NSO Orchestration for 4G CUPS

Revision History

Revision Details	Release
First introduced.	21.25

Feature Description

The Cisco Network Service Orchestrator (NSO) based VNF orchestration enables you to manage the lifecycle of newly created Virtual Network Function (VNF) devices such as CP, UP, and RCM.

The Cisco NSO Orchestration for 4G CUPS solution provides the following functions:

- Instantiation via NSO CLI, Web-Interface, or NSO RESTCONF API
- Onboarding of VNF devices such as CP, UP, and RCM upon successful instantiation
- Pushing of Day-0.5, and Day-1 CUPS configuration after successful instantiation
- Decommission of the VNF devices

Use Cases

The NSO orchestration solution caters to the following use cases:

1. Instantiation of new CP, UP, and RCM

Instantiating new 4G-based VNFs (CPs, UPs, or RCMs) for CUPS. CP can be a Virtualized Packet Core-Single Instance (VPC-SI) or Virtualized Packet Core-Distributed Instance (VPC-DI), but UP can only be a VPC-SI.

Users are notified if there are any failures.

2. Termination of CP, UP, and RCM

   Terminating 4G-based VNFs (CPs, UPs, or RCMs) for CUPS.

   Users are notified if there are any failures.

3. Updating current status on the VNF dashboard

   Providing the current status on the dashboard of VNFs.

4. Configuration of logical group for CP, UPs, and RCMs

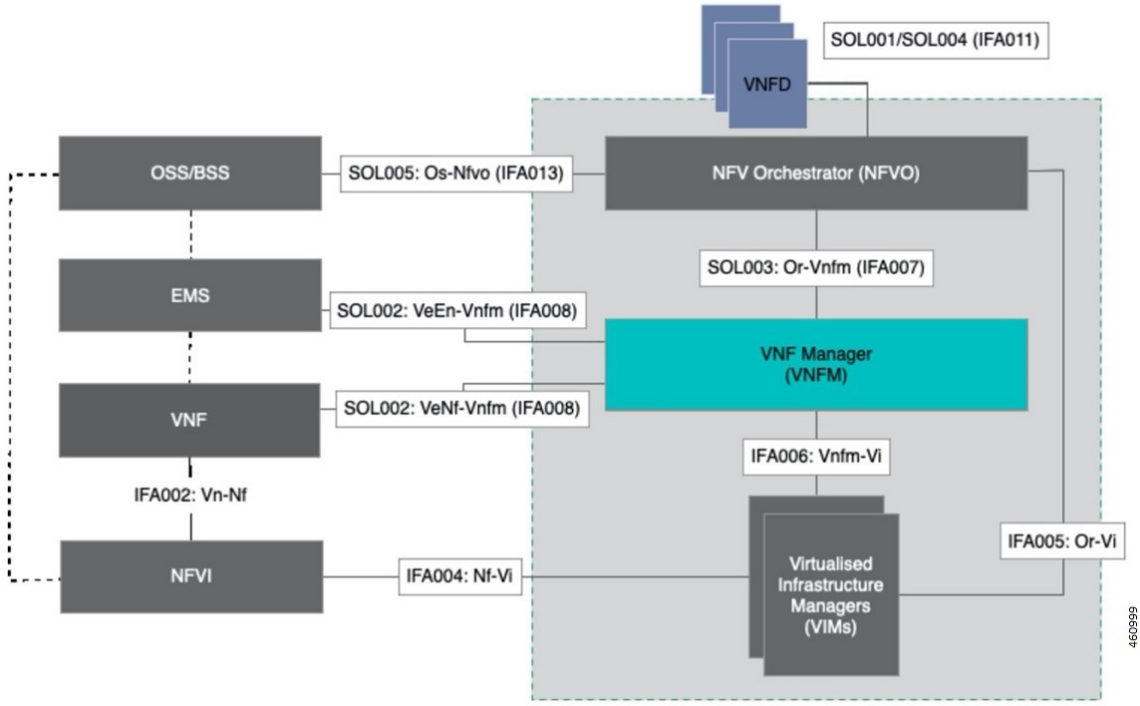
   Configuring device group in NSO to group the CP, UPs, and RCMs, and adding the corresponding VNFs to the device group.

How it Works

Architecture

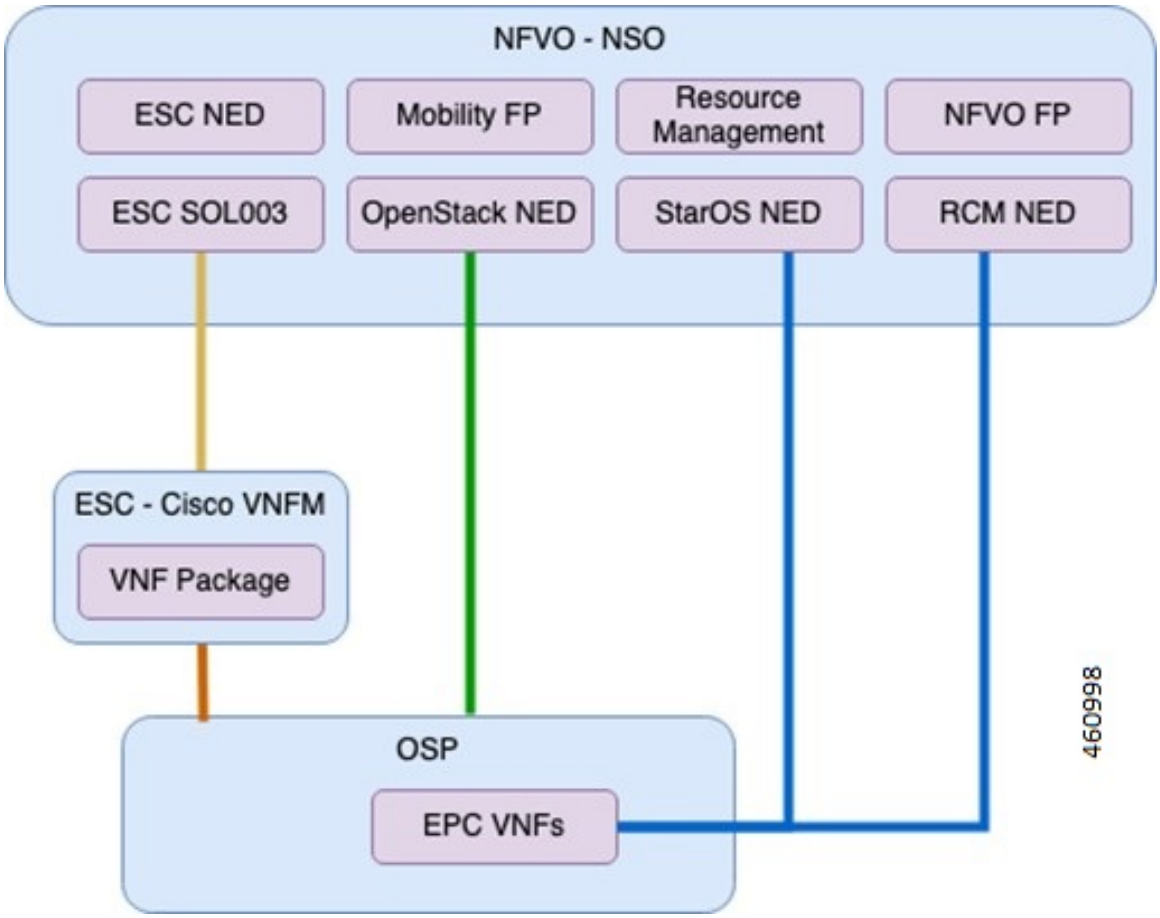
The Cisco NSO orchestration engine software modules handle the Network Functions Virtualization Orchestrator (NFVO) functions. The NFV solution follows the ETSI NFV Management and Orchestration (MANO) model, as shown in the following figure.

Figure 1: NFV Solution Architecture



The following diagram illustrates, at a high-level, the components and frameworks involved in the solution.

Figure 2: NFV Solution Components



Components

The following are some of the important components of NSO:

- **Cisco NFVO Functional Pack**:

  Cisco NFVO Functional Pack contains the YANG models according to the MANO specification (SOL006).

  Cisco NFVO Functional Pack contains models for **cisco-etsi-nfvo**, which implements the instantiation logic of MANO descriptors on VNF Managers (VNFM) and OpenStack. Virtual Network Function (VNF) and Network Service (NS) are the main services in this package. Northbound users interact with these services to start VNFs or network services.

  It also includes models for **cisco-etsi-nfvo-ro**, which contains the Resource Orchestration (RO) functionality. Resource Orchestration manages the allocation of physical resources in the Virtualized Infrastructure Managers (VIMs). These physical resources are used by a VNF or an NS.

- **StarOS NED for NSO**:

  StarOS-based Network Element Driver (NED) interfaces with the Cisco 4G CUPS VNFs for configuration push.

- **RCM NED for NSO**:

RCM-based NETCONF NED is used to establish communication between NSO and RCM devices.

- **Cisco ESC SOL003 NED**:

  This NED is used for ETSI SOL3 compliant devices. Elastic Services Controller (ESC) is also added as SOL3 compliant device to NSO.

- **NFV Apps Mobility Package:**

  This is a custom package that provides VNF life-cycle management, and VNF dashboard update.

Minimum Platform and Software Requirements

The following are the minimum platform and software requirements to support NSO Orchestration:

- Supported VIM: OpenStack
- Supported VNFM: Cisco ESC
- Supported Orchestrator: NSO
- Network Elements:
  - RCM
  - VPC-SI (UP/CP)
  - VPC-DI (CP)

Table 1: Software Versions

Software	Minimum version
Redhat OpenStack	13 (Queens) **Note** VMWare or OSP 16 isn't supported or validated.
Cisco ESC	5.5.0
Cisco NSO	5.5.2
OpenStack NED	4.2.23
ESC NED	5.3.0.94
StarOS NSO NED	5.38.1
Cisco NFVO FP	4.6.0
Mobility FP	3.0.0
NSO Resource Management	3.5.2
Cisco NSO HCC	5.0.0

This feature supports the following ETSI MANO specifications:

Table 2: ETSI MANO Specifications

Specification	Supported Version	Description
SOL001	v2.5.1	Defines the format and structure for the VNF Descriptor
SOL003	v2.4.1	Defines all interactions over the Or-Vnfm reference point

Network and Hardware Requirements

Network Requirements:

The following table demonstrates the NSO and ESC network requirements:

Table 3: NSO and ESC Network Requirements

Application	Management IP	Orchestration	Connection between HA Pair
NSO (2 VMs + VIP)	3	3	L2 connection of 100 Mbps with latency less than 30 ms
ESC (2 VMs + VIP)	3	3	L2 connection of 100 Mbps with latency less than 30 ms

Hardware Requirements

The following table demonstrates the specifications for NSO and ESC Virtual Machine to support maximum of 250 VNFs.

Table 4: NSO and ESC VM Specifications

Application	Number of VMs	VM CPU Cores	VM RAM	VM Storage	VM Connectivity
NSO	2	8	16 GB RAM baseline + 10 MB RAM for every StarOS device to be supported	100 GB disk (preferably SSD)	One 10 Gbps network link
ESC	2	4	16 GB	100 GB	

Licensing

The NSO Orchestration for 4G CUPS is a licensed Cisco feature. Contact your Cisco Account representative for detailed information on specific licensing requirements.

Call Flows

This section describes the key call flows for the 4G CUPS orchestration functionality.

VNF Onboarding

This section describes the VNF Onboarding flow.

Figure 3: VNF Onboarding

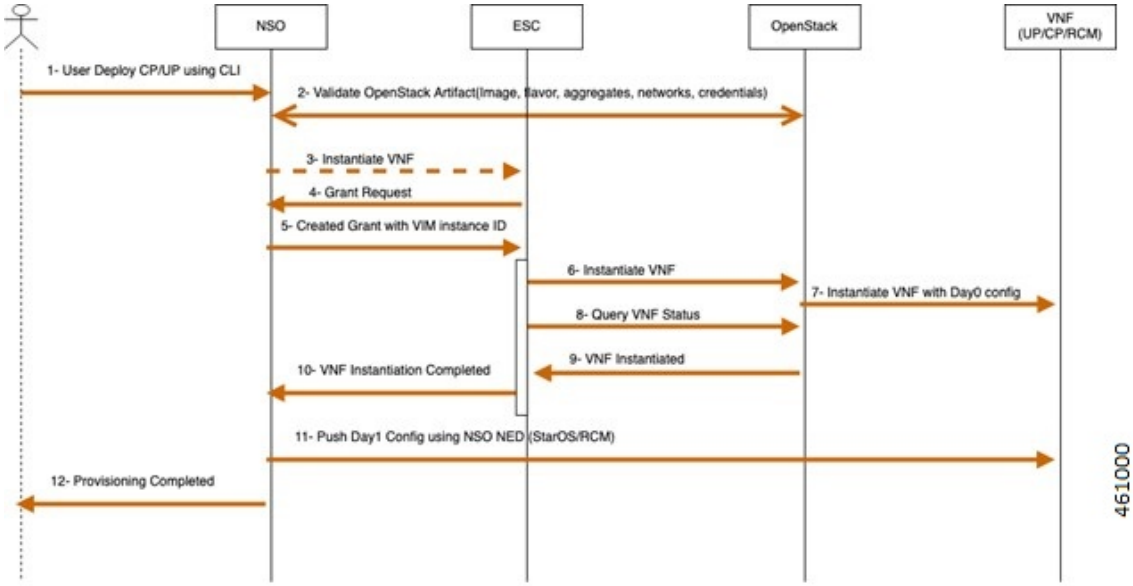


Table 5: Call Flow Description

Step	Description
1	Network operator uses NSO CLI to instantiate a VNF (CP, UP, or RCM). This includes the VIM ID to host the VNF, and the ESC.
2	NSO validates the data provided by the user via OpenStack.
3	NSO sends a SOL.003 request to instantiate the VNF on the ESC.
4	ESC sends a Grant Request to the NSO.
5	NSO sends a resource grant message to the ESC with VIM InstanceId.
6	ESC uses OpenStack API to instantiate the VNF.
7	OpenStack brings up the VNF.
8	ESC queries the OpenStack about the VNF status.
9	OpenStack replies with VNF-Up message.
10	ESC notifies the NSO about VNF instantiation.
11	NSO pushes Day-1 configuration onto the VNF.
12	NSO notifies the Operator that the VNF provisioning is complete.

P2P Module Installation

The mobility function pack supports installing a P2P module as part of VNF deployment. The P2P module is installed after the device is onboarded. The P2P module file must be uploaded to the NSO prior to the VNF deployment. The configurable parameters indicate whether P2P installation is required, and the file location.

VNF Termination

This section describes the VNF Termination flow.

Figure 4: VNF Termination Flow

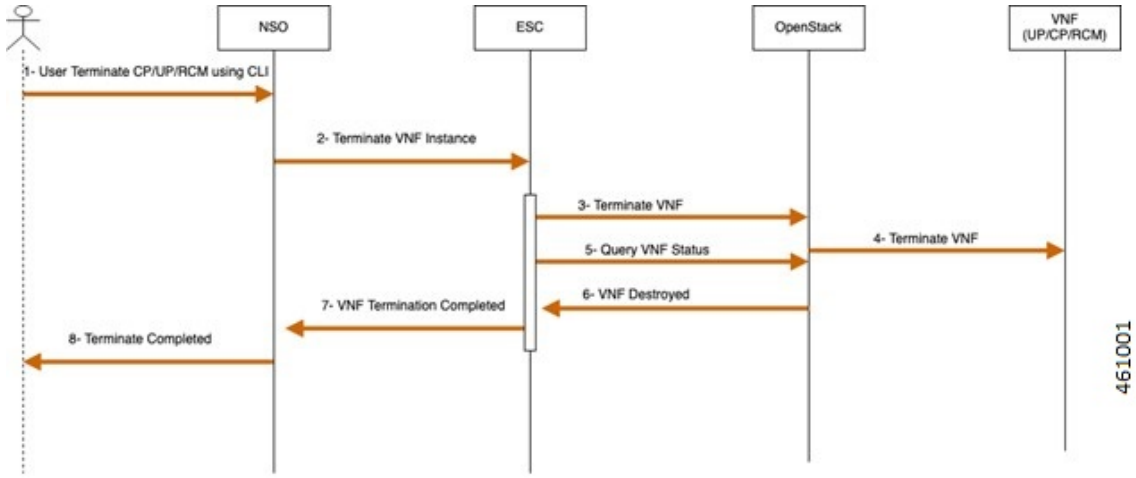


Table 6: Call Flow Description

Steps	Description
1	Operator uses NSO CLI to terminate a VNF (CP, UP, or RCM). This includes the VNF ID, VIM ID to host the VNF, and the ESC.
2	NSO sends a SOL.003 request to terminate the VNF on the ESC.
3	ESC uses OpenStack API to terminate the VNF.
4	OpenStack terminates the VNF.
5	ESC queries the OpenStack about the VNF status.
6	OpenStack replies with a VNF Destroyed message.
7	ESC notifies the NSO about VNF termination.
8	NSO notifies the Operator that the VNF termination is complete.

Recovery

Auto-healing isn’t supported currently.

To recover from fault state to previous state, perform the following steps:

- Cancel or terminate the VNF instantiation. The system returns to its original state
- Cancel or recreate VNF termination process. The system returns to its original state

Limitation

The NSO Orchestration for 4G CUPS feature has the following limitation in this release:

- Production NSO instance can run only on popular Linux flavors (for example, RedHat, Cisco Linux, Ubuntu, CentOS, and so on).
- A VNF deployment may fail if the NSO/ESC instance handling the deployment goes down. This is applicable for both ESC/NSO HA as well as standalone ESC/NSO deployments. Operator intervention is required depending on the exact nature of the failure. In case of deployment followed by automated configuration push, it's possible that the deployment succeeds but the subsequent configuration push fails depending on the timing of the NSO failure.

Installing NSO Packages

The NSO Orchestration solution uses a collection of NEDs and other NSO packages. The following is a detailed list of various packages and their roles. For installation instructions of these packages, see the "Packages" chapter in the *NSO Administration Guide* for the appropriate NSO version.

1. **NSO NED packages:**

   Most of the NSO NED packages are published for downloading independently. Contact your Cisco representative for the details on how to download.

   cisco-rcm-1.0.tar.gz - RCM NETCONF-based NED for RCM device communication from NSO

   ncs-5.5.2-cisco-staros-5.38.tar.gz - CLI-based NED for StarOS device (SI or DI) communication from NSO

   ncs-5.5.1-etsi-sol003-1.13.14.tar.gz - ETSI SOL003 based NED for ESC communication from NSO

   ncs-5.5.1-openstack-cos-4.2.23.tar.gz - Openstack NED for Openstack communication from NSO

   ncs-5.5.2.3-cisco-etsi-nfvo-4.6.0.tar.gz - NETCONF-based NED for ESC communication from NSO

   ncs-5.5.2.3-esc-5.5.0.57.tar.gz - ETSI SOL-based NED for ESC communication from NSO

2. **NSO Custom packages:**

   These are custom-built packages for Mobility VNF orchestration. NSO custom packages are bundled in the mobility function pack tar archive.

   mobility-common.tar.gz - Common package for config and device metadata

   nfv-common.tar.gz - Common packages for VNF orchestration-related common utilities

   nfv-device-onboarding.tar.gz - Package to support NSO device onboarding

   nfv-vim.tar.gz - Package for Openstack related precheck functionality

   nfv-vnf-lcm.tar.gz - Package for VNF Instantiation and termination logic

mop-common.tar.gz - Common packages for config MOP-related common utilities

mobility-mop.tar.gz - Package for Mobility MOP Config Push

3. **VNF Packages required for Orchestration (SOL003/SOL004):**

   These are VNF packages that are used for onboarding a specific VNF. These packages are provided only as guidelines. Mostly, a given package is customized to suit the deployment environment.

   VPC-SI-2P-IMAGE-BOOT - Reference SOL003/SOL004 CSAR package for SI instantiation

   RCM-IMAGE-BOOT - Reference SOL003/SOL004 CSAR package for RCM instantiation

   VPC-DI-2P-1DI-ENCRYPTVOLBOOT - Reference SOL003/SOL004 CSAR package for VPC DI instantiation with two CF and four SF. SF has two service networks.

   VPC-DI-2P-1DI-ENCRYPTVOLBOOT-LTD - Reference SOL003/SOL004 CSAR package for VPC DI instantiation with two CF and two SF. SF has two service networks.

   VPC-DI-2P-1DI-ENCRYPTVOLBOOT-LTD-1S-NETWORK - Reference SOL003/SOL004 CSAR package for VPC DI instantiation with two CF and two SF. SF has only one service network.

   create-zip.sh - Shell script to rebuild the SOL003 package, if there are any changes to SOL001 definitions or Day-0 scripts.

VNF Orchestration/Deployment and Automatic Configuration Management

This solution includes the following tasks:

- Pre-population of config metadata for VNF orchestration.
- Orchestration/Deployment of VNFs (CP, UP, or RCM)
- Automatic device onboarding post VNF deployment
- Post-deployment automatic configuration push

Pre-population of Config Metadata for VNF Orchestration

Pre-population of Config Metadata is important to achieve any post-deployment configuration push from NSO in an automated mode. If there are no prepopulated data for this device, NSO instantiates the VNF and on-boards as a device in NSO.

Prepopulating of config metadata has the following structure, and population of this data is based on the network scheme and data set:

```
container
metadata-store {
         list config-metadata {
             key device-name;
             leaf device-name {
                 tailf:info "onboarding device name";
                 type string;
             }
             leaf redundancy_scheme {
```

```
                tailf:info "cluster-topology 1:1, N:M and N+2";
                type string;
            }
            leaf device-type {
                tailf:info "Onboarding device type vpc or rcm";
                type string;
            }
            list attributes {
                key attribute-name;
                leaf attribute-name {
                    tailf:info "Attribute Name";
                    type string;
                }
                leaf attribute-value {
                    tailf:info "Attribute Value";
                    type string;
                }
            }
            list configuration-type {
                key config-type;
                tailf:info "Configuration type Day0.5, Day1 or DayN";
                leaf config-type {
                    type string;
                }
                list files {
                    key file-name;
                    tailf:info "file name";
                    leaf file-name {
                        type string;
                    }
                    leaf config-scheme {
                        type string;
                    }
                    // CP device info
                    list additional-files {
                        key device;
                        //cp device
                        leaf device {
                            tailf:info "device name";
                            type string;
                        }
                        list additional-file {
                            key additional-file-name;
                            leaf additional-file-name {
                                tailf:info "file name";
                                type string;
                            }
                        }
                    }
                }
            }
        }
    }
```

The following table provides a description of the parameters:

Parameter	Description
device-name	Name of the NSO device corresponding to the VNF. Same as VNF name.
redundancy_scheme	Type of redundancy scheme. N + 2 is standalone (no redundancy)
device-type	vpc (for SI and DI) or rcm (for RCM)

Parameter	Description
configuration-type	Day-0.5 is a special configuration for N:M redundancy. This configuration enables the UP to contact the RCM. This configuration is expected to be saved persistently.
	Day-1 is the bulk of the configuration
	Day-N generally changes to a working configuration. Does not apply to NSO Orchestration flows.
file-name	Primary configuration file(s) to be pushed
config-scheme	This parameter can have one of the following values: **Common**: Configuration is pushed to all UPs regardless of role (Active or Standby). **host-specific**: This scheme is similar to “Common”, as the configuration is pushed to all UPs (Active or Standby). However, the configuration is pushed only after “common” configuration. This enables you to provide any configuration that is dependent on the “common” configuration. For example, the control-plane group configuration. **allHostSpecific**: Contains the union of host-specific configurations for all active UPs. The configuration is pushed to all the standby UPs for N:M. **"Active1", "Active2",... "ActiveN”**: Host-specific configuration for the respective active UP. It is pushed only to the specific UP. **"Active1-rcm", "Active2-rcm", .. "ActiveN-rcm"**: This configuration is in RCM format, and is pushed to the RCMs. RCM needs this scheme to perform configuration negation when a standby takes over for a specific active device.
additional-files	This parameter pushes the related configuration to other devices (for example, pushing configuration to CP when onboarding UP). This is not yet supported.
attribute-name	This parameter identifies any attribute (variables) in the config files for dynamic substitution. Formatted as $attribute_name
attribute-value	Value for the attribute

The following is an example of NSO action to populate or modify the config meta-data:

```
container
  config-metadata {
         // config true;
         tailf:action config-metadata-request {
             tailf:info "Invoke upgrade action on the selected devices";
             tailf:actionpoint config-metadata-request;
             input {
                 list config-metadata {
                     key device-name;
                     leaf device-name {
                         tailf:info "onboarding device name";
                         type string;
                     }
                     leaf device-type {
                         tailf:info "Onboarding device type vpc or rcm";
                         type enumeration {
                             enum vpc;
                             enum rcm;
                         }
                     }
                     leaf redundancy_scheme {
```

```
    tailf:info "cluster-topology 1:1, N:M and N+2";
    type enumeration {
        enum 1:1;
        enum N:M;
        enum RCUPS;
    }
}


list configuration-type {
    key config-type;
    tailf:info "Configuration type Day0.5, Day1 or DayN";
    leaf config-type {
        type enumeration {
            enum Day0.5;
            enum Day1;
            enum DayN;
        }
    }
    list files {
        key file-name;
        tailf:info "file name";
        leaf file-name {
            type string;
        }
        leaf config-scheme {
            type enumeration {
                enum common;
                enum host-specific;
                enum host-specific-common;
            }
        }
        // CP device info
        list additional-files {
            key device;
            //cp device
            leaf device {
                tailf:info "device name";
                type string;
            }
            list additional-file {
                key additional-file-name;
                leaf additional-file-name {
                    tailf:info "file name";
                    type string;
                }
            }


            }
        }
    }
}
list attributes {
    key attribute-name;
    leaf attribute-name {
        tailf:info "Attribute Name";
        type string;
    }
    leaf attribute-value {
        tailf:info "Attribute Value";
        type string;
    }
}
```

```
                }
                leaf delete-config-data {
                    type boolean;
                    default false;
                }
            }
            output {
                leaf status {
                    type string;
                }
                leaf message {
                 type string;
                }
            }
        }
    }
```

You can call this action from RESTCONF, as shown in the following example:

URI:
http://<NSO-IP>:<NSO-REST-PORT>/restconf/data/mobility-common:config-metadata/config-metadata-request

Method: POST

Content-Type: application/yang-data+json

Payload:

```
{
        "config-metadata": {
            "device-name": "test2",
     "schema" : "1:1",
            "attributes":{
                "attribute-name":"test",
                 "attribute-value": "gh"
                  },
            "configuration-type":{
                "config-type": "Day0.5",
                "files":{
                    "file-name":"/home/ubuntu/tmo_action/test.txt"
                    },
                "files":{
                    "file-name":"/home/ubuntu/tmo_action/day0.5.txt"
                    }

            }
        }
}
```

Result:

```
{
    "mobility-common:output": {
        "status": "Success
/home/ubuntu/tmo_action/test.txt ==> syntax error: unknown command,Error: on line 3: kkkl,
/home/ubuntu/tmo_action/day0.5.txt ==> Success"
    }
}
```

You can call this action using NCS CLI, as shown in the following example:

```
ubuntu@ncs> request config-metadata config-metadata-request config-metadata { device-name
staros-1 attributes { attribute-name hostname attribute-value TEST } configuration-type {
config-type Day0.5 files { file-name /home/ubuntu/tmo_action/test.txt } files { file-name
/home/ubuntu/tmo_action/day0.5.txt } } schema 1:1  }
status Success
```

```
/home/ubuntu/tmo_action/test.txt ==> syntax error: unknown command,Error: on line 3: kkkl,
/home/ubuntu/tmo_action/day0.5.txt ==> Success
[ok][2021-07-12 08:05:01]
```

NOTES:

- Config-metadata-request action has internal config validator. Config validator allows detection of syntax or certain semantic errors (for example, out of range values) in advance before pushing the configuration. Config validation requires at least a device which is onboarded in NSO (Either real-one or NetSim).

  The configurable parameter is as follows:

  ```
  container
  configurable-parameters {
     leaf config-pre-validation-vpc-device-name {
             type string;
     leaf config-pre-validation-rcm-device-name {
             type string;
     }
  }
  ```

  This config validation of files is also optional. If you do not want to validate the configs, you can turn-off this feature using configurable parameter. If config validation is turned off, then any error in the configuration files results in a config push error, and should be rolled back.

  ```
  container
  configurable-parameters {
     leaf config-pre-validation-required {
             type boolean;
             default false;
     }
     }
    }
  ```

  This config metadata contains all configurable parameters.

Onboarding ESC and Openstack as Devices

For ESC installation, see ESC documentation. Prior to configuration or onboarding and instantiation of VNFs, perform the following setup steps:

NSO and ESC Environment Setup for NFV

1. SSH to ESC host using username and password

   ```
   ssh esc@<esc-ip>
   ```

2. Become Sudo user

   sudo su

3. Edit the following file: `vi /opt/cisco/esc/esc_database/etsi-production.properties`

4. Edit the information as shown below and save the file (Don't change anything in spring user and password). Change the NSO details accordingly. Use only local subnet management IP for communication, and not the floating-IP between ESC/NSO communication.

   ```
   spring.security.user.name=esc
   spring.security.user.password=$1$J7BUBX$Ce4vqA6JcrWCggRpYrPYg1
   ```

```
security.pam.service=
server.additionalConnector.port=8253
server.additionalConnector.key-alias=esc
server.esc.key-alias=esc

nfvo.apiRoot=<NSO-IP>:9191
nfvo.httpScheme=http
nfvo.userName=<NSO-User-name>
nfvo.password=<NSO-Password>
nfvo.authenticationType=BASIC

server.host=<ESC-Orch-IP>
http.enabled=true
https.enabled=false
certificate.validation=false
spring.datasource.password=${PGSQL_PASSWORD}
spring.flyway.password=${PGSQL_PASSWORD}
```

5. Restart the **escadm** service, as shown below:

```
escadm restart
Stopping esc_service: [OK]
Stopping escadm service: [OK]
Starting escadm service: [OK]
#
```

6. Check for the escadm health till it becomes healthy, as shown below (It may take few minutes):

```
escadm health
============== ESC =================
vimmanager (pgid 18651) is running
monitor (pgid 18688) is running
mona (pgid 18741) is running
snmp is disabled at startup
etsi (pgid 19316) is running
pgsql (pgid 18944) is running
portal (pgid 19355) is running
confd (pgid 18978) is running
escmanager (pgid 19131) is running
=====================================
ESC HEALTH PASSED
```

7. Login to the NSO and modify the configs according to the environment and save it into a file:

```
<config xmlns="http://tail-f.com/ns/config/1.0">
  <nfv xmlns="urn:etsi:nfv:yang:etsi-nfv-descriptors">
  <settings xmlns="http://cisco.com/ns/nso/cfp/cisco-etsi-nfvo">
    <image-server>
      <ip-address><NSO-IP></ip-address>
      <port>8010</port>
      <document-root>/var/opt/ncs/vnfpackages</document-root>
    </image-server>
    <etsi-sol3>
      <server>
        <ip-address><NSO-IP></ip-address>
        <port>9191</port>
        <use-ssl>false</use-ssl>
        <document-root>/var/opt/ncs</document-root>
        <auth-enabled>true</auth-enabled>
        <auth-types>
          <basic>
            <username><NSO-USERNAME></username>
            <password><NSO-PASSWORD></password>
          </basic>
        </auth-types>
```

```
      </server>
      <vnfm-behaviour>
        <vnfm-behaviour-override>
          <id>default-sol3</id>
          <rpc-behaviour>
            <rpc>
              <include>
                <vim-info>false</vim-info>
              </include>
            </rpc>
            <modify>
              <pre>
                <rpc>false</rpc>
              </pre>
              <post>
                <rpc>true</rpc>
              </post>
            </modify>
          </rpc-behaviour>
          <grant>
            <store-history>false</store-history>
            <heal>
              <authorise-grant>true</authorise-grant>
            </heal>
          </grant>
          <onboarding>
            <store-details>true</store-details>
          </onboarding>
        </vnfm-behaviour-override>
      </vnfm-behaviour>
    </etsi-sol3>
  </settings>
  </nfv>
</config>
```

8. Compile all the packages in package folder and perform package reload.

```
ubuntu@test-nso:/var/opt/ncs/packages$ ncs_cli -C
User ubuntu last logged in 2021-09-23T08:00:34.649202+00:00, to test-nso, from
209.165.200.225 using cli-ssh
ubuntu connected from 209.165.200.225 using ssh on test-nso
ubuntu@ncs# packages reload
```

9. Load merge the file, as shown below. This step enables NSO as NFVO and runs NFVO service in 9191 port:

```
ubuntu@test-nso:~$ vi config.xml
ubuntu@test-nso:~$ ncs_cli -C

User ubuntu last logged in 2021-08-04T09:10:55.819283+00:00, to test-nso, from
209.165.200.226 using cli-ssh
ubuntu connected from 209.165.200.227 using ssh on test-nso
ubuntu@ncs# config
Entering configuration mode terminal
ubuntu@ncs(config)# load merge config.xml
Loading.
1.54 KiB parsed in 0.01 sec (128.38 KiB/sec)
ubuntu@ncs(config)# commit
```

10. Update NACM rule by adding NSO username to “ncsadmin” group

```
ubuntu@test-nso:~$ ncs_cli -C

User ubuntu last logged in 2021-08-06T09:56:26.370979+00:00, to test-nso, from
```

```
209.165.200.227 using cli-ssh
ubuntu connected from 209.165.200.227 using ssh on test-nso
ubuntu@ncs# config
Entering configuration mode terminal
ubuntu@ncs(config)# nacm groups group ncsadmin user-name ubuntu
ubuntu@ncs(config-group-ncsadmin)# commit
```

11. Copy the necessary packages to the standard location on the NSO (typically /var/opt/ncs/packages).
12. Perform package reload and check for package status. Status should be UP for all packages.

```
ubuntu@test-nso:~$ ncs_cli -C

User ubuntu last logged in 2021-08-06T09:58:39.866838+00:00, to test-nso, from
209.165.200.227 using cli-ssh
ubuntu connected from 209.165.200.227 using ssh on test-nso
ubuntu@ncs# packages reload
ubuntu@ncs# show packages package oper-status

NAME                 UP  PROGRAM CODE ERROR  JAVA UNINITIALIZED PYTHON UNINITIALIZED
 *
-------------------------------------------------------------------------------
cisco-etsi-nfvo       X      -                    -             -
cisco-rcm-nc-1.0      X      -                    -             -
cisco-staros-cli-5.38 X      -                    -             -
esc                   X      -                    -             -
etsi-sol003-gen-1.13  X      -                    -             -
mobility-common       X      -                    -             -
mop-automation        X      -                    -             -
mop-common            X      -                    -             -
nfv-common            X      -                    -             -
nfv-device-onboarding X      -                    -             -
nfv-vim               X      -                    -             -
nfv-vnf-lcm           X      -                    -             -
openstack-cos-gen-4.2 X      -                    -             -
```

13. Setup the notification stream: Update /etc/ncs/ncs.conf file to add "nfv-events" stream.

```
<ncs-config>
    <event-streams>
      <notifications>
        <stream>
          <name>nfv-events</name>
         <description>Generic netconf notification stream for NFV events</description>

          <replay-support>true</replay-support>
          <builtin-replay-store>
            <enabled>true</enabled>
            <dir>${NCS_RUN_DIR}/state</dir>
            <max-size>S10M</max-size>
            <max-files>50</max-files>
          </builtin-replay-store>
        </stream>
      </event-streams>
    </notifications>
</ncs-config>
```

14. Restart NSO as sudo user.

```
/etc/init.d/ncs stop
Stopping ncs (via systemctl):                              [  OK  ]
/etc/init.d/ncs start
Starting ncs (via systemctl):                              [  OK  ]
```

15. Onboard NETCONF, ESC, ETSI SOL003 ESC, and Openstack as devices in NSO via device onboarding APIs.

    a. Onboard Openstack as a device. The following is an example. Customize to the specific deployment. This can be configured via NSO CLI in the configuration mode. See NSO documentation for information about authgroup.

    ```
    devices device openstack
    address 209.165.200.228
    port 5000
    authgroup openstack
    device-type generic ned-id openstack-cos-gen-4.2
    ```

    b. Onboard ESC ETSI interface as a device. The following is an example. Customize to the specific deployment.

    ```
    devices device esc-etsi
    address 209.165.200.229
    port 8250
    authgroup esc-etsi
    device-type generic ned-id etsi-sol003-gen-1.13
    ```

    c. Onboard ESC native NETCONF interface as a device. The following is an example. Customize to the specific deployment.

    ```
    devices device esc-netconf
    address 209.165.200.229
    ssh host-key ssh-rsa
    key-data "AAAAB3NzaC1yc2EAAAADAQABAAABAQDYwNCaa3ghJtnJSvn/
    aSPjCuoMKmssZds+J5d9JCOS\n3h3V/fCtJwiH7qMgMXnNc0LEr1fZhxQ4kg5o/
    IafmoYD7N+w/ECqWEp68sjeN+AftiZ9J74D\n+/KDonffgBCHxIVEo0XHYlojrtmpg/
    EH9/N3fQgoSzEhGItGG4uMaAzbWr1pO8AApOPlPi4r\nciL4Qemi6u4i/
    HGFr8MqQp5qcMFd8O30OlB1q1vKn9sq/9sL6EzqyUd2lMounDglEQYMgi8J\
    nyG6upsOFuvhiYRC9qfHML45quyepsJdVi2Li2QwUJLa89EDh148RlhLTJs4s2iAwBGNdvLdK\ntzLu2VGyWKqH"
    !
    authgroup esc-netconf
    device-type netconf ned-id esc
    ```

16. Track the device addition status as shown below (for different devices):

```
ubuntu@test-nso:~$ ncs_cli -C

User ubuntu last logged in 2021-08-06T10:09:23.550686+00:00, to test-nso, from
209.165.200.227 using cli-ssh
ubuntu connected from 209.165.200.227 using ssh on test-nso
ubuntu@ncs# show vnf-status instances esc-netconf

INSTANCE ID  TIMESTAMP   FUNCTION TYPE OPERATION  STATUS   STATUS MESSAGE
-------------------------------------------------------------------------------
esc-netconf  2021-07-21 *    -          init   success   Device Onboarding initialized

             2021-07-21 *    -          init   success   Device Onboarding initialized

             2021-07-21 *    - fetch-ssh-keys  success   fetch-ssh-keys was successful

             2021-07-21 *    -         connect success   connect was successful
             2021-07-21 *    -       sync-from success   sync-from was successful
             2021-07-21 *    - device-config   success   Subscribed to ESC Netconf
 notification escEvent Stream
             2021-07-21 *    -          ready  success   Device Successfully onboarded
```

Prerequisites for VNF Instantiation

Before submitting the VNF deployment request, make the following configuration changes:

1. **Configurable parameters**

   Set the following configurable parameters, if required:

   - configurable-parameters device-ping-sleeptimesec 30 (default value is 30 sec)
   - configurable-parameters device-ping-retries 150 (default value is 30. In case of RCM, configure it to some higher value, for example, 150)
   - configurable-parameters p2p-required true (default value is false)
   - configurable-parameters p2p-soFile-path /var/opt/ncs/patch_libp2p-2.64.1418.so.tgz

2. **Prepopulating of config metadata**

   When you configure Config-metadata, the device name must be the same as VNF instance name.

   You can call this action from RESTCONF, as shown in the following example:

   URI:
   http://<NSO-IP>:<NSO-REST-PORT>/restconf/data/mobility-common:config-metadata/config-metadata-request

   Method: POST

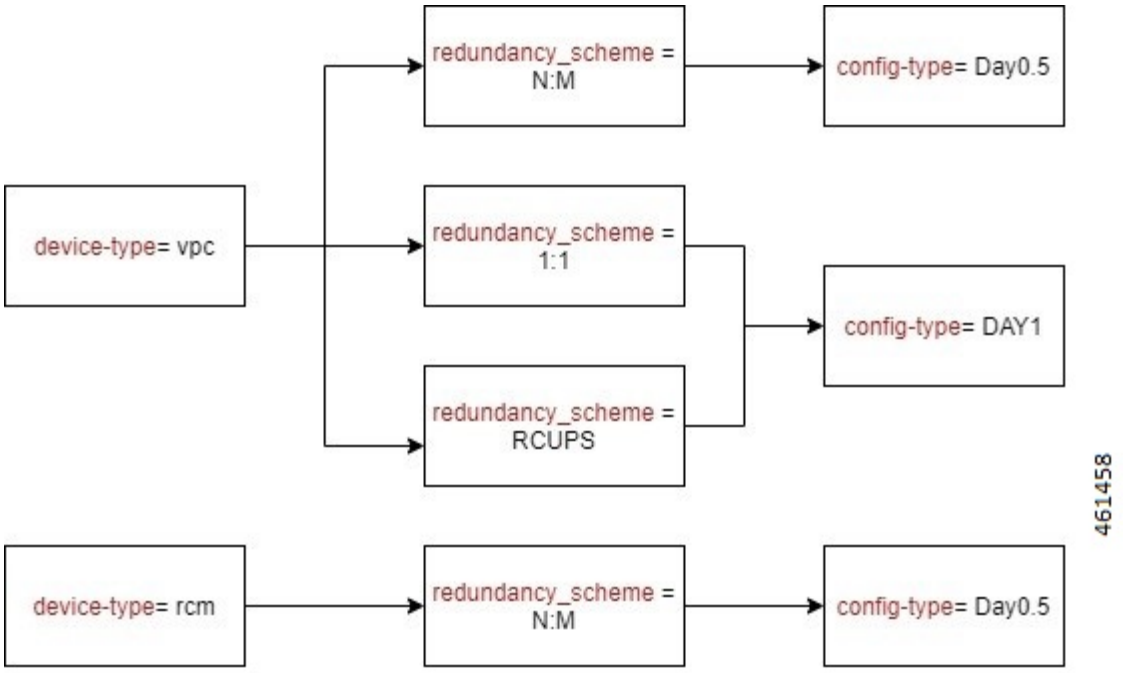
   Content-Type: application/yang-data+json

   Sample Payload:

```
{
        "config-metadata": {
            "device-name": "test2",
     "schema" : "1:1",
            "attributes":{
                "attribute-name":"test",
                 "attribute-value": "gh"
                  },
            "configuration-type":{
                "config-type": "Day1",
                "files":{
                    "file-name":"/home/ubuntu/tmo_action/test.txt"
                    },
                "files":{
                    "file-name":"/home/ubuntu/tmo_action/day0.5.txt"
                    }

            }
        }
    }
```

Ensure to follow the criteria described in the following figure while pre-populating config metadata:



VNF Instantiation

VNF is instantiated upon configuration. So, to instantiate the VNF, you must load the VNF configuration into the NSO. The VNF has references to the SOL006 VNFD. It also has references to VIM artifacts like Openstack tenant networks, and IP addresses. For details about YANG definition of VNF, see Appendix A: YANG definition of VNF.

Instantiating a VNF involves many components:

- An ETSI SOL001 VNFD template packaged as a TOSCA VNF package
- An ETSI SOL006 VNFD with the same name or ID as the VNF package
- A VNF instance that is proprietary to the NSO

The Mobility function pack ships with some example VNF packages, which also contain the corresponding SOL006 VNFD. These examples can be used as a base, but additional customization is required to fit the deployment. An example VNF configuration is given below:

```
{
   "nfv-vnf-lcm:nfv-vnf":[
      {
         "network-function-type":"VPC-SI",
         "name":"test026",
         "vnfd":"VPC-SI-2P-IMAGE-BOOT",
         "instantiation-level":"default",
         "deployment-flavor":"default",
         "mgmt-user-name":"admin",
         "mgmt-password":"Csco@123",
         "host-name":"vpc-si",
         "domain-name":"cisco.com",
         "ntp-server":"209.165.201.1",
```

```
"name-server":"209.165.201.2",
"location":{
   "vim":{
      "name":"openstack",
      "project":"test",
      "zone-id":"nova"
   },
   "vnfm":"esc-etsi"
},
"network":[
   {
      "type":"VIM_NETWORK_MANAGEMENT",
      "extent":"external",
      "name":"test-mgmt",
      "subnet-name":"test-mgmt-subnet"
   },
   {
      "type":"VIM_NETWORK_ORCHESTRATION",
      "extent":"external",
      "name":"test-orch",
      "subnet-name":"test-orch-subnet"
   },
   {
      "type":"VIM_NETWORK_SERVICE_1",
      "extent":"external",
      "name":"service1",
      "subnet-name":"service1"
   },
   {
      "type":"VIM_NETWORK_SERVICE_2",
      "extent":"external",
      "name":"service2",
      "subnet-name":"service2"
   }
],
"unit":[
   {
      "type":"VPC-SI",
      "image":"core-si-21.23",
      "flavor":"core-si",
      "connection-point":[
         {
            "name":"nic0",
            "ip-address":[
               {
                  "id":0,
                  "fixed-address":[
                     "209.165.201.3"
                  ]
               }
            ],
            "security-group":[
               "default"
            ],
            "network-type":"VIM_NETWORK_ORCHESTRATION"
         },
         {
            "name":"nic1",
            "ip-address":[
               {
                  "id":0,
                  "fixed-address":[
                     "209.165.201.4"
                  ]
```

```
                        }
                     ],
                     "security-group":[
                        "default"
                     ],
                     "network-type":"VIM_NETWORK_MANAGEMENT"
                  },
                  {
                     "name":"nic2",
                     "ip-address":[
                        {
                           "id":0,
                           "fixed-address":[
                              "209.165.201.5"
                           ]
                        }
                     ],
                     "security-group":[
                        "default"
                     ],
                     "network-type":"VIM_NETWORK_SERVICE_1"
                  },
                  {
                     "name":"nic3",
                     "ip-address":[
                        {
                           "id":0,
                           "fixed-address":[
                              "209.165.201.6"
                           ]
                        }
                     ],
                     "security-group":[
                        "default"
                     ],
                     "network-type":"VIM_NETWORK_SERVICE_2"
                  }
               ]
            }
         ],
         "extra-parameters":[
            {
               "name":"BOOTUP_TIME",
               "value":"100"
            },
            {
               "name":"LICENSE_KEY",
               "value":"\"VER=1|DOI=1624646484|DOE=1640457684|ISS=3|NUM=212017|
CMT=SWIFT_License|LSG=5000000|LEC=10000000|LGT=5000000|FIS=Y|FR4=Y|FTC=Y|FSR=Y|
FPM=Y|FID=Y|FI6=Y|FLI=Y|FFA=Y|FCA=Y|FTP=Y|FTA=Y|FDR=Y|FDC=Y|FGR=Y|FAA=Y|FDQ=Y|
FEL=Y|BEP=Y|FAI=Y|FCP=Y|LCF=5000000|LPP=5000000|LSF=5000000|FLS=Y|FSG=Y|
LGW=5000000|HIL=XT2|LSB=5000000|LMM=5000000|FIB=Y|FND=Y|FAP=Y|FRE=Y|FHE=Y|
FUO=Y|FUR=Y|FOP=Y|FRB=Y|FCF=Y|FVO=Y|FST=Y|FSI=Y|FRV=Y|F6D=Y|F13=Y|FIM=Y|
FLP=Y|FSE=Y|FMF=Y|FEE=Y|FHH=Y|FIT=Y|FSB=Y|FDS=Y|LSE=5000000|FLR=Y|FLG=Y|
FMC=Y|FOC=Y|FOS=Y|FIR=Y|FNE=Y|FGD=Y|LIP=5000000|FOE=Y|FAU=Y|FEG=Y|FL2=Y|
FSH=Y|FLF=Y|FSP=Y|FNI=Y|FCI=Y|FME=Y|FCN=Y|FUB=Y|FSF=Y|FGO=Y|FPE=Y|FWI=Y|
FAC=Y|FIE=Y|FSM=Y|FAG=Y|FNQ=Y|FEW=Y|FAR=Y|FOX=Y|FPW=Y|FAM=Y|FGX=Y|FWT=Y|
FUA=Y|LDT=5000000|LEX=5000000|LVL=5000000|LQP=5000000|LMP=5000000|
LCU=10000000|LUU=10000000|FXS=Y|FLC=Y|FRT=Y|FSX=Y|FBS=Y|FRD=Y|FXM=Y|
LTO=10000000|FNS=Y|LNS=5000000|SIG=MC0CFBge/
0TZha2Ta7c1L5CLOL2tgDIDAhUAhIKwZxXEJJpr9Xk5buNyzZStrNM\""
            }
         ]
      }
```

```
    ]
}
```

The following is another example to instantiate RCM VNF:

```
{
    "nfv-vnf-lcm:nfv-vnf": [
      {
        "network-function-type": "RCM",
        "name": "RCM-ahhashem-sol003-78",
        "vnfd": "RCM-IMAGE-BOOT",
        "instantiation-level": "default",
        "deployment-flavor": "default",
        "mgmt-user-name": "luser",
        "mgmt-password": "$8$40/jVMTHJY+Jrd7mZiwqdrKEIz6Kc5Pt2Qvnwi0/65g=;”
        "host-name": "rcm",
        "domain-name": "cisco.com",
        "ntp-server": "209.165.201.1",
        "name-server": "209.165.201.1",
        "location": {
          "vim": {
            "name": "openstack",
            "project": "ahhashem",
            "zone-id": "nova"
          },
          "vnfm": "esc-etsi"
        },
        "network": [
          {
            "type": "VIM_NETWORK_MANAGEMENT",
            "name": "ahhashem-mgmt",
            "extent": "external",
            "subnet-name": "ahhashem-mgmt-subnet"
          },
          {
            "type": "VIM_NETWORK_ORCHESTRATION",
            "name": "ahhashem-orch",
            "extent": "external",
            "subnet-name": "ahhashem-orch-subnet"
          },
          {
            "type": "VIM_NETWORK_SERVICE_1",
            "name": "service1",
            "extent": "external",
            "subnet-name": "service1"
          } ,
          {
            "type": "VIM_NETWORK_SERVICE_2",
            "name": "service2",
            "extent": "external",
            "subnet-name": "service2"
          }
        ],
        "unit": [
          {
            "type": "RCM",
            "image": "core-rcm-21.23",
            "flavor": "mkal-rcm-hugepages",
            "connection-point": [
              {
                "name": "nic0",
                "ip-address":{
                    "id": 1,
                    "fixed-address": ["209.165.201.7"]
                },
```

```
          "security-group": ["default"],
          "network-type": "VIM_NETWORK_ORCHESTRATION"
        },
        {
          "name": "nic1",
          "ip-address":{
              "id": 1,
              "fixed-address":["209.165.201.8"]
          } ,
          "security-group": ["default"],
          "network-type": "VIM_NETWORK_MANAGEMENT"
        },
        {
          "name": "nic2",
          "ip-address": {
              "id": 1,
              "fixed-address": ["209.165.201.9"]
          },
          "security-group": ["default"],
          "network-type": "VIM_NETWORK_SERVICE_1"
        } ,
        {
          "name": "nic3",
          "ip-address": {
              "id": 1,
              "fixed-address": ["209.165.201.10"]
          },
          "security-group": ["default"],
          "network-type": "VIM_NETWORK_SERVICE_2"
        }

      ]
    }
  ],
  "extra-parameters": [
    {
      "name": "VIM_VM_NAME",
      "value": "RCM-ahhashem-sol003-78"
    },
    {
      "name": "HOST_NAME",
      "value": "rcm"
    },
      {
        "name": "NIC0_TYPE",
        "value": "virtual"
      },
      {
        "name": "NIC1_TYPE",
        "value": "virtual"
      },
      {
        "name": "NIC2_TYPE",
        "value": "direct"
      },
      {
        "name": "NIC3_TYPE",
        "value": "direct"
      },
      {
        "name": "MGMT_USER_NAME",
        "value": "luser"
      },
      {
```

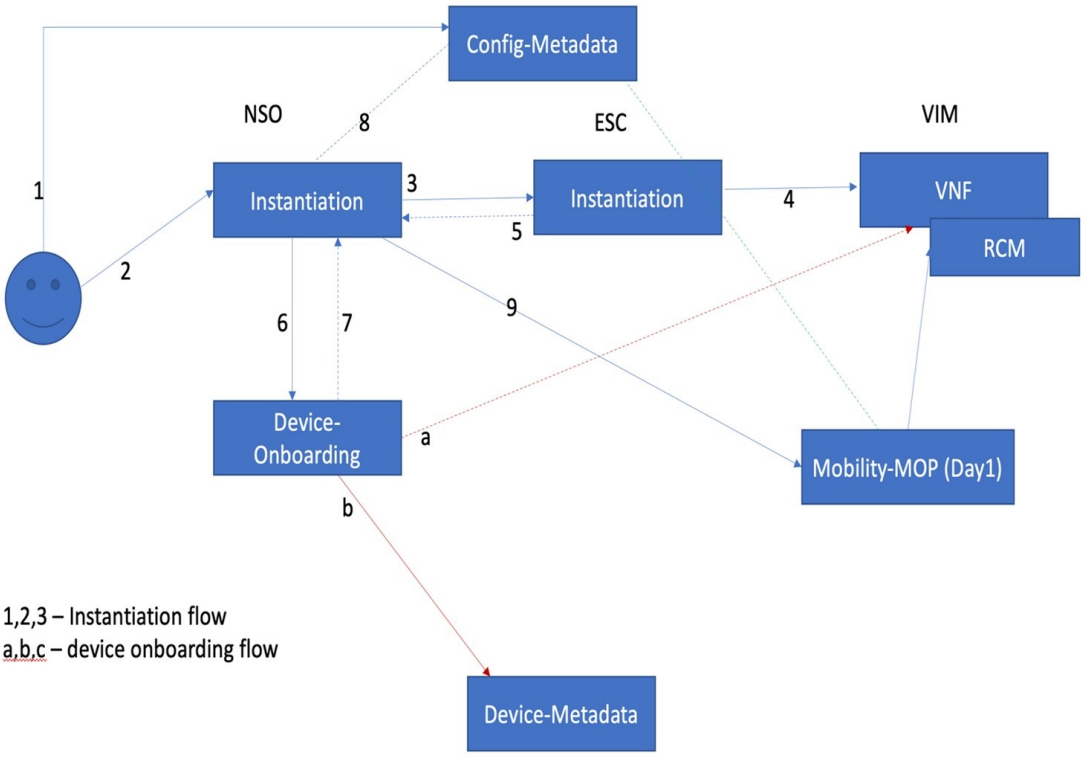
```
                "name": "MGMT_PASSWORD_ROUND4096",
                "value": "$6$rounds=4096$P2wdTbEBO0LHmHi$OwbVEIarMbt
Qxbu5Us5kW0n0MOWp3QN9eVRX7WjvLm4xTJvFpl6vHez3XkKm39XJJ7dGRRIsZqXfcZRjQBA7E."
            },
            {
                "name": "SERVICE_INTERFACE_IP_1",
                "value": "209.165.201.9"
            },
            {
                "name": "SERVICE_INTERFACE_IP_2",
                "value": "209.165.201.11"
            },
            {
              "name": "NTP_SERVER",
              "value": ["209.165.201.12","209.165.201.13","209.165.201.14"]
            }

        ]
      }
    ]
  }
```

VNF Instantiation - Component Interactions and Flows

The following figure illustrates the complete flow of end-to-end instantiation automation:

Figure 5: VNF Instantiation Interactions



Detailed Steps:

1. The Network operator has all the required details for VNF instantiation including the name, type, dynamic attributes, and the configuration files. The network operator places the config files into the NSO filesystem, and registers the details with NSO config DB for automation.

   This step includes the following tasks:

   - The network operator Secure copy (SCP) the config files into NSO filesystem. This location must be an NFS, or it's replicated in NSO HA environment.
   - Registers all attribute value pairs, dynamic substitution values, Day-0.5, or Day-1 configurations.
   - Enables the validation of config files, and provides the testing device details.
   - If the revalidation flag is set to true, config metadata action internally validates all config files. Otherwise, it fails while applying the configuration.

2. The Network operator prepares the payload for VNF instantiation with all the details. Then, the network operator invokes the payload to create an instance. It does the basic validation and processes the order.

   This step includes the following task:

   - Validates the inputs such as password length, image, flavor, and network existence in Openstack before invoking an order

3. NSO processes the order internally and prepares the ESC VNF instantiation order.

   This step includes the following tasks:

   - Creates NSO footprint of the service
   - Does CSAR validation
   - Invokes ESC VNF instantiation order using SOL3/SOL4 input
   - Starts listening to ESC notifications (both ETSI and NETCONF)

4. ESC performs input validation of SOL3/SOL4 and creates the order in VIM.

   This step includes the following tasks:

   - ESC invokes VNF instantiation.
   - On Successful invocation of VNF, it creates mono monitors to monitor the VNF.
   - Returns the updates via ETSI and NETCONF notifications to NSO (both Success and Failure).

5. ESC returns the periodic updates on progress to NSO via ETSI or NETCONF notifications.

   This step includes the following tasks:

   - ESC constantly sends the ETSI and NETCONF notifications on progress.
   - ETSI notification comprises deploy – init, processing, and completed notification.
   - NETCONF notifications provide more granular information on VM status.
   - On Failure, it gives appropriate error message.

6. On receiving VNF instantiation completion message from ESC, NSO onboards as an NSO device.

This step includes the following tasks:

- Instantiation logic fetches the details from input payload, and invokes device onboarding logic.
- NSO performs the fetch-ssh-host-key from the device.
- NSO performs connection check.
- NSO performs sync-from.
- NSO executes post check command such as “show version” on device.
- NSO adds the device into NSO device tree.

7. NSO instantiation logic waits for device addition to complete.

   This step includes the following tasks:

   - NSO checks if device onboarding process is complete.
   - If device onboarding fails, NSO stops the execution.

8. NSO instantiation logic reads the prepopulated config metadata to interpret the config to be pushed.

   This step includes the following tasks:

   - Reads the prepopulated config metadata and interprets the Day-0.5 or Day-1 configuration files based on the device-name (Device name is based on VNF name)
   - For RCM-based N:M scheme, Day-0.5 is pushed.
   - On 1:1 case, Day-1 is pushed.
   - On missing information, instantiation completes and stops processing.

9. NSO takes the config files from config metadata, formulates the Mobility MOP input format, and invokes the MOP for config push.

   This step includes the following tasks:

   - Invokes the Mobility MOP and gets the task-id.
   - Periodically checks for the status on task-id.
   - Saves the config permanently in device flash if its 1:1 CP or UP pairs (via MOP).
   - Completion status is updated in vnf-status ledger.

Checking the VNF Instantiation Status

You can check the status of VNF instantiation using **vnf-status** command periodically.

Any failure, processing, or completion related messages are appended in the status message.

```
show vnf-status instances vnf-instance-name
INSTANCE ID TIMESTAMP TYPE OPERATION STATUS STATUS MESSAGE
------------------------------------------------------------------------
<VNF-Name> <Time-Stamp> <type> <function> <status> <message-if-any>
```

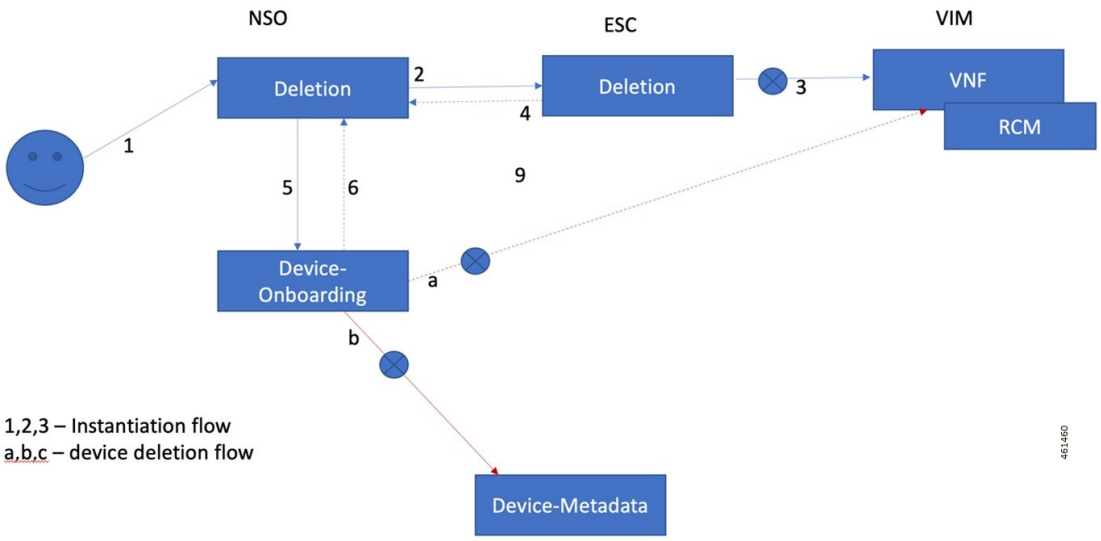
VNF Dashboard

VNF instantiation steps and current status of the VNF are displayed in NSO based dashboard.

VNF Deletion

The following flow diagram illustrates the complete flow of end-to-end deletion automation.

Figure 6: VNF Deletion Interactions



Detailed Steps:

1. Network operator decides to decommission or delete the existing instance, which is in running or failed state.

   This step includes the following tasks:

   - Network operator provides the VNF-name with type for the deletion
   - NSO does the validation of the VNF existence

2. NSO checks for the VNF instance status, and if there is a failed instance at NSO end, NSO invokes ESC for deletion from VIM or perform rollback.

   This step includes the following tasks:

   - NSO decides to push it to ESC or perform rollback (in case of failed instance within NSO)
   - NSO does the asynchronous request to ESC and waits for notifications.

3. ESC does the clean-up and removes the VNF monitors.
4. ESC generates ETSI/NETCONF notifications to NSO.
5. NSO processes ESC notifications and performs the following:

- Invokes the device-onboarding package for deletion of the instance
- Removes entry from “nfv-vnf-inventory”

6. Device onboarding package deletes the device from NSO, and the status is updated in the VNF ledger.

Checking the VNF Deletion Status

You can check the status of VNF deletion using **vnf-status** command.

Any failure, processing, or completion related messages are appended in the status message

```
show vnf-status instances vnf-instance-name
INSTANCE ID TIMESTAMP TYPE OPERATION STATUS STATUS MESSAGE
------------------------------------------------------------------------
<VNF-Name> <Time-Stamp> <type> <function> <status> <message-if-any>
```

Removing Configuration Metadata

This is a manual step, and you need to remove the config metadata using NSO action. Keeping this data doesn’t have any impact.

Cleaning Config Files from NSO Filesystem

You need to remove the config files manually from NSO filesystem. Keeping this data doesn’t have any impact.

Automation Process - VNF Deployment, Onboarding, and Configuration Push

The Automation process includes the following sections:

Instantiation of VNF using Input Payload

After making necessary changes, submit the instantiation request using input payload. The automation process for VNF instantiation starts.

For input payload sample, see the section VNF Instantiation.

Onboarding VNF as a Device in NSO

Upon successful instantiation, the VNF is onboarded as a device on the NSO. The device name is the same as the VNF name.

Upgrading the P2P Module in VPC Device

If the “device-type” is of VPC type and the configurable-parameters “p2p-required” is set to “true” with the “p2p-soFile-path” defined, copy the P2P file to device flash directory, and then upgrade the P2P module.

The P2P module is installed on the device.

Configuration Push to the Onboarded Device

The following are the static parameters that are used during automated config push:

- operation-type =Commit

- mop-type=Common
- save-config-permanently= default is false, and it is set to true when the device type is "vpc"

Once the config push is done using the configuration files, a task-id is generated. Using the task-id, it checks the status of config-push, and based on the status, the ledger entry is updated.

Note NSO doesn't perform configuration audit on RCM. If an RCM reboots when NSO is in the process of pushing configuration to it, the NSO doesn't re-push the configuration upon reboot completion. The configuration must be re-pushed manually. NSO alerts the operator about configuration push failure. Any configuration successfully pushed to the RCM is persistent across reboots of that RCM.

Appendix A: YANG definition of VNF

This section provides a sample YANG definition of VNF.

```
module nfv-vnf-lcm {
  namespace "http://com/cisco/cx/servicepack/nfv/vnflcm";
  prefix nfv-vnf-lcm;

  import ietf-inet-types { prefix inet; }
  import tailf-common { prefix tailf; }
  import tailf-ncs { prefix ncs; }
  import nfv-common { prefix nfv-common; }
  import tailf-kicker { prefix kicker; }
  include nfv-vnf-lcm-nano {
    revision-date 2020-02-14;
  }

  organization "Cisco-AS";

  contact "Cisco AS";

  description "Generic NFV VNF LCM service package";

  revision 2020-10-22 {
    description "Active Inventory and LCM Auto/on-demand heal support";
  }

  revision 2020-07-01 {
    description "Re-branded per new naming convention";
  }

  revision 2020-02-14 {
    description "First version, ready for testing";
  }

  notification vnf-lcm {
    description "Notification about Network Function Operation";
    uses nfv-common:network-function-notification;
  }

  notification vnf-alarm {
    description "VNF alarms";
    uses nfv-common:vnf-alarm;
  }
```

```
container nfv-vnf-inventory {
  tailf:info "CDB model to persist the VNFs, associated project, VIM and the
      VM details";
  config false;
  tailf:cdb-oper {
    tailf:persistent true;
  }

  list vnf {
    tailf:info "VNFs with associated VMs and status";
    key name;
    leaf name {
      tailf:info "VNF Name";
      type string;
    }
    leaf vnfd {
      type string;
      tailf:info "Associated VNFD name";
    }
    leaf project {
      type string;
      tailf:info "Associated vim tenant/project";
    }
    leaf vim {
      type string;
      tailf:info "Associated VIM";
    }
    leaf status {
      type string;
      tailf:info "Overall VNF status";
    }
    list vm {
      tailf:info "Associated VMs and the status";
      key name;
      leaf name {
        type string;
        tailf:info "VM name";
      }
      leaf type {
        type string;
        tailf:info "VM Type";
      }
      leaf flavor {
        type string;
        tailf:info "VIM flavor that is used to deploy the VM";
      }
      leaf host {
        type string;
        tailf:info "Compute host where the VM has been deployed";
      }
      list connection-point {
        key nic-id;
        leaf nic-id {
          type uint8;
          tailf:info "NIC id of the connection point";
        }
        leaf ip-address {
          type inet:ip-address;
          tailf:info "IP address of the connection point";
        }
      }
      leaf status {
        type string;
        tailf:info "VM status";
```

```
      }
    }

    leaf netconf-notification-done {
        tailf:hidden nfv-internal;
        type empty;
    }
  }
}

list nfv-vnf {
  description "Generic RFS model for VNF LCM";

  key "network-function-type name";

  leaf network-function-type {
    tailf:info "virtual network function type";
    type enumeration {
      enum "VPC-SI";
      enum "VPC-DI";
      enum "CSR1KV";
      enum "GENERIC";
      enum "VCU";
      enum "VDU";
      enum "EMS";
      enum "RCM";
    }
  }

  leaf name {
    tailf:info "Unique service id";
    type string;
  }

  leaf vnfd {
    mandatory true;
    type string;
    tailf:info "VNFD to use for this type of Network Function that has to be
          onboarded on the target VIM.";
  }

  uses ncs:service-data;
  ncs:servicepoint nfv-vnf-lcm;
  uses ncs:nano-plan-data;

  tailf:action heal {
    tailf:info "Heal VNF";
    tailf:actionpoint nfv-lcm-heal-ap;
    input {
    }
    output {
      uses nfv-common:standard-action-response;
    }
  }

tailf:action start {
          tailf:info "Start VNF";
          tailf:actionpoint nfv-lcm-start-ap;
          input {
          }
          output {
              uses nfv-common:standard-action-response;
          }
      }
```

```
 tailf:action stop {
           tailf:info "Stop VNF";
           tailf:actionpoint nfv-lcm-stop-ap;
           input {
           }
           output {
               uses nfv-common:standard-action-response;
           }
       }

tailf:action scale {
           tailf:info "Scale-In VNF";
           tailf:actionpoint nfv-lcm-scale-ap;
           input {
               leaf scale-type {
                   mandatory true;
                   tailf:info "SCALE IN or OUT";
                   type enumeration {
                       enum "OUT";
                       enum "IN";
                   }
               }

               leaf no-of-instances {
                   tailf:info "Number of scale IN or OUT instances. Default is 1";
                   type uint32;
                   default 1;
               }

               leaf vdu-type {
                   mandatory true;
                   tailf:info "vdu-type as CF/SF/VPC-SI etc";
                   type string;
               }

           }
           output {
               uses nfv-common:standard-action-response;
           }
       }

     tailf:action retry {
           tailf:info "Stop VNF";
           tailf:actionpoint nfv-lcm-retry-ap;
           input {
           }
           output {
               uses nfv-common:standard-action-response;
           }
       }

   leaf instantiation-level {
     type string;
     default "default";
     tailf:info "Instantiation level defined in VNFD to use. This will determine
           the number of VMs/VDUs to be deployed.";
   }

   leaf deployment-flavor {
     type string;
     default "default";
     tailf:info "Deployment flavor defined in the VNFD to use. Describes a specific
           deployment version of a VNF with specific requirements for capacity
```

```
        and performance.";
}
leaf mgmt-user-name {
  type nfv-common:identifier;
  description " Management login username specific to this VNF. Default values
        can be configured per VNF type.";
}

leaf mgmt-password {
  tailf:suppress-echo "true";
  type tailf:aes-cfb-128-encrypted-string;
  description "Management login password specific to this VNF.";
}

leaf host-name {
  type inet:domain-name;
  description "Hostname to use to communicate with this network function";
}

leaf domain-name {
  type inet:domain-name;
  description "Domain name used to construct Fully Qualified Domain Name by
        concatenating with VM hostname: <hostname>.<domain>";
}

leaf ntp-server {
  description "NTP server to use for VNFs deployed in this data center";
  type inet:host;
}

leaf name-server {
  type inet:ip-address;
  description "Name server";
}

container location {
  container vim {
    leaf name {
      description "NFVI this Network Function is deployed on.";
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
      //must "/ncs:devices/ncs:device[ncs:name=current()]/ncs:platform/ncs:name
      //          = 'Openstack'" {
      //  error-message "Please select Openstack devices only";
      //}
    }
    leaf project {
      type nfv-common:identifier;
      description "VIM project used to instantiate VNFs";
      mandatory true;
    }
    leaf zone-id {
      type string;
      default "nova";
      description "VIM zone id";
    }
    //TODO might need to support user domain and project domain
  }
  leaf vnfm {
    mandatory true;
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
```

```
    //must "/ncs:devices/ncs:device[ncs:name=current()]/ncs:platform/ncs:name
    //        = 'ETSI SOL'" {
    //  error-message "Please select ETSI-SOL VNFM devices only";
    //}
    description "ESC VNFM onboarded";
  }
}
list network {
  key type;
  leaf type {
    type nfv-common:identifier;
  }
  leaf name {
    type nfv-common:identifier;
    mandatory true;
  }
  leaf extent {
    type nfv-common:network-extent;
  }
  leaf subnet-name {
    when "../extent='external'";
    type nfv-common:identifier;
    mandatory true;
  }
}

list unit {
  description "Virtual Deployment Unit, a single VM.";
  key type;

  leaf type {
    description "VDU type as defined in the VNFD of this Network Function.";
    type nfv-common:identifier;
  }
  leaf image {
    type string;
    description " Image to use for this type of Network Function. Must have been
            be onboarded on the target VIM.";
  }
  leaf flavor {
    mandatory true;
    type string;
    description " Flavor to use for this type of Network Function. Must have been
            onboarded on the target VIM.";
  }
  list storage-volume {
    key id;
    description "Out of band Storage volumes to use for this network function";
    leaf id {
      type string;
    }
    leaf volume-name {
      type string;
      description "Storage Volume to use for this type of Network function";
    }
  }
  list connection-point {
    key name;
    description " Network connection point such as a network interface card, as
            defined in the descriptor.";
    leaf name {
      mandatory true;
      type nfv-common:identifier;
    }
```

```
        list ip-address {
            key id;
            ordered-by user;
            leaf id {
             type uint8;
                tailf:info "IP Address ID for connection points";
            }
            leaf-list fixed-address {
                ordered-by user;
                description " IP address(es) to assign this network interface for both
scaled and non-scaled VNF's. Both IPv4 and
                        IPv6 is possible to allow for dual-stack cases if this VNF requires

                          it for Internet access.";
                type inet:ip-address;
             }
        }

        list vip {
          key address;
          ordered-by user;
          description " Virtual IP address(es) to assign this network interface. Both
                    IPv4 and IPv6 is possible to allow for dual-stack cases if this
                    VNF requires it for Internet access. Setting this will populate
                    allowed-address-pair list in the CVIM";

          leaf address {
            type inet:ip-address;
          }
          leaf netmask {
            type inet:ip-address;
            mandatory true;
          }
        }
        leaf-list security-group {
          type nfv-common:identifier;
          description "Security group(s) to apply to this network interface.";
        }
        leaf network-type {
          type leafref {
            path "../../../network/type";
          }
          description "Network used for this connection-point.";
        }
      }
    }
    list extra-parameters {
      description "VNF instance specific additional parameters defined in the VNFD.
            This will override the values configured in the VNFD";
      key name;
      leaf name {
        type string {
          pattern "[A-Za-z0-9_]+";
        }
      }
      leaf value {
        type string;
      }
    }
  }

   list nfv-retry-vnfs {
        tailf:info "Retry VNF's to tweak the notifications";
```

```
            config false;
            tailf:cdb-oper {
                tailf:persistent true;
            }
            tailf:hidden nfv-internal;

            key name;
            leaf name {
                tailf:info "VNF Name";
                type string;
            }
        }
    }
```