/address` on the interface `/ha/vip/member/vip-interfaces/vip-interface[vip-interface address]/vip-interface`. When the node is not Master, or if the node goes down, it will bring down the interface again.

Alarms

tailf-hcc can generate the following three alarms:

node-failure

device-node-failure

bgp-failure

Which corresponds to the three types of failure with the same name. See the section called “Three types of failure” for more information on what triggered the failure and raised the alarm.

See the section called “NSO Tailf HCC Alarms model” for the yang-definition on the alarm types.

The alarms will always be generated with the severity level *CRTICITAL*. The generated alarm(s) will be cleared when the *role-revert* action has been triggered.

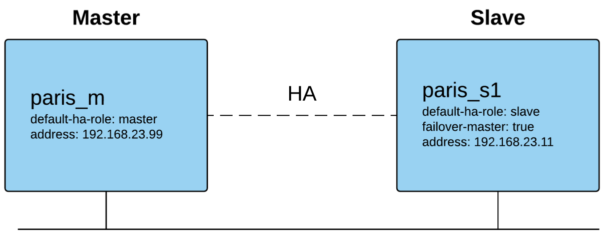
CHAPTER 3

NSO Tailf HCC Examples

Basic deployment

In this example there is one Master node, named 'paris_m', and one Slave node, 'paris_s1'. 'paris_s1' will act as the fail-over master.

Figure 8. NCS HA



Node OS Configuration

The ha member names match the machine hostname, and is resolvable. The hosts-file on 'paris_m':

```
$ cat /etc/hosts
127.0.1.1     paris_m
127.0.0.1     localhost
192.168.23.99 paris_m
192.168.23.11 paris_s1
```

Node NSO Configuration

Each node has enabled 'ha' in 'ncs.conf':

```
<ha>
  <enabled>true</enabled>
    </ha>
```

tailf-hcc is properly loaded:

```
# On 'paris_m'
user@ncs> show packages package tailf-hcc
packages package tailf-hcc
 package-version 4.0
 description     "NED package for Tail-f HA Cluster Control Interface"
 ncs-min-version [ 4.0.1 ]
 ...
 oper-status up
[ok]
# On 'paris_s1'
user@ncs> show packages package tailf-hcc
packages package tailf-hcc
 package-version 4.0
 description     "NED package for Tail-f HA Cluster Control Interface"
 ncs-min-version [ 4.0.1 ]
 ...
 oper-status up
[ok]
```

Configure the following ha-configuration:

```
# On 'paris_m'
user@ncs> configure
Entering configuration mode private
user@ncs% set ha token sometoken
user@ncs% set ha failure-limit 10
user@ncs% set ha interval 4
user@ncs% set ha member paris_m address 192.168.23.99 default-ha-role master
user@ncs% set ha member paris_s1 address 192.168.23.11 default-ha-role slave failover-master tru
user@ncs% commit
Commit complete.
user@ncs% show ha
token         sometoken;
interval      4;
failure-limit 10;
member paris_m {
    address         192.168.23.99;
    default-ha-role master;
}
member paris_s1 {
    address         192.168.23.11;
    default-ha-role slave;
    failover-master true;
}

# On 'paris_s1' do exactly the same thing
```

Show status before activation

```
# On 'paris_m'
user@paris_m> show ncs-state ha
ncs-state ha mode none

# On 'paris_s1'
user@paris_s1> show ncs-state ha
ncs-state ha mode none
```

Activate ha

```
# On 'paris_m'
user@paris_m> request ha commands activate
status activated

# On 'paris_s1'
user@paris_s1> request ha commands activate
status activated
```

Show status after activation

```
# On 'paris_m'
user@paris_m> show ncs-state ha
ncs-state ha mode master
ncs-state ha node-id paris_m
ncs-state ha connected-slave [ paris_s1 ]

# On 'paris_s1'
user@paris_s1> show ncs-state ha
ncs-state ha mode slave
ncs-state ha node-id paris_s1
ncs-state ha master-node-id paris_m
```

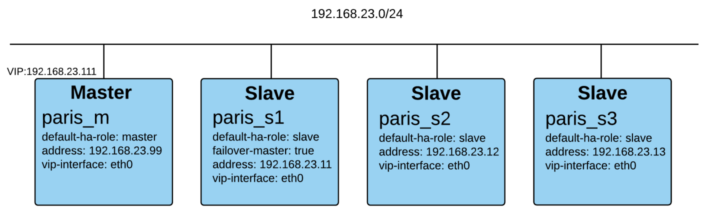
Advanced

The following examples make use of the tool 'nct'. This tool is not required, but makes it easier to communicate and configure several nodes at once. For each example the corresponding CLI command will be shown.

The tool is well documented in the NSO man pages.

In this example there is one Master node, named 'paris_m', and three Slave nodes, 'paris_s1', 'paris_s2' and 'paris_s3'. Node 'paris_s1' will act as the fail-over master. Two VIPs are configured.

Figure 9. NCS HA



ncs-cluster-

The hostsfile for nct used in this example:

```
$ cat hostsfile
{"192.168.23.99", [
```

tool hostsfile

```
                  {name, "paris_m"},
                  {groups, ["all", "master"]},
                  {ssh_user, ...}, {ssh_pass, ...}
                 ]}.
{"192.168.23.11", [
                  {name, "paris_s1"},
                  {groups, ["all", "slave"]},
                  {ssh_user, ...}, {ssh_pass, ...}
                 ]}.
{"192.168.23.12", [
                  {name, "paris_s2"},
                  {groups, ["all", "slave"]},
                  {ssh_user, ...}, {ssh_pass, ...}
                 ]}.
{"192.168.23.13", [
                  {name, "paris_s3"},
                  {groups, ["all", "slave"]},
                  {ssh_user, ...}, {ssh_pass, ...},
                 ]}.
```

Node OS Configuration

The ha member names match the machine hostname, and is resolvable. The hosts-file on 'paris_m':

```
$ cat /etc/hosts
127.0.1.1     paris_m
127.0.0.1     localhost
192.168.23.99 paris_m
192.168.23.11 paris_s1
192.168.23.12 paris_s2
192.168.23.13 paris_s3
192.168.23.111 ha-vip
```

Node NSO Configuration

Each node has enabled 'ha' in 'ncs.conf'

```
<ha>
  <enabled>true</enabled>
</ha>
```

tailf-hcc is properly loaded:

```
# Corresponding CLI command: 'user@ncs> show packages package tailf-hcc'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show packages package tailf-hcc'

Cli command to 192.168.23.99 [paris_m]
packages package tailf-hcc
 package-version 4.0
 description     "NED package for Tail-f HA Cluster Control Interface"
 ncs-min-version [ 4.0.1 ]
 ...
 oper-status up
[ok]

Cli command to 192.168.23.11 [paris_s1]
...
```

Each node share the following ha-configuration:

```
# Corresponding CLI command: 'user@ncs> show configuration ha'
```

```
$ nct cli-cmd --hostsfile hostsfile -c 'show configuration ha'

Cli command to 192.168.23.99 [paris_m]
token         sometoken;
interval      4;
failure-limit 10;
vips {
    vip 10.0.0.111 {
        netmask 255.255.0.0;
    }
    vip 10.0.0.112 {
        netmask 255.255.0.0;
    }
}
member paris_m {
    address         192.168.23.99;
    default-ha-role master;
    vip-interfaces {
        vip-interface eth0 10.0.0.111;
        vip-interface lo 10.0.0.112;
    }
}
member paris_s1 {
    address         192.168.23.11;
    default-ha-role slave;
    failover-master true;
    vip-interfaces {
        vip-interface eth0 10.0.0.111;
        vip-interface lo 10.0.0.112;
    }
}
member paris_s2 {
    address         192.168.23.12;
    default-ha-role slave;
    vip-interfaces {
        vip-interface eth0 10.0.0.111;
        vip-interface lo 10.0.0.112;
    }
}
member paris_s3 {
    address         192.168.23.13;
    default-ha-role slave;
    vip-interfaces {
        vip-interface eth0 10.0.0.111;
        vip-interface lo 10.0.0.112;
    }
}

Cli command to 192.168.23.11 [paris_s1]
token         sometoken;
interval      4;
failure-limit 10;
...
```

Note There can be only one 'default-ha-role = master' and only one 'failover-master = true'. 'paris_s2' and 'paris_s3' serves as 'disaster recovery' copies of CDB in case both 'paris_m' and 'paris_s1' were lost.

Activate HA

Show status before activation

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode none

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode none

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode none

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode none
```

Activating HA by invoking the action 'activate', first on the master node, then on the slaves:

```
# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action activate --group master

HA Node 192.168.23.99:8080 [paris_m]
activated

# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on each slave node
$ nct ha --hostsfile hostsfile --action activate --group slave

HA Node 192.168.23.11:8080 [paris_s1]
activated

HA Node 192.168.23.12:8080 [paris_s2]
activated

HA Node 192.168.23.13:8080 [paris_s3]
activated
```

Show status after activation:

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode master
ncs-state ha node-id paris_m
ncs-state ha connected-slave [ paris_s1 paris_s2 paris_s3 ]

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode slave
ncs-state ha node-id paris_s1
```

```
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode slave
ncs-state ha node-id paris_s2
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode slave
ncs-state ha node-id paris_s3
ncs-state ha master-node-id paris_m
```

Note Slaves do not know about the status of the other slaves

Check VIP Interfaces

Check that the VIP interfaces is up on the master but not on the slaves (grep returns nothing if not present):

```
# Corresponding terminal command: '$ sudo ifconfig | grep ncsvip'
# issued on each node
$ nct ssh-cmd --hostsfile hostsfile -c 'sudo ifconfig | grep ncsvip'

SSH command to 192.168.23.99:22 [paris_m]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
eth0:ncsvip Link encap:Ethernet  HWaddr 52:54:00:fa:61:99
lo:ncsvip Link encap:Local Loopback

SSH command to 192.168.23.11:22 [paris_s1]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:

SSH command to 192.168.23.12:22 [paris_s2]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:

SSH command to 192.168.23.13:22 [paris_s3]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
```

HA Failover

Simulate that the master node is down by inactivating it:

```
# Corresponding CLI command: 'user@ncs> request ha commands deactivate'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action deactivate --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
deactivated
```

Check that the failover-master transitioned to master:

Note Failover will take (interval*failure limit) seconds before the failover master confirms loss of 'master' and initiates a failover. If the Master-Slave communication is re-established during this period, HA operation would continue as normal

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'
```

```
Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode none

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode master
ncs-state ha node-id paris_s1

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode none

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode none
```

Note The nodes 'paris_s2' and 'paris_s3' will not automatically transition to 'slave' to the new 'master'. They assume the role 'none' upon connectivity loss to 'paris_m'

Check that the VIP interfaces is now up on the new master:

```
# Corresponding terminal command: '$ sudo ifconfig | grep ncsvip'
# issued on each node
$ nct ssh-cmd --hostsfile hostsfile -c 'sudo ifconfig | grep ncsvip'

SSH command to 192.168.23.99:22 [paris_m]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
encap:Ethernet  HWaddr 52:54:00:fa:61:99

SSH command to 192.168.23.11:22 [paris_s1]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
eth0:ncsvip Link encap:Ethernet  HWaddr 52:54:00:fa:61:98
lo:ncsvip Link encap:Local Loopback

SSH command to 192.168.23.12:22 [paris_s2]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:

SSH command to 192.168.23.13:22 [paris_s3]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
```

HA Fail-back to original configuration

After rectifying the cause of the Master-slave communication failure, the approach is to bring the original Master back on-line initially as a Slave, which will then connect to the new Master to complete a CDB re-sync prior to being transitioned to the operational Master.

There are no provisions in the tailf-hcc application to automatically sense and initiate a revert back to the original HA Cluster configuration.

Note It is **very important** that the original master is setup to be Slave to the failover master before activation after a fail-over. If not, potential configuration loss may occur.

Override to slave

Override the ha role of *paris_m* to *slave* and then activate *paris_m*:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-override role slave'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action role-override --role slave
--name paris_m
```

```
HA Node 192.168.23.99:8080 [paris_m]
override

# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action activate --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
activated
```

Verify override

Verify that *paris_m* is now slave to *paris_s1*:

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode slave
ncs-state ha node-id paris_m
ncs-state ha master-node-id paris_s1

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode master
ncs-state ha node-id paris_s1
ncs-state ha connected-slave [ paris_m ]

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode none

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode none
```

Role-revert nodes

Role-revert all nodes. First *paris_m*, then the original *slave* nodes:

Note Allow adequate time for CDB on re-activated *paris_m* to sync with CDB on the failed-over Master. The time depends on the CDB size and how much of the CDB needs to be updated.

```
# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action role-revert --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
reverted

# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on each slave
$ nct ha --hostsfile hostsfile --action role-revert --group slave

HA Node 192.168.23.11:8080 [paris_s1]
reverted

HA Node 192.168.23.12:8080 [paris_s2]
reverted
```

```
HA Node 192.168.23.13:8080 [paris_s3]
reverted
```

Verify role-revert

Verify that *paris_m* is the master once again:

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'
Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode master
ncs-state ha node-id paris_m
ncs-state ha connected-slave [ paris_s1 paris_s2 paris_s3 ]

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode slave
ncs-state ha node-id paris_s1
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode slave
ncs-state ha node-id paris_s2
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode slave
ncs-state ha node-id paris_s3
ncs-state ha master-node-id paris_m
```

Check VIP interface

Check that the VIP interfaces is up on the master but not on the slaves (grep returns nothing if not present):

```
# Corresponding terminal command: '$ sudo ifconfig | grep ncsvip'
# issued on each node
$ nct ssh-cmd --hostsfile hostsfile -c 'sudo ifconfig | grep ncsvip'

SSH command to 192.168.23.99:22 [paris_m]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
eth0:ncsvip Link encap:Ethernet  HWaddr 52:54:00:fa:61:99
lo:ncsvip Link encap:Local Loopback

SSH command to 192.168.23.11:22 [paris_s1]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:

SSH command to 192.168.23.12:22 [paris_s2]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:

SSH command to 192.168.23.13:22 [paris_s3]
SSH OK : 'ssh sudo ifconfig | grep ncsvip' returned:
```

HA Disaster Recovery

Disaster Recovery (DR) when the master CDB is no longer viable and the CDB on one of the Disaster Recovery Slaves, *paris_s2* and *paris_s3* has the most up-to-date configuration/operational data in the HA group. To recover the DR Slave CDB to the configured Master follow the following procedure.

Simulate disaster failover

Simulate that both *paris_m* and *paris_s1* goes down, by deactivating them:

```
# Corresponding CLI command: 'user@ncs> request ha commands deactivate'
# on node 'paris_s1'
$ nct ha --hostsfile hostsfile --action deactivate --name paris_s1

HA Node 192.168.23.11:8080 [paris_s1]
deactivated

# Corresponding CLI command: 'user@ncs> request ha commands deactivate'
# on node 'paris_m'
$ nct ha --hostsfile hostsfile --action deactivate --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
deactivated
```

Check HA status

Check HA status on all nodes:

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode none

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode master
ncs-state ha node-id paris_s1

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode none

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode none
```

Set a new master

Make *paris_s2* the new *master* with the action *role-override*:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-override role master'
# on node 'paris_s2'
$ nct ha --hostsfile hostsfile --action role-override --role master
--name paris_s2

HA Node 192.168.23.12:8080 [paris_s2]
override
```

Force to be slave

Force *paris_m* to be slave with the action *role-override*, then set *paris_s2* to be its master with the action *force-be-slave-to*, and finally activate *paris_m*:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-override role slave'
# on node 'paris_m'
$ nct ha --hostsfile hostsfile --action role-override --role slave
--name paris_m

HA Node 192.168.23.99:8080 [paris_m]
override

# Corresponding CLI command: 'user@ncs> request ha commands force-be-slave-to member paris_s2
# on node 'paris_m'
```

```
$ nct ha --hostsfile hostsfile --action force-be-slave-to
--member paris_s2 --name paris_m

HA Node 192.168.23.11:8080 [paris_m]
Trying to be slave to paris_s2

# Corresponding CLI command: 'user@ncs> request ha commands activate'
# on node 'paris_m'
$ nct ha --hostsfile hostsfile --action activate --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
activated
```

Verify new master

Verify that *paris_s1* now is *master* over *paris_m*:

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode slave
ncs-state ha node-id paris_m
ncs-state ha master-node-id paris_s2

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode none

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode master
ncs-state ha node-id paris_s2
ncs-state ha connected-slave [ paris_m ]

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode none
```

Role-revert nodes

Role-revert all nodes. First *paris_m*, then the original *slave* nodes, and activate *paris_s1* again:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on 'paris_m'
$ nct ha --hostsfile hostsfile --action role-revert --name paris_m

HA Node 192.168.23.99:8080 [paris_m]
reverted

# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on each slave
$ nct ha --hostsfile hostsfile --action role-revert --group slave

HA Node 192.168.23.11:8080 [paris_s1]
reverted

HA Node 192.168.23.12:8080 [paris_s2]
reverted

HA Node 192.168.23.13:8080 [paris_s3]
reverted

# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on 'paris_s1'
$ nct ha --hostsfile hostsfile --action activate --name paris_s1

HA Node 192.168.23.11:8080 [paris_s1]
activated
```

Verify role-revert

Verify that *paris_m* is the master once again:

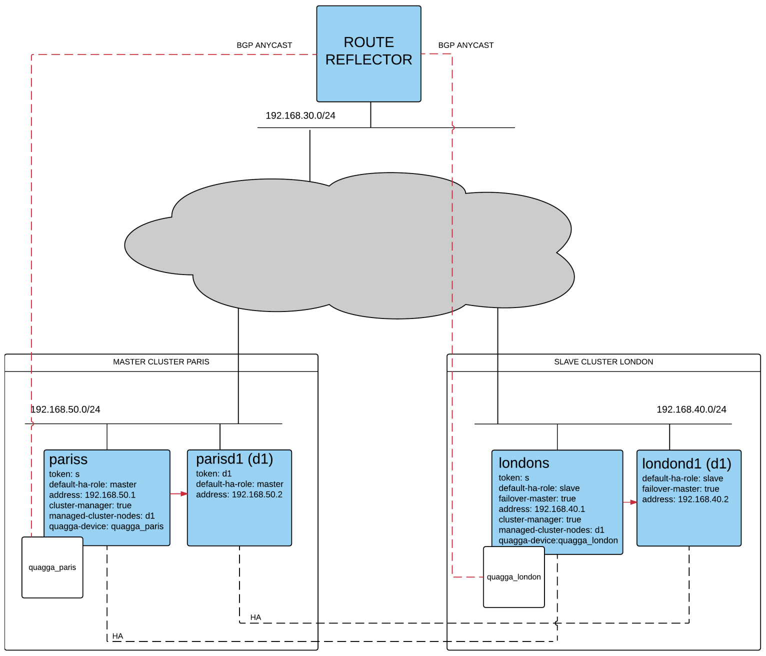
```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c 'show ncs-state ha'

Cli command to 192.168.23.99 [paris_m]
ncs-state ha mode master
ncs-state ha node-id paris_m
ncs-state ha connected-slave [ paris_s1 paris_s2 paris_s3 ]

Cli command to 192.168.23.11 [paris_s1]
ncs-state ha mode slave
ncs-state ha node-id paris_s1
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.12 [paris_s2]
ncs-state ha mode slave
ncs-state ha node-id paris_s2
ncs-state ha master-node-id paris_m

Cli command to 192.168.23.13 [paris_s3]
ncs-state ha mode slave
ncs-state ha node-id paris_s3
ncs-state ha master-node-id paris_m
```

NSO Cluster HA

In this example there are 2 Clusters, **PARIS** and **LONDON**. **PARIS** is the *master* cluster, and **LONDON** the *slave* cluster. Cluster **PARIS** consist of the *service node pariss* and the *device node parisd1*. Cluster **LONDON** consist of the *service node londons* and the *device node londond1*. *pariss* is the master to slave *londons*, *parisd1* is the master to slave *londond1*.

The service nodes will require some additional configuration for remote node communication.

BGP will be enabled for the service nodes, so *pariss* and *londons* will require some additional configuration for the quagga device, using the NED *quagga-bgp*.



ncs-cluster-

The hostsfile for nct used in this example:

```
  $ cat hostsfile
{"192.168.50.1", [
```

tool hostsfile

```
                   {name, "pariss"},
                   {groups, ["all", "service", "paris"]},
                   {ssh_user, ...}, {ssh_pass, ...}
                  ]}.
{"192.168.40.1", [
                   {name, "londons"},
                   {groups, ["all", "service", "london"]},
                   {ssh_user, ...}, {ssh_pass, ...},
                  ]}.
{"192.168.50.2", [
                   {name, "parisd1"},
                   {groups, ["all", "device", "paris"]},
                   {ssh_user, ...}, {ssh_pass, ...},
                  ]}.
{"192.168.40.2", [
                   {name, "londond1"},
                   {groups, ["all", "device", "london"]},
                   {ssh_user, ...}, {ssh_pass, ...},
                  ]}.
```

Node OS configuration

Hostname Resolution

The ha member names match the machine hostname, and is resolvable.

The cluster configuration needs to point to the Device node(s) as part of the ‘remote-node’ configuration. As the service nodes *pariss* and *londons* share any configuration stored in CDB, an additional hostname is added for the device nodes. The Service Node in each cluster needs to resolve the hostname *d1* to the appropriate address for its referenced cluster, as shown below.

The hosts-file on each node:

```
# Corresponding terminal command: '$ cat /etc/hosts'
# issued on each node
$ nct ssh-cmd --hostsfile hostsfile -c 'cat /etc/hosts'

SSH command to 192.168.50.1:22 [pariss]
SSH OK : 'ssh cat /etc/hosts' returned:
127.0.1.1     pariss
127.0.0.1     localhost
192.168.50.1  pariss
192.168.40.1  londons
192.168.50.2  parisd1 d1
192.168.50.21 s-vip

SSH command to 192.168.40.1:22 [londons]
SSH OK : 'ssh cat /etc/hosts' returned:
127.0.1.1     londons
127.0.0.1     localhost
192.168.50.1  pariss
192.168.40.1  londons
192.168.40.2  londond1 d1
192.168.50.21 s-vip

SSH command to 192.168.50.2:22 [parisd1]
SSH OK : 'ssh cat /etc/hosts' returned:
127.0.1.1    parisd1
127.0.0.1    localhost
192.168.50.1 pariss
```

```
192.168.50.2 parisd1
192.168.40.2 londond1

SSH command to 192.168.40.2:22 [londond1]
SSH OK : 'ssh cat /etc/hosts' returned:
127.0.1.1    londond1
127.0.0.1    localhost
192.168.40.1 londons
192.168.50.2 parisd1
192.168.40.2 londond1
```

Cluster remote-node SSH Keys

Cluster configuration needs to maintain the SSH Keys for authentication to the ‘remote-node’ in its configuration. Therefore, the SSH Keys fetched from Device nodes and maintained in the cluster remote-node configuration of the Service nodes will need to be the same for *d1* in both clusters. The only way to achieve this is to use the same SSH Keys on both Device nodes, *parisd1* and *londond1*. This can be done by copying the SSH Keys from NCS on one of the Device nodes to the other after installation of NCS on each node (SSH Keys get generated during NCS installation).

Node NSO configuration

Each node has enabled *ha* in *ncs.conf*:

```
<ha>
  <enabled>true</enabled>
</ha>
```

tailf-hcc is properly loaded on each node:

```
# Corresponding CLI command: 'user@ncs> show packages package tailf-hcc'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show packages package tailf-hcc"

Cli command to 192.168.50.1 [pariss]
packages package tailf-hcc
 package-version 4.0
 description     "NED package for Tail-f HA Cluster Control Interface"
 ncs-min-version [ 4.0.1 ]
 ...
 oper-status up

Cli command to 192.168.40.1 [londons]
packages package tailf-hcc
 package-version 4.0
 description     "NED package for Tail-f HA Cluster Control Interface"
 ncs-min-version [ 4.0.1 ]
 ...
 oper-status up
... etc
```

quagga-bgp is properly loaded on the service nodes:

```
# Corresponding CLI command: 'user@ncs> show packages package quagga-bgp'
# issued on each service node
$ nct cli-cmd --hostsfile hostsfile
  -c "show packages package quagga-bgp" --group service

Cli command to 192.168.50.1 [pariss]
packages package quagga-bgp
 package-version 3.3.0
 description     "NED package for Quagga BGP daemon"
 ncs-min-version [ 3.1 3.2 3.3 3.4 4.0 ]
 ...
 oper-status up

Cli command to 192.168.40.1 [londons]
packages package quagga-bgp
 package-version 3.3.0
 description     "NED package for Quagga BGP daemon"
 ncs-min-version [ 3.1 3.2 3.3 3.4 4.0 ]
 ...
 oper-status up
```

Cluster configuration on the Service nodes

The service nodes have configuration to communicate with its remote nodes (the SSH keys for the remote nodes have been fetched):

```
# Corresponding CLI command: 'user@ncs> show configuration cluster'
# issued on each service node
$ nct cli-cmd --hostsfile hostsfile -c "show configuration cluster"
--group service

Cli command to 192.168.50.1 [pariss]
remote-node d1 {
    address   d1;
    port      2022;
    ssh {
        host-key ssh-dss {
            key-data ...;
        }
    }
    authgroup default;
    trace     pretty;
}
authgroup default {
    default-map {
        same-user;
        same-pass;
    }
    umap admin {
        same-user;
        remote-password ...;
    }
    umap hcctest {
        remote-name     admin;
        remote-password ...;
    }
    umap oper {
        same-user;
        remote-password ...;
    }
}
```

Note In the authgroup *default* there is a umap for the user *hcctest*, which is used by tailf-hcc to authenticate towards the remote nodes during monitoring. When a *human* user is invoking an action, the current session user will be used instead.

BGP configuration on the Service nodes

The service nodes have configuration to communicate and configure quagga-devices. As they share configuration, the quagga-device for *pariss* will be present in *londons* as well. Although the quagga-device for *pariss* will be out-of-sync on *londons* , and vice versa. The nodes will be configured with the quagga-devices it should communicate with in its ha-configuration.

The quagga-device configuration for both service nodes:

```
# Corresponding CLI command: 'user@ncs> show configuration devices device'
# issued on each service node
$ nct cli-cmd --hostsfile hostsfile -c "show configuration devices device"
--group service

device quagga_london {
    address   londons;
    port      2605;
    authgroup quagga;
    device-type {
        cli {
            ned-id   quagga-bgp;
            protocol telnet;
        }
    }
    state {
        admin-state unlocked;
    }
    config {
    ...
    }
}
device quagga_paris {
    address   pariss;
    port      2605;
    authgroup quagga;
    device-type {
        cli {
            ned-id   quagga-bgp;
            protocol telnet;
        }
    }
    state {
        admin-state unlocked;
    }
    config {
    ...
    }
}

# Corresponding CLI command: 'user@ncs> show configuration devices authgroups group quagga'
# issued on each service node
$ nct cli-cmd --hostsfile hostsfile -c "show configuration devices authgroups group quagga"
--group service

umap admin {
    remote-name     bgpd;
    remote-password ...;
}
umap hcctest {
    remote-name     bgpd;
    remote-password ...;
}
```

Note In the authgroups group *quagga* there is a umap for the user *hcctest*, which is used by tailf-hcc to authenticate towards the remote nodes during monitoring.

The service nodes will reconfigure its quagga device depending on its current HA state, by applying one of four device templates, which need to be present in the configuration as well:

```
# Corresponding CLI command: 'user@ncs> show configuration devices template'
# issued on 'pariss'
$ nct cli-cmd --hostsfile hostsfile -c "show configuration devices
template" --name pariss

Cli command to 192.168.50.1 [pariss]
template hcc-failover-master {
    config {
        quagga-bgp:hostname FAILOVERMASTER;
        quagga-bgp:route-map SET-MED 10 {
            set {
                metric 10;
            }
        }
    }
}
template hcc-master {
    config {
        quagga-bgp:hostname MASTER;
        quagga-bgp:route-map SET-MED 10 {
            set {
                metric 12;
            }
        }
    }
}
template hcc-none {
    config {
        quagga-bgp:hostname NONE;
        quagga-bgp:route-map SET-MED 10 {
            set {
                metric 20;
            }
        }
    }
}
template hcc-slave {
    config {
        quagga-bgp:hostname SLAVE;
        quagga-bgp:route-map SET-MED 10 {
            set {
                metric 15;
            }
        }
    }
}
```

Note The above templates are very generic and only change the metric value for the device. In reality, these templates will be bigger, and not possible to generalize. The above templates can be generated with the action ha action *create-bgp-templates* to give a starting point.

It is also quite possible that the device nodes need to apply different device-templates for the same state. This can be achieved by adding templates with a hostname suffix, which will then have precedence. For example the *hcc-master-pariss* will have precedence over the template *hcc-master*

HA configuration on the Service nodes

The ha configuration for both service nodes:

```
# Corresponding CLI command: 'user@ncs> show configuration ha'
# issued on each service node
$ nct cli-cmd --hostsfile hostsfile -c "show configuration ha"
--group service

Cli command to 192.168.50.1 [pariss]
token      s;
local-user hcctest;
bgp {
    anycast-prefix   192.168.60.100/32;
    anycast-path-min 3;
    clear-enabled    true;
}
member londons {
    address               192.168.40.1;
    default-ha-role       slave;
    failover-master       true;
    cluster-manager       true;
    quagga-device         quagga_london;
    managed-cluster-nodes [ d1 ];
}
member pariss {
    address               192.168.50.1;
    default-ha-role       master;
    cluster-manager       true;
    quagga-device         quagga_paris;
    managed-cluster-nodes [ d1 ];
}

Cli command to 192.168.40.1 [londons]
<same as for pariss>
```

The local-user *hcctest* will be used for authentication towards both the remote-nodes and quagga-devices. *pariss* is configured to use the device *quagga_paris* and *londons* is configured to use the device *quagga_london*. Both nodes are configured to manage the remote-node *d1*, which will be resolved to *parisd1* on *pariss* and to *londond1* on *londons*.

Configuration on the Device nodes

The device nodes have less complicated configuration as they do not manage other nodes, nor need to communicate with a quagga device:

```
# Corresponding CLI command: 'user@ncs> show configuration ha'
# issued on each device node
$ nct cli-cmd --hostsfile hostsfile -c "show configuration ha"
  --group device
```

```
Cli command to 192.168.50.2 [parisd1]
token d1;
member londond1 {
    address         192.168.40.2;
    default-ha-role slave;
    failover-master true;
}
member parisd1 {
    address         192.168.50.2;
    default-ha-role master;
}

Cli command to 192.168.40.2 [londond1]
<same as for parisd1>
```

Activate HA

Show status before activation

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show ncs-state ha"

Cli command to 192.168.50.1 [pariss]
ncs-state ha mode none

Cli command to 192.168.40.1 [londons]
ncs-state ha mode none

Cli command to 192.168.50.2 [parisd1]
ncs-state ha mode none

Cli command to 192.168.40.2 [londond1]
ncs-state ha mode none
```

When not activated, the *none*-template have been applied to the quagga devices. Verify that the hostname is NONE:

```
$ telnet 192.168.50.1 2605 #pariss quagga
...
NONE> exit                 #<-- Hostname NONE
Connection closed by foreign host.

$ telnet 192.168.40.1 2605 #londons quagga
...
NONE> exit                 #<-- Hostname NONE
Connection closed by foreign host.
```

Activate HA

Activate HA by invoking the action *activate* First on the **PARIS** cluster, then on the **LONDON** cluster:

```
# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on each node in *PARIS* cluster
$ nct ha --hostsfile hostsfile --action activate --group paris

HA Node 192.168.50.1:8080 [pariss]
activated

HA Node 192.168.50.2:8080 [parisd1]
activated
```

```
# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on node each in *LONDON* cluster
$ nct ha --hostsfile hostsfile --action activate --group london

HA Node 192.168.40.1:8080 [londons]
activated

HA Node 192.168.40.2:8080 [londond1]
activated
```

Show status after activation

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show ncs-state ha"

Cli command to 192.168.50.1 [pariss]
ncs-state ha mode master
ncs-state ha node-id pariss
ncs-state ha connected-slave [ londons ]

Cli command to 192.168.40.1 [londons]
ncs-state ha mode slave
ncs-state ha node-id londons
ncs-state ha master-node-id pariss

Cli command to 192.168.50.2 [parisd1]
ncs-state ha mode master
ncs-state ha node-id parisd1
ncs-state ha connected-slave [ londond1 ]

Cli command to 192.168.40.2 [londond1]
ncs-state ha mode slave
ncs-state ha node-id londond1
ncs-state ha master-node-id parisd1
```

Check Quagga

When activated, the *master*-template has been applied to the quagga_paris device, and the *slave*-template has been applied to the quagga_london device:

```
$ telnet 192.168.50.1 2605 #pariss quagga
...
MASTER> exit                #<-- Hostname MASTER
Connection closed by foreign host.

$ telnet 192.168.40.1 2605 #londons quagga
...
SLAVE> exit                 #<-- Hostname MASTER
Connection closed by foreign host.
```

HA Failover

Simulate failover

Simulate that the node *pariss* is down by deactivating it:

```
# Corresponding CLI command: 'user@ncs> request ha commands deactivate'
# issued on node 'pariss'
$ nct ha --hostsfile hostsfile --action deactivate --name pariss

HA Node 192.168.50.1:8080 [pariss]
```

```
deactivated
```

Check failover

Check that the failover cluster transitioned to master

Note Failover will take (interval*failure limit) seconds before the failover master confirms loss of *master* and initiates a failover. If the Master-Slave communication is re-established during this period, HA operation would continue as normal

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show ncs-state ha"

Cli command to 192.168.50.1 [pariss]
ncs-state ha mode none

Cli command to 192.168.40.1 [londons]
ncs-state ha mode master
ncs-state ha node-id londons

Cli command to 192.168.50.2 [parisd1]
ncs-state ha mode master
ncs-state ha node-id parisd1

Cli command to 192.168.40.2 [londond1]
ncs-state ha mode master
ncs-state ha node-id londond1
```

londons noticed that it lost connectivity with its master, and transitioned to master. It also instructed its remote-node *londond1* to transition to master.

Note *parisd1* is unaware that *pariss* is down, and will not automatically change its HA state.

Check Quagga

During a failover, the *failover-master*-template has been applied to the quagga_london device, and since this was a *controlled* failure of *pariss*, the *none*-template has been applied to the quagga_paris device:

```
$ telnet 192.168.50.1 2605 #pariss quagga
...
NONE> exit                 #<-- Hostname NONE
Connection closed by foreign host.

$ telnet 192.168.40.1 2605 #londons quagga
...
FAILOVERMASTER> exit       #<-- Hostname FAILOVERMASTER
Connection closed by foreign host.
```

HA Fail-back to original configuration

After rectifying the cause of the Master-slave communication failure, the approach is to bring the original Master cluster back online initially as a Slave, which will then connect to the new Master to complete a CDB re-sync prior to being transitioned to the operational Master.

There are no provisions in the tailf-hcc application to automatically sense and initiate a revert back to the original HA Cluster configuration.

Note It is **very** important that the original master cluster is setup to be Slave to the failover master cluster before activation after a fail-over. If not, potential configuration loss may occur.

Override to slave

Override the ha role of the cluster nodes in **PARIS** to slave then activate:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-override role slave'
# issued on each node in *PARIS* cluster
$ nct ha --hostsfile hostsfile --action role-override --role slave
--group paris

HA Node 192.168.50.1:8080 [pariss]
override

HA Node 192.168.50.2:8080 [parisd1]
override

# Corresponding CLI command: 'user@ncs> request ha commands activate'
# issued on each node in *PARIS* cluster
$ nct ha --hostsfile hostsfile --action activate --group paris

HA Node 192.168.50.1:8080 [pariss]
activated

HA Node 192.168.50.2:8080 [parisd1]
activated
```

Verify override

Verify that cluster **PARIS** now is slave to cluster **LONDON**

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show ncs-state ha"

Cli command to 192.168.50.1 [pariss]
ncs-state ha mode slave
ncs-state ha node-id pariss
ncs-state ha master-node-id londons

Cli command to 192.168.40.1 [londons]
ncs-state ha mode master
ncs-state ha node-id londons
ncs-state ha connected-slave [ pariss ]

Cli command to 192.168.50.2 [parisd1]
ncs-state ha mode slave
ncs-state ha node-id parisd1
ncs-state ha master-node-id londond1

Cli command to 192.168.40.2 [londond1]
ncs-state ha mode master
ncs-state ha node-id londond1
ncs-state ha connected-slave [ parisd1 ]
```

Check Quagga

During a role-override, the *slave*-template has been applied to the quagga_paris device:

```
$ telnet 192.168.50.1 2605 #pariss quagga
...
SLAVE> exit                #<-- Hostname SLAVE
Connection closed by foreign host.

$ telnet 192.168.40.1 2605 #londons quagga
...
FAILOVERMASTER> exit       #<-- Hostname FAILOVERMASTER
Connection closed by foreign host.
```

Role-revert nodes

Role-revert all nodes

Note Allow adequate time for CDB on re-activated *paris_m* to sync with CDB on the failed-over Master. The time depends on the CDB size and how much of the CDB needs to be updated.

First role-revert cluster **PARIS** then cluster **LONDON**:

```
# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on each node in *PARIS* cluster
$ nct ha --hostsfile hostsfile --action role-revert --group paris

HA Node 192.168.50.1:8080 [pariss]
reverted

HA Node 192.168.50.2:8080 [parisd1]
reverted

# Corresponding CLI command: 'user@ncs> request ha commands role-revert'
# issued on each node in *LONDON* cluster
$ nct ha --hostsfile hostsfile --action role-revert --group london

HA Node 192.168.40.1:8080 [londons]
reverted

HA Node 192.168.40.2:8080 [londond1]
reverted
```

Verify role-revert

Verify cluster **PARIS** is master once again

```
# Corresponding CLI command: 'user@ncs> show ncs-state ha'
# issued on each node
$ nct cli-cmd --hostsfile hostsfile -c "show ncs-state ha"

Cli command to 192.168.50.1 [pariss]
ncs-state ha mode master
ncs-state ha node-id pariss
ncs-state ha connected-slave [ londons ]

Cli command to 192.168.40.1 [londons]
ncs-state ha mode slave
ncs-state ha node-id londons
ncs-state ha master-node-id pariss

Cli command to 192.168.50.2 [parisd1]
ncs-state ha mode master
ncs-state ha node-id parisd1
```

```
ncs-state ha connected-slave [ londond1 ]

Cli command to 192.168.40.2 [londond1]
ncs-state ha mode slave
ncs-state ha node-id londond1
ncs-state ha master-node-id parisd1
```

Check Quagga

After role-revert, the *master*-template has been applied to the quagga_paris device, and the *slave*-template has been applied to the quagga_london device:

```
$ telnet 192.168.50.1 2605 #pariss quagga
...
MASTER> exit                #<-- Hostname MASTER
Connection closed by foreign host.

$ telnet 192.168.40.1 2605 #londons quagga
...
SLAVE> exit                 #<-- Hostname SLAVE
Connection closed by foreign host.
```

FAQ and known issues

FAQ

1. After a failover, the new Master cannot connect to its devices, with a authentication failure

As the Master and Slave(s) share the same configuration stored in CDB, they also share the encrypted passwords for authentication. Therefore, the encryption keys **must** be the same on the master and slave nodes in ncs.conf

2. After activating tailf-hcc, nothing happens, the node(s) does not transition to its configured HA-role

Most likely there is a hostname <→ ha member-name mismatch. tailf-hcc will use the node hostname to identify which member instance configuration to apply. It is therefore **required** that the hostname and member's name is the same

3. Which log files are of interest?

The package is mainly written in Java, so when something goes wrong, start by looking in the `ncs-java-vm.log` in the log-directory for ncs (default under `/var/log/ncs`). If you are using the VIP-functionality, you also need to look at the `devel.log`, as the VIP code is written in erlang and uses the devel.log for logging.

4. Any other debugging tips?

NCS Master and Slave communicates over TCP. The port used is configured in ncs.conf, and **must** be the same on the Master and the Slave(s). Make sure that no firewall etc is blocking this port, with your flavour of *iptables* command, for example, and you can see which ports the host is listening to with *netstat -anp | grep tcp*

Issues

Issues with HA in Openstack VM environment

- Openstack VMs have a VM-level psuedo-firewall capability called *Security Groups* that needs to allow the appropriate protocols/ports.
- Issues with unidirectional master-slave traffic - return packet lost in Openstack infrastructure. Customers have seen, with *tcp dump*, that the Slave sends a SYN as soon as the master goes

down, Master sends back a RST,ACK but the Slave never gets that last packet, so the Slave keeps waiting until the timeout*retries expires. This could be caused by the Neutron bug (https://bugs.launchpad.net/neutron/+bug/1460741), but this is unconfirmed.

CHAPTER 4

The NSO Tailf HCC Models

NSO Tailf HCC model

Example 10. NSO Tailf HCC YANG Model

```
module tailf-hcc {
  namespace "http://tail-f.com/pkg/tailf-hcc";
  prefix hcc;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }
  include tailf-hcc-typedefs {
    revision-date "2015-09-14";
  }
  include tailf-hcc-alarms {
    revision-date "2015-09-14";
  }
  include tailf-hcc-actions {
    revision-date "2015-09-14";
  }

  organization "Tail-f Systems";
  description
    "This module contains a collection of YANG definitions for
     configuring and monitoring the NCS high availability
     API. This yang file is used by the Tail-f Cluster Manager (tailf-hcc)
     package.";

  revision 2017-12-22 {
    description
    "Support of multiple VIP";
}

revision 2017-05-05 {
  description
     "Add the netmask leaf to the VIP configuration";
}

revision 2016-03-10 {
  description
     "Removed the operational leaf 'current-ha-role', as the ncs operational
      leaf '/ncs-state/ha' contains the same information";
}

revision 2015-09-14 {
  description
    "Initial revision.";
}

grouping vip-content {
  leaf address {
    mandatory true;
    tailf:info "The Virtual IP address to bring up on a labeled interface";
    type inet:ip-address;
  }
  leaf netmask {
    tailf:info "The netmask used for the labeled interface";
    type inet:ip-address;
    description
      "The netmask used for the brought up VIP. If it is not present
       in the configuration, the netmask will be the same as the one
       from the underlying interface.";
  }
  leaf broadcast-address {
    tailf:info "The Broadcast address used for the labeled interface";
    type inet:ip-address;
    description
      "The broadcast address used for the brought up VIP. If it is not present
       in the configuration, the broadcast address will be calculated from the
       address and netmask (configured or taken from the underlying interface).";
  }
}

augment "/hcc:ha" {
  uses local-actions;
}
augment "/ncs:cluster/ncs:remote-node" {
  uses remote-actions;
}
container ha {
  presence "Enable hcc ha";
  tailf:info "High availability cluster configuration";
  leaf token {
    tailf:info "A shared secret within an HA group.";
    type string;
    mandatory true;
    description
      "This value is used as the shared secret when setting
        up HA between nodes. All nodes within the same HA group
        (e.g a master-node and slave(s)) must share this token,
        and the token must be unique";
  }
```

```
leaf local-user {
  when "../member/cluster-manager = 'true' or ../bgp";
  tailf:info "local-user used for remote device and/or quagga device authentication";
  type string;
  description
    "The local user used by tailf-hcc to issue actions on remote nodes
     and/or configure the quagga device. It must match a local user under
     /cluster/authgroup/umap and/or /devices/authgroups/group/umap";
}

leaf interval {
  tailf:info "Interval on which HA state is checked (seconds)";
  type int32 {
    range "1..60";
  }
  default "4";
  description
    "Interval on which a (slave) node check its HA status.
     The node will check if it has connection with its master.
     If the node is a cluster-manager, it will also check the HA-status
     of its managed remote-nodes. If the node is a cluster-manager and BGP
     is enabled, it will also check the anycast prefix path length on the
     BGP device.";
}

leaf failure-limit {
  tailf:info "Number of failed HA state check interval(s) before declaring a failure";
  type int32 {
    range "1..100";
  }
  default "10";
  description
    "This is the number of times a slave node notice that it, or its
     managed remote-nodes, lost HA connection with its master. When the limit is hit,
     the node will declare a failure and a failover-process will initiate.";
}
choice l2orl3 {
  case l3 {
    container bgp {
      must '/ncs:devices/ncs:template[starts-with(., "hcc-master")]' {
        error-message "You must have a device template starting with the name 'hcc-master
      }
      must '/ncs:devices/ncs:template[starts-with(., "hcc-failover-master")]' {
        error-message "You must have a device template starting with the name 'hcc-failov
      }
      must '/ncs:devices/ncs:template[starts-with(., "hcc-slave")]' {
        error-message "You must have a device template starting with the name 'hcc-slave'
      }
      must '/ncs:devices/ncs:template[starts-with(., "hcc-none")]' {
        error-message "You must have a device template starting with the name 'hcc-none'"
      }

      must " ../local-user" {
        error-message "You must specify a local-user for quagga device authentication";
      }
      tailf:info "BGP anycast failover configuration";
      presence "Enable BGP";
      description
        "The configuration needed for BGP monitoring and configuring. The node
         will apply different device templates depending on which HA state it is in.
         The (slave) node will monitor a configured anycast prefix paths,
           and when the number of paths fall below a set minimum, the node will
           start the failover process, which include applying a failover device template.";
        leaf failure-limit {
          tailf:info "Number of failed anycast path state check before declaring a failure";
          type int32 {
            range "1..100";
          }
          default "10";
          description
            "This is the number of times a slave node notice that the prefix anycast
             path is lower than the configured minimum.
             When the limit is hit, the node will declare a failure and
             a failover-process will initiate.";
        }
        leaf anycast-prefix {
          tailf:info "BGP anycast prefix to monitor";
          type inet:ip-prefix;
          mandatory true;
        }
        leaf anycast-path-min {
          tailf:info "Minimum number of valid anycast paths before the declaring loss of conne
          type int32;
          default "3";
          description
            "The minimum number of valid anycast paths. When below this number
             more times than the allowed failure limit, configured under
             /ha/bgp/failure-limit, a loss of connectivity to the master
             node is declared, and a fail over is initiated";
        }
        leaf clear-enabled {
          tailf:info "If set, all BGP sessions and routes are cleared each time a new configur
          type boolean;
          default "false";
          description
            "In the case of applying configuration towards a quagga device, should the node
             also clear BGP sessions and routes on the quagga";
        }
      }
    }
    case l2 {
      choice vip-ordinal-choice {
        case vip-single {
          container vip {
            status deprecated;
            presence "Enable VIP";
            tailf:info "VIP failover configuration (deprecated)";
            description
              "If enabled, a VIP with the label $interface:ncsvip will be
               added for the current master node with the address specified.";

            uses vip-content;
          }
        }
        case vip-multiple {
          container vips {
            tailf:info "VIPs failover configuration";
            list vip {
              key "address";
              description
                "If list contains VIPs, then with the label $interface:ncsvip they will be
                 added for the current master node with the address specified.";
            uses vip-content;
          }
        }
      }
    }
  }
}
list member {
  must "count(../member[default-ha-role = 'master']) <= 1" {
    error-message "At most one node can serve as a master";
  }
  must "count(../member[failover-master = 'true']) <= 1" {
    error-message "At most one node can serve as a failover master";
  }
  must "not(./cluster-manager = 'true') or ../local-user" {
    error-message "You must specify a local-user for remote node authentication";
  }
  must "not(../bgp) or ./quagga-device" {
    error-message "You must specify a device used for BGP monitoring";
  }
  must "not(../vip) or ./vip-interface" {
    error-message "You must specify a vip interface when vip is enabled";
  }
  must "not(../vips/vip) or (count(./vip-interfaces/vip-interface) = count(../vips/vip))"
    error-message "You must specify a vip interface for each vip enabled";
  }
  key name;
  unique "address";
  max-elements 64;
  tailf:info "HA Cluster member configuration";
  description
    "This table is the cluster member nodes";
  leaf name {
    tailf:info "The name of the member. Must be the same as the member's hostname";
    type string;
  }
  leaf address {
    tailf:info "Ip Address of the ncs instance";
    type inet:ip-address;
    mandatory true;
  }
  leaf default-ha-role {
    tailf:info "The preferred HA role for this member";
    type cluster-role-type;
    default none;
    description
      "This is the HA role the node should have when everything is
       working as it should. This is also the role the node will try
       to take when given the action 'role-revert'";
  }
  leaf failover-master {
    tailf:info "HA node will assume role of HA master when default HA master is down";
    when "../default-ha-role = 'slave'";
    type boolean;
    default "false";
  }
  leaf relay-name {
    type leafref {
      path "/hcc:ha/hcc:member/hcc:name";
    }
    tailf:info "This member's relay-node";
    description "The name of the node that should act as a relay
```

```
                      between this node and the master";
          }
          leaf cluster-manager {
            type boolean;
            default "false";
            tailf:info "A cluster manager will manage HA within its NCS cluster";
            description
              "A cluster node acting as Cluster manager
               will monitor and manage HA for all its remote nodes
               specified under /ha/member/managed-cluster-nodes";
          }
          leaf-list managed-cluster-nodes {
            when "../cluster-manager = 'true'";
            type leafref {
              path "/ncs:cluster/ncs:remote-node/ncs:name";
            }
            tailf:info "Remote nodes in the cluster managed by the cluster-manager";
            description
              "The remote nodes for which the cluster-manager should monitor HA status,
               and manage HA roles depending on changes in the HA network.";
          }
          leaf quagga-device {
            when "../../bgp";
            type leafref {
              path "/ncs:devices/ncs:device/ncs:name";
            }
            tailf:info "Device used for BGP anycast monitoring and managing";
          }
          choice vip-ordinal-choice {
            case vip-multiple {
              container vip-interfaces {
                list vip-interface {
                  key "vip-interface address";
                  when "../../../hcc:vips/vip";
                  tailf:info "The interfaces used for the vips";

                  leaf vip-interface {
                    type string;
                  }

                  leaf address {
                    type leafref {
                      path "../../../../hcc:vips/hcc:vip/hcc:address";
                    }
                  }
                }
              }
            }
            case vip-single {
              leaf vip-interface {
                status deprecated;
                when "../../vip";
                tailf:info "The interface used for the vip (deprecated)";
                type string;
              }
            }
          }
        }
      }
    }
```

NSO Tailf HCC typedefs model

Example 11. NSO Tailf HCC notif typedefs Model

```
submodule tailf-hcc-typedefs {
  belongs-to tailf-hcc {
    prefix hcc;
  }

  revision "2015-09-14" {
    description "Initial revision";
  }

  typedef cluster-role-type {
    type enumeration {
      enum unknown {value 0;}
      enum none {value 1;}
      enum slave {value 2;}
      enum master {value 3;}
      enum relay {value 4;}
    }
  }
}
```

NSO Tailf HCC Actions model

Example 12. NSO Tailf HCC actions YANG Model

```
submodule tailf-hcc-actions {
  belongs-to tailf-hcc {
    prefix hcc;
  }

  import tailf-common {
    prefix tailf;
  }

  import tailf-ncs {
    prefix ncs;
  }

  include tailf-hcc-typedefs;

  description
    "This submodule contains all tailf:actions used for managing the ha.";

  revision "2015-09-14" {
    description "Initial revision";
  }

  //Actions used both locally and remotely.
  grouping shared-actions {
    tailf:action activate {
      tailf:actionpoint hcc-action-point;
      tailf:info "Activate HA";
      output {
        leaf status {
          type string;
        }
      }
      description
```

```
        "The node will be activated and take the HA role configured
         under /ha/member/default-ha-role";
    }

    tailf:action deactivate {
      tailf:actionpoint hcc-action-point;
      tailf:info "deactivate HA";
      output {
        leaf status {
          type string;
        }
      }
      description
        "The node will be deactivated, and the HA role 'none'";
    }

    tailf:action role-override {
      tailf:actionpoint hcc-action-point;
      tailf:info "Override the current HA role";
      input {
        leaf role {
          type cluster-role-type;
        }
      }
      output {
        leaf status {
          type string;
        }
      }
    }

    tailf:action role-revert {
      tailf:actionpoint hcc-action-point;
      tailf:info "Revert the HA role to the default-ha-role value";
      output {
        leaf status {
          type string;
        }
      }
    }

    tailf:action status {
      tailf:actionpoint hcc-action-point;
      tailf:info "Retrieve node HA status";
      output {
        leaf status {
          type string;
        }
      }
    }

    tailf:action connect-state {
      tailf:actionpoint hcc-action-point;
      tailf:hidden "hcc-remote-actions";
      tailf:info "Retrieve node connection state on a remote node";
      output {
        leaf status {
          type string;
        }
      }
    }
  }
```

```
grouping cluster-wide-actions {
  tailf:action cluster-status {
    tailf:actionpoint hcc-action-point;
    tailf:info "Retrieve cluster wide HA status";
    output {
      leaf status {
        type string;
      }
    }
  }

  tailf:action cluster-activate {
    tailf:actionpoint hcc-action-point;
    tailf:info "Issue activation command to all nodes in the cluster";
    output {
      leaf status {
        type string;
      }
    }
  }

  tailf:action cluster-deactivate {
    tailf:actionpoint hcc-action-point;
    tailf:info "Stop all HA activity on all nodes";
    output {
      leaf status {
        type string;
      }
    }
  }

  tailf:action cluster-role-revert {
    tailf:actionpoint hcc-action-point;
    tailf:info "Revert the HA role to the default-ha-role value on all nodes";
    output {
      leaf status {
        type string;
      }
    }
  }

  tailf:action cluster-role-override {
    tailf:actionpoint hcc-action-point;
    tailf:info "Revert the HA role to the default-ha-role value on all nodes";
    input {
      leaf role {
        type cluster-role-type;
      }
    }
    output {
      leaf status {
        type string;
      }
    }
  }

  tailf:action cluster-connect-state {
    tailf:actionpoint hcc-action-point;
    tailf:hidden "hcc-remote-actions";
    tailf:info "Retrieve cluster wide HA connection state";
    output {
          leaf status {
            type string;
          }
        }
      }
    }

    //Local actions
    grouping local-actions {
      container commands {
        config false;
        tailf:info "Modify HA behavior modifying commands/actions";

        uses shared-actions;
        uses cluster-wide-actions;

        tailf:action force-be-slave-to {
          tailf:actionpoint hcc-action-point;
          tailf:info "Set the node in slave-mode and try to connect to a set master";
          input {
            leaf member {
              type leafref {
                path "/hcc:ha/hcc:member/hcc:name";
              }
            }
          }
          output {
            leaf status {
              type string;
            }
          }
        }

        tailf:action create-bgp-templates {
          tailf:actionpoint hcc-action-point;
          tailf:info "Creates example device templates for BGP";
          output {
            leaf status {
              type string;
            }
          }
        }

        tailf:action readonly {
          tailf:actionpoint hcc-action-point;
          tailf:info "Configure the node to be readonly mode";
          input {
            leaf mode {
              type boolean;
              default false;
            }
          }
          output {
            leaf status {
              type string;
            }
          }
        }

        tailf:action reactivate {
          tailf:actionpoint hcc-action-point;
          tailf:info "Issue reactivation command to node";
        output {
          leaf status {
            type string;
          }
        }
      }
    }
  }

  //Remote actions
  grouping remote-actions {
    container commands {
      tailf:info "Modify HA behavior modifying commands/actions";
      tailf:hidden "hcc-remote-actions";
      uses shared-actions {
        refine activate {
          tailf:actionpoint hcc-rm-ap;
        }

        refine deactivate {
          tailf:actionpoint hcc-rm-ap;
        }

        refine role-override {
          tailf:actionpoint hcc-rm-ap;
        }

        refine role-revert {
          tailf:actionpoint hcc-rm-ap;
        }

        refine status {
          tailf:actionpoint hcc-rm-ap;
        }

        refine connect-state {
          tailf:actionpoint hcc-rm-ap;
        }
      }
    }
  }
}
```

NSO Tailf HCC Alarms model

Example 13. NSO Tailf HCC alarms YANG Model

```
submodule tailf-hcc-alarms {
  belongs-to tailf-hcc {
    prefix hcc;
  }

  import tailf-ncs-alarms {
    prefix al;
  }

  organization "Tail-f Systems";

  revision "2015-09-14" {
    description "Initial revision";
  }
```

```
    identity hcc-alarm {
      base al:alarm-type;
      description "Alarms raised by the tailf-hcc package.";
    }

    identity node-failure {
      base hcc-alarm;
      description
        "The node lost HA connection with its master";
    }

    identity device-node-failure {
      base hcc-alarm;
      description
        "A service node noticed one of its device nodes lost
         HA connection with its master";
    }

    identity bgp-failure {
      base hcc-alarm;
      description
        "A service node noticed it lost its BGP prefix path to
         its master";
    }
  }
```

CHAPTER 5

Resources

References for further reading

NSO Packages chapter in NSO 4.7 Administration Guide.

The AAA Infrastructure chapter in NSO 4.7 Administration Guide.