Cisco
Customer Experience

Ask the Experts

NSO のスケールアウトと冗長構成の導入方法
（ Scaling NSO Deployment ）

[２０２４年 １月 ２５日]

Disclaimer

This document is Cisco Confidential information provided for your internal business use in connection with the Cisco Services purchased by you or your authorized reseller on your behalf. This document contains guidance based on Cisco's recommended practices.

You remain responsible for determining whether to employ this guidance, whether it fits your network design, business needs, and whether the guidance complies with laws, including any regulatory, security, or privacy requirements applicable to your business.

免責

この文書は、お客様またはお客様の代理人である認定リセラーが購入したシスコサービスに関連して、お客様が社内業務において使用することを目的としてシスコが提供するシスコの機密情報です。この文書にはシスコが推奨するプラクティスに基づく手引きが記載されています。

お客様は、この手引きを使用するか否かやお客様のネットワーク設計および業務上のニーズにこの手引きが適合しているか否か、さらにはこの手引きが法律（お客様の業務に適用される規制上の要件、セキュリティ上の要件およびプライバシーに関する要件を含みます）に準拠しているか否かを判断する責任を引き続き負います。

本日の学習内容：

- LSA (Layered Service Architecture) による NSO のスケールアウト方法
- 堅牢性向上のための Raft HA も含めた HA セットアップ方法

本日のトピック

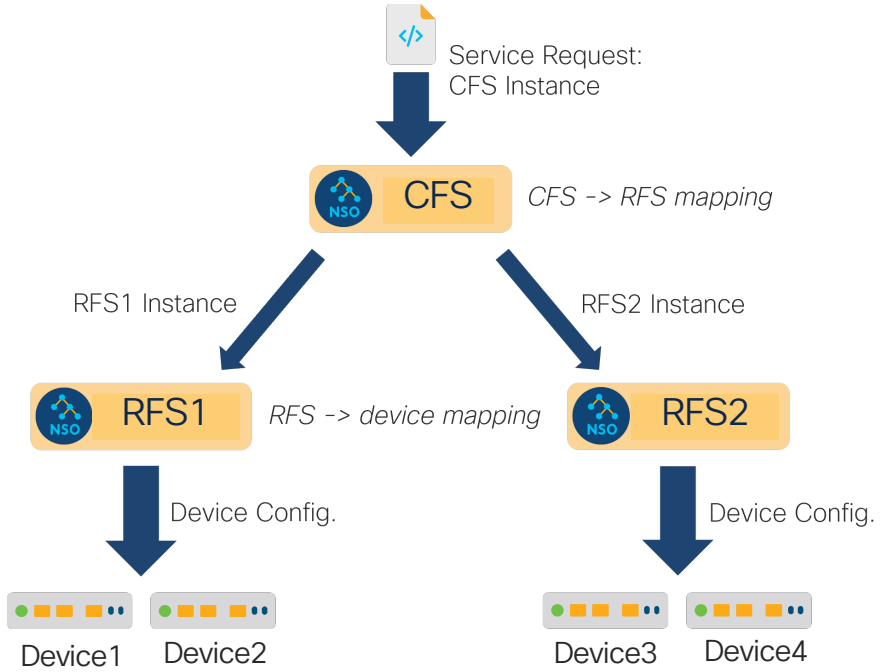
1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

LSA (Layered Service Architecture) とは

Layered Service Architecture (LSA) は、NSO を階層化し、スケールアウトさせる設計アプローチです。LSA を使用し、数十万以上のデバイスに対する数百万のサービスを管理できます。



CFS (Customer Facing Service)
ユーザが操作する NSO

RFS (Resource Facing Service)
デバイスを操作する NSO

- NSO 5.3 まで
  CFS/RFS は同一バージョン
- NSO 5.4 以降
  CFS/RFS で異なるバージョン可能
- NSO 5.5 以降
  ログに Trace ID が付与され、CFS/RFS でのデバッグが容易に

ATX「NSO のインストール方法 」からの再掲

単体 NSO サーバ / VM のサイジング

管理デバイス数や実行するサービス数、そのサービスの複雑さによって必要なリソースが異なります。

CPU

- 少なくとも 8コア、できるだけ高いクロック周波数を使用することを推奨します。
- 2GHz 16コア < 3GHz 8コア

メモリ

- CDB (Configuration Database) の全てのデータが RAM に保持されます。
- CDB とクエリを処理するための作業メモリを必要とします。
- 少なくとも 32 GB RAM の準備を推奨しますが、CDB サイズに応じて必要リソースは変わります。

デバイス数	サービス数	CPU 数	メモリ (GB)
100	5,000	1	32
1,000	10,000	2	32
1,000	25,000	4	32
10,000	25,000	6	64
10,000	100,000	8	64
50,000	1,000,000	16	224

```
admin@ncs> show ncs-state internal cdb datastore running
| select ram-size | select disk-size
NAME     DISK SIZE  RAM SIZE
------------------------------
running  28.51 KiB  240.99 KiB
```

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

RFS (下位 NSO) のセットアップ

RFS ノード (下位NSO) のセットアップは、通常のスタンドアロン NSO とほとんど違いがありません。唯一の違いは、ncs.conf に上位 CFS ノードと対話するために NETCONF Northbound を有効にする必要があることです。

ncs.conf 設定例
(抜粋)

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

RFS (下位 NSO) のセットアップの注意点

注意点として、Commit queue を LSA で使用する場合でも、RFS ノードでは Commit queue を有効化する必要はありません。上位 CFS ノードは適切なコミットフラグを下位 RFS ノードに自動的に伝搬します。

ただし、RFSノードから直接設定変更を実施する可能性がある場合、RFSノードでも下記のように Commit queue をグローバル有効化し、全てのノードで共通にすることをおすすめします。

```
admin@lower-nso(config)# devices global-settings commit-queue enabled-by-default true
```

LSA 構成全体で Commit queue の一貫性がない場合、下記のように rollback-on-error オプションを設定していても、Commit queue を通過していない部分はロールバックできず、元のトランザクション全体をロールバックできなくなる可能性があります。

```
admin@lower-nso(config)# devices global-settings commit-queue error-option rollback-on-error
```

Commit queue の詳細は、下記をご参照ください。
https://developer.cisco.com/docs/nso/guides/#!the-nso-device-manager/commit-queue

CFS (上位 NSO) のセットアップ

CFS ノードには、デバイスとして 下位 RFS ノードを登録します。その際に、通常の IP アドレスやポート番号の代わりに、"lsa-remote-node" を指定することで、LSA RFS ノードであることを認識します。lsa-remote-node は、"cluster remote-node" に設定された値を参照します。

```
admin@upper-nso# show running-config ncs:devices device lower-nso-*
ncs:devices device lower-nso-1
 lsa-remote-node lower-nso-1
 ssh host-key ssh-ed25519
  key-data AAAAC3NzaC1lZDI1NTE5AAAAIL+
 !
 authgroup       default
 device-type netconf ned-id lsa-netconf
 state admin-state unlocked
!
```

```
admin@upper-nso# show cluster connection
                                        LOCAL  REMOTE
REMOTE NODE  ADDRESS    PORT  CHANNELS  USER   USER    STATUS  TRACE
-------------------------------------------------------------------------
lower-nso-1  127.0.0.1  2023  -         admin  admin   up      disabled
```

CFS (上位 NSO) のセットアップ

前頁の構成のため、CFS ノードのセットアップは、下記の流れで実施します。

1. LSA NED の読み込み (マルチバージョン LSA の場合のみ)

   RFS ノードのバージョンに対応する LSA NED を読み込みます。CFS と RFS が同じバージョンである場合は、組み込み NED を利用可能なため、LSA NED の読み込みは不要です。

2. Cluster remote-node の設定

   RFS ノードへの接続情報を cluster remote-node に設定します。

3. LSA デバイス登録

   RFS ノードをデバイスとして登録します。

① LSA NED の読み込み（マルチバージョン LSA の場合のみ）

NSO 5.4 以降では、CFS ノードと RFS ノードが異バージョンでも LSA の構築が可能となりました。マルチバージョン LSA を構築する場合は、下記の手順を実行してください。

LSA NED はインストールディレクトリ内 $NCS_DIR/packages/lsa にあります。
下記フォルダ名 cisco-nso-ncs-x.y の x.y がこの CFS ノードがサポートしている RFS ノードのバージョンです。

```
$ ls ~/nso-6.1.4/packages/lsa/
cisco-nso-nc-5.7  cisco-nso-nc-5.8  cisco-nso-nc-6.0  cisco-nso-nc-6.1
```

RFS ノードのバージョンに対応する LSA NED を実行環境ディレクトリの packages フォルダにコピーまたはシンボリックリンクを作成し、読み込みます。

```
$ ln -s nso-6.1.4/packages/lsa/cisco-nso-nc-5.7 ncs-run/packages
$ ncs-cli -C -u admin
admin@upper-nso# packages reload
```

② Cluster remote-node の設定

下記のように、RFS ノードへの接続情報を cluster remote-node に設定します。

```
admin@upper-nso# config t
admin@upper-nso(config)# load merge terminal
Loading.
cluster remote-node lower-nso-1 authgroup default username admin
cluster remote-node lower-nso-1 address 127.0.0.1 port 2023
cluster remote-node lower-nso-2 authgroup default username admin
cluster remote-node lower-nso-2 address 127.0.0.1 port 2024
cluster device-notifications enabled
cluster commit-queue enabled
(Ctrl+D)
0 bytes parsed in 129.05 sec (0 bytes/sec)
admin@upper-nso(config)# commit
```

cluster device-notifications enabled:
CFS ノードが RFS ノードから転送された
デバイス通知を受信できるようにします。

cluster commit-queue enabled:
LSA の Commit queue を有効化します。

RFS ノードの SSH ホストキーを取得します。

```
admin@upper-nso# cluster remote-node lower-nso-* ssh fetch-host-keys
```

② Cluster remote-node の設定

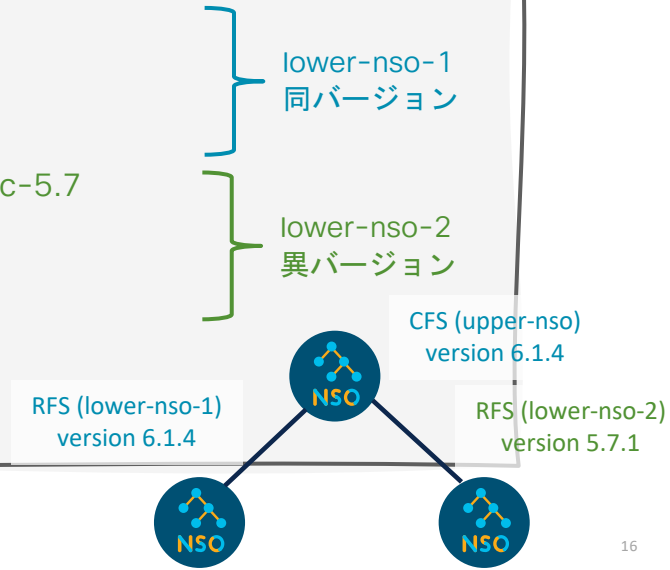
下記のように、Cluster 設定を確認できます。

```
admin@upper-nso# show cluster connection
                              LOCAL  REMOTE
REMOTE NODE  ADDRESS    PORT  CHANNELS  USER   USER    STATUS  TRACE
-------------------------------------------------------------------------
lower-nso-2  127.0.0.1  2024  -         admin  admin   up      disabled
lower-nso-1  127.0.0.1  2023  -         admin  admin   up      disabled

admin@upper-nso#
admin@upper-nso# show cluster remote-node
                                                 RECEIVED
NAME        NAME                STATUS  LAST EVENT                NOTIFICATIONS
-------------------------------------------------------------------------------
lower-nso-1  device-notifications  up      0000-01-01T00:00:00-00:00  0
             ncs-events            up      0000-01-01T00:00:00-00:00  0
lower-nso-2  device-notifications  up      0000-01-01T00:00:00-00:00  0
             ncs-events            up      0000-01-01T00:00:00-00:00  0

admin@upper-nso#
```

③ LSA デバイス登録

下記のように RFS ノードをデバイスとして登録します。ned-id は RFS ノードが同バージョンの場合は組み込みの ”lsa-netconf”、異バージョンの場合は手順①で読み込んだNED “cisco-nso-ncs-x.y” を指定します。

```
admin@upper-nso# config t
admin@upper-nso(config)# load merge terminal
Loading.
devices device lower-nso-1 device-type netconf ned-id lsa-netconf
devices device lower-nso-1 authgroup default
devices device lower-nso-1 lsa-remote-node lower-nso-1
devices device lower-nso-1 state admin-state unlocked
devices device lower-nso-2 device-type netconf ned-id cisco-nso-nc-5.7
devices device lower-nso-2 authgroup default
devices device lower-nso-2 lsa-remote-node lower-nso-2
devices device lower-nso-2 state admin-state unlocked
(Ctrl+D)
0 bytes parsed in 152.12 sec (0 bytes/sec)
admin@upper-nso(config)# commit
```



③ LSA デバイス登録

RFS ノードの SSH ホストキーを取得し、/devices/device のパスに対しても ホストキーの更新を行い、sync-from を実行します。

```
admin@upper-nso# ncs:devices device lower-nso-* ssh fetch-host-keys
admin@upper-nso# ncs:devices sync-from
sync-result {
    device lower-nso-1
    result true
}
sync-result {
    device lower-nso-2
    result true
}
```

RFS ノードに CFS ノードからは認識できない設定投入があった場合、CFS と RFS の間に非同期が発生します。非同期でも CFS ノードから設定投入できるように、下記設定をお勧めします。

```
admin@upper-nso(config)# ncs:devices device lower-nso-* out-of-sync-commit-behaviour accept
```

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

RFS サービス設定用 NED の作成 (単一バージョン LSA)

CFS ノードから RFS のサービスインスタンスの作成を実施するためには、CFS ノードにて RFS のサービスモデル (YANG) から NETCONF NED パッケージを作成し、読み込む必要があります。
ここでは、RFS ノードにてサービスパッケージ (mybgp) の作成及び読み込み済み前提で説明します。

単一バージョン LSA の場合は、下記のように ncs-make-package コマンドで作成します。

```
$ ncs-make-package --no-netsim --no-java --no-python \
   --lsa-netconf-ned package-store/mybgp/src/yang \
   --dest upper-nso/packages/mybgp-ned --build mybgp-ned
```

ncs-make-package オプション

--no-netsim, --no-java, --no-python:
netsim directory, java / python class を生成しない。

--lsa-netconf-ned:
ここで指定されたディレクトリ内の RFS サービスモデル (YANG) から NETCONF NED を生成。
デフォルトで、lsa-netconf を使用してコンパイルされます。

--dest:
生成されたパッケージの置き場所。RFS のサービスパッケージとは別名にすることをお勧めします。

--build:
パッケージの生成と同時にビルドします。

RFS サービス設定用 NED の作成 (マルチバージョン LSA)

マルチバージョン LSA の場合は、下記のようなオプションでパッケージを作成します。

```
$ ncs-make-package --no-netsim --no-java --no-python \
   --lsa-netconf-ned package-store/mybgp/src/yang \
   --lsa-lower-nso cisco-nso-nc-5.7 \
   --package-version 5.7 --dest upper-nso/packages/mybgp-5.7 \
   --build mybgp-5.7
```

ncs-make-package オプション

--lsa-netconf-ned:
ここで指定されたディレクトリ内の RFS サービスモデル (YANG) から NETCONF NED を生成。
デフォルトで、lsa-netconf を使用してコンパイルされます。

--lsa-lower-nso:
下位 RFS ノードの ned-id を指定します。

--package-version:
下位 RFS ノードのバージョンを指定します。

ncs-make-package オプションの詳細は下記をご参照ください。
https://developer.cisco.com/docs/nso/guides/#!ncs-man-pages-volume-1/ncs-make-package

RFS サービス設定用 NED コンパイル時のエラー

RFS のサービスモデルが他の YANG モデルへの依存関係を持つ場合、コンパイルに失敗し、下記のようなエラーが発生します。

```
augmented/mybgp@2016-01-01.yang:1471: error: the node 'devices' from module 'tailf-ncs' (in
node 'config' from 'tailf-ncs-internal-mount') is not found
augmented/mybgp@2016-01-01.yang:3091: error: the node 'devices' from module 'tailf-ncs' (in
node 'live-status' from 'tailf-ncs-internal-mount') is not found
make: *** [Makefile:31: ncsc-out/.done] Error 1
```

下記の例では、type leafref を用いて RFS ノードに登録したデバイスからデバイス名を選択できるようにしていますが、CFS ノードからは RFS ノードのデバイスリストを参照できないため、エラーが発生します。

```
leaf device {
  type leafref {
    path " /ncs:devices/ncs:device/ncs:name" ;
  }
 }
```

RFS サービス設定用 NED コンパイル時のエラー

前頁のようなエラーが発生した場合は、CFS ノードの サービスモデル (YANG) におけるパラメータ定義を type leafref から type string などに変更することで、正常にコンパイルできるようになります。RFS ノードでの修正は不要です。

```
37      // may replace this with other ways of refering
38      leaf device {
39        type leafref {
40          path "/ncs:devices/ncs:device/ncs:name";
41        }
42      }
43
44      // replace with your own stuff here
45      leaf neighbor {
46        type inet:ipv4-address;
47      }
48      leaf routerid {
49        type inet:ipv4-address;
50      }
51      leaf as {
52        type inet:as-number;
```

```
37      // may replace this with other ways of refering
38      leaf device {
          type string;
        }
41
42      // replace with your own stuff here
43      leaf neighbor {
44        type inet:ipv4-address;
45      }
46      leaf routerid {
47        type inet:ipv4-address;
48      }
49      leaf as {
50        type inet:as-number;
51      }
52    }
```

修正後は、CFS ノードの packages/src ディレクトリにて make コマンドを実行し、コンパイルを忘れずに行ってください。

RFS サービス設定用 NED の読み込み

パッケージを作成、コンパイルが正常に完了したら、CFS ノードにてパッケージを読み込み、sync-from を実行します。

```
admin@upper-nso# packages reload
reload-result {
    package mybgp-ned
    result true
}
admin@upper-nso# ncs:devices sync-from
sync-result {
    device lower-nso-1
    result true
}
sync-result {
    device lower-nso-2
    result true
}
```

CFS ノードからのサービス設定

準備が完了すると、下記のように CFS ノードから RFS ノードのサービスインスタンス作成を行うことができるようになります。

上位 CFS ノードでのサービス作成

```
admin@upper-nso(config)# ncs:devices device lower-nso-1
config mybgp test device ex0 as 100 routerid 10.1.1.1
neighbor 10.1.1.2
admin@upper-nso(config)# commit dry-run outformat native
--- omitted ---
    device {
        name ex0
        data router bgp 100
             bgp router-id 10.1.1.1
             neighbor 10.1.1.2 remote-as 100
             address-family ipv4 unicast
              neighbor 10.1.1.2 activate
              exit-address-family
             !
            !
    }
}
admin@upper-nso(config-mybgp-test)# commit
```

下位 RFS ノードでの設定確認

```
admin@lower-nso-1# mybgp test get-modifications
outformat cli-c
cli-c {
    local-node {
        data devices device ex0
                config
                 router bgp 100
                  bgp router-id 10.1.1.1
                  neighbor 10.1.1.2 remote-as 100
                  address-family ipv4 unicast
                   neighbor 10.1.1.2 activate
                   exit-address-family
                  !
                 !
                !
               !
    }
}
```

複数の RFS ノードをまたぐサービスの設定

複数の RFS ノードにわたるサービス設定を行うためには、CFS にサービスパッケージを作成する必要があります。パッケージの作成は下記のような流れで実施することができ、通常のスタンドアロン NSO でのサービスパッケージ作成とあまり違いはありません。

1

パッケージスケルトンの作成

ncs-make-package コマンドを用いて、パッケージのスケルトンを作成します。

これにより、サービスに必要なファイルが自動生成されます。

2

サービスモデル (YANG) の編集

サービス設定のために必要な入力パラメータを定義します。

3

XML テンプレートの編集

Sync した下位 RFS ノードから生成した XML に、サービスモデルで定義したパラメータを適切な箇所に埋め込むことでマッピングロジックを作成します。

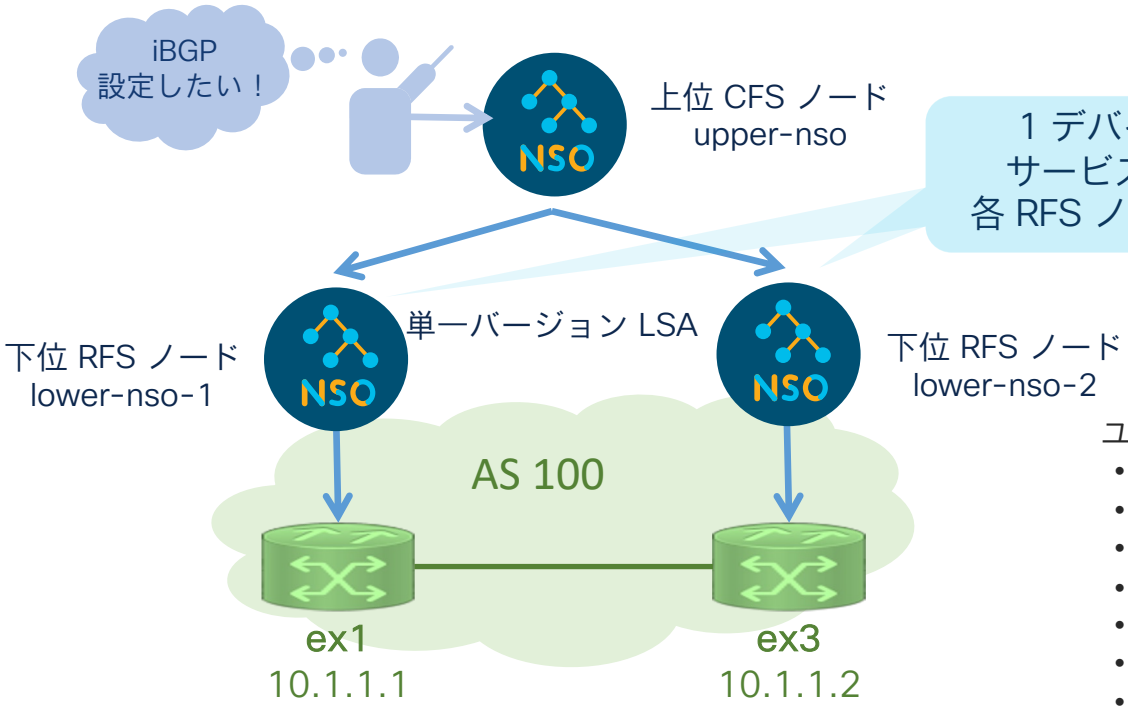
4

パッケージの読み込み

NSO CLI にて packages reload コマンドを実行し、作成したサービスを適用可能な状態にします。

シナリオ

上位 CFS ノードから下記 7 つのパラメータで異なる RFS ノードに接続された 2 デバイスに対して iBGP 設定を行う例を用いて、CFS ノードでのサービスパッケージ作成方法をご説明します。



ユーザからの入力（7 項目）

- AS 番号: 100
- 下位 RFS ノード1: lower-nso-1
- デバイス1: ex1
- デバイス1 IPアドレス: 10.1.1.1
- 下位 RFS ノード2: lower-nso-2
- デバイス2: ex3
- デバイス2 IPアドレス: 10.1.1.2

Step1: パッケージスケルトンの作成

CFS ノードの実行環境ディレクトリ (今回は upper-nso) の packages フォルダにて、
ncs-make-package コマンドを用いて、パッケージのスケルトンを作成します。

```
$ cd upper-nso/packages
$ ncs-make-package --service-skeleton python-and-template cfs-bgp
$ ls cfs-bgp
README  package-meta-data.xml  python  src  templates  test
```

Step2: サービスモデル (YANG) の編集

サービスモデル (upper-nso/packages/cfs-bgp/src/yang/cfs-bgp.yang) を編集し、
ユーザ入力が必要なパラメータを定義します。今回は、下記 7 つのパラメータを定義します。

- AS 番号: as
- 下位 RFS ノード1: dev1-lower-nso
- デバイス1: dev1
- デバイス1 IP アドレス: dev1-addr
- 下位 RFS ノード2: dev2-lower-nso
- デバイス2: dev2
- デバイス2 IP アドレス: dev2-addr

RFS ノードに登録したデバイスリストから直接設定対象のデバイス名を参照することができないため、ここでは type string (文字列入力) としています。

編集が完了したら、
upper-nso/packages/bgpmgr/src
フォルダに移動し、make を実行し、
YANG をコンパイルします

```
37    // may replace this with other ways of refering to the devices.
38    leaf dev1-lower-nso {
39      type leafref {
40        path "/ncs:devices/ncs:device/ncs:name";
41      }
42    }
43    leaf dev2-lower-nso {
44      type leafref {
45        path "/ncs:devices/ncs:device/ncs:name";
46      }
47    }
48
49    // replace with your own stuff here
50    leaf dev1 {
51      type string;
52    }
53    leaf dev2 {
54      type string;
55    }
56    leaf dev1-addr {
57      type inet:ipv4-address;
58    }
59    leaf dev2-addr {
60      type inet:ipv4-address;
61    }
62    leaf as {
63      type inet:as-number;
64    }
```

cfs-bgp.yang 設定例

Step3: XML テンプレートの編集

XML テンプレート (upper-nso/packages/bgpmgr/templates/cfs-bgp.xml) を編集し、
サービスモデルとデバイスモデルとのマッピングロジックを作成します。
まず、CFS ノードにて RFS ノードに対して mybgp サービスインスタンスを作成するために必要な設定をXML形式に変換します。

```
admin@upper-nso(config)# ncs:devices device lower-nso-1 config
mybgp test device ex0 as 100 routerid 10.1.1.1 neighbor 10.1.1.2
admin@upper-nso(config-mybgp-test)# commit dry-run outformat
xml
result-xml {
    local-node {
        data <devices xmlns=" http://tail-f.com/ns/ncs" >
               <device>
                 <name>lower-nso-1</name>
                 <config>
                   <mybgp xmlns=" http://example.com/mybgp" >
                     <name>test</name>
                     <device>ex0</device>
                     --- omitted ---
               </device>
             </devices>
    }
```

CFS ノードの CDB に書き込まれる設定

RFS ノードの CDB に書き込まれる設定

```
lsa-node {
        name lower-nso-1
        data <devices xmlns=" http://tail-f.com/ns/ncs" >
               <device>
                 <name>ex0</name>
                 <config>
                 --- omitted ---
                 </config>
               </device>
             </devices>
             <mybgp xmlns=" http://example.com/mybgp" >
               <name>test</name>
               <device>ex0</device>
                --- omitted ---
             </mybgp>
    }
}
admin@upper-nso(config-mybgp-test)#
```

Step3: XML テンプレートの編集

XML 変換後のコンフィグの local-node の <device> ~ </device> をコピーし、 XML テンプレート (upper-nso/packages/cfs-bgp/templates/cfs-bgp-template.xml) の <device> タグ部分を置き換えます。

```
admin@upper-nso(config-mybgp-test)# commit dry-run
outformat xml
result-xml {
    local-node {
        data <devices xmlns=" http://tail-f.com/ns/ncs" >
               <device>
                <name>lower-nso-1</name>
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>test</name>
                    <device>ex0</device>
                    --- omitted ---
               </device>
             </devices>
    }
    lsa-node {
        --- omitted ---
}
admin@upper-nso(config-mybgp-test)#
```

コピー

-version-deployment-test > upper-nso > packages > cfs-bgp > templates > cfs-bgp-te

```
1  <config-template xmlns="http://tail-f.com/ns/config/1.0">
2    <devices xmlns="http://tail-f.com/ns/ncs">
3      <device>
4        <!--
5          Select the devices from some data structure in the service
6          model. In this skeleton the devices are specified in a leaf-
7          Select all devices in that leaf-list:
8        -->
9        <name>{/device}</name>
10       <config>
11         <!--
12           Add device-specific parameters here.
13           In this skeleton the, java code sets a variable DUMMY, use
14           to set something on the device e.g.:
15           <ip-address-on-device>{$DUMMY}</ip-address-on-device>
16         -->
17       </config>
18     </device>
19   </devices>
20 </config-template>
```

残す

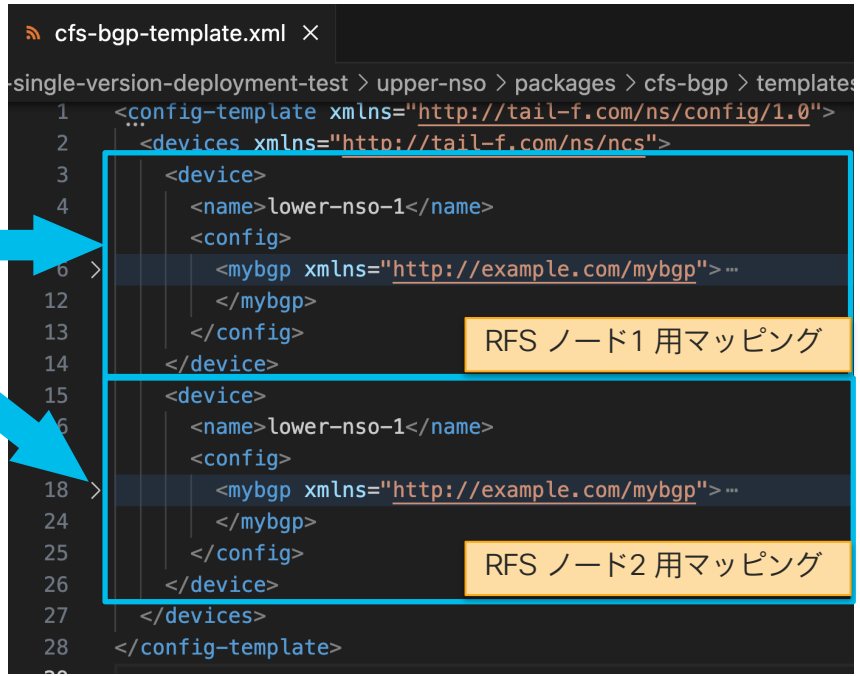
置換

残す

Step3: XML テンプレートの編集

今回、いずれの RFS ノードも同じバージョンかつ同じサービスモデル (mybgp) の使用を想定するため、<device> タグ部分を下記のようにもう一度ペーストします。RFS ノードごとに異なるサービスモデルを使用する場合は、 NED 作成から実施してください。

```
admin@upper-nso(config-mybgp-test)# commit dry-run
outformat xml
result-xml {
    local-node {
        data <devices xmlns=" http://tail-f.com/ns/ncs" >
              <device>
                <name>lower-nso-1</name>
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>test</name>
                    <device>ex0</device>
                    --- omitted ---
              </device>
            </devices>
    }
    lsa-node {
        --- omitted ---
}
admin@upper-nso(config-mybgp-test)#
```



Step3: XML テンプレートの編集

サービスモデルで定義したパラメータを {/変数名} に置き換えます。

- AS 番号: {/as}
- デバイス1: {/dev1}
- デバイス1 ループバック: {/dev1-addr}
- 下位 RFS ノード1: {/dev1-lower-nso}
- デバイス2: {/dev2}
- デバイス2 ループバック: {/dev2-loop}
- 下位 RFS ノード2: {/dev2-lower-nso}

```
3     <device>
4       <name>lower-nso-1</name>
5       <config>
6         <mybgp xmlns="http://example.com/mybgp">
7           <name>test</name>
8           <device>ex0</device>
9           <neighbor>10.1.1.2</neighbor>
10          <routerid>10.1.1.1</routerid>
11          <as>100</as>
12        </mybgp>
13      </config>
14    </device>
```

RFS ノード1 用マッピング

```
3     <device>
4       <name>{/dev1-lower-nso}</name>
5       <config>
6         <mybgp xmlns="http://example.com/mybgp">
7           <name>{/name}</name>
8           <device>{/dev1}</device>
9           <neighbor>{/dev2-addr}</neighbor>
10          <routerid>{/dev1-addr}</routerid>
11          <as>{/as}</as>
12        </mybgp>
13      </config>
14    </device>
```

RFS ノード1 用マッピング

<name></name> は、RFS サービスインスタンスの識別子です。ここでは、CFS サービスインスタンス名 (name という名前で自動定義)と同じ値としています。

Step3: XML テンプレートの編集

RFS ノード 2 用のマッピングも同様に置き換えます。

- AS 番号: {/as}
- デバイス1: {/dev1}
- デバイス1 ループバック: {/dev1-addr}
- 下位 RFS ノード1: {/dev1-lower-nso}
- デバイス2: {/dev2}
- デバイス2 ループバック: {/dev2-loop}
- 下位 RFS ノード2: {/dev2-lower-nso}

```
16        <name>lower-nso-1</name>
17        <config>
18          <mybgp xmlns="http://example.com/mybgp">
19            <name>test</name>
20            <device>ex0</device>
21            <neighbor>10.1.1.2</neighbor>
22            <routerid>10.1.1.1</routerid>
23            <as>100</as>
24          </mybgp>
25        </config>
26      </device>
27    </devices>
```

RFS ノード2 用マッピング

```
16        <name>{/dev2-lower-nso}</name>
17        <config>
18          <mybgp xmlns="http://example.com/mybgp">
19            <name>{/name}</name>
20            <device>{/dev2}</device>
21            <neighbor>{/dev1-addr}</neighbor>
22            <routerid>{/dev2-addr}</routerid>
23            <as>{/as}</as>
24          </mybgp>
25        </config>
26      </device>
27    </devices>
```

RFS ノード2 用マッピング

<name></name> は、RFS サービスインスタンスの識別子です。ここでは、CFS サービスインスタンス名 (name という名前で自動定義)と同じ値としています。

Step4: パッケージの読み込み

CFS ノードにて packages reload コマンドを実行し、作成したサービスを読み込みます。

```
$ ncs_cli -C -u admin
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has completed.
>>> System upgrade has completed successfully.
reload-result {
    package cfs-bgp
    result true
}
admin@ncs#
admin@ncs# show packages package oper-status
packages package cfs-bgp
 oper-status up
admin@ncs#
```

CFS ノードからのサービス設定

作成したサービスパッケージ cfs-bgp を用いて、上位 CFS ノードから下記 7 つのパラメータで異なる RFS ルータに接続された 2 デバイスに対して iBGP 設定を実行します。

```
admin@upper-nso(config)# cfs-bgp ex1_ex3 dev1-lower-nso
lower-nso-1 dev1 ex1 dev1-addr 10.1.1.1 dev2-lower-nso
lower-nso-2 dev2 ex3 dev2-addr 10.1.1.2 as 100
admin@upper-nso(config)# commit dry-run outformat native
native {
    device {
        name lower-nso-1
            --- omitted ---
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>ex1_ex3</name>
                    <neighbor>10.1.1.2</neighbor>
                    <routerid>10.1.1.1</routerid>
                    <as>100</as>
                    <device>ex1</device>
                  </mybgp>
                  --- omitted ---
    device {
        name ex1
        data router bgp 100
             bgp router-id 10.1.1.1
             neighbor 10.1.1.2 remote-as 100
             address-family ipv4 unicast
              neighbor 10.1.1.2 activate
              exit-address-family
```

```
--- omitted ---
device {
        name lower-nso-2
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>ex1_ex3</name>
                    <neighbor>10.1.1.1</neighbor>
                    <routerid>10.1.1.2</routerid>
                    <as>100</as>
                    <device>ex3</device>
                  </mybgp>
                  --- omitted ---
 device {
        name ex3
        data router bgp 100
             bgp router-id 10.1.1.2
             neighbor 10.1.1.1 remote-as 100
             address-family ipv4 unicast
              neighbor 10.1.1.1 activate
              exit-address-family
             !
            !
    }
}
admin@upper-nso(config)#
```

下位 RFS ノードを自動入力させる方法

前述の例では、下位 RFS ノードを指定していましたが、dispatch-map を作成し、デバイス名から下位 RFS ノードを自動入力することができます。設定の流れは下記の通りです。

1. **サービスモデル (YANG) の編集**
   Dispatch-map 用の定義を追加し、下位 RFS ノード用のパラメータ定義を削除します。
2. **XMLテンプレートの編集**
   Dispatch-map を参照し、デバイス名から RFS ノード名を取得できる様にします。
3. **パッケージの読み込み**
   NSO CLI にて packages reload コマンドを実行し、作成したサービスを適用可能な状態にします。
4. **Dispatch-map の作成**
   デバイス名とそれが接続する RFS ノードのマッピングを設定します。

Step1: サービスモデル (YANG) の編集

サービスモデル (upper-nso/packages/cfs-bgp/src/yang/cfs-bgp.yang) を編集し、
Dispatch-map 用の定義を追加し、デバイス名を Dispatch-map 参照に変更します。

```
19  revision 2016-01-01 {
20    description
21      "Initial revision.";
22  }
23
24  container devices {
25    list device {
26      key name;
        1 reference
27      leaf name {
28        type string;
29      }
30      leaf lower-node {
31        mandatory true;
32        type leafref {
33          path "/ncs:cluster/ncs:remote-node/ncs:name";
34        }
35      }
36    }
37  }
38
39  list cfs-bgp {
40    description "This is an RFS skeleton service";
41
```

追加

cfs-bgp.yang 設定例

```
52  // may replace this with othe
53  /*
54  leaf dev1-lower-nso {
55    type leafref {
56      path "/ncs:devices/ncs:device/ncs:name";
57    }
58  }
59  leaf dev2-lower-nso {
60    type leafref {
61      path "/ncs:devices/ncs:device/ncs:name";
62    }
63  }
64  */
65
66  // replace with your own stuff here
67  leaf dev1 {
68    type leafref {
69      path "/cfs-bgp:devices/cfs-bgp:device/cfs-bgp:name";
70    }
71  }
72  leaf dev2 {
73    type leafref {
74      path "/cfs-bgp:devices/cfs-bgp:device/cfs-bgp:name";
75    }
76  }
```

コメントアウト

Dispatch-map を参照に変更

編集が完了したら、upper-nso/packages/bgpmgr/src フォルダに移動し、make を実行し、
YANG をコンパイルします

Step2: XML テンプレートの編集

XML テンプレート (upper-nso/packages/cfs-bgp/templates/cfs-bgp-template.xml) の 下位 RFS ノード名に当たる部分を Dispatch-map にてデバイス名に対応する下位 RFS ノード名を取得するように変更します。

cfs-bgp-template.xml

developing-with-ncs > 22-lsa-single-version-deployment-test > upper-nso > pac

```
1  <config-template xmlns="http://tail-f.com/ns/config/1.0">
2    <devices xmlns="http://tail-f.com/ns/ncs">
3      <device>
4        <name>{/dev1-lower-nso}</name>
5      > <config> ⋯
13       </config>
14     </device>
15     <device>
16       <name>{/dev2-lower-nso}</name>
17     > <config> ⋯
25       </config>
26     </device>
27   </devices>
28 </config-template>
```

cfs-bgp-template.xml

ncs > 22-lsa-single-version-deployment-test > upper-nso > packages > cfs-bgp

```
1  <config-template xmlns="http://tail-f.com/ns/config/1.0">
2    <devices xmlns="http://tail-f.com/ns/ncs">
3      <device>
4        <name>{deref(dev1)/../lower-node}</name>
5      > <config> ⋯
13       </config>
14     </device>
15     <device>
16       <name>{deref(dev2)/../lower-node}</name>
17     > <config> ⋯
25       </config>
26     </device>
27   </devices>
28 </config-template>
```

サービスモデルは右図のような階層となっており、deref(dev1) は、dev1 のパスを取得し、同じ階層の lower-node の値を取得します。

```
$ pyang -f tree cfs-bgp.yang
module: cfs-bgp
  +--rw devices
  |  +--rw device* [name]
  |     +--rw name          string
  |     +--rw lower-node    -> /ncs:cluster/remote-node/name
```

Step3: パッケージの読み込み

CFS ノードにて packages reload コマンドを実行し、作成したサービスを再度読み込みます。

```
$ ncs_cli -C -u admin
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has completed.
>>> System upgrade has completed successfully.
reload-result {
    package cfs-bgp
    result true
}
admin@ncs#
admin@ncs# show packages package oper-status
packages package cfs-bgp
 oper-status up
admin@ncs#
```

Step4: Dispatch-map の作成

下記のように、デバイス名とそれが接続する RFS ノードのマッピングを設定します。

```
admin@upper-nso# config t
Entering configuration mode terminal
admin@upper-nso(config)# load merge terminal
Loading.
cfs-bgp:devices device ex0 lower-node lower-nso-1
cfs-bgp:devices device ex1 lower-node lower-nso-1
cfs-bgp:devices device ex2 lower-node lower-nso-1
cfs-bgp:devices device ex3 lower-node lower-nso-2
cfs-bgp:devices device ex4 lower-node lower-nso-2
(Ctrl+D)
0 bytes parsed in 7.03 sec (0 bytes/sec)
admin@upper-nso(config)# commit
Commit complete.
```

CFS ノードからのサービス設定（dispatch-map 利用時）

サービスパッケージ cfs-bgp を用いて、下位 RFS ノードを指定せずに、上位 CFS ノードから下記５つのパラメータで異なる RFS ルータに接続された 2 デバイスに対して iBGP 設定を実行できるようになりました。

```
admin@upper-nso(config)# cfs-bgp ex1_ex3 dev1 ex1 dev1-addr 10.1.1.1 dev2 ex3 dev2-addr 10.1.1.2 as 100
admin@upper-nso(config)# commit dry-run outformat native
native {
    device {
        name lower-nso-1
            --- omitted ---
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>ex1_ex3</name>
                    <neighbor>10.1.1.2</neighbor>
                    <routerid>10.1.1.1</routerid>
                    <as>100</as>
                    <device>ex1</device>
                  </mybgp>
                  --- omitted ---
    device {
        name ex1
        data router bgp 100
             bgp router-id 10.1.1.1
             neighbor 10.1.1.2 remote-as 100
             address-family ipv4 unicast
              neighbor 10.1.1.2 activate
              exit-address-family
```

デバイス ex1 が接続する RFS ノードを自動取得

```
--- omitted ---
device {
        name lower-nso-2
                <config>
                  <mybgp xmlns=" http://example.com/mybgp" >
                    <name>ex1_ex3</name>
                    <neighbor>10.1.1.1</neighbor>
                    <routerid>10.1.1.2</routerid>
                    <as>100</as>
                    <device>ex3</device>
                  </mybgp>
                  --- omitted ---
 device {
        name ex3
        data router bgp 100
             bgp router-id 10.1.1.2
             neighbor 10.1.1.1 remote-as 100
             address-family ipv4 unicast
              neighbor 10.1.1.1 activate
              exit-address-family
             !
            !
    }
}
admin@upper-nso(config)#
```

デバイス ex3 が接続する RFS ノードを自動取得

Trace ID

NSO 5.5 より、デフォルトで CLI を含む全ての Northbound からのリクエストに対して、Trace ID と呼ばれる固有の ID が自動付与されるようになりました。上位 CFS ノードの Trace ID と同じ ID が下位 RFS ノードでも使用されるため、トラブルシューティングがしやすくなっています。

```
admin@upper-nso(config)# cfs-bgp ex1_ex3 dev1 ex1 dev1-addr 10.1.1.1 dev2 ex3 dev2-addr 10.1.1.2 as 100
admin@upper-nso(config-cfs-bgp-ex1_ex3)# commit | details
applying transaction for running datastore usid=408 tid=1409 trace-id=85b4c053-0ad4-4b16-9ff9-eb8c6ce4fec9
```

upper-nso devel.log

```
<INFO> 11-Jan-2024::23:04:09.174 cw-nso ncs[3315334][<0.11538.1>]: ncs progress usid=408 tid=1409 datastore=running context=cli
trace-id=85b4c053-0ad4-4b16-9ff9-eb8c6ce4fec9 subsystem=device-manager device=lower-nso-1 push configuration: ok (0.573 s)
<INFO> 11-Jan-2024::23:04:09.174 cw-nso ncs[3315334][<0.11539.1>]: ncs progress usid=408 tid=1409 datastore=running context=cli
trace-id=85b4c053-0ad4-4b16-9ff9-eb8c6ce4fec9 subsystem=device-manager device=lower-nso-2 push configuration: ok (0.573 s)
```

lower-nso-1 devel.log

```
<INFO> 11-Jan-2024::23:04:09.048 cw-nso ncs[3315457][<0.10486.1>]: ncs progress usid=311 tid=1244 datastore=running
context=netconf trace-id=85b4c053-0ad4-4b16-9ff9-eb8c6ce4fec9 subsystem=device-manager device=ex1 push configuration: ok
(0.155 s)
```

lower-nso-2 devel.log

```
<INFO> 11-Jan-2024::23:04:09.048 cw-nso ncs[3315610][<0.5742.1>]: ncs progress usid=196 tid=504 datastore=running
context=netconf trace-id=85b4c053-0ad4-4b16-9ff9-eb8c6ce4fec9 subsystem=device-manager device=ex3 push configuration: ok
(0.241 s)
```

本日のトピック

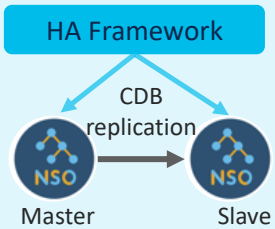
1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

堅牢性の向上のための HA 構成

一部のノードで障害発生した場合でも、システムを稼働させ続けるために、商用環境では HA 構成をとることを推奨しています。NSO 6.1.2 以降ではより堅牢な Raft HA をサポートしています。

NSO 5.3 以前

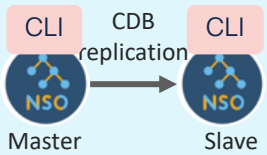
NSO ノードの役割 (Master/Slave) 毎の動作についてのみ内蔵。HA Framework API を使用した制御アプリケーションの実装が必要。Ciscoは、実装例として、tailf-hccパッケージを公開。



① 外部アプリによるHA

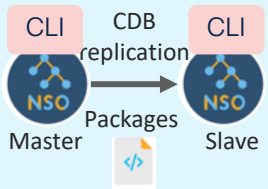
NSO 5.4

Master / Slaveの切替機能をCLIにより実行可能に。Tailf-hcc 5.0がリリースされ、L3冗長の設定が簡易になりました。L2/L3冗長には、制御パッケージが必要です。



NSO 5.6 / 5.7

Master / Slave間でのパッケージのSyncが可能に。パッケージアップグレードもシンプル化。事前設定ルールに従い、Split Brainを回避する仕組みを導入。



② ルールベースHA

NSO 6.1.2 以降

推奨

Raftアルゴリズム (HAクラスター参加ノード間での投票でリーダー選出や変更を実行する) をベースとしたHAをサポート。 HAクラスターは、最小３台構成。

③ Raft HA

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

Raft アルゴリズム

Raft HA は下記のようなコンセンサスベースの Raft アルゴリズムを採用し、 Split Brain の問題に対処します。リーダー選出・変更反映には過半数の合意が必要なため、最小 3 台構成となります。

Raft アルゴリズムの概要

リーダー選挙 (Leader Election)

- クラスター内で**過半数の票**を獲得した 1 つのノードがリーダーとなり、それ以外はフォロワーとなる
- リーダーは定期的に heartbeat をフォロワーに送り、フォロワーは election timeout 内に heartbeat を受信できない場合、リーダー候補者となり選挙を開始する
- リーダー選挙開始毎に Term が +1 される
- Election timeout は、指定範囲 (ex. 150-300ms) 内でノード毎に都度ランダムに設定される

ログ複製 (Log Replication)

- 変更はまずリーダーのログにエントリとして追加される
- リーダーからheartbeat を利用しフォロワーに複製が送信され、**過半数がそれを承認**すると変更が反映される
- 変更反映後、リーダーはフォロワーにその旨を通知する

Raft HA のメリット

- ネットワーク分断による Split Brain の状態に陥っても、**過半数を満たすグループのみを機能**させることができる
- 分断解消後は、自動でクラスターが再構成される
- Term の値が大きい方のリーダーの変更を全体に適用することで、クラスター内の**データの一貫性**を保つことができる

Raft HA のデメリット

- **最小 3 ノード**が必要
  - 2 ノード構成の場合、過半数は 2 のため、 1 ノードに障害が発生するとクラスターが機能しない
- 2 ノードの同時障害に対応するには、最小 5 ノード必要

Raft Consensus Algorithm: https://raft.github.io/
Raft Visualization: https://thesecretlivesofdata.com/raft/

Raft HA のセットアップ

Raft HA のセットアップは、下記の流れで実施します。

1. 各ノードの設定更新

   クラスターの各参加ノードの設定 ncs.conf を更新し、HA Raft を有効化、SSL/TLS パラメータなどを設定します。

2. クラスター作成アクションの呼び出し

   リーダーとしたいノードで ha-raft create-cluster アクションを実行し、クラスターを初期化します。

① 各ノードの設定更新 (基本設定)

下記のように、各参加ノードに対して ncs.conf を設定します。cluster-name / seed-node は
ノード間で一致させる必要があります。また、HA ノード間の通信は SSL/TLS がデフォルトで
有効化されています。

ncs.conf (抜粋)

```
<ha-raft>
  <enabled>true</enabled>                                              1. Raft HA の有効化
  <cluster-name>sherwood</cluster-name>                                2. クラスター名（クラスター識別名）
  <listen>
    <node-address>ash.example.org</node-address>                       3. ノードアドレス（ノード識別名）
  </listen>
  <ssl>                                                                4. SSL/TLS 通信の設定
    <ca-cert-file>${NCS_CONFIG_DIR}/dist/ssl/cert/myca.crt</ca-cert-file>   4-1. CA 証明書ファイルの指定
    <cert-file>${NCS_CONFIG_DIR}/dist/ssl/cert/ash.crt</cert-file>      4-2. ノード証明書ファイルの指定
    <key-file>${NCS_CONFIG_DIR}/dist/ssl/cert/ash.key</key-file>        4-3. 秘密鍵ファイルの指定
  </ssl>
  <seed-nodes>
    <seed-node>birch.example.org</seed-node>                           5. 起動時に接続するノードのアドレス
  </seed-nodes>
</ha-raft>
```

① 各ノードの設定更新 (詳細)

1. Raft HA の有効化: ncs-config/ha-raft/enabled

Raft HA はデフォルトで無効化されているため、true で有効化します。

2. クラスター名: ncs-config/ha-raft/cluster-name (string)

クラスターを識別するための名前です。クラスター名が一致しないノードからの接続は拒否されます。LSA 構成などで複数のクラスターが稼働している際に意図しないクラスターにノードが接続されるのを防ぎます。

3. ノードアドレス: /ncs-config/ha-raft/listen/node-address

ノードを識別するための名前で、n1.acme.com、10.45.22.11 などの形式で指定します。少なくとも一つはドット(.) が含まれている必要があります。同じホストで複数の HA ノードを稼働させる場合は、 ncsd@n1.acme.com な形でノード ID を@で指定することもできます。

① 各ノードの設定更新 (詳細)

4. SSL/TLS 通信の設定: /ncs-config/ha-raft/ssl

ノード間の通信は、SSL/TLS で保護されます。プライベート認証局 (CA) の立ち上げやノード証明書の作成などに関しては、NSO インストールディレクトリ内の examples.ncs/high-availability/raft-cluster にある openssl コマンド実行例をご参照ください。

ノードアドレスを IP アドレスで設定した場合は、証明書のサブジェクト代替名 (subjectAltName; SAN) に IP アドレスを追加する必要があります。

5. 起動時に接続するノードのアドレス: /ncs-config/ha-raft/seed-nodes

seed-nodes に1 つ以上のノードを指定するとノードは、クラスターに参加する可能性のあるノードリストを取得するために、このノードへの接続を試行します。完全なノードリストを取得するためには、他のすべてのノードは同じシードノードを設定している必要があります。

① 各ノードの設定更新 (その他)

Raft HA クラスター内のノードは互いに、他のノードの下記ポートへの TCP 接続ができる必要があります。ポート4369 以外は ncs.conf で範囲の変更が可能です。

- 4369
- 4370 ～ 4399 (変更可能)

ncs.conf (抜粋)

```
<ha-raft>
  <listen>
    <min-port>4370</min-port>
    <max-port>4399</max-port>
  </listen>
</ha-raft>
```

また、/ncs-config/cdb/operational/enabled および ncs-config/rollback/enabled はレプリケーションの動作に影響を与えるため、ノード間で一致する必要があります。

/ncs-config/encrypted-strings 内の値もリーダー切替わり後にデバイスと接続できない問題を回避するためにノード間で同じ値の設定が必要です。

② クラスター作成アクションの呼び出し

リーダーとしたいノードで、ha-raft create-cluster アクションを実行し、クラスターを初期化します。seed-nodes を適切に設定していると、下記のようにメンバーノードの候補が表示されます。

```
admin@ncs# ha-raft create-cluster member [ ?
Possible completions:
  ] birch.example.org cedar.example.org
admin@ncs# ha-raft create-cluster members [ birch.example.org cedar.example.org ]
admin@ncs# show ha-raft
ha-raft status role leader
ha-raft status leader ash.example.org
ha-raft status member [ ash.example.org birch.example.org cedar.example.org ]
ha-raft status connected-node [ birch.example.org cedar.example.org ]
ha-raft status local-node ash.example.org
...
```

トラブルシューティング

Raft HA 関連のエラーメッセージおよび Raft アルゴリズムの全体的な動作に関する詳細情報は、logs/raft.log に出力されます。デフォルトで有効になっており、ncs.conf で出力レベルを変更できます。

ncs.conf (抜粋)

```
<raft-log>
  <enabled>true</enabled>
  <file>
    <name>${NCS_LOG_DIR}/raft.log</name>
    <enabled>true</enabled>
  </file>
  <level>info</level>
</raft-log>
```

Raft HA は、変更反映に過半数の同意が必要なため、クラスターノード間の往復遅延が長いと、トランザクションスループットに悪影響を及ぼす可能性があります。

メンバーノードの追加・削除

リーダーは、ha-raft adjust-membership アクションを用いて、クラスターに新しいノードを追加したり、削除したりできます。削除されたノードは削除されたことを認識せず、candidate ステータスとなりリーダー選挙を開始します。そのため、削除したノードはシャットダウンする必要があります。

```
admin@ncs# show ha-raft status member
ha-raft status member [ ash.example.org birch.example.org cedar.example.org ]
--- ノード削除 ---
admin@ncs# ha-raft adjust-membership remove-node birch.example.org
admin@ncs# show ha-raft status member
ha-raft status member [ ash.example.org cedar.example.org ]
--- ノード追加 ---
admin@ncs# ha-raft adjust-membership add-node dollartree.example.org
admin@ncs# show ha-raft status member
ha-raft status member [ ash.example.org cedar.example.org dollartree.example.org ]
```

リーダーの再選出

Raft HA はリーダーが自動で決定します。リーダーを再選出したい場合、もしくは特定のノードにリーダーを任せたい場合は、リーダーノードにて、ha-raft handover アクションを実施します。

```
admin@ncs# show ha-raft status leader
ha-raft status leader ash.example.org
admin@ncs# ha-raft handover to-member dollartree.example.org
admin@ncs# show ha-raft status leader
ha-raft status leader dollartree.example.org
```

ha-raft handover を引数なしで実行すると、リーダー選挙の開始がトリガー （ランダムにリーダーが決定）されます。

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

ルールベース HA とは

ルールベース HA (組み込み HA) は、NSO 5.4 で追加された事前に定義されたルールに従って HA クラスターを管理する機能です。ncs.conf にクラスター参加時やフェールオーバー時の動作を事前に設定しておきます。

None

アップグレード時などに利用するロール
ローカルに保存された CDB の読み取り・書き込み可能

ルールベース HA のセットアップ 1

各メンバーノードに対して、下記のように ncs.conf の設定を行います。

ncs.conf (抜粋)

```
<ha>
  <enabled>true</enabled>
  <ip>192.168.1.1</ip>
  <port>4570</port>
  <extra-listen>
    <ip>fe80::1</ip>
    <port>4569</port> >
  </extra-listen>
  <tick-timeout>PT20S</tick-timeout>
</ha>
```

1. HA を有効化
2. HA ノード間の接続に使用する IP アドレス
3. HA ノード間の接続に使用するポート番号
4. HA ノード間の接続に使用する追加の IP アドレス
5. HA ノード間の接続に使用する追加の ポート番号
6. 死活監視の間隔（デフォルト20秒）

   Primary – Secondary 間でデフォルトで20秒間隔で tick message を送ります。Tick-timeout の 3 倍の時間以内に tick message を受信しなかったら、そのノードは停止しているとみなされます。

ルールベース HA のセットアップ 2

各メンバーノードに対して、下記のように設定を行います。

```
admin@n1# show running-config high-availability | display xml
  <high-availability xmlns=" http://tail-f.com/ns/ncs" >
    <token>$9$C8FdSyaPbs4w84a81DWI6Z/4clZLAO0=</token>
    <ha-node>
      <id>n1</id>
      <address>192.168.1.1</address>
      <nominal-role>primary</nominal-role>
    </ha-node>
    <ha-node>
      <id>n2</id>
      <address>192.168.1.2</address>
      <nominal-role>secondary</nominal-role>
      <failover-primary>true</failover-primary>
    </ha-node>
    <ha-node>
      <id>n3</id>
      <address>192.168.1.3</address>
      <nominal-role>secondary</nominal-role>
      <failover-primary>false</failover-primary>
    </ha-node>
```

config (抜粋)

- 1. HA クラスター内で共有するトークン
- 2. HA クラスターメンバー情報
  - 2-1. ノードの識別子
  - 2-2. ノードの IP アドレス
  - 2-3. 名目上のノードの役割
  - 2-4. フェールオーバー時にプライマリになる（1 ノードだけ設定可）

ルールベース HA のセットアップ 3

各メンバーノードに対して、下記のように設定を行います。

```
<settings>
  <enable-failover>true</enable-failover>                    3. フェールオーバーを自動で実施
  <reconnect-secondaries>true</reconnect-secondaries>        4. フェールオーバー前にプライマリに再接続を試行
  <start-up>
    <assume-nominal-role>true</assume-nominal-role>          5. nominal-role を動作ロールと仮定してクラスタに参加
    <join-ha>true</join-ha>                                  6. クラスター参加時に Primary への接続を試行
  </start-up>
  <reconnect-interval>10</reconnect-interval>                7. Primary への接続試行間隔
  <reconnect-attempts>6</reconnect-attempts>                 8. Primary への接続試行回数
  <consensus>
    <enabled>true</enabled>                                  9. Split Brain の発生を回避するアルゴリズムの有効化
    <algorithm>rule-based</algorithm>                        10. ルールベースのコンセンサスアルゴリズムを使用
  </consensus>
 </settings>
</high-availability>
```

config (抜粋)

フェールオーバー時の動作 (NSO 5.7 以降でサポート)

- **Failover Primary:**
  Primary になるが、読み取り専用モードが有効になる。Secondary が参加したら、読み取り専用を無効にする。
- **名目上の Primary:**
  すべての Secondary への接続が失われると、ロールを none に変更。1 つでも Secondary ノードが接続されている場合は、Primary のままになる。

起動時の動作

起動時の動作は、ncs.conf の high-availability/settings/start-up の設定に従って実行されます。

assume-nominal-role	join-ha	nominal-role	起動時の動作
true	false	primary	primary となる
true	false	secondary	nominal-role: primary のノードに接続を試みる
true	false	none	何も役割を引き受けない
false	true	primary	現在の primary への接続を試行し、secondary としてクラスターに参加
false	true	secondary	現在の primary への接続を試行し、secondary としてクラスターに参加
false	true	none	何も役割を引き受けない
true	true	primary	現在の primary への接続を 1 回だけ実施。接続可能な場合は secondary として、接続不可な場合は primary としてクラスターに参加
true	true	secondary	現在の primary への接続を試行し、secondary としてクラスターに参加
true	true	none	何も役割を引き受けない
false	false	-	何も役割を引き受けない

HA 有効化 / ステータスの確認

各メンバーノードで下記のコマンドを実行し、 HA を有効化します。

```
admin@n1# high-availability enable
result enabled
```

show high-availability により、ルールベース HA の各種ステータスを確認できます。

```
admin@n1# show high-availability
high-availability enabled
high-availability status mode primary
high-availability status current-id n1
high-availability status assigned-role primary
high-availability status read-only-mode false
ID  ADDRESS
---------------
n2  192.168.1.2
n3  192.168.1.3

admin@n2#
```

ルールベース HA の制御

下記のように、ロールの切替え、HA 有効化 / 無効化、読み取りモードへの移行などを行うことができます。

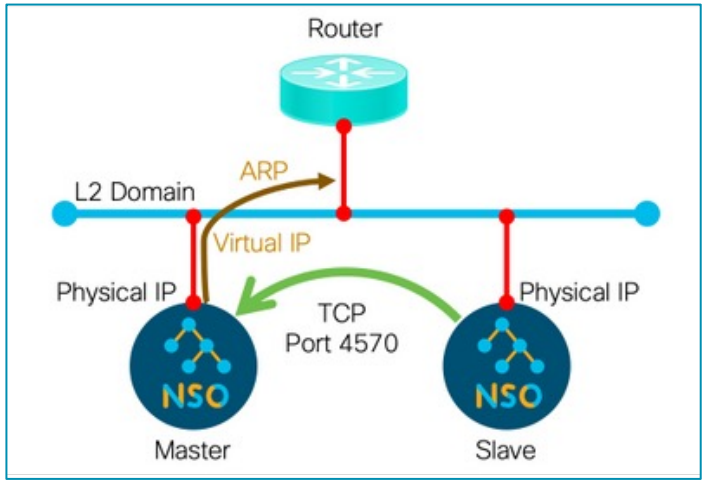
```
admin@n1# high-availability ?
Possible completions:
  be-none              Order the local node to assume ha role none
  be-primary           Order the local node to assume ha role primary
  be-secondary-to      Order the local node to connect as secondary to the provided ha node
  disable              Disable NCS built in HA and assume a ha role none
  enable               Enable NCS built in HA and optionally assume a ha role according to
                       /high-availability/settings/start-up/ parameters
  local-node-id        Identify the which of the nodes in /high-availability/ha-node (if any)
                       corresponds to the local NCS instance
  read-only            Toggle read-only mode, if the mode is 'true' no configuration changes can occur
admin@n1#
```

本日のトピック

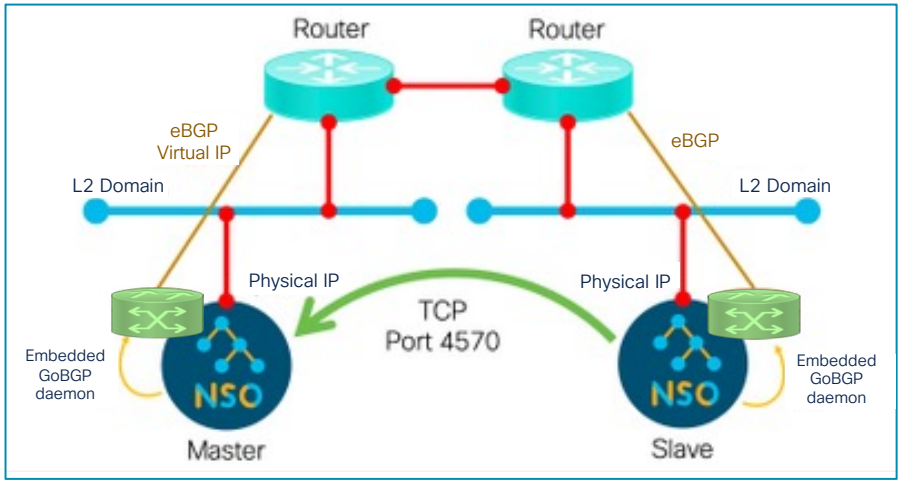
1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

Tail-f HCC パッケージ

Tail-f HCC パッケージは、NSO HA クラスターの Primary ノードへ接続する仮想 IP アドレス (VIP) を提供します。バージョン 6.0.0 より、Raft HA にも対応しています。
L2 HA / L3 HA の 2 つの構成をサポートしています。



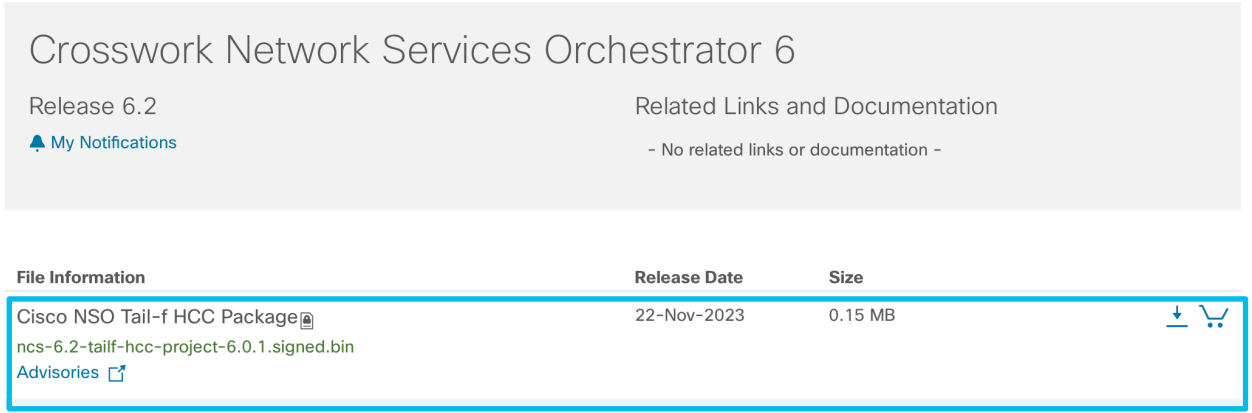
L2 HA



L3 HA

Tail-f HCC パッケージ

Tail-f HCC パッケージは、Cisco Software Download (https://software.cisco.com/download/home) よりダウンロード可能です。



パッケージの利用方法に関しては、下記をご参照ください。
https://developer.cisco.com/docs/nso/guides/#!high-availability/tail-f-hcc-package

本日のトピック

1. LSA とは
2. LSA のセットアップ
3. LSA におけるサービス作成
4. NSO HA の種類
5. Raft HA
6. ルールベース HA
7. Tail-f HCC パッケージ
8. デモ

本日のポイント

01 Layered Service Architecture (LSA) により、NSO を階層化し、スケールアウトさせることができます。

02 障害時もシステムを稼働させ続けるために、NSO ではバージョンに応じて、いくつかの HA 構成がサポートされています。

03 Raft HA は、リーダー選出・変更反映に過半数の合意が必要なため、最も堅牢性の高い構成です。

04 ルールベース HA は、事前に設定したルールに従い、プライマリの選択、クラスターの形成、およびフェールオーバーが実行される構成です。

05 NSO HA クラスターの Primary ノードへ接続する仮想 IP アドレス (VIP) を設定するには、 Tail-f HCC パッケージを利用します。

参考リンク

- NSO Layered Service Architecture: LSA Overview (英語)
  https://developer.cisco.com/docs/nso/guides/#!lsa-overview
- NSO Administration Guide: High Availability (英語)
  https://developer.cisco.com/docs/nso/guides/#!high-availability

examples.ncs

NSO 学習用コンテンツが、インストールディレクトリの examples.ncs フォルダに格納されています。実際に手を動かして LSA / HA 構成を学ぶことができます。

<LSA 関連>
- examples.ncs/getting-started/developing-with-ncs/22-lsa-single-version-deployment
- examples.ncs/getting-started/developing-with-ncs/24-layered-service-architecture-scaling
- examples.ncs/getting-started/developing-with-ncs/28-lsa-multi-version-deployment

<HA 関連>
- examples.ncs/high-availability/raft-cluster – Raft HA
- examples.ncs/development-guide/high-availability – ルールベースHA


Cisco
Customer Experience