NSO Deployment

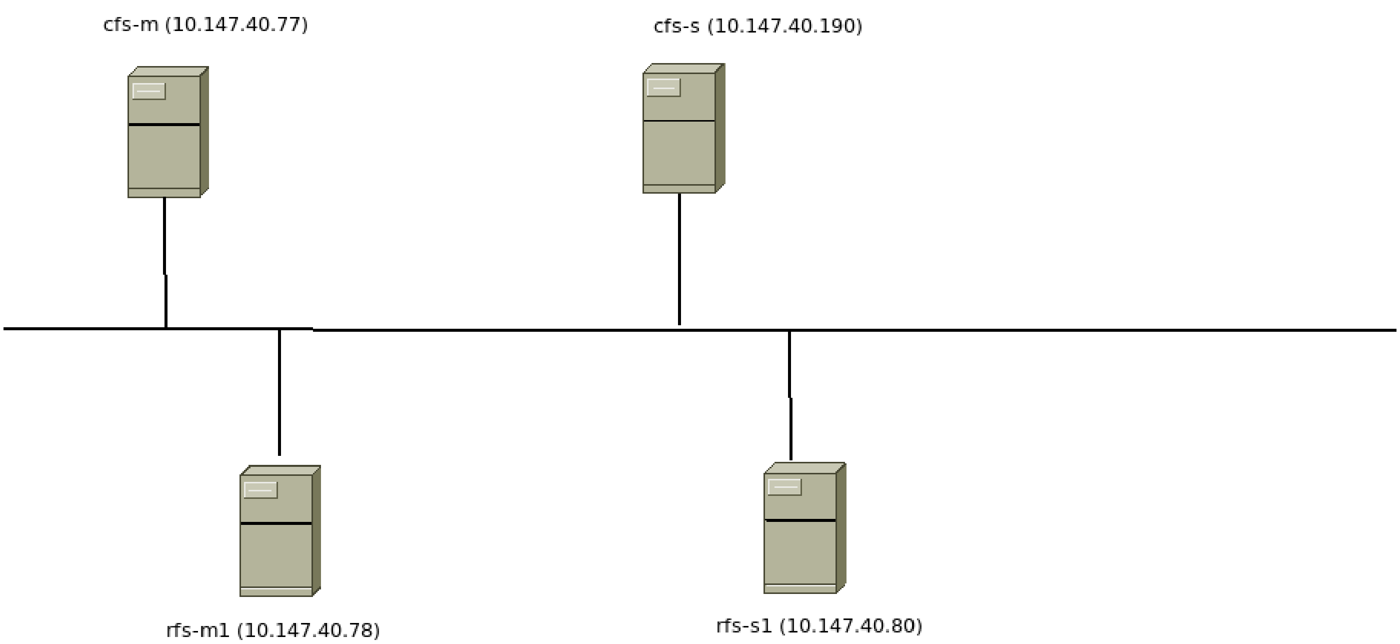
Introduction

This article is written as a series of examples. We'll be describing a typical large scale deployment where the following topics will be covered:

- Installation of NSO on all hosts
- Initial configuration of NSO on all hosts
- Upgrade of NSO on all hosts
- Upgrade of NSO packages/NEDs on all hosts
- Monitoring the installation
- Trouble shooting, backups and disaster recovery
- Security considerations

We'll be using a Layered Service Architecture cluster as an example deployment. The deployment consists of four hosts, a CFS node pair and a RFS node pair. The two CFS nodes are an NSO HA-pair, and so are the two RFS nodes. The following picture shows our deployment.

Figure 1. The deployment network



Thus the two NSO hosts cfs-m and cfs-s make up one HA pair, one active and one standby, and similarly for the two so called RFS nodes, rfs-m1 and rfs-s1. The HA setup as well as the cluster setup will be thoroughly described later in this article.

The cluster part is really optional, it's only needed if you are expecting the amount of managed devices and/or the amount of instantiated services to be so large so that it doesn't fit in memory on a single NSO host. If for example the expected number of managed devices and services is less than 20k, it's recommended to *not* use clustering at all and instead equip the NSO hosts with sufficient RAM. Installation, performance, bug search, observation, maintenance, all become harder with clustering turned on.

HA on the other hand is usually not optional for customer deployment. Data resides in CDB, which is a RAM database with a disk based journal for persistence. One possibility to run NSO without the HA component could be to use a fault tolerant filesystem, such as CEPH. This would mean that provisioning data survives a disk crash on the NSO host, but fail over would require manual intervention. As we shall see, the HA component we have *tailf-hcc* also requires some manual intervention, but only after an automatic fail over.

In this article we shall describe a complete HA/cluster setup though, you will have to decide for your deployment weather HA and/or clustering is required.

Initial NSO installation

We will perform an NSO system installation on 4 NSO hosts. NSO comes with a tool called `nct` which is ideal for the task at hand here. `nct` has it's own documentation and will not be described here. nct is shipped together with NSO. The following prerequisites are needed for the `nct` operations.

- We need a user on the management station which has sudo rights on the four NSO hosts. Most of the `nct` operations that you'll execute towards the 4 NSO hosts require root privileges.
- Access to the NSO .bin install package as well as access to The NEDs and all packages you're planning to run. The packages shall be on the form of tar.gz packages.

We'll be needing an `NCT_HOSTSFILE`. In this example it looks like:

```
$ echo $NCT_HOSTFILE
/home/klacke/nct-hosts

$ cat $NCT_HOSTFILE

{"10.147.40.80", [{name, "rfs-s1"},
                  {rest_pass, "MYPASS"},
                  {ssh_pass, "MYPASS"},
                  {netconf_pass, "MYPASS"},
                  {rest_port, 8888}]}.

{"10.147.40.78", [{name, "rfs-m"},
                  {rest_pass, "MYPASS"},
                  {ssh_pass, "MYPASS"},
                  {netconf_pass, "MYPASS"},
                  {rest_port, 8888}]}.

{"10.147.40.190", [{name, "cfs-m"},
                   {rest_pass, "MYPASS"},
```



```
                  {ssh_pass, "MYPASS"},
                  {netconf_pass, "MYPASS"},
                  {rest_port, 8888}]}.

{"10.147.40.77",  [{name, "cfs-s"},
                  {rest_pass, "MYPASS"},
                  {ssh_pass, "MYPASS"},
                  {netconf_pass, "MYPASS"},
                  {rest_port, 8888}]}.

$ ls -lat /home/klacke/nct-hosts
-rw-------  1 klacke  staff  1015 Jan 22 13:12 /home/klacke/nct-hosts
```

The different passwords in `nct-hosts` file are all my regular Linux password on the target host.

We can use SSH keys, especially for normal SSH shell login, however, unfortunately, the `nct` tool doesn't work well with *ssh-agent*, thus the keys shouldn't have a pass phrase, if they do, we'll have to enter the pass phrase over and over again while using `nct`. Since *ssh-agent* doesn't work, and we'll be needing the password for the REST api access anyway, the recommended setup is to store the password for the target hosts in a read-only file. This is for ad-hoc `nct` usage.

This data is needed on the management station. That can be one of the 4 NSO hosts, but it can also be another host, e.g an operator laptop. One convenient way to get easy access to the `nct` command is to do a "local install" of NSO on the management station. To test nct, and the SSH key setup you can do:

```
$ nct ssh-cmd -c 'sudo id'

SSH command to 10.147.40.80:22 [rfs-s1]
SSH OK : 'ssh sudo id' returned: uid=0(root) gid=0(root) groups=0(root)

SSH command to 10.147.40.78:22 [rfs-m1]
SSH OK : 'ssh sudo id' returned: uid=0(root) gid=0(root) groups=0(root)

SSH command to 10.147.40.190:22 [cfs-m]
SSH OK : 'ssh sudo id' returned: uid=0(root) gid=0(root) groups=0(root)

SSH command to 10.147.40.77:22 [cfs-s]
SSH OK : 'ssh sudo id' returned: uid=0(root) gid=0(root) groups=0(root)
```

Now you are ready to execute the NSO installer on all the 4 NSO hosts. This is done through the `nct` command `install`.

```
$ nct install --file ./nso-4.1.linux.x86_64.installer.bin --progress true
............
Install NCS to 10.147.40.80:22
Installation started, see : /tmp/nso-4.1.linux.x86_64.installer.bin.log

Install NCS to 10.147.40.78:22
Installation started, see : /tmp/nso-4.1.linux.x86_64.installer.bin.log

Install NCS to 10.147.40.190:22
Installation started, see : /tmp/nso-4.1.linux.x86_64.installer.bin.log

Install NCS to 10.147.40.77:22
Installation started, see : /tmp/nso-4.1.linux.x86_64.installer.bin.log
```


If you for some reason want to undo everything and start over from scratch, the following command cleans up everything on all the NSO hosts.

```
$ nct ssh-cmd \
-c 'sudo /opt/ncs/current/bin/ncs-uninstall --non-interactive --all'
```

At this point NSO is properly installed on the NSO hosts. The default options were used for the NSO installer, thus files end up in the normal places on the NSO hosts. We have:

- Boot files in `/etc/init.d`, NSO configuration files in `/etc/ncs` and shell files under `/etc/profile.d`
- NSO run dir, with CDB database, packages directory, NSO state directory in `/var/opt/ncs`
- Log files in `/var/log/ncs`
- The releases structure in `/opt/ncs` with man pages for all NSO related commands under `/opt/ncs/current/man`

To read more about this, see man page ncs-installer(1)

Initial NSO configuration - ncs.conf

After installation the configuration needs to be updated to be in sync on all 4 hosts. The configuration file `/etc/ncs/ncs.conf` should be identical on all hosts. Note that the configuration for encrypted strings is generated during installation. The keys are stored in the file `/etc/ncs/ncs.crypto_keys` and should be copied from one of the hosts to the remaining three.

The required services and authentication needs to be configured taking security requirements into account. It is recommended to use PAM for authenticating users although it is possible to have users in NSO CDB database.

To keep configuration in sync between the hosts, copy `/etc/ncs/ncs.conf` and `/etc/ncs/ncs.crypto_keys` from one of the hosts to a management station and edit it there. See NSO man page `ncs.conf(1)` for all the settings of `ncs.conf`

- Enable the NSO ssh CLI login. `/ncs-config/cli/ssh/enabled`
- Modify the CLI prompt so that the hostname is part of the CLI prompt. `/ncs-config/cli/prompt`

```
<prompt1>\u@\H> </prompt1>
<prompt2>\u@\H% </prompt2>

<c-prompt1>\u@\H# </c-prompt1>
<c-prompt2>\u@\H(\m)# </c-prompt2>
```

- Enable the NSO https interface `/ncs-config/webui/`

  The SSL certificates that get distributed with NSO are self signed.

```
$ openssl x509 -in /etc/ncs/ssl/cert/host.cert -text -noout
Certificate:
Data:
Version: 1 (0x0)
Serial Number: 2 (0x2)
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=US, ST=California, O=Internet Widgits Pty Ltd, CN=John Smith
Validity
Not Before: Dec 18 11:17:50 2015 GMT
```

```
Not After : Dec 15 11:17:50 2025 GMT
Subject: C=US, ST=California, O=Internet Widgits Pty Ltd
Subject Public Key Info:
.......
```

Thus, if this is a real production environment, and the Web/REST interface is used for something which is not solely internal purposes it's a good idea to replace the self signed certificate with a properly signed certificate.

- Disable `/ncs-config/webui/cgi` unless needed.
- Enable the NSO netconf SSH interface `/ncs-config/netconf-northbound/`
- Enable the NSO ha in `ncs.conf`.

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

- PAM - the recommended authentication setting for NSO is to rely on Linux PAM. Thus all remote access to NSO must be done using real host privileges. Depending on your Linux distro, you may have to change `/ncs-config/aaa/pam/service`. The default value is `common-auth`. Check the file `/etc/pam.d/common-auth` and make sure it fits your needs.
- Depending on the type of provisioning applications you have, you might want to turn `/ncs-config/rollback/enabled` off. Rollbacks don't work that well with reactive-fastmap applications. If your application is a classical NSO provisioning application, the recommendation is to enable rollbacks, otherwise not.

Now that you have a proper `ncs.conf` - the same config files can be used on all the 4 NSO hosts, we can copy the modified file to all hosts. To do this we use the `nct` command:

```
$ nct copy --file ncs.conf
$ nct ssh-cmd -c 'sudo mv /tmp/ncs.conf /etc/ncs'
$ nct ssh-cmd -c 'sudo chmod 600 /etc/ncs/ncs.conf'
```

Or use the builtin support for the ncs.conf file:

```
$ nct load-config --file ncs.conf --type ncs-conf
```

The ncs.crypto_keys file must also be copied if the standard encrypted-strings configuration is used:

```
$ nct copy --file ncs.crypto_keys
$ nct ssh-cmd -c 'sudo mv /tmp/ncs.crypto_keys /etc/ncs'
$ nct ssh-cmd -c 'sudo chmod 400 /etc/ncs/ncs.crypto_keys'
```

Note that the `ncs.crypto_keys` is highly sensitive. The file contains the encryption keys for all CDB data that is encrypted on disk. This usually contains passwords etc for various entities, such as login credentials to managed devices. In YANG parlance, this is all YANG data modeled with the types `tailf:des3-cbc-encrypted-string`, `tailf:aes-cfb-128-encrypted-string` or `tailf:aes-256-cfb-128-encrypted-string`

Setting up AAA

As we saw in the previous section, the REST HTTPS api is enabled. This API is used by a few of the crucial `nct` commands, thus if we want to use `nct`, we must enables password based REST login (through PAM)

The default AAA initialization file that gets shipped with NSO resides under `/var/opt/ncs/cdb/aaa_init.xml`. If we're not happy with that, this is a good point in time to modify the initialization data for AAA.

The NSO daemon is still not running, and we have no existing CDB files. The defaults are restrictive and fine though, so we'll keep them here.

Looking at the `aaa_init.xml` file we see that two groups are referred to in the NACM rule list, *ncsadmin* and *ncsoper*. The NSO authorization system is group based, thus for the rules to apply for a specific user, the user must be member of the right group. Authentication is performed by PAM, and authorization is performed by the NSO NACM rules. Adding myself to *ncsadmin* group will ensure that I get properly authorized.

```
$ nct ssh-cmd -c 'sudo addgroup  ncsadmin'
$ nct ssh-cmd -c 'sudo adduser $USER  ncsadmin'
```

Henceforth I will log into the different NSO hosts using my own login credentials. There are many advantages to this scheme, the main one being that all audit logs on the NSO hosts will show who did what and when. The common scheme of having a shared *admin* user with a shared password is not recommended.

To test the NSO logins, we must first start NSO:

```
$ nct ssh-cmd -c 'sudo /etc/init.d/ncs start'
```

Or use the nct command `nct start`:

```
$ nct start
```

At this point we should be able to curl login over REST, and also directly log in remotely to the NSO cli. On the admin host:

```
$ ssh -p 2024 cfs-m
klacke connected from 10.147.40.94 using ssh on cfs-m
klacke@cfs-m> exit
Connection to cfs-m closed.
```

Checking the NSO audit log on the NSO host *cfs-m* we see at the end of `/var/log/ncs/audit.log`

```
<INFO> 5-Jan-2016::15:51:10.425 cfs-m ncs[666]: audit user: klacke/0
       logged in over ssh from 10.147.40.94 with authmeth:publickey
<INFO> 5-Jan-2016::15:51:10.442 cfs-m ncs[666]: audit user:
       klacke/21 assigned to groups: ncsadmin,sambashare,lpadmin,
       klacke,plugdev,dip,sudo,cdrom,adm
<INFO> 5-Jan-2016::16:03:42.723 cfs-m ncs[666]: audit user:
       klacke/21 CLI 'exit'
<INFO> 5-Jan-2016::16:03:42.927 cfs-m ncs[666]: audit user:
       klacke/0 Logged out ssh <publickey> user
```

Especially the group assignment is worth mentioning here, we were assigned to the recently created *ncsadmin* group. Testing the REST api we get:

```
$ curl -u klacke:PASSW http://cfs-m:8080/api -X GET
curl: (7) Failed to connect to cfs-m port 8080: Connection refused
$ curl -k -u klacke:PASSW https://cfs-m:8888/api -X GET
<api xmlns="http://tail-f.com/ns/rest" xmlns:y="http://tail-f.com/ns/rest">
  <version>0.5</version>
  <config/>
  <running/>
  <operational/>
  <operations/>
  <rollbacks/>
```



```
</api>
```

The `nct check` command is a good command to check all 4 NSO hosts in one go:

```
nct check --rest-pass PASSW  --rest-port 8888 -c all
```

Cisco Smart Licensing

NSO uses Cisco Smart Licensing, described in detail in the NSO Administration Guide. After you have registered your NSO instance(s), and received a token, by following step 1-6 as described in the Create a License Registration Token section of the NSO Administration Guide, you need to enter a token from your Cisco Smart Software Manager account on each host. You can use the same token for all instances. We can use the `nct cli-cmd` tool to do this on all NSO hosts:

```
$ nct cli-cmd --style cisco -c 'license smart register idtoken YzY2Yj...'
```

Note The Cisco Smart Licensing CLI command is present only in the Cisco Style CLI, so make sure you use the `--style cisco` flag with `nct cli-cmd`

Global settings and timeouts

Depending on your installation, the size and speed of the managed devices, as well as the characteristics of your service applications - some of the default values of NSO may have to be tweaked. In particular some of the timeouts.

- Device timeouts. NSO has connect/read/and write timeouts for traffic that goes from NSO to the managed devices. The default value is 20 seconds for all three. Some routers are slow to commit, some are sometimes slow to deliver it's full configuration. Adjust timeouts under `/devices/global-settings` accordingly.
- Service code timeouts. Some service applications can sometimes be slow. In order to minimize the chance of a service application timing out - adjusting `/services/global-settings/service-callback-timeout` might be applicable - depending on the application.

There are quite a few different global settings to NSO, the two mentioned above usually needs to be changed. On the management station:

```
$ cat globs.xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <global-settings>
      <connect-timeout>120</connect-timeout>
      <read-timeout>120</read-timeout>
      <write-timeout>120</write-timeout>
      <trace-dir>/var/log/ncs</trace-dir>
  </global-settings>
</devices>
<services xmlns="http://tail-f.com/ns/ncs">
  <global-settings>
    <service-callback-timeout>180</service-callback-timeout>
  <global-settings>
</java-vm>
</config>
```



```
$ nct load-config --file globs.xml --type xml
```

Enabling SNMP

For real deployments we usually want to enable SNMP. Two reasons:

- When NSO alarms are created - SNMP traps automatically get created and sent - thus we typically want to enable SNMP and also set one or more trap targets
- Many organizations have SNMP based monitoring systems, in order to enable SNMP based system to monitor NSO we need SNMP enabled.

There is already a decent SNMP configuration in place, it just needs a few extra localizations: We need to enable SNMP, and decide:

- If and where-to send SNMP traps
- Which SNMP security model to choose.

At a minimum we could have:

```
klacke@cfs-s% show snmp
agent {
    enabled;
    ip               0.0.0.0;
    udp-port         161;
    version {
        v1;
        v2c;
        v3;
    }
    engine-id {
        enterprise-number 32473;
        from-text         testing;
    }
    max-message-size 50000;
}
system {
    contact  Klacke;
    name     nso;
    location Stockholm;
}
target test {
    ip       3.4.5.6;
    udp-port 162;
    tag      [ x ];
    timeout  1500;
    retries  3;
    v2c {
        sec-name test;
    }
}
community test {
    sec-name test;
}
notify test {
```

```
    tag  x;
    type trap;
}
```

Loading the required NSO packages

We'll be using a couple of packages to illustrate the process of managing packages over a set of NSO nodes. The first prerequisite here is that all nodes *must* have the same version of all packages. If not, havoc will wreak. In particular HA will break, since a check is run while establishing a connection between the secondary and the primary, ensuring that both nodes have exactly the same NSO packages loaded.

On our management station we have the following NSO packages.

```
$ ls -lt packages
total 15416
-rw-r--r-- 1 klacke klacke     8255 Jan  5 13:10 ncs-4.1-nso-util-1.0.tar.gz
-rw-r--r-- 1 klacke klacke 14399526 Jan  5 13:09 ncs-4.1-cisco-ios-4.0.2.tar.gz
-rw-r--r-- 1 klacke klacke  1369969 Jan  5 13:07 ncs-4.1-tailf-hcc-4.0.1.tar.gz
```

Package management in an NSO system install is a three-stage process.

- First, all versions of all all packages, all reside in `/opt/ncs/packages` Since this is the initial install, we'll only have a single version of our 3 example packages.
- The version of each package we *want* to use will reside as a symlink in `/var/opt/ncs/packages/`
- And finally, the package which is actually running, will reside under `/var/opt/ncs/state/packages-in-use.cur`

The tool here is `nct packages`, it can be used to upload and install our packages in stages. The `nct packages` command work over the REST api, thus in the following examples I have added `{rest_user, "klacke"}` and also `{rest_port, 8888}` to my $NCT_HOSTSFILE. We upload all our packages as:

```
$ for p in packages/*; do
     nct packages --file $p -c fetch --rest-pass PASSW
  done
  Fetch Package at 10.147.40.80:8888
  OK
  ......
```

Verifying on one of the NSO hosts:

```
$ ls /opt/ncs/packages/
ncs-4.1-cisco-ios-4.0.2.tar.gz  ncs-4.1-tailf-hcc-4.0.1.tar.gz
ncs-4.1-nso-util-1.0.tar.gz
```

Verifying with the `nct` command

```
$ nct packages --rest-pass PASSW list
Package Info at 10.147.40.80:8888
  ncs-4.1-cisco-ios-4.0.2 (installable)
  ncs-4.1-nso-util-1.0 (installable)
  ncs-4.1-tailf-hcc-4.0.1 (installable)
Package Info at 10.147.40.78:8888
  ncs-4.1-cisco-ios-4.0.2 (installable)
  ncs-4.1-nso-util-1.0 (installable)
  ncs-4.1-tailf-hcc-4.0.1 (installable)
```

```
Package Info at 10.147.40.190:8888
  ncs-4.1-cisco-ios-4.0.2 (installable)
  ncs-4.1-nso-util-1.0 (installable)
  ncs-4.1-tailf-hcc-4.0.1 (installable)
Package Info at 10.147.40.77:8888
  ncs-4.1-cisco-ios-4.0.2 (installable)
  ncs-4.1-nso-util-1.0 (installable)
  ncs-4.1-tailf-hcc-4.0.1 (installable)
```

Next step is to install the packages. As stated above, package management in NSO is a three-stage process, we have now covered step one. The packages reside on the NSO hosts. Step two is to *install* the 3 packages. This is also done through the `nct` command as:

```
$ nct packages --package ncs-4.1-cisco-ios-4.0.2 --rest-pass PASSW -c install
$ nct packages --package ncs-4.1-nso-util-1.0 --rest-pass PASSW -c install
$ nct packages --package  ncs-4.1-tailf-hcc-4.0.1 --rest-pass PASSW -c install
```

This command will setup the symbolic links from `/var/opt/ncs/packages` to `/opt/ncs/packages`. NSO is still running with the previous set of packages. Actually, even a restart of NSO will run with the previous set of packages. The packages that get loaded at startup time reside under `/var/opt/ncs/state/packages-in-use.cur`.

To force a single node to restart using the set of *installed* packages under `/var/opt/ncs/packages` we can do:

```
/etc/init.d/ncs restart-with-package-reload
```

This is a full NSO restart, depending on the amount of data in CDB and also depending on which data models are actually updated, it's usually faster to have the NSO node reload the data models and do the schema upgrade while running. The NSO CLI has support for this using the CLI command.

```
$ ncs_cli

klacke connected from 10.147.40.113 using ssh on cfs-m
klacke@cfs-m> request packages reload
```

Here however, we wish to do the data model upgrade on all 4 NSO hosts, the `nct` tool can do this as:

```
$ nct packages --rest-pass PASSW -c reload
Reload Packages at 10.147.40.80:8888
  cisco-ios            true
  nso-util             true
  tailf-hcc            true
Reload Packages at 10.147.40.78:8888
  cisco-ios            true
  nso-util             true
  tailf-hcc            true
Reload Packages at 10.147.40.190:8888
  cisco-ios            true
  nso-util             true
  tailf-hcc            true
Reload Packages at 10.147.40.77:8888
  cisco-ios            true
  nso-util             true
  tailf-hcc            true
```

To verify that all packages are indeed loaded and also running we can do the following in the CLI:

```
$ ncs_cli
klacke@cfs-m> show status packages package oper-status
package cisco-ios {
    oper-status {
    up;
  }
}
package nso-util {
    oper-status {
    up;
  }
}
package tailf-hcc {
    oper-status {
    up;
  }
}
```

We can use the `nct` tool to do it on all NSO hosts

```
$ nct cli-cmd  -c 'show status packages package oper-status'
```

This section covered initial loading of NSO packages, in a later section we will also cover upgrade of existing packages.

Preparing the HA of the NSO installation

In this example we will be running with two HA-pairs, the two CFS nodes will make up one HA-pair and the two RFS nodes will make up another HA-pair. We will use the `tailf-hcc` package as a HA framework. The package itself is well documented thus that will not be described here. Instead we'll just show a simple standard configuration of *tailf-hcc* and we'll focus on issues when managing and upgrading an HA cluster.

One simple alternative to the *tailf-hcc* package is to use completely manual HA, i.e HA entirely without automatic failover. An example of code that accomplish this can be found in the NSO example collection under `examples.ncs/web-server-farm/ha/packages/manual-ha`

I have also modified the $NCT_HOSTSFILE to have a few groups so that we can do `nct` commands to groups of NSO hosts.

If we plan to use VIP fail over, a prerequisite is the `arping` command and the `ip` command

```
$ nct ssh-cmd -c 'sudo aptitude -y install arping'
$ nct ssh-cmd -c 'sudo aptitude -y install iprout2'
```

The *tailf-hcc* package gives us two things and only that:

- All CDB data becomes replicated from the primary to the secondary
- If the primary fails, the secondary takes over and starts to act as primary. I.e the package automatically handles one fail over. At fail over, the *tailf-hcc* either brings up a Virtual alias IP address using gratuitous ARP or be means of Quagga/BGP announce a better route to an anycast IP address.

Thus we become resilient to NSO host failures. However it's important to realize that the *tailf-hcc* is fairly primitive once a fail over has occurred. We shall run through a couple of failure scenarios in this section.

Following the *tailf-hcc* documentation we have the same HA configuration on both *cfs-m* and *cfs-s*. The tool to use in order to push identical config to two nodes, is `nct load-config`. We prepare the configuration as XML data on the management station:

```
$ dep cat srv-ha.xml
<ha xmlns="http://tail-f.com/pkg/tailf-hcc">
  <token>xyz</token>
  <interval>4</interval>
  <failure-limit>10</failure-limit>
  <member>
    <name>cfs-m</name>
    <address>10.147.40.190</address>
    <default-ha-role>master</default-ha-role>
  </member>
  <member>
    <name>cfs-s</name>
    <address>10.147.40.77</address>
    <default-ha-role>slave</default-ha-role>
    <failover-master>true</failover-master>
  </member>
</ha>
$ nct load-config --file srv-ha.xml --type xml --group srv

Node 10.147.40.190 [cfs-m]
load-config result : successfully loaded srv-ha.xml with ncs_load

Node 10.147.40.77 [cfs-s]
load-config result : successfully loaded srv-ha.xml with ncs_load
```

The last piece of the puzzle here is now to activate HA. The configuration is now there on both the service nodes. We use the `nct ha` command to basically just execute the CLI command `request ha commands activate` on the two service nodes.

```
$ nct ha --group srv --action activate --rest-pass PASSW
```

To verify the HA status we do:

```
$ nct ha --group srv --action status --rest-pass PASSW

HA Node 10.147.40.190:8888 [cfs-m]
cfs-m[master] connected cfs-s[slave]

HA Node 10.147.40.77:8888 [cfs-s]
cfs-s[slave] connected cfs-m[master]
```

To verify the whole setup, we can now also run the `nct check` command, now that HA is operational

```
$ nct check --group srv --rest-pass PASSW --netconf-user klacke all

ALL Check to 10.147.40.190:22 [cfs-m]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=37%
REST OK
NETCONF OK
NCS-VSN : 4.1
HA : mode=master, node-id=cfs-m, connected-slave=cfs-s
```

```
ALL Check to 10.147.40.77:22 [cfs-s]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=37%
REST OK
NETCONF OK
NCS-VSN : 4.1
HA : mode=slave, node-id=cfs-s, master-node-id=cfs-m
```

Handling tailf-hcc HA fallout

As previously indicated, the *tailf-hcc* is not especially sophisticated. Here follows a list of error scenarios after which the operator must act. This section applies to tailf-hcc 4.x and earlier.

NSO secondary node failure

If the *cfs-s* node reboots, NSO will start from the /etc boot scripts. The HA component cannot automatically decide what to do though. It will await an explicit operator command. After reboot, we will see:

```
klacke@cfs-s> show status ncs-state ha
mode none;
[ok][2016-01-07 18:36:41]
klacke@cfs-s> show status ha
member cfs-m {
    current-ha-role unknown;
}
member cfs-s {
    current-ha-role unknown;
}
```

On the designated secondary node, and the preferred primary node *cfs-m* will show:

```
klacke@cfs-m> show status ha
member cfs-m {
    current-ha-role master;
}
member cfs-s {
    current-ha-role unknown;
}
```

To remedy this, the operator must once again *activate* HA. In suffices to to it on *cfs-s*, but we can in this case safely do it on both nodes, even doing it using the `nct ha` command. Re-activating HA on *cfs-s* will ensure that

- All data from *cfs-m* is copied to *cfs-s*.
- Ensure that all future configuration changes (that have to go through *cfs-m* are replicated.

NSO primary node failure

This is the interesting fail over scenario. Powering off the *cfs-m* primary node, we see the following on *cfs-s*:

- An alarm gets created on the designated secondary

```
alarm-list {
    number-of-alarms 1;
    last-changed     2016-01-11T12:48:45.143+00:00;
    alarm ncs node-failure /ha/member[name='cfs-m'] "" {
```



```
        is-cleared              false;
        last-status-change      2016-01-11T12:48:45.143+00:00;
        last-perceived-severity critical;
        last-alarm-text         "HA connection lost. 'cfs-s' transitioning to HA MASTER role.
                                 When the problem has been fixed, role-override the old MASTER to SLAVE
                                 to prevent config loss, then role-revert all nodes.
                                 This will clear the alarm.";
    .....
```

- Fail over occurred:

```
klacke@cfs-s> show status ha
member cfs-m {
    current-ha-role unknown;
}
member cfs-s {
    current-ha-role master;
}
```

This is a critical moment, HA has failed over. When the original primary *cfs-m* restarts, the operator MUST manually decide what to do. Restarting *cfs-m* we get:

```
klacke@cfs-m> show status ha
member cfs-m {
    current-ha-role unknown;
}
member cfs-s {
    current-ha-role unknown;
}
```

If we now activate the original primary, it will resume it's former primary role. Since *cfs-s* already is primary, this will be a mistake. Instead we must

```
klacke@cfs-m> request ha commands role-override role slave
status override
[ok][2016-01-11 14:28:27]
klacke@cfs-m> request ha commands activate
status activated
[ok][2016-01-11 14:28:42]
klacke@cfs-m> show status ha
    member cfs-m {
current-ha-role slave;
}
member cfs-s {
    current-ha-role master;
}
```

This means that all config from *cfs-s* will be copied back to *cfs-m*. Once HA is once again established, we can easily go back to original situation by executing:

```
klacke@cfs-m> request ha commands role-revert
```

on both nodes. This is recommended in order to have the running situation as normal as possible.

Note: This is indeed a critical operation. It's actually possible to loose all or some data here. For example, assume that the original primary *cfs-m* was down for a period of time, the following sequence of events/commands will loose data.

1 *cfs-m* goes down at time *t0*
2 Node *cfs-s* continues to process provisioning request until time *t1* when it goes down.
3 Node *cfs-s* goes down
4 The original primary *cfs-m* comes up and the operator activates *cfs-m* manually. At which time it can start to process provisioning requests.

The above sequence of events/commands loose all provisioning requests between *t0* and *t1*

Setting up the VIP or L3 anycast BGP support

The final part of configuring HA is enabling either IP layer 2 VIP (Virtual IP) support or IP layer 3 BGP anycast fail over. Here we will describe layer 2 VIP configuration, details about anycast setup can be found in the *tailf-hcc* documentation.

We modify the HA configuration so that it looks as:

```
klacke@cfs-m% show ha
token         xyz;
interval      4;
failure-limit 10;
vip {
    address 10.147.41.253;
}
member cfs-m {
    address         10.147.40.190;
    default-ha-role master;
    vip-interface   eth0;
}
member cfs-s {
    address         10.147.40.77;
    default-ha-role slave;
    failover-master true;
    vip-interface   eth0;
}
```

Whenever a node is primary, it will also bring up a VIP.

```
$ ifconfig eth0:ncsvip
eth0:ncsvip Link encap:Ethernet  HWaddr 08:00:27:0c:c3:48
inet addr:10.147.41.253  Bcast:10.147.41.255  Mask:255.255.254.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

The purpose of the VIP is to have northbound systems, northbound provisioning systems, activate NSO services through the VIP which will always be reachable as long as one NSO system is still up.

Referring to previous discussion above on manual activation, if the operator reactivates HA on the designated primary after a primary reboot, we will end up with two primaries, both activating the VIP using gratuitous ARP. This must be avoided at all costs - thus HA activation must be done with care. It cannot be automated.

Preparing the clustering of the NSO installation

Clustering is a technique whereby we can use multiple NSO nodes for the data. Note: if all our data fits on one node, i.e all service data and all device data, we recommend not using clustering. Clustering is non-trivial to configure, harder to search for errors and also has some performance issues. The clustering has nothing to do with HA, it is solely a means of using multiple machines when our dataset is too big to fit in RAM one machine.

Communication between CFS node and RFS nodes is over SSH and the NETCONF interface. Thus the first thing we must do is to decide which user/password to use when service node establish the SSH connection to the RFS node. One good solution is to create a specific user solely for this purpose.

```
$ sudo adduser netconfncs --ingroup ncsadmin --disabled-login
$ sudo passwd netconfncs
...
$ id netconfncs
uid=1001(netconfncs) gid=1001(ncsadmin) groups=1001(ncsadmin)
```

At service node level we need some cluster configuration

```
    klacke@cfs-m% show cluster
authgroup rfsnodes {
    default-map {
    remote-name     netconfncs;
    remote-password $4$T8hh78koPrja9Hggowrl2A==;
  }
}
```

Our complete cluster configuration looks as:

```
klacke@cfs-m> show configuration cluster
remote-node rfs1 {
    address   10.147.41.254;
    port      2022;
    ssh {
        host-key-verification none;
    }
    authgroup rfsnodes;
    username  netconfncs;
}
authgroup rfsnodes {
    default-map {
        remote-name     netconfncs;
        remote-password $4$T8hh78koPrja9Hggowrl2A==;
    }
}
```

Three important observations on this configuration:

- The address of the remote node is the VIP of the two RFS nodes.
- We turned off ssh host verification. If we need this, we must also make sure that the SSH keys of the two RFS nodes *rfs-m1* and *rfs-s1* are identical. Otherwise we'll not be able to connect over SSH after a fail over.

Testing the cluster configuration

Testing the cluster configuration involves testing a whole chain of credentials. To do this we must add at least one managed device, the RFS node, on the CFS node.

To hook up this device to our cluster we need to add this device to `/devices/device` on CFS the node(s). On the CFS node acting as HA primary, *cfs-m1*, we add:

```
klacke@cfs-m1> show configuration devices
authgroups {
    group default {
        default-map {
            remote-name     admin;
            remote-password $4$QMl45chGWNa5h3ggowrl2A==;
        }
    }
}
device rfs1 {
    lsa-remote-node rfs1;
    authgroup       default;
    device-type {
        netconf {
            ned-id lsa-netconf;
        }
    }
    state {
        admin-state unlocked;
    }
}
```

The device *rfs1* has username *admin* with password *admin*. The `lsa-remote-node` leaf points to the node named *rfs1* in the cluster configuration and will use its IP address, the VIP of the RFS node HA-pair, and port. At this point we can read the data on device *rfs1* all the way from the top level service node *cfs-m*. The HA secondary service node is read-only, and all data is available from there too.

When trouble shooting the cluster setup, it's a good idea to have cluster tracing turned on at the CFS nodes. On the CFS node:

```
klacke@cfs-m% set cluster remote-node rfs1 trace pretty
```

NSO system and packages upgrade

NSO major upgrade

Upgrading NSO in the cluster is done as a staged process. To do this we need to have good group information in the $NCT_HOSTSFILE so that we can easily run `nct` commands to the designated primaries and the designated secondaries. See man-page for `nct`. There are three different forms of upgrade, the simpler is when we only upgrade some packages but retain the same NSO release, or when we do an NSO minor upgrade. The upgrade including the most steps is the NSO major upgrade because it also involves upgrading all packages. Maybe the packages are the same version as before, but the packages MUST be compiled with the right NSO version.

If using the Layered Service Architecture (LSA) feature, the packages that require recompilation include so-called *device compiled RFS services*. Please consult ???? for the correct procedure.

The stages of an NSO upgrade are:

1. Take a full backup on all nodes.
2. Disconnect the HA pairs by deactivating HA on the designated secondaries. This means that the VIP will still be up on the designated primaries.
3. Set the designated primaries into read-only mode. VIP is up, but we cannot allow any updates while the upgrade is in progress.

4. Upgrade the designated secondaries.
5. Deactivate HA on the designated primaries, thereby removing the VIPs.
6. Activate HA on the designated secondaries, and do *role-override*, ensuring that the VIP comes back up as soon as possible. At this point the system is up and online again, however without HA.
7. Upgrade the designated primaries.
8. Force the primary to become secondary to the original designated secondaries, connecting HA again. Once HA is operational, do HA role-revert on all nodes.

This is an error prone process, and scripting the process is highly recommended. One major caveat here is that all packages we have MUST match the NSO release. If they don't the `nct upgrade` command fails. Thus, we must not just `nct install` the right NSO release but we must also make sure that we by means of `nct packages` fetch all the packages we're currently running, but compiled for the new (or old) NSO release.

A script to upgrade our example environment here can look like:

```
#!/bin/bash

set -ex

vsn=$1
restpass=$2
img=nso-${vsn}.linux.x86_64.installer.bin

args="--rest-pass ${restpass}"

function fetch() {
    grp=$1
    for p in packages-${vsn}/*.gz; do
        nct packages --group ${grp} --file $p -c fetch ${args}
    done
}


nct backup
nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
                       --mode enable
nct ha --hostsfile hostsfile --group secondary --action deactivate ${args}
nct install --hostsfile hostsfile --file ${img} --group secondary
fetch secondary
nct upgrade --hostsfile hostsfile --ncs-vsn ${vsn} --backup false -c upgrade \
                            --group secondary ${args}

nct ha --hostsfile hostsfile --group primary --action deactivate ${args}
nct ha --hostsfile hostsfile --group secondary --action role-override ${args} \
                       --role master
nct ha --hostsfile hostsfile --group secondary --action activate ${args}

nct install --hostsfile hostsfile --file ${img} --group primary
fetch primary
nct upgrade --hostsfile hostsfile --ncs-vsn ${vsn} --backup false -c upgrade \
                            --group primary ${args}

nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
```



```
                        --mode disable ${args}
nct ha --hostsfile hostsfile --group primary --action role-override ${args} \
                        --role slave
nct ha --hostsfile hostsfile --group primary --action activate ${args}
```

Once we have run the script, it's paramount that we manually check the outcome. The `nct check` is ideal.

```
$ nct check --rest-pass PASSW

ALL Check to 10.147.40.80:22 [rfs-s1]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=58%
REST OK
NETCONF OK
NCS-VSN : 4.0.3
HA : mode=master, node-id=rfs-s1, connected-slave=rfs-m1

ALL Check to 10.147.40.78:22 [rfs-m1]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=55%
REST OK
NETCONF OK
NCS-VSN : 4.0.3
HA : mode=slave, node-id=rfs-m1, master-node-id=rfs-s1

ALL Check to 10.147.40.190:22 [cfs-m]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=54%
REST OK
NETCONF OK
NCS-VSN : 4.0.3
HA : mode=slave, node-id=cfs-m, master-node-id=cfs-s

ALL Check to 10.147.40.77:22 [cfs-s]
SSH OK : 'ssh uname' returned: Linux
SSH+SUDO OK
DISK-USAGE FileSys=/dev/sda1 (/var,/opt) Use=57%
REST OK
NETCONF OK
NCS-VSN : 4.0.3
HA : mode=master, node-id=cfs-s, connected-slave=cfs-m
```

The above script leaves the system with reversed HA roles. If this is not desired, we must also do

```
$  nct ha --rest-pass  PASSW --action role-revert
```

It's very important that we don't do this without ensuring that the designated primary truly have a complete latest copy of all the configuration data.

NSO minor upgrade

This is considerably easier than the major upgrade. A minor NSO upgrade is when the third number in the NSO release is changed, for example upgrading from NSO 5.6.1 to NSO 5.6.2. A major upgrade is when the first or the second number is changed, for example upgrading to 5.6.1 to 5.7 is a major upgrade.

With minor upgrades we can keep all the packages as they are. Packages (actually .fxs files) are guaranteed to be compatible within a major release. Thus the upgrade scenario for simple upgrades look similar to the major upgrade, with the exception of packages - that can be retained as is.

A script perform an NSO minor upgrade of our example environment here can look like:

```
#!/bin/bash

set -ex

vsn=$1
restpass=$2
img=nso-${vsn}.linux.x86_64.installer.bin

args="--rest-pass ${restpass}"

nct backup
nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
                       --mode enable
nct ha --hostsfile hostsfile --group secondary --action deactivate ${args}
nct install --hostsfile hostsfile --file ${img} --group secondary
nct ssh-cmd --group secondary \
     -c 'sudo /etc/init.d/ncs restart-with-package-reload'


nct ha --hostsfile hostsfile --group primary --action deactivate ${args}
nct ha --hostsfile hostsfile --group secondary --action role-override ${args} \
                     --role master
nct ha --hostsfile hostsfile --group secondary --action activate ${args}

nct install --hostsfile hostsfile --file ${img} --group primary
nct ssh-cmd --group primary \
     -c 'sudo /etc/init.d/ncs restart-with-package-reload'


nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
                     --mode disable ${args}
nct ha --hostsfile hostsfile --group primary --action role-override ${args} \
                     --role slave
nct ha --hostsfile hostsfile --group primary --action activate ${args}
```

Package upgrade

Note In NSO 5, to perform a major NED upgrade, the new procedure is to load the new NED and then run the new **migrate** action. This action will migrate data and change the ned-id. For a detailed description, refer to the NED Migration section in the NED Development guide.

Similarly to upgrading NSO, upgrading packages is also a staged process. We have the exact same sequence of steps as in upgrading NSO. This must also be scripted.

Here is a small script to do this.

```
#!/bin/bash
```



```
set -ex

package=$1
fromver=$2
tover=$3
ncsver=$4
restpass=$5

file=ncs-${ncsver}-${package}-${tover}.tar.gz
frompack=ncs-${ncsver}-${package}-${fromver}
topack=ncs-${ncsver}-${package}-${tover}

args="--rest-pass $restpass "

function pkg() {
    grp=$1
    nct packages --group ${grp} --file ${file} -c fetch ${args}
    nct packages --group ${grp} --package ${frompack} -c deinstall ${args}
    nct packages --group ${grp} --package ${topack} -c install ${args}
    nct cli-cmd -c 'request packages reload' --group ${grp}
}

nct backup
nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
                       --mode enable
nct ha --hostsfile hostsfile --group secondary --action deactivate ${args}
pkg secondary

nct ha --hostsfile hostsfile --group primary --action deactivate ${args}
nct ha --hostsfile hostsfile --group secondary --action role-override ${args} \
                       --role master
nct ha --hostsfile hostsfile --group secondary --action activate ${args}
nct ha --hostsfile hostsfile --group primary --action readonly ${args} \
                       --mode disable
pkg primary
nct ha --hostsfile hostsfile --group primary --action role-override ${args} \
                       --role slave
nct ha --hostsfile hostsfile --group primary --action activate ${args}
```

This script also leaves the system with reversed HA roles, same argument as in previous section apply.

The script can be expanded to handle multiple packages in one go. It's more efficient to upgrade several packages in one go than in several steps. For multiple packages distribution, it is more efficient to use `nct cli-cmd --group primary -c 'request packages ha sync'` to distribute all packages from the primary node to secondary nodes in one go. Another important efficiency note here is the use of `nct cli-cmd -c 'request packages reload'`. There are two ways to load new data models into NSO. This is one, the other is to invoke:

```
$ sudo /etc/init.d/ncs restart-with-package-reload
```

The former is considerably more efficient than the latter. If the amount of data in CDB is huge, the time difference to upgrade can be considerable.

In some cases NSO may give warnings when the upgrade looks "suspicious". See more information on this in the NSO Administration Guide, Loading Packages. In order to ignore warnings and proceed with the upgrade set `NCS_RELOAD_PACKAGES` environment variable to `force`.

An important note to make on NED upgrades is that care must be exercised when upgrading NEDs in production environments. If there exists NSO service instances that have provisioned services towards the old NED, the services may become out of sync after a NED upgrade - followed by a *sync-from*. Worst case scenario is that once services have been provisioned, some NED cannot ever be updated without explicit cleanup code attached to the installation.

It all depends on the type of changes in the NED, if there are major structural changes in YANG model data that is used by service provisioning code, we cannot simply upgrade NEDs without also doing the accompanying changes to the service packages.

Patch management

NSO has the ability to install - during runtime - emergency patches. These get delivered on the form of `.beam` files. The `nct patch` command can be used to install such patches.

Manual upgrade

This section is included so that the reader can understand what is done under the hood by the above examples. It's important to understand this in order to be able to repair a system which for some reason is broken.

Here we have a system running NSO 4.0.3, and we wish to manually upgrade to 4.1. System is running with the 3 packages we have been using above:

```
klacke@cfs-m> show status packages package package-version
package cisco-ios {
    package-version 4.0.3;
}
package nso-util {
    package-version 1.0;
}
package tailf-hcc {
    package-version 4.0.1;
}
[ok][2016-01-20 13:05:52]
klacke@cfs-m> show status ncs-state version
version 4.0.3;
```

To manually upgrade this system we must:

1. Do a system install of the new NSO release. This must be done as root. Download the `nso-4.1.linux.x86_64.installer.bin` to the host and install it as:

   ```
   # sh nso-4.1.linux.x86_64.installer.bin --system-install
   ```

   After this, the 4.1 NSO code resides under `/opt/ncs/4.1`, however the symbolic link `/opt/ncs/current` must also be updated. As root:

   ```
   # rm /opt/ncs/current
   # ln -s /opt/ncs/ncs-4.1 /opt/ncs/current
   ```


2 Update all the packages. For all packages we're running, we must first ensure that we have the exact same version of each package, but compiled for the new NSO release. All packages, for all versions of NSO that we have used, reside under `/opt/ncs/packages`

```
# ls  /opt/ncs/packages
ncs-4.0.3-cisco-ios-4.0.3.tar.gz  ncs-4.1-cisco-ios-4.0.2.tar.gz
ncs-4.0.3-nso-util-1.0.tar.gz     ncs-4.1-nso-util-1.0.tar.gz
ncs-4.0.3-tailf-hcc-4.0.1.tar.gz  ncs-4.1-tailf-hcc-4.0.1.tar.gz
```

Thus just download the right packages and put them all in `/opt/ncs/packages`. Next step is to manually rearrange all the symlinks in `/var/opt/ncs/packages` Initially we have links:

```
ncs-4.0.3-cisco-ios-4.0.3.tar.gz -> /opt/ncs/packages/ncs-4.0.3-cisco-ios-4.0.3.tar.gz
ncs-4.0.3-nso-util-1.0.tar.gz -> /opt/ncs/packages/ncs-4.0.3-nso-util-1.0.tar.gz
ncs-4.0.3-tailf-hcc-4.0.1.tar.gz -> /opt/ncs/packages/ncs-4.0.3-tailf-hcc-4.0.1.tar.gz
```

Instead we want:

```
# cd /var/opt/ncs/packages/
# rm -f *
# for p in /opt/ncs/packages/*4.1*; do ln -s $p; done
```

resulting in links:

```
ncs-4.1-cisco-ios-4.0.2.tar.gz -> /opt/ncs/packages/ncs-4.1-cisco-ios-4.0.2.tar.gz
ncs-4.1-nso-util-1.0.tar.gz -> /opt/ncs/packages/ncs-4.1-nso-util-1.0.tar.gz
ncs-4.1-tailf-hcc-4.0.1.tar.gz -> /opt/ncs/packages/ncs-4.1-tailf-hcc-4.0.1.tar.gz
```

3 Final step is to restart NSO, and tell it to reload packages. Again as root:

```
# /etc/init.d/ncs restart-with-package-reload
Stopping ncs: Starting ncs: .
```

Note: if the restart with reload takes more than 90 seconds, you might encounter the following error message when restarting NSO:

```
Starting ncs (via systemctl): Job for ncs.service failed
because a timeout was exceeded. See "systemctl status
ncs.service" and "journalctl -xe" for details. [FAILED]
```

This does not imply that NSO failed to start, just that it took longer than 90 seconds. It is recommended to wait some additional time before verifying.

Then check the result:

```
$ ncs_cli
klacke@cfs-m> show status packages package oper-status
package cisco-ios {
    oper-status {
        up;
    }
}
package nso-util {
    oper-status {
        up;
    }
}
```

```
package tailf-hcc {
    oper-status {
        up;
    }
}
[ok][2016-01-20 13:33:59]
klacke@cfs-m> show status ncs-state version
version 4.1;
```

System is upgraded, and all packages loaded fine.

Log management

We already covered some of the logging settings that are possible to set in `ncs.conf`. All ncs.conf settings are described in the man page for ncs.conf

```
$ man ncs.conf
.....
```

Log rotate

The NSO system install that we have performed on our 4 hosts also install good defaults for logrotate. Inspect `/etc/logrotate.d/ncs` and ensure that the settings are what you want. Note: The NSO error logs, i.e the files `/var/log/ncs/ncserr.log*` are internally rotated by NSO and MUST not be rotated by `logrotate`

NED logs

A crucial tool for debugging NSO installations are NED logs. These logs are very verbose and are for debugging only. Do not have these logs enabled in production. Note that everything, including potentially sensitive data, is logged. No filtering is done. The NED trace logs are controlled through in CLI under: `/device/global-settings/trace`. It's also possible to control the NED trace on a per device basis under `/devices/device[name='x']/trace`.

There are 3 different levels of trace, and for various historic reasons we usually want different settings depending on device type.

- For all CLI NEDs we want to use the *raw* setting.
- For all ConfD based NETCONF devices we want to use the pretty setting. ConfD sends the NETCONF XML unformatted, pretty means that we get the XML formatted.
- For Juniper devices, we want to use the raw setting, Juniper sends sometimes broken XML that cannot be properly formatted, however their XML payload is already indented and formatted.
- For generic NED devices - depending on the level of trace support in the NED itself, we want either pretty or raw.
- For SNMP based devices, we want the pretty setting.

Thus it's usually not good enough to just control the NED trace from `/devices/global-settings/trace`

Java logs

User application Java logs are written to `/var/log/ncs/ncs-java-vm.log` The level of logging from Java code is controlled on a per Java package basis. For example if, we want to increase the level of logging on e.g the *tailf-hcc* code, we need to look into the code and find out the name of the corresponding Java package.

Unpacking the *tailf-hcc* tar.gz package, we see in file `tailf-hcc/src/java/src/com/tailf/ns/tailfHcc/TcmApp.java` that package is called *com.tailf.ns.tailfHcc*. We can then do:

```
klacke@cfs-s% show java-vm java-logging
logger com.tailf.ns.tailfHcc {
    level level-all;
}
```

Internal NSO log

The internal NSO log resides at `/var/log/ncs/ncserr.*`. The log is written in a binary format, to view the log run command:

```
$ ncs --printlog /var/log/ncs/ncserr.log
```

to view the internal error log.

The command `nct get-logs` grabs all logs from all hosts. This is good when collecting data from the system.

Monitoring the installation

All large scale deployments employ monitoring systems. There are plenty good tools to choose. Open source and commercial. Examples are *Cacti* and *Nagios*. All good monitoring tools has the ability to script (using various protocols) what should be monitored. Using the NSO REST api is ideal for this. We also recommend setting up a special read-only Linux user without shell access for this. The command `nct check` summarizes well what should be monitored.

Alarms

The REST api can be used to view the NSO alarm table. NSO alarms are not events, whenever an NSO alarm is created - an SNMP trap is also sent (assuming we have configured a proper SNMP target) All alarms require operator invention. Thus a monitoring tool should also *GET* the NSO alarm table.

```
curl -k -u klacke:PASSW https://cfs-m:8888/api/operational/alarms/alarm-list -X GET
```

Whenever there are new alarms, an operator MUST take a look.

Security considerations

The AAA setup that we have described so far in this deployment document are the recommended AAA setup. To reiterate:

- Have all users that need access to NSO in PAM, this may then be through `/etc/passwd` or whatever. Do not store any users in CDB.
- Given the default *NACM* authorization rules we should have three different types of users on the system
  - Users with shell access that are members of ncsadmin Linux group. These users are considered fully trusted. They have full access to the system as well as the entire network.
  - Users without shell access that are members of ncsadmin Linux group. These users have full access to the network. They can SSH to the NSO SSH shell, they can execute arbitrary REST calls etc. They cannot manipulate backups and perform system upgrades. If we have provisioning systems north of NSO, it's recommended to assign a user of this type for those operations.

- Users without shell access that are members of ncsoper Linux group. These users have read-only access to the network. They can SSH to the NSO SSH shell, they can execute arbitrary REST calls etc. They cannot manipulate backups and perform system upgrades.

If you have more fine grained authorization requirements than read-write all and read all, additional Linux groups can be created and the *NACM* rules can be updated accordingly. Since the *NACM* are data model specific, We'll do an example here. Assume we have a service that stores all data under `/mv:myvpn`. These services - once instantiated - manipulates the network. We want two new sets of users - apart from the *ncsoper* and *ncsadmin* users we already have. We want one set of users that can read everything under `/mv:myvpn` and one set of users that can read-write everything there. They're not allowed to see anything else in the system as a whole. To accomplish this we recommend:

- Create two new Linux groups. One called *vpnread* and one called *vpnwrite*.
- Modify `/nacm` by adding to all 4 nodes:

```
$ cat nacm.xml
 <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
    <groups>
      <group>
        <name>vpnread</name>
      </group>
      <group>
        <name>vpnwrite</name>
      </group>
    </groups>
    <rule-list>
      <name>vpnwrite</name>
      <group>vpnwrite</group>
      <rule>
        <name>rw</name>
        <module-name>myvpn</module-name>
        <path>/myvpn</path>
        <access-operations>create read update delete</access-operations>
        <action>permit</action>
      </rule>
      <cmdrule xmlns="http://tail-f.com/yang/acm">
        <name>any-command</name>
        <action>permit</action>
      </cmdrule>
    </rule-list>
    <rule-list>
      <name>vpnread</name>
      <group>vpnread</group>
      <rule>
        <name>ro</name>
        <module-name>myvpn</module-name>
        <path>/myvpn</path>
        <access-operations>read</access-operations>
        <action>permit</action>
      </rule>
      <cmdrule xmlns="http://tail-f.com/yang/acm">
        <name>any-command</name>
        <action>permit</action>
```



```
      </cmdrule>
    </rule-list>
  </nacm>

$ nct load-config --file nacm.xml --type xml
```

The above command will merge the data in `nacm.xml` on top of the already existing NACM data in CDB.

For a detailed discussion of the configuration of authorization rules via NACM, see the NSO Administration Guide, The AAA Infrastructure, in particular The AAA Infrastructure, Authorization.

A considerably more complex scenario is when you need/want to have users with shell access to the host, but those users are either untrusted, or shouldn't have any access to NSO at all. NSO listens to a port called `/ncs-config/ncs-ipc-address`, typically on localhost. By default this is *127.0.0.1:4569*. The purpose of the port is to multiplex several different access methods to NSO. The main security related point to make here is that there are no AAA checks done on that port at all. If you have access to the port, you also have complete access to all of NSO. To drive this point home, when we invoke the command `ncs_cli`, that is a small C program that connects to the port and *tells* NSO who you are - assuming that authentication is already performed. There is even a documented flag *--noaaa* which tells NSO to skip all NACM rules checks for this session.

To cover the scenario with untrusted users with SHELL access, we must thus protect the port. This is done through the use of a file in the Linux file system. At install time, the file `/etc/ncs/ipc_access` gets created and populated with random data. Enable `/ncs-config/ncs-ipc-access-check/enabled` in ncs.conf and ensure that trusted users can read the `/etc/ncs/ipc_access` file for example by changing group access to the file.

```
$ cat /etc/ncs/ipc_access
cat: /etc/ncs/ipc_access: Permission denied
$ sudo chown root:ncsadmin /etc/ncs/ipc_access
$ sudo chmod g+r /etc/ncs/ipc_access
$ ls -lat /etc/ncs/ipc_access
$ cat /etc/ncs/ipc_access
.......
```

American Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam.
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at `www.cisco.com/go/offices`.

