NSO-based Configuration Management

Feature Description

The Cisco Network Service Orchestrator (NSO) based configuration management for 4G CUPS supports:

- Onboarding of Cisco Virtual Network Function (VNF) devices—CP, UP, and RCM
- Centralized configuration management of 4G-based CPs, UPs, and RCMs for Day-N, Day-1, and Day-0.5 CUPS configuration push.
  - Day-0.5 applies to N:M UP redundancy scheme that uses RCM. The Day-0.5 configuration is intended for the UP to communicate to the RCM, so that it's role can be defined and suitable configuration be pushed subsequently.

Managing customer configuration management for 4G CUPS deployments using NSO automation also exhibits reusability, standard notification management, and systematic device configuration governance.

Use Cases

The NSO configuration-handling caters to the following use cases:

1. NSO on-boarding of VNFs (CPs, UPs, and RCMs) that are already deployed using Management IPs:
   - Onboarding of already-running VNFs (CPs, UPs, and RCMs) as devices into NSO and perform post-check to ensure the reachability and functioning of the devices. This is preliminary step to push/sync any configuration and establish communication for notifications.

     Orchestration of VNF devices (Instantiation and Destroy) is a separate module, and it doesn't have any dependency on configuration module. We need certain details (IP, Port, Management Username/Password) to onboard the device and supporting configuration management.

2. Allowing to store native-configs or device-templates for CPs, UPs, and RCMs:
   - Providing interface through RESTCONF/NSO-CLI to manage the reusable configurations for devices with logical name. Providing flexibility to network SMEs/Operators to create, modify, delete, and disable/enable the configurations. Aim is to pick those active configurations and apply to the device set as part of Day-0.5, Day-1, or Day-N for CPs, UPs, and RCMs.

3. Providing CLI/REST interface to apply Day-N/Day-1/Day-0.5 configurations to device logical groups (including CPs/UPs/RCMs) or custom list of target devices:
   - Providing interface through RESTCONF/NSO-CLI to Network SMEs/Operators to push Day-N/Day-1/Day-0.5 configurations to single or set of devices (CPs, UPs, or RCMs). This interface exhibits notifications/status on the progress of configuration push to the end users.

4. Logistic management of configuration management per device basis (Day-N, Day-1, or Day-0.5 pushed):
   - Providing dashboard utility and managing the configuration logs per device basis. This log is useful to know the most recent activity done on the device.

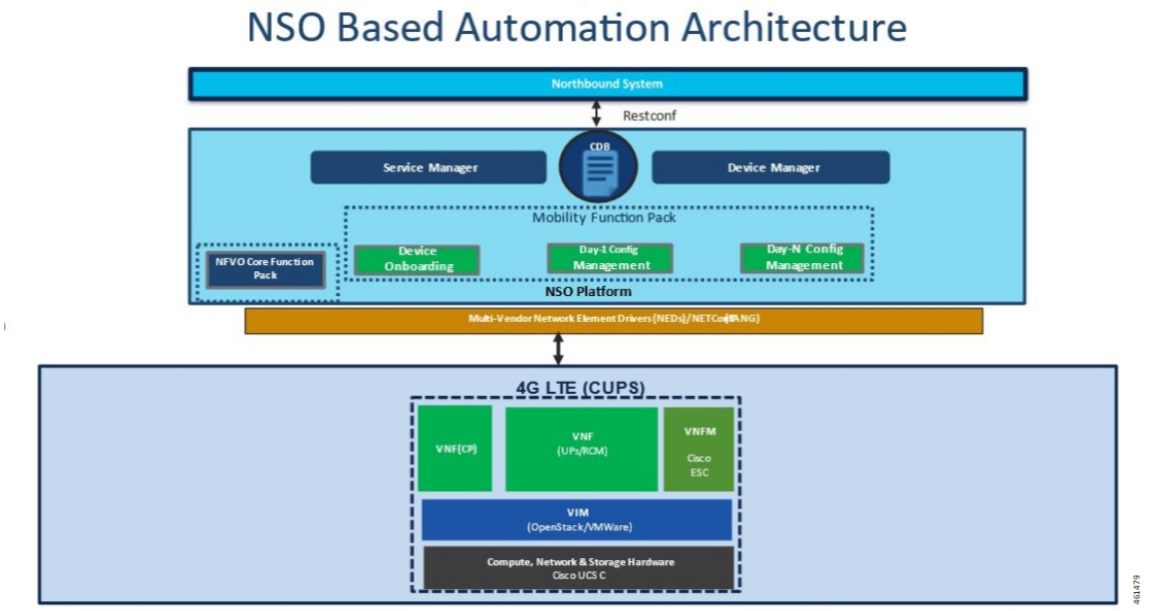
5. 5. Build the notification framework for RCM notification management in (N:M cases) and automate the configuration push for UPs for Day-N, Day-1, and Day-0.5:
   - Building notification framework in NSO to listen to RCM NetConf notifications on status changes, and push configurations automatically based on scenarios.

How it Works

-

Architecture

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



RCM and NSO

In the N:M UP redundancy scenario, while the NSO manages configuration, the RCM will continue to arbitrate the role of the UPs (Active or Standby) and handle the switchover of an Active UP. Hence, this solution only moves the configuration function out of the RCM into the NSO; RCM is still required. For details about RCM, refer to the *RCM Configuration and Administration Guide*.

Components

Cisco 4G CUPS VNF deployment and configuration workflows are driven from the NSO. The following are some of the important components of NSO:

- **NSO Device Manager**: Manages each VNF component (CP, UP, RCM), keeps the copy of each device configurations, and manages the integrity of device configuration push.
- **NSO Service Manager**: Provides YANG standard to define high-level abstraction network service model for the customer/user input.
- **CDB**: Persistent Configuration Database for storing network configurations and operational data.
- **Mobility Function Packs**: Custom-built NSO packages to manage the 4G CUPS-based VNF orchestration and configuration management.
- **NFVO Core Function Pack**: NSO core NFV FP is a driver software to communicate with Cisco or other 3rd party VNFMs and VIMs, like OpenStack/VMWare, to deploy VNFs.
- **StarOS NSO NED**: StarOS-based NSO Network Element Driver (NED) that interfaces with the Cisco 4G CUPS VNFs for configuration push. This NED is based on Cisco CLI. The StarOS NSO NED communicates with the StarOS management CLI instance using Secure Shell (SSH).
- **RCM NSO NED**: RCM-based NETCONF NED is used to communicate with RCM devices for configuration management.

Minimum Platform, Hardware, and Software Requirements

The following are the minimum platform and software requirements to support centralized configuration management:

- Supported Orchestrator: NSO
- Configuration management for following Network Element's:
  - RCM: Redundancy and Configuration Management
  - VPC-SI: As 4G CUPS CP or UP
  - VPC-DI: As 4G CUPS CP only
- Minimum hardware requirements:
  - VM CPU: 8 CPU cores
  - VM RAM: 16 GB RAM baseline + 10 MB RAM for every StarOS device to be supported
  - VM connectivity: One 10 GBps network link. This can be used for both NSO HA and config/deployment by using separate VLANs or other mechanisms
  - VM Storage: 100 GB disk (preferably, SSD)
- Minimum software version

Software	Minimum Version
Cisco NSO	6.1.6.1
StarOS NSO NED	5.52.4
Cisco NSO HCC	6.0.1
Mobility Function Pack	3.5

Note The recommended StarOS software image version for UP, CP, and RCM is 21.23 and later releases. The release versions are not tightly coupled and only depend on the NEDs.

Licensing

The NSO-based Configuration Management is a licensed Cisco feature. Contact your Cisco Account representative for detailed information on specific licensing requirements.

NSO Installation

Call Flows

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

The call flows refer to NSO primitives like "connect", "sync-from", and so on. For detailed information on these primitives, refer to the *NSO User Guide*.

Note In the following call flow diagrams, “NSO Northbound” implies NCS CLI or RESTCONF interface.

Onboarding Existing 4G CUPS VNFs into NSO

This section describes the flow for adding existing 4G CUPS devices to NSO.

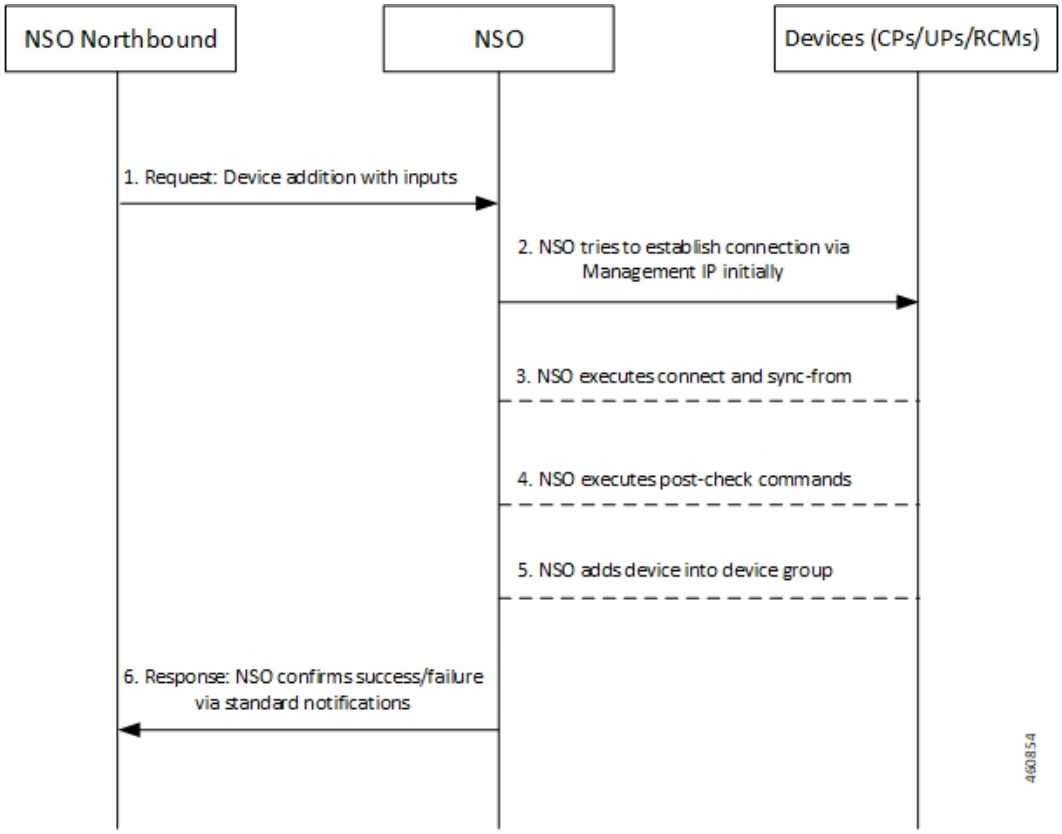


Table 1: Call Flow Description

Step	Description
1	NSO Northbound sends a request to NSO for adding devices into device-group. A device-group is a logical grouping of devices (VNFs) that share nearly identical configurations. This simplifies configuration in certain cases.

Step	Description
2	NSO initially attempts to establish a connection via Management IP.
3	NSO executes connect and sync-from commands. A sync-from operation pulls the existing configuration from the device/VNF into the NSO so that NSO is aware of the exact configuration on the device. The device configuration is not changed in a sync-from.
4	NSO executes post-check commands.
5	NSO adds device into device group.
6	NSO updates NSO Northbound about the success or failure of device addition via standard notifications.

4G CUPS Device Configuration Push – Manual

This section describes the flow for manual configuration push for 4G CUPS device.

This scenario applies to CPs, UPs, or RCMs in standalone or 1:1 redundancy configuration.

Prior to any configuration push, the device(s) must be onboarded. See Onboarding Existing 4G CUPS VNFs into NSO, on page 5.

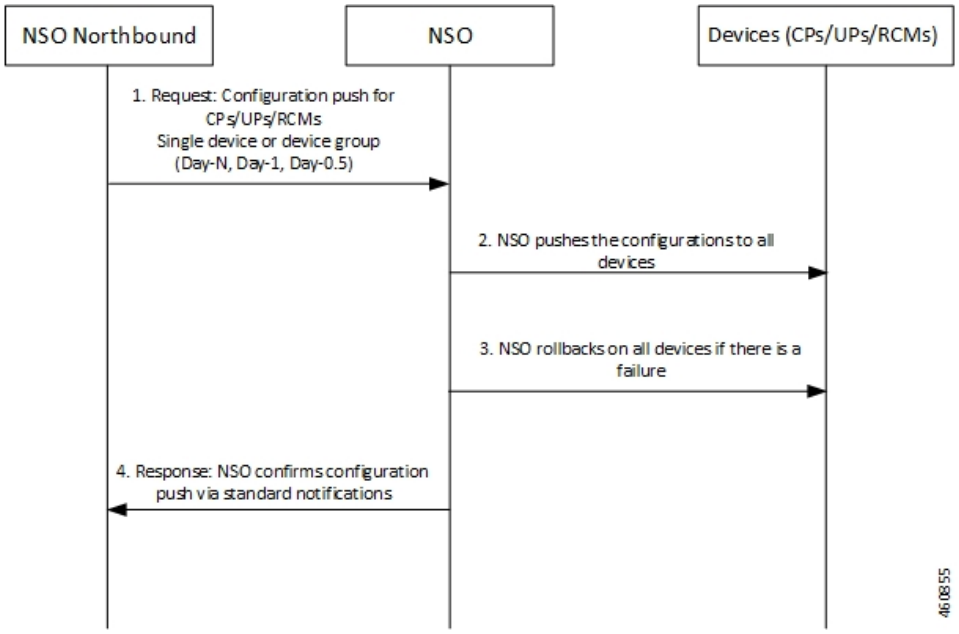


Table 2: Call Flow Description

Step	Description
1	NSO Northbound requests the NSO for a configuration push on devices such as CPs or Ups or RCMs. The devices can be a single device or a device group (Day-N, Day-1, Day-0.5).

Step	Description
2	NSO pushes the configurations to all devices.
3	If there is a failure, NSO rolls back the configuration on all devices. A rollback operation undoes the configuration applied to the device so as to restore it to the previous state (prior to application of the configuration).
4	NSO updates NSO Northbound about the configuration push via standard notifications.

Configuration Push from NSO to 4G CUPS UPs in N:M Redundancy – Automated

This section describes the flow for automatic configuration push from NSO to 4G CUPS devices.

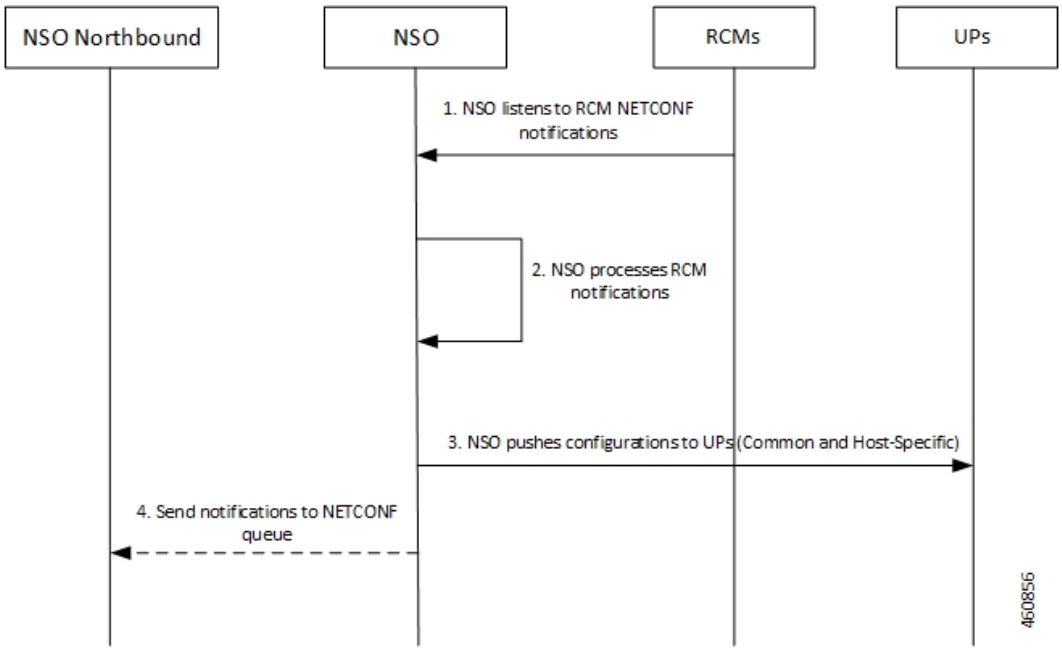


Table 3: Call Flow Description

Step	Description
1	NSO listens to RCM NETCONF notifications. When a UP connects to the RCM, the RCM decides what the UPs role should be (Active or Standby). This role is communicated in the notification. The configuration pushed to the UP is dependent on its role. So, the automated configuration push is driven based on the RCM notification.
2	NSO processes the received RCM notifications.

Step	Description
3	NSO pushes the common and host-specific configurations to UPs. Common configuration refers to the configuration that is shared across all the UPs in a redundancy group. This is typically Enhanced Charging Service (ECS) and Access Point Name (APN) configurations. Host-specific configuration is unique to an Active UP. Each Active UP needs its host-specific configuration. All Standby UPs need the host-specific configurations of all Active UPs, since Standby UP needs to be able to take over for any Active UP.
4	NSO sends notifications to NETCONF queue.

Configuration Metadata Pre-population

This section describes the flow for pre-population of configuration metadata

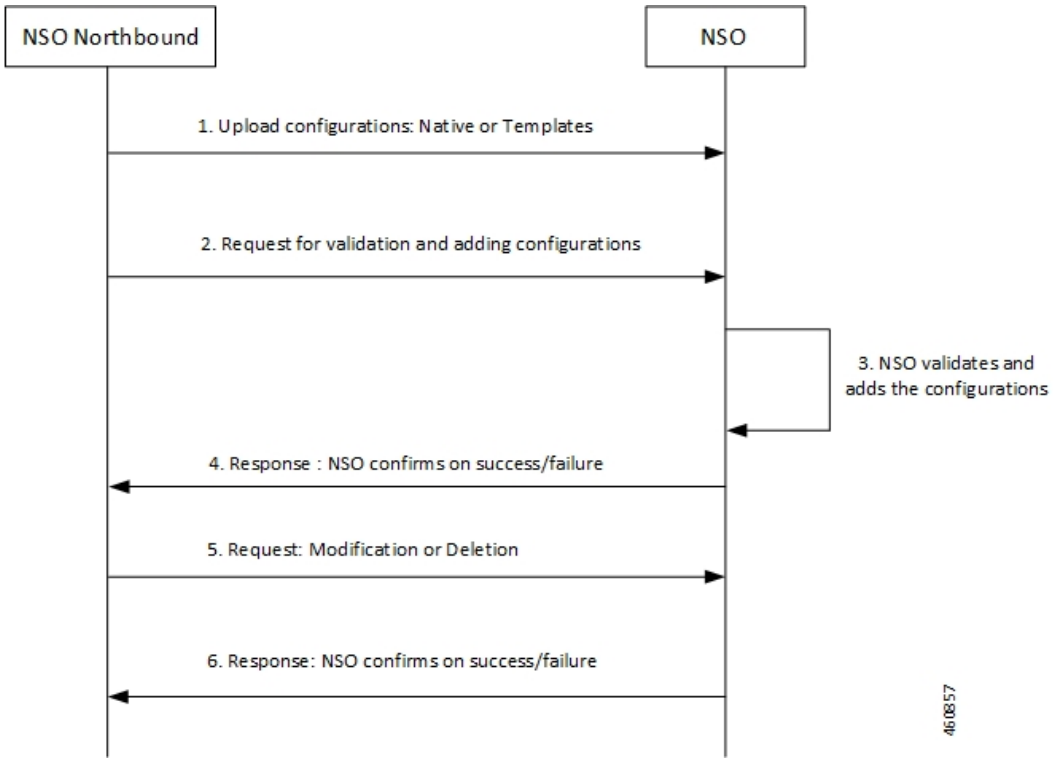


Table 4: Call Flow Description

Steps	Description
1	NSO Northbound uploads the configurations (native or templates) to NSO.
2	NSO Northbound requests NSO to validate and add configurations.
3	NSO validates and adds the configurations.
4	NSO updates NSO Northbound about the success or failure of device addition.

Steps	Description
5	NSO Northbound requests NSO for modification or deletion of configurations.
6	NSO updates NSO Northbound about the success or failure.

NSO HA Switchover Handling

This section describes the flow of handling NSO HA switchover.

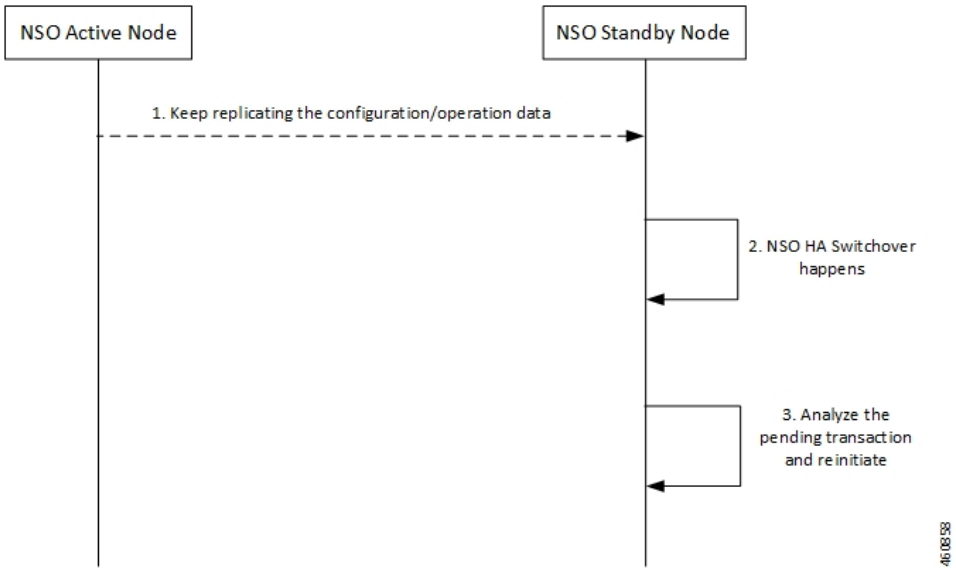


Table 5: Call Flow Description

Step	Description
1	NSO Standby Node keeps replicating the configuration or operation data from the NSO Active Node.
2	NSO HA switchover happens at the NSO Standby Node.
3	NSO Standby Node analyses the pending transaction and reinitiates the process.

Recovery

To recover from fault state to previous state, NSO provides in-built rollback mechanism for the pushed configurations. The following options are available for pushing the configuration to one or more devices:

- Commit or Dry-Runs only
- Commit with Rollback generated
- Scheme of Single or Multiple transactions
- Scheme for failure-handling on multiple transactions

- Scheme for pushing only stand-by nodes, active nodes, or common

CP Switchover (1:1)

The Mobility Function Pack does not actively track the active CP. It tracks on demand, if required, when a configuration push is initiated from northbound. Any configuration pushed to either CP is expected to be stored persistently as a boot configuration on that CP.

Note You must use the MOP option to save the configuration permanently.

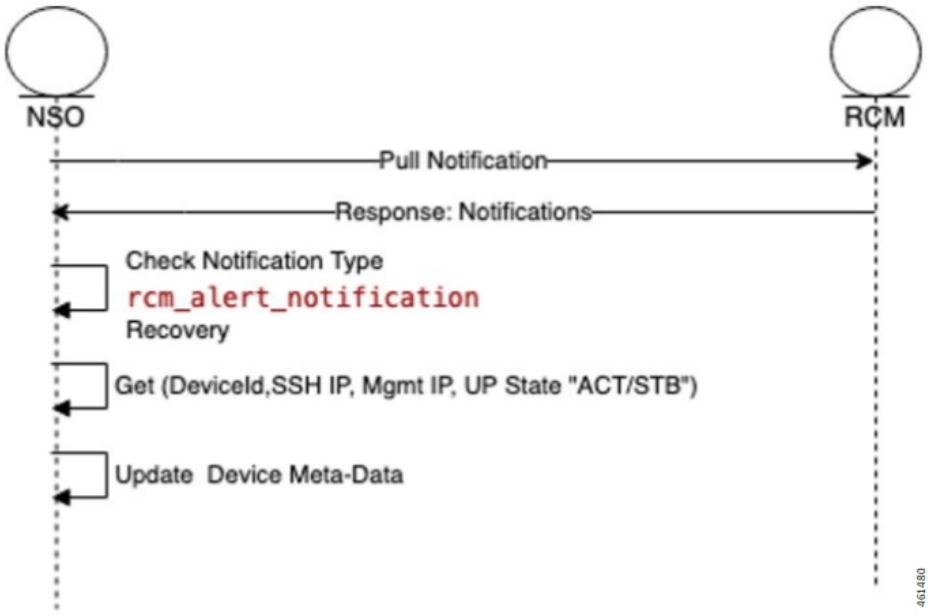
On a CP switchover, the CP that rebooted is expected to boot up with all the necessary configurations. The Mobility Function Pack does not perform any special handling in this scenario.

UP Switchover (1:1)

Like the CP scenario, any configuration pushed to either UP instance is saved persistently as part of the boot configuration. You must use the MOP option to save the configuration permanently. On a UP switchover, the UP that reboots are expected to come up with all the necessary configuration automatically. The Mobility Function Pack does not perform any special actions in this case.

UP Switchover (N:M)

The following figure illustrates the UP Recovery notification flow:



Once the NSO is subscribed to RCM device notification, NSO gets all notifications published to the stream “rcm-alert-notification”.

NSO performs the following steps whenever there is UP Recovery notification:

1. Waits for alert-status “Recovery”

2. Gets device details (Device Name, SSH IP, Management IP, UP Status)
3. Updates the device meta-data in NSO

Out-of-Band Configuration

A fundamental aspect of NSO-based configuration management is that the NSO maintains a copy of each device's (VNFs) configuration. When any configuration changes are applied from Northbound, the NSO compares its local copy of the configuration with the applied configuration to determine precisely what configuration needs to be pushed into the device. For this to occur, the NSOs copy of the configuration has to be in-sync with the actual configuration on the device/VNF.

There can be reasons due to which VNF configuration may be performed out-of-band, bypassing the NED. For example, any Day-0 configuration necessarily precedes onboarding the device into the NSO and thus is out-of-band (it is pushed through the appropriate VNF Manager). The configuration push MOP performs a "sync-from" operation prior to pushing any configuration to a device. This ensures that the NSO pulls any out-of-band configuration into the NSOs local copy, and the attempted configuration push is applied to the most current device configuration. The sync-from can only pull configuration that is known to the NED. Also, there are caveats when dealing with encrypted data.

Sensitive Elements in Configuration

StarOS encrypts sensitive elements in the configuration like passwords, keys, and so on. The encrypted items can only be decrypted by StarOS and are thus opaque to the NSO. Furthermore, the encrypted form of the sensitive item can change even when the underlying cleartext does not. As a result, the NSO cannot reliably detect any out-of-band changes made to such items.

The recommendation is to either:

- Completely manage the corresponding configuration out-of-band

—Or—

- Use only the NSO to manage the corresponding configuration, meaning the cleartext form of the command must be configured into NSO from Northbound initially, and for every subsequent change.

Do not mix NSO-based configuration management and out-of-band management for the same configuration.

Lawful Intercept

Lawful Intercept (LI) can be configured in a couple of different ways. One deployment involves all LI configuration in a single context (without using the dedicated LI context feature), and providing the general system administrator with LI administrator privileges. The other deployment involves a dedicated LI context, segregated LI configuration, and a dedicated LI administrator separate from general system administrator. There are other variations that likely fall in-between.

For the NSO to be able to manage LI configuration, it needs to:

- Have LI privileges and general system administrator privileges
- Be able to view and pull LI configuration in cleartext

For the deployment that involves all LI configuration in a single context scenario, the NSO must manage the LI configuration. For the other cases, it is recommended that the LI configuration be maintained out-of-band and provided as part of Day-0 configuration.

CUPS Configuration MOP

The Configuration MOP is the Method of Procedure (MOP) by which Configuration is applied to Cisco StarOS devices or RCM. This operation is invoked by network operator and in response, NSO provides the unique task-id for the request. Later, network operator can poll NSO using task-id to retrieve the status.

Configuration MOP broadly consists of the following three steps:

1. Device onboarding
2. Prepopulating Config metadata
3. Configuration push through Mobility MOP

Device Onboarding

The Device onboarding step is required only for the devices which are instantiated or orchestrated outside the Mobility orchestration solution. Otherwise, the instantiated VNF is implicitly onboarded onto the NSO as a device by the NSO-based Mobility orchestration solution.

Note This step is required only for the first time. Subsequent configuration pushes should skip this step.

The following examples illustrate how to onboard the VNF using RESTCONF or CLI respectively.

RESTCONF

Patch request for NSO URL: http://<NSO-IP>:<PORT>/restconf/data

The following is an example configuration:

```
{
  "data": {
    "nfv-device-onboarding:nfv-devices": {
      "device": [
        {
          "name": "<Device-or-VNF-Name>",
          "address": "<Management-Address>",
          "username": "<Management-Username>",
          "password": "<Management-Password>",
          "ned-type": "<cisco-staros/RCM>",
          "retry-options": {
            "number-of-attempts": <no-of-attempts-to-ping>,
            "delay": <delay-between-pings>
          }
        }
      ]
    }
  }
}
```

The following is an example configuration:

```
{
  "data": {
    "nfv-device-onboarding:nfv-devices": {
      "device": [
        {
          "name": "vpc-si25",
          "address": "209.165.200.225",
          "username": "admin",
          "password": "Cisco@123",
          "ned-type": "cisco-staros",
          "retry-options": {
            "number-of-attempts": 2,
            "delay": 10
          }
        }
      ]
    }
  }
}
```

CLI

You can also create/populate onboarding of the device using the following NSO CLI commands:

```
configure
  nfv-devices device device_name address ip_address username user_name password
 password ned-type cisco-staros retry-options delay delay_value
number-of-attempts value
  commit
```

The following is an example configuration:

```
nfv-devices device dummy-device address 209.165.200.225 username admin password cisco@123
ned-type cisco-staros retry-options delay 10 number-of-attempts 2
```

Existing device in NSO can be deleted using the Delete request URL, no payload:
http://<NSO-IP>:<NSO-PORT>/restconf/data/nfv-device-onboarding:nfv-devices/device=<device-name>

It can also be deleted using NSO CLI:

```
configure
  no nfv-devices device device_name
  commit
```

Prepopulating Configuration Metadata

The configuration push MOP allows for variable substitution. This is useful in cases where nearly identical configuration is pushed to multiple devices (for example an ICSR active/standby pair). The differences can be represented as variables in the input configuration file. You can then populate the specific values for each device as metadata in a "variable: value" format. The MOP dynamically substitutes the right variable values at runtime.

If there are no prepopulated data for the device, Config MOP assumes that there are no dynamic substitution variables in config files, which are given for configuration push. If any attribute values that are referred in config files are missing, this step fails at runtime.

Prepopulating of config metadata has the following structure, and population of this data is based on the network scheme and data set. Highlighted items are mandatory for config push, and other items are optional.

```
container metadata-store{
    list config-metadata {
        key device-name;
        leaf device-name {
          tailf:info "onboarding device name";
          type string;
         }
 leaf schema {
   tailf:info "cluster-topology 1:1, N:M and N+2";
   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;
        }
      }
        }
    }
  }
   }
}
```

Configuration meta-data is populated using the configuration meta-data request. This request follows the following YANG schema. The items in the "input" section are to be provided by the operator. The "output" section represents what is returned by the NSO after execution the action request.

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

```
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;
   }

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

RESTCONF

The following is an example to call this action from RESTCONF:

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

Content-Type: application/yang-data+json

Payload:

```
{
        "config-metadata": {
            "device-name": "test2",
     "schema" : "1:1",
            "attributes":{
                "attribute-name":"hostname",
                 "attribute-value": "TEST"
                  },
                "attribute-name":"BACKHAUL_IP",
                 "attribute-value": "209.165.200.225"
                }
        }
}
```

Result:

```
{
    "mobility-common:output": {
        "status": "Success"
    }
}
```

CLI

The following is an example to call this action using NCS CLI:

```
ubuntu@ncs> request config-metadata config-metadata-request config-metadata { device-name
staros-1 attributes { attribute-name hostname attribute-value TEST }
status Success
[ok][2021-07-12 08:05:01]
```

Configuration Push through Mobility MOP

This step is the final step in the configuration MOP. It allows you to push a fresh configuration or rollback a previously pushed configuration. The configuration to be pushed is present in one or more files as mentioned previously

Configuration MOP Push Request Flow

Network Operator invokes NSO API to start the process of config MOP automation for devices

NSO performs the below steps:

- Perform check-sync and sync-from or partial sync (if required) for the device. The check-sync determines if the NSOs copy of the device configuration is already in-sync with the actual device configuration.
- If specified in the MOP, NSO replaces the device attributes (variables) with node specific values read from device tree of the config metadata.
- NSO applies the configuration from the input files specified in the MOP to the device or list of devices in the order specified in the request. If there is a failure when pushing the configuration towards a device, then no further configuration is pushed to that device.
- NSO applies MOP(s) based on the mop type provided in the request (active/standby/common).
    - If the mop type is “common”, then NSO applies MOP(s) for all the devices provided in the request.

      In case of a failure, configuration push to the device(s) that encountered the failure is halted. Configuration push to other devices in the request continues. The Status shows the details of the failed devices. The operator then gets the option of rolling back the configuration on the failed device(s) as a separate action.
    - If the mop type is “active”, then NSO applies MOP(s) for all the “active” devices provided in the request

      The mop type “active” applies only to 1:1 redundancy scenario.

      In case of a failure, any pushed configuration is rolled back.
    - If the mop type is “standby”, then NSO applies MOP(s) for all the “standby” devices provided in the request.

      The mop type “standby” applies only to 1:1 redundancy scenario.

In case of a failure, any pushed configuration is rolled back.

- If the mop type is “pair”, then NSO applies MOP(s) first on the “standby” device and if successful, MOP(s) is applied on the “active” device. It performs the atomic transaction, so the configuration is applied to either both or neither device.

  The mop type “pair” applies only to 1:1 redundancy scenario.

  In case of a failure, any applied configuration is rolled back from both instances of the pair.

- If the mop type is “rcm-upf”, then NSO applies MOP(s) on the input device. Additionally, it identifies the RD-group of the input device and finds out the other UPF devices present in the same RD-group. Then it saves the ECS/APN config on the input device.

  In case of a failure, configuration push to the device(s) that encountered the failure is halted. Configuration push to other devices in the request continues. The Status shows the details of the failed devices. The operator then gets the option of rolling back the configuration on the failed device(s) as a separate action.

- NSO generates dry-run and reverse (rollback) configuration in native format for the MOP(s) supplied and stores in two separate files. In response, NSO returns both the file names along with absolute file path to the network operator.
  - Dry-run file is named as <MOP File Name>-<Device Name>-dryrun.txt.
  - Rollback file name is named as <MOP File Name>-<Device Name>-rollback.txt. Files are generated under the task id folder.

- If the network operator sends a request only for dry-run, then NSO generates dry-run and rollback files, but does not apply the MOP towards the device.
- If the network operator sends a request to apply the MOP, NSO generates dry-run and rollback files, and then applies the MOP towards the device.
- Network operator keeps on polling NSO for MOP automation status.
- NSO returns the list of hosts (devices) along with dry-run and rollback file location, and the status (Completed/In-Progress/Failed).

Configuration MOP Rollback Request Flow

- Network Operator invokes NSO API to start the process of rollback of a previously applied configuration.
- NSO performs the following steps:
  - Perform check-sync and sync-from or partial sync (if required) for the device.
  - NSO performs the rollback of MOP files in the reverse order for the task ID, MOP file name, and device name supplied by the network operator.

- If the MOP type is “pair”, then NSO performs rollback first on the “standby” device and after successful rollback, NSO performs rollback on the “active” device.
- If only task ID is supplied, then the whole transaction is rolled back. If task ID and MOP file name(s) are supplied, then only supplied MOPs are rolled back for all the devices. If task ID, MOPs file name, and devices names are supplied, then only supplied MOP for supplied devices are rolled back.

- NSO generates dry-run and reverse (rollback) configuration in native format for the rollback to be done, and stores in files. In response, NSO returns both file names along with absolute file path to the network operator.
  - Dry-run file is named as <MOP File Name>-<Device Name>-dryrun.txt
  - Rollback file is named as <MOP File Name>-<Device Name>-rollback.txt

  Files are generated under the task ID folder.
- If the network operator sends a request only for dry-run, then NSO generates dry-run and rollback files.
- If the network operator sends a request to roll back the MOP, NSO generates dry-run and rollback files, and then performs rollback.
- Network operator keeps on polling NSO for rollback status.
  - NSO returns the list of hosts (devices) along with dry-run and rollback file location, and the status (Completed/In-Progress/Failed)

MOP Automation

The Mobility configuration MOP is a set of commands that can be used to configure mobility devices from the NSO. This allows end user to specify locations to find or save the MOP-related input files and output files. It also allows the end user to setup global configurable parameters for MOP.

Configuration Prerequisites

- Navigate to NSO CLI and use static action to set the below parameters:
  1. Dry-run-mop location: Dry-run-mop file contains the configurations pushed to the device. Enter the location to save dry-run files of the MOP.
  2. Rollback-mop location: Rollback files are the configuration files generated that are required to roll back the configuration on device. Enter the location to save rollback files of the MOP.
  3. Config-mop-file location: Enter the location to fetch input configuration MOP files.
  4. Netconf-to-cli Conversion: Set the flag as “true” to convert NETCONF configuration to device CLI format. If the flag is set as “false”, then the dry-run file is generated in native NETCONF xml format.
- Static action call command in configuration:
- **static dry-run-mop /var/opt/ncs/**
  static rollback-mop /var/opt/ncs/

  To verify, use the following CLI command:

  show full-configuration static
- Global parameter configuration for mop-file location:

  configure
  configurable-parameters config-mop-file-loc /var/opt/ncs/

  To verify, use the following CLI command:

  show full-configuration configurable-parameters config-mop-file-loc

- StarOS-level NED setting examples

1. To prevent configuration update in the system cfg boot files of the devices, ensure that the write-memory-setting is disabled in the NCS CLI using the following command:

   ```
   devices global-settings ned-settings cisco-staros
   write-memory-setting disabled
   ```

2. Use the following command to exclude warnings while committing the configurations to the device:

   ```
   devices global-settings ned-settings cisco-staros behaviour
   config-warning-ignore.*Standby card not ready.*
   ```

Note Here, .*Standby card not ready.* can be replaced with the warning message to be ignored.

Mop-type Pair Prerequisites

- One of the device names, based on the identification of the device state (Active/Standby), can be specified as target-device-name.
- Configure the peer device and srp_loopback using the following commands:

```
config-metadata config-metadata-request config-metadata { device-name
up2-SI device-type vpc attributes { attribute-name srp_loopback
attribute-value 209.165.200.225 } scheme 1:1 }
config-metadata config-metadata-request config-metadata { device-name
up2-SI device-type vpc attributes { attribute-name Peer_Device_Name
attribute-value up1-SI } scheme 1:1 }
```

NSO APIs

NSO APIs are exposed by the NSO Mobility function pack that relate to configuration push functionality. These APIs are accessible either over RESTCONF or CLI.

Configuration Push MOP Automation

This API is used to start the MOP for pushing configuration to one or more devices. This is an asynchronous operation, and the status can be queried using a separate API.

API:

```
mop-automation
```

Request Details

Parameter	Format	Required	Description
mop-file-name	List	—	
file-name	String	Key	Name of the device that corresponds to the UP, CP, or RCM.

Parameter	Format	Required	Description
execution-order	Int	Mandatory	MOP execution order. 1 means first – order used is 1, 2, 3….
target-devices	List	Mandatory	Devices list
target-device-name	Leafref	Key	The NSO device name for VNF. If NSO is used for orchestration, the device name is the same as the VNF name.
operation-type	String	Mandatory	**dry-run** or **commit**

Parameter	Format	Required	Description
mop-type	Enum	—	

Parameter	Format	Required	Description
			Determines which device(s) to push the configuration to. The allowed values are: • **active** Push to the active instance of a 1:1 redundancy pair. You can enter the device name for one of the instances of the redundancy pair (regardless of whether it is active or standby). The MOP automatically determines the currently active instance of that pair and pushes. • **standby** Push to the standby instance of 1:1 redundancy pair. You can enter the device name for one of the instances of the redundancy pair (regardless of whether it is active or standby). The MOP automatically determines the currently standby instance of that pair and pushes. • **pair** Push to both active and standby instances of a 1:1 redundant pair. You can enter the device name for one of the instances of the redundancy pair (regardless of whether it is active or standby). The MOP automatically determines both instances of the pair using the supplied instance, pushes to the standby instance first, and then to the active instance. • **common**

Parameter	Format	Required	Description
			Push to all devices provided in the request. This is the **default**. • **rcm-upf** Push configuration to either single or all associated UPFs.

Parameter	Format	Required	Description
transaction-type	Enum	—	The allowed values are: • **single-transaction** The configuration in all the supplied input files is combined and pushed to the device as a single transaction. • **multiple-transaction** Each input file is pushed to the device as a separate transaction. This is the default value. A transaction is an atomic unit of configuration change. All configuration in a transaction will either be pushed successfully or will be rolled back automatically if there was a failure during the push. **Note** A transaction will not span multiple devices. Each transaction is specific to a single device. For example, if the operator pushes 3 files each to 2 devices, then: • with the **multiple-transactions** option, a total of 3x2 = 6 transactions, one per device, per file are present. • with the **single-transaction** option, a total of 2 transactions, 1 per device is present.
save-config-permanently	Boolean	—	The default value is "false". When set to "true", you can save the configuration to "system.cfg".

Parameter	Format	Required	Description
timeout	Int	Optional; default value is 600 seconds	The maximum number of seconds to wait for the device to be locked.

Timeout Parameter

The NSO 6.0 version uses optimistic concurrency to improve parallelism. However, transaction conflicts can occur when services are executed in parallel. When a single device is configured concurrently, the initial transaction locks the device, causing subsequent transactions to fail.

The timeout parameter determines how long MFP will wait for some operations related to the device such as getting a device lock. This is relevant if there are multiple operations that push configuration to the same device simultaneously.

Note It is not recommended to configure a single device parallelly at the same time.

If the **timeout** parameter is not used while pushing config with the **mobility-mop** automation CLI or postman API, the system will automatically invoke a default value of 600 seconds.

If you want to push many configs or larger size of config, you can make a call to set the timeout value beyond the default value using the timeout parameter.

You can specify any value as shown in the following configuration example:

```
cloud-user@ncs# mobility-mop:action mop-automation generate-dry-run true operation-type
       commit save-config-permanently true mop-type common mop-file-name { file-name ABC.cfg
 order
        1 target-devices-list { target-device-name XYZ } } timeout 900
```

You can set the timeout parameter as infinity by specifying the timeout value as -1:

```
cloud-user@ncs# mobility-mop:action mop-automation generate-dry-run true operation-type
       commit save-config-permanently true mop-type common mop-file-name { file-name ABC.cfg
 order
        1 target-devices-list { target-device-name XYZ } } timeout -1
```

As soon as the config is pushed, the device is freed for another user or for another round of config push. For example, if timeout is 600 seconds and config push completes in 100 seconds, the device can be used by another user for config push after 100 seconds.

Response details

Parameter	Format	Required	Description
Task-id	String	Mandatory	Task unique identifier. The Task-id is to be used to query the status of the operation.
Time stamp	String	Mandatory	Time stamp
Error Code	String		Error Code
Error Message	String		Error Message

CLI

The following is an example of a request with NCS CLI:

```
mobility-mop:action mop-automation mop-type common transaction-type
multiple-transaction operation-type commit mop-file-name { file-name
dayN.txt order 1 target-devices-list { target-device-name up2-SI } }
```

REST API Request – Without Specifying Transaction Type

The following is an example of REST API request using postman without specifying the transaction type:

```
POST  /restconf/data/mobility-mop:action/mop-automation
   Host: localhost:8080
      Authorization: Basic YWRtaW46YWRtaW4= Content-Type: application/vnd.yang.data+json
cache-control: no-cache

         {
            “mop-automation”: {
            “operation-type”: “commit”,
            “mop-type”: “active”,
            “generate-dry-run”: “true”,
            “save-config-permanently”: “true”,
            “mop-file-name”: [
               {
                  “file-name”: “up_dayN.txt” ,
                  “order”: 1,
                  “target-devices-list”: [
                     {
                        “target-device-name”: “up2-SI”
                     }
                  ]
               }
          ]
      }
}
```

REST API Request – With Specifying Transaction Type

The following is an example of REST API request with specifying the transaction type:

```
POST    /restconf/data/mobility-mop:action/mop-automation
   Host: localhost:8080
      Authorization: Basic YWRtaW46YWRtaW4= Content-Type: application/vnd.yang.data+json
cache-control: no-cache
         Postman-Token: d2d2ddbe-5dff-4917-972a-146db6dc175f

            {
               "mop-automation": {
               "operation-type": "commit",
               "mop-type": "active",
               “transaction-type”: “single-transaction”,
               "mop-file-name": [
                  {
                     "file-name": "load3.txt",
                     "order": 1,
                     "target-devices-list": [
                        {
                           "target-device-name": "test3"
                        }
                     ]
                  },
               {
                   "file-name": "load4.txt",
```

```
                    "order": 2,
                    "target-devices-list": [
                {
                    "target-device-name": "test3"
                }
            ]
        }
    ]
  }
}
```

REST API Request – Without Specifying Transaction Type and mop-type as Pair

The following is an example of REST API request without specifying the transaction type and mop-type as pair:

```
 {
    "mop-automation": {
                        "operation-type": "commit",
                        "mop-type": "pair",
                        "generate-dry-run": "true",
                        "save-1-1-config": "true",
                        "mop-file-name": [
                    {
                       "file-name": "up_dayN.txt" ,
                       "order": 1,
                       "target-devices-list": [
                {
                    "target-device-name": "up2-SI"
                }
            ]
        }
    ]
  }
}
```

In response to successful invocation of above asynchronous requests, a unique task-id and time stamp is returned which is used to check the status of the mop-automation request.

```
{
    "mobility-mop:output": {
        "task-id": "1a1f62f0-487a-4c8c-bdeb-a760c26925cc",
        "time-stamp": "2021-07-19T11:10:51+0000",
        "time-zone": "Coordinated Universal Time"
    }
}
```

MOP Automation with mop-type as rcm-upf

Mop-type rcm-upf is used to push configuration to either single or all associated UPFs. The following two scenarios are applicable:

1. Apply MOP on single UPF.

   The following are ways to specify the UPF device:

   - Specify the upf-device in target-device-name.
   - Specify the rcm-vip, group, and device-id corresponding to upf-device.

   For the above two ways, the “only-to-target-devices” must set to “true” in the request.

   Payload Examples:

a. Specify the upf-device in target-device-name:

```
{
    "mop-automation": {
        "operation-type": "commit",
        "mop-type": "rcm-upf",
        "generate-dry-run": "true",
        "save-config-permanently": "true",
        "only-to-target-devices": "true",
        "mop-file-name": [
            {
                "file-name": "simpleStarOsChange.txt",
                "order": 1,
                "target-devices-list": [
                    {
                        "target-device-name": "up1-device"
                    }
                ]
            }
        ]
    }
}
```

b. Specify the rcm-vip, group, and device-id corresponding to upf-device:

```
{
    "mop-automation": {
        "operation-type": "commit",
        "mop-type": "rcm-upf",
        "generate-dry-run": "true",
        "save-config-permanently": "true",
        "only-to-target-devices": "true",
        "mop-file-name": [
            {
                "file-name": "simpleStarOsChange.txt",
                "order": 1,
                "rcm-vip" : "rcmvip01",
                "group" : "group03",
                "device-id" : "device-id1"
            }
        ]
    }
}
```

2. Apply MOP on all the associated UPF devices.

The following are the ways to identify the UPF devices:

- Specify a sample upf-device in target-device-name.
- Specify a rcm-vip and group.

Payload Examples:

a. Specify a sample upf-device in target-device-name:

```
  {
"mop-automation": {
    "operation-type": "commit",
    "mop-type": "rcm-upf",
    "generate-dry-run": "true",
    "save-config-permanently": "true",
    "only-to-target-devices": "false",
    "mop-file-name": [
```

```
                {
                    "file-name": "simpleStarOsChange.txt",
                    "order": 1,
                    "rcm-vip" : "rcmvip01",
                    "group" : "group03"
                }
            ]
        }
    }
```

b. Specify a rcm-vip and group.

```
{
    "mop-automation": {
        "operation-type": "commit",
        "mop-type": "rcm-upf",
        "generate-dry-run": "true",
        "save-config-permanently": "true",
        "only-to-target-devices": "false",
        "mop-file-name": [
            {
                "file-name": "simpleStarOsChange.txt",
                "order": 1,
                "rcm-vip" : "rcmvip01",
                "group" : "group03"
            }
        ]
    }
}
```

For the above two ways, the “only-to-target-devices” must be set to “false” in the request.

To retrieve the UPF device using rcm-vip and group, the data available in the following lists in CDB are used:

- device-id-up-mapping
- up-rcm-mapping
- rcm-upf-mapping

MOP Automation Status

NSO provides device status results for the task-id passed by the network operator.

API:

`mop-automation-status`

Request details

Parameter	Format	Required	Description
Task-id	String	Mandatory	Task unique identifier obtained from the "MOP Automation" API.

Response details

Parameter	Format	Required	Description
task-id	String	key	Task ID

Parameter	Format	Required	Description
task-status	String		Task status
Start-date	String		Start Date Time
End-date	String		End Date Time
Time-zone	String		Time zone
Operation-type	String		Commit/Dry-run
Action-type	String		Save
devices-list	list		Devices
Device-name	leafref	key	Device name
Start-date	String		Start Date Time
End-date	String		End Date Time
device-status	leafref		Device Status (Completed/In-Progress/Failed)
device-state	String		Active/Standby/Common /Pair/rcm-upf
files	list		Files
file-name	String	key	MOP file name
Order	Uint8		Order in which MOP has been performed
dry-run-mop	String		Dry-run output file location
rollback-mop	String		Rollback MOP location
Commit-queue-status	String		Commit queue status
Commit-queue-id	String		Commit queue ID
Error Code	String		Error Code
Error Message	String		Error Message
Error Code	String		Error Code
Error Message	String		Error Message

CLI

The following is an example of a request with NCS CLI:

```
mobility-mop:action mop-automation-status task-id
8d08e359-0bd2-48de-9a34-9192a986a486
```

REST API Request

The following is an example of REST API request to know the status of the mop-automation:

```
POST /restconf/data/mobility-mop:action/mop-automation-status
   Host: localhost:8080
```

```
      Authorization: Basic YWRtaW46YWRtaW4= Content-Type: application/vnd.yang.data+json
cache-control: no-cache

      {
         "task-id": "22301071-9a6c-4f27-a0dc-b50c24124806"
      }
```

The following is an example of response format generated, post invocation of the above request:

```
{
    "mobility-mop:output": {
        "task-id": "8d08e359-0bd2-48de-9a34-9192a986a486",
        "task-status": "COMPLETED",
        "start-date": "2021-09-06T09:08:54+0000",
        "end-date": "2021-08-06T09:09:10+0000",
        "time-zone": "Coordinated Universal Time",
        "operation-type": "commit",
        "action-type": "save",
        "devices-list": [
            {
                "device-name": "up2-SI",
                "device-status": "COMPLETED",
                "start-date": "2021-08-06T09:08:55+0000",
                "end-date": "2021-08-06T09:08:59+0000",
                "device-state": "active",
                "files": [
                    {
                        "file-name": "up_dayN.txt",
                        "order": "1",
                        "dry-run-mop":
"/var/opt/ncs//8d08e359-0bd2-48de-9a34-9192a986a486/up2-SI/up_dayN_commit_2021-08-06T090854+0000.txt",

                        "rollback-mop":
"/var/opt/ncs//8d08e359-0bd2-48de-9a34-9192a986a486/up2-SI/up_dayN_rollback_commit_2021-08-06T090854+0000.txt",

                        "commit-queue-status": "completed",
                        "commit-queue-id": "1628240937590"
                    }
                ]
            }
        ]
    }
}
```

MOP Rollback

NSO starts the process of rollback for task ID, MOP file, and devices provided in the input of the request.

This is the only option to roll back the MOP-configured configs or rolled-back configs.

This API rolls back a previously applied configuration. This uses the rollback files creating while originally applying the configuration. Rollback can be done per file, or for all files, and per device, or for all devices.

Note The success of a rollback is highly dependent on what changes have been made to the system since the relevant configuration was pushed. Subsequent changes may have changed the system such that the rollback configuration may not make sense.

API:

`mop-rollback`

Request details

Parameter	Format	Required	Description
Task-id	String	Mandatory	Task Unique Identifier
mop-file-name	List	Optional	MOP files along with device
file-name	String	Key	The original file name, used for identifying the corresponding rollback file.
target-devices	List	Optional	Device list. If not provided, the rollback is performed on all devices to which configuration was pushed in the original transaction.
target-device-name	Leafref	Key	
operation-type	String	Mandatory	Dry-run/Commit
timeout	Int	Optional; default value is 600 seconds	The maximum number of seconds to wait for the device to be locked.

Response details

Parameter	Format	Required	Description
Task-id	String	Mandatory	Task unique identifier. The task-id is to be used to check the status of the rollback via a separate API.
Time stamp	String	Mandatory	Time stamp
Error Code	String	Error Code	
Error Message	String	Error Message	

CLI

The following is an example of a request with NCS CLI:

```
mobility-mop:action mop-rollback task-id
8d08e359-0bd2-48de-9a34-9192a986a486 generate-dry-run true operation-type
 commit save-config-permanently true mop-file-name { file-name up_dayN.txt
 target-devices-list { target-device-name up2-SI } }
```

REST API Request – With Operation Type "commit"

The following is an example of REST API request with operation-type "commit":

```
POST /restconf/data/mobility-mop:action/mop-rollback
   Host: localhost:8080
      Authorization: Basic YWRtaW46YWRtaW4= Content-Type: application/vnd.yang.data+json
cache-control: no-cache
         Postman-Token: 1b687031-dc32-41 14-a69f-5984130c36a5
         {
            "mop-rollback": {
            "task-id": "0891655c-642b-4ba3-9392-6f05d4e77a63",
            "operation-type": "commit",
            "generate-dry-run": "true",
            "save-config-permanently": "true",
            "mop-file-name": [
            {
               "file-name": "up_dayN.txt",
               "target-devices-list": [
               {
                  "target-device-name": "up2-SI"
               }
            ]
         }
      ]
   }
}
```

In response to a successful invocation of above request, a unique task-id and time stamp is returned which is used to check the status of the mop-rollback request.

```
{
    "mobility-mop:output": {
        "task-id": "8d08e359-0bd2-48de-9a34-9192a986a486",
        "time-stamp": "2021-08-06T09:08:44+0000",
        "time-zone": "Coordinated Universal Time"
    }
}
```

REST API Request – With Operation Type “dry-run”

The following is an example of REST API request with operation-type "dry-run":

```
POST /restconf/data/mobility-mop:action/mop-rollback
   Host: localhost:8080
      Authorization: Basic YWRtaW46YWRtaW4= Content-Type:
application/vnd.yang.data+json cache-control: no-cache
      Postman-Token: 1b687031-dc32-41 14-a69f-5984130c36a5
      {
         "mop-rollback": {
         "task-id": "0891655c-642b-4ba3-9392-6f05d4e77a63",
         "operation-type": "dry-run",
         "generate-dry-run": "true",
         "save-config-permanently": "true",
         "mop-file-name": [
         {
            "file-name": "up_dayN.txt",
            "target-devices-list": [
            {
                  "target-device-name": "up2-SI"
               }
            ]
         }
      ]
   }
}
```

In response to a successful invocation of above request, a unique task-id and time stamp is returned which is used to check the status of the mop-rollback request.

```
{
    "mobility-mop:output": {
        "task-id": "1a1f62f0-487a-4c8c-bdeb-a760c26925cc",
        "time-stamp": "2021-07-19T11:10:51+0000",
        "time-zone": "Coordinated Universal Time"
}
```

MOP Rollback Status

NSO provides device status results for the task-id passed by the network operator. API to query the status of an ongoing or completed rollback operation.

API:

`mop-rollback-status`

Request details

Parameter	Format	Required	Description
Task-id	String	Mandatory	Task unique identifier of the rollback operation.

Response details

Parameter	Format	Required	Description
task-id	String	key	Task unique identifier
task-status	String		Task status
Start-date	String		Start Date Time
End-date	String		End Date Time
Time-zone	String		Time zone
Operation-type	String		Commit/Dry-run
Action-type	String		Rollback
devices-list	list		Devices
Device-name	leafref	key	Device name
Start-date	String		Start Date Time
End-date	String		End Date Time
device-status	leafref		Device Status (Completed/In-Progress/Failed)
device-state	String		Active/Standby/Common/Pair
files	list		Files
file-name	String	key	MOP file name
Order	Uint8		Order in which MOP has been performed
dry-run-mop	String		Dry run output file location

Parameter	Format	Required	Description
rollback-mop	String		Rollback MOP location
Commit-queue-status	String		Commit queue status
Commit-queue-id	String		Commit queue ID
Error Code	String		Error Code
Error Message	String		Error Message
Error Code	String		Error Code
Error Message	String		Error Message

CLI

The following is an example of a request with NCS CLI:

```
mobility-mop:action mop-rollback-status task-id
fd0fb9ae-8685-420e-9490-0c6858d14148
```

REST API Request

The following is an example of REST API request to know the status of the mop-rollback:

```
POST /restconf/data/mobility-mop:action/mop-rollback-status
Host: localhost:8080
Authorization: Basic YWRtaW46YWRtaW4= Content-Type: application/vnd.yang.data+json
cache-control: no-cache
Postman-Token: 0e2c4bd3-2dc6-4ddb-aea9-1 1 0ccf622da7
"mop-rollback -status": {
"task-id": "5733d661-9242-4867-8320-a314da592c93"
}

Below is response format generated, post invocation of the above request -
task-id fd0fb9ae-8685-420e-9490-0c6858d14148
task-status COMPLETED
start-date 2021-08-06T09:24:14+0000
end-date 2021-08-06T09:24:30+0000
time-zone Coordinated Universal Time
operation-type commit
action-type rollback
devices-list {
    device-name up2-SI
    device-status COMPLETED
    start-date 2021-08-06T09:24:14+0000
    end-date 2021-08-06T09:24:19+0000
    device-state active
    files {
        file-name up_dayN_rollback_commit_2021-08-06T090854+0000.txt
        order 1
        dry-run-mop /var/opt/ncs//fd0fb9ae-8685-420e-9490-0c6858d14148/up2-SI
/up_dayN_2021-08-06T090854+0000_rollback_commit_2021-08-06T092414+0000.txt
        rollback-mop /var/opt/ncs//fd0fb9ae-8685-420e-9490-0c6858d14148/up2-SI
/up_dayN_2021-08-06T090854+0000_commit_2021-08-06T092414+0000.txt
        commit-queue-status completed
        commit-queue-id 1628241856973
    }
}
```

Verifying the dry-run and Reverse dry-run MOP

To verify the dry-run MOP and reverse dry-run MOP, go to respective file location which was provided while configuring static data for dry-run MOP and reverse dry-run MOP.

Adding Variables to Configuration File for MOP Execution

The MOP automation package supports specifying variables in the MOP so that they are populated at runtime based on what device the MOP is being applied to. For example, if the following MOP was specified and was executed on device TXPCF003:

```
config context local administrator $Host_name password Nsotest123$ exit
end
```

The Host Name can be configured using the following action call:

```
config-metadata config-metadata-request config-metadata { device-name
up2-SI device-type vpc attributes { attribute-name Host_name
attribute-value TXPCF003} scheme 1:1 }
```

The dry-run MOP that would be generated is as follows:

```
config context local administrator TXPCF003 password Nsotest123$ exit end
```

UP Configuration Push and Recovery in N:M Redundancy

In the N:M scenario, the RCM determines the role of each UP (active versus standby). Since any of the M standby instances must be capable of taking over for any of the N active instances, the configurations to be pushed are different and dynamic. This also means not all configuration can be saved on the UPs persistently.

RCM issues NETCONF notifications for relevant events such as a UP booting up or UP switchover. NSO listens to those notifications and applies the necessary configuration as appropriate.

The configuration for a UP consists of the following logical components:

- Day-0 configuration: This is primarily basic configuration for the UP's management interface to be reachable. This is pushed at the time of UP deployment by the VNFM. This configuration is expected to be persistent across reboot.

Note The require

```
rcm-configmgr CLI
```

command must be configured on the UP as part of Day-0 configuration for the NSO-based configuration push to work. This command is required irrespective of whether RCM is used in the solution or not. Without configuring this command, the ECS configuration push appears hidden.

- Day-0.5 configuration: This is configuration that allows the UP to contact the RCM. This configuration can be pushed either along with day 0, or pushed separately from the NSO, either automatically, right after UP deployment (if NSO is deploying the VM), or by a manual execution of the config push MOP. This configuration is also expected to be saved persistently across reboots.
- Common configuration: This is configuration that is common to all UPs regardless of whether they are active or standby. This is ECS and APN configuration only. This configuration needs to be pre-populated

in the NSO. NSO will push this upon receiving notifications from the RCM. This configuration is not saved persistently as part of the boot configuration but is saved locally as a file on each UP and re-applied on every reboot by the NSO automatically.

- Host-specific configuration: This is configuration that is unique to each active UP. This is primarily the various service IP addresses. Each active UP is pushed the configuration specific to that active instance. Each standby instance is pushed the combined host-specific configurations of all active UPs. This configuration is expected to be pre-populate don the NSO. NSO will push this to each UP as appropriate based on the notification from the RCM.
- Host-specific configuration - RCM copy: This is the host-specific configuration of each UP, however, formatted in RCM compatible format. This needs to be pushed to the RCM. While RCM is not involved in configuration for the most part, it is still involved in performing config negation during UP switchover. Config negation means removing the configuration of all the other active UPs from the standby UP that is about to take over for a given active UP. So, say, in a 3:1 scenario, Active3 UP goes, down. The standby has the host-specific configurations of Active1, Active2, and Active3. Since the standby now takes over for Active3, the RCM negates the configs of Active1 and Active2 from that standby as part of the switchover.

NETCONF Notification Subscription on NSO

All notifications sent from RCM are captured by NSO. NSO filters the notifications and handles RCM related notifications.

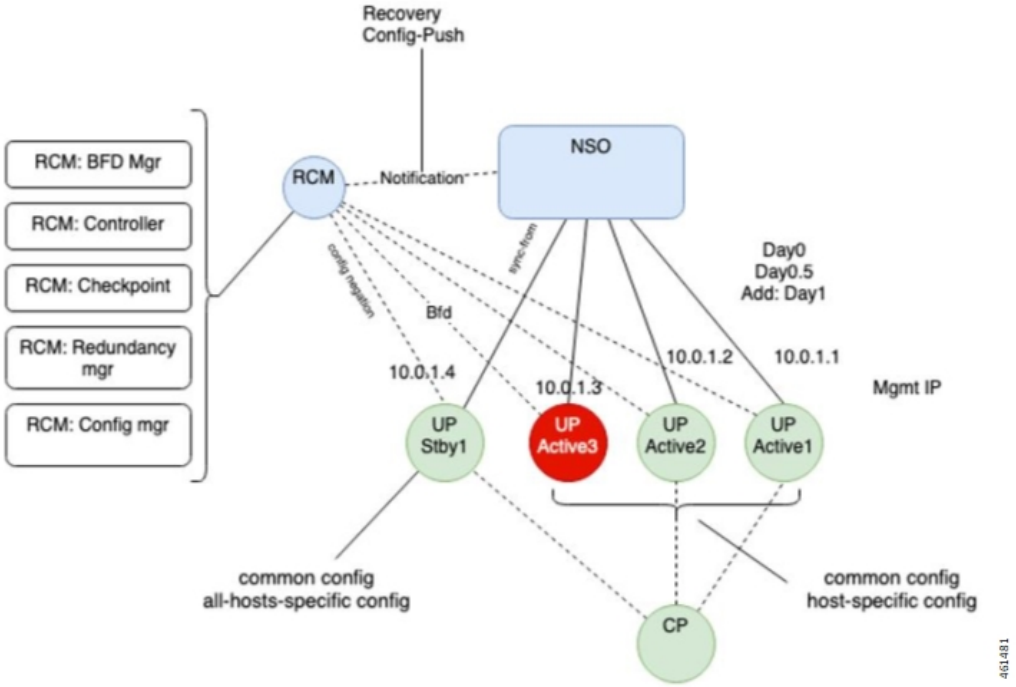
The following table explains the possible types of RCM UP notifications, and how they are handled by NSO.

	Recovery	**Config-Push**	
	UP Recovery	**UP Reboot**	**New UP**
Active UP	Update Device meta-data in NSO.	**Push host** specific config. If common config is not on device, re-push the common config file. Update device meta-data in NSO.	
Standby UP	N/A	Push **all hosts** specific config. If common config is not on device, re-push the common config file. Update device meta-data in NSO.	

Handle RCM UP Recovery Notification

In case of UP failure, RCM detects the failure via BFD Manager, and pushes the notification, which is received by NSO. RCM handles the switchover of the UP to make an elected standby UP to an active one. This configuration management process for the standby UP to switchover does not take much time because the standby UP already has all the required configuration.

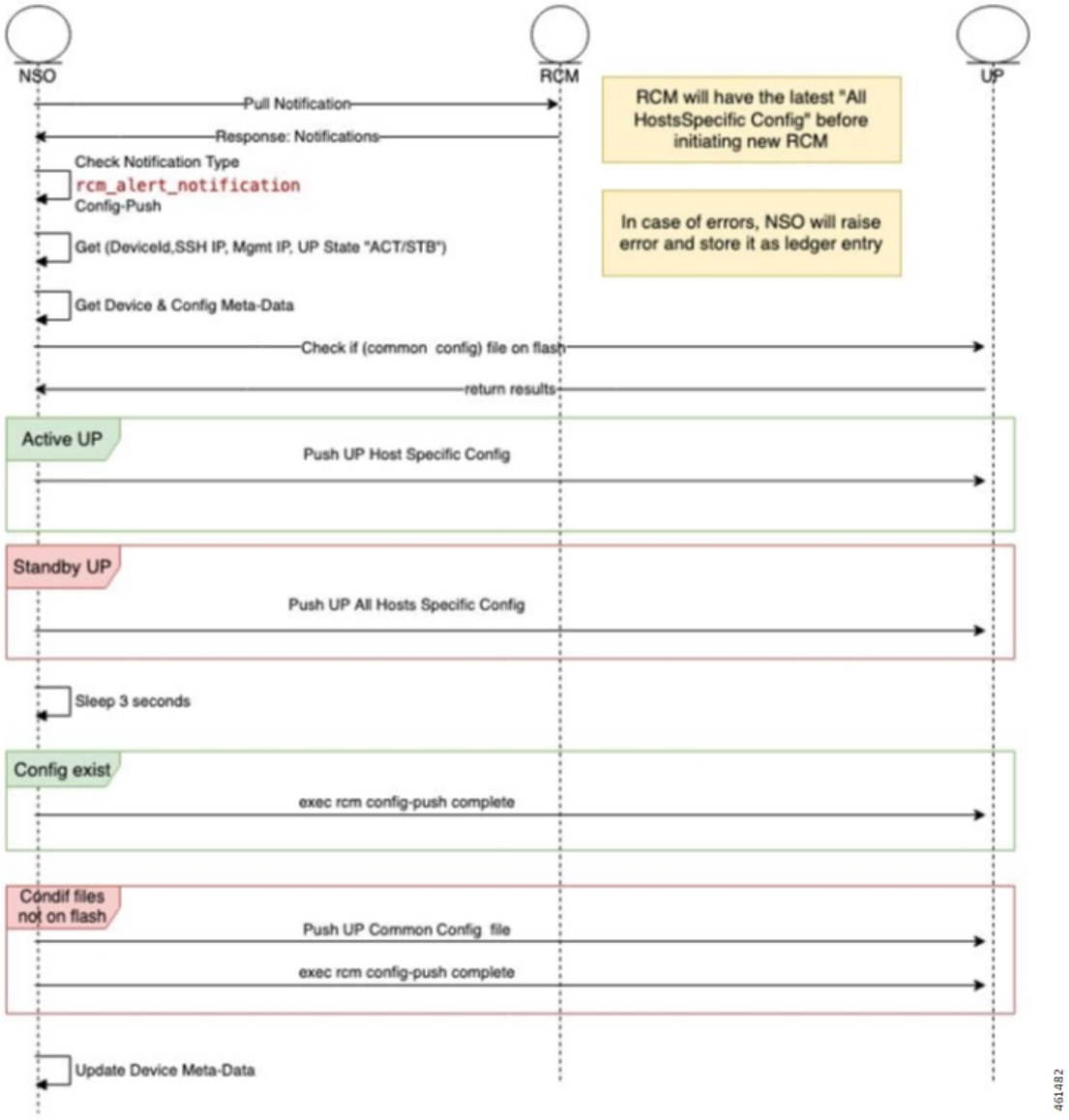
The following figure illustrates the RCM UP notification handling:



RCM UP Config-Push Notification

RCM generates config-push notification if there is a new UP coming up or existing UP is rebooting for recovery.

The following figure illustrates the RCM UP Configuration Push notification handling



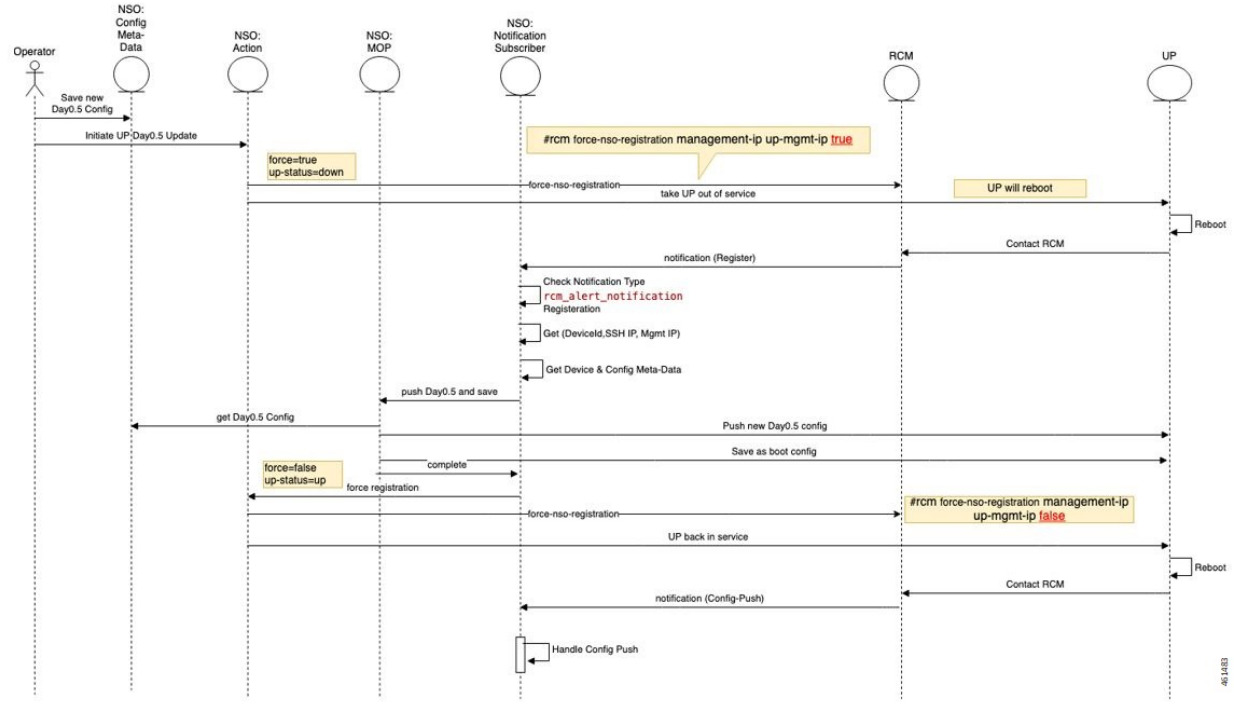
NSO performs the following steps whenever there is UP Config-Push notification:

1. Waits for alert-status “config-push”
2. Gets the device details (Device Name, SSH IP, Management IP, UP Status)
3. Gets the device meta-data from NSO
4. Checks if common file exists in UP flash.
5. If the UP State is Active, NSO pushes the UP host specific config file using the Mobility MOP.
6. If the UP State is Standby, NSO pushes the UP all hosts specific config file using the Mobility MOP.
7. Sleeps for 3 seconds.
8. If the common config file exists, NSO applies it by running live-status command on UP

9. If the common config file doesn't exist, NSO pushes the common config file to the UP using the Mobility MOP, and then applies it by running live-status command on UP.
10. Updates the device meta-data in NSO.

UP Day-0.5 Update

To change UP Day-0.5 configurations, UP must be rebooted which requires UP to be down during the change. RCM supports this use case through a command to force sending of notification whenever a specific UP gets rebooted. This notification triggers NSO to push the new Day-0.5 configurations.



1. You must update the UP Day-0.5 configuration in NSO config-metadata.
2. You must start the UP Day-0.5 change action by providing the UP device-name and management IP address.
3. NSO Action runs RCM command to force NSO registration when the UP reboots:

   `rcm force-nso-registration management-ip` *`MGMT-IP`* **`true`**
4. NSO action brings down the UP, which can be one of the following two scenarios:
   a. Standby UP: NSO action runs reload command on UP.
   b. Active UP: NSO Action runs planned switchover command on RCM by providing the UP management-ip as well as the Standby UP management-ip. The Standby UP management-ip can be retrieved from NSO Device-Metadata of the UP Group
5. After UP reboot (Active or Standby UP), RCM generates notification of type "Registration" which is received by the NSO.

6. After NSO RCM Subscriber receives the Registration notification, it starts new MOP process to push Day-0.5 configuration for the target UP.
7. NSO RCM Subscriber keeps pulling the MOP Status. After the MOP is completed, NSO runs the following command UP:

   rcm-config-push-complete
8. NSO RCM Subscriber calls the NSO Action to disable the force-notification command by running the following command:

   rcm force-nso-registration management-ip *MGMT-IP* **false**
9. NSO Action brings up the UP by running the reload command.
10. After the UP reboots, RCM generates the Config-Push notification, which is handled by NSO as usual.

Prerequisites for Configuration Push

The following are some prerequisites for config-push:

1. For UPs, the

   require rcm-configmgr

   CLI command must be preconfigured as part of Day-0 configuration. This is required for N:M, 1:1, and standalone scenarios. This enables appropriate behavior of ECS configuration.
2. For UPs, PFD push must be disabled from the CPs and in the UPs, wherever applicable. All UP configurations are pushed from the NSO directly. This is applicable for N:M, 1:1, and standalone scenarios.
3. The RCM OpsCenter Configuration mode CLIs must be configured as follows:

   k8 smf profile rcm-config-ep config-mode NSO

   k8 smf profile rcm-config-ep switchover deployment false
4. There are some default StarOS NED settings that must be overridden:
   a. Any configuration change is automatically saved to the devices boot configuration (system.cfg). This is not desirable when N:M redundancy is used as the configuration of a UP changes depends on its role. This must be disabled globally using the following CLI configuration command:

      devices global-settings ned-settings cisco-staros write-memory-setting disabled

      If using only 1:1 or standalone deployments, then this setting can be left as is.

      If using a mix of N:M and 1:1/standalone, then disable config save as above, and then fuse the "save-config-permanently" parameter in any manual configuration push for 1:1/standalone. For automated configuration pushes, the mobility function pack automatically saves the configuration whenever required.

   b. The NED treats any warnings as errors and fails the configuration push. In a lot of cases, the warnings can be ignored, and the configuration push needs to proceed. The NED can be configured to ignore select warnings using a regular expression for that warning. Here's an example.

      devices global-settings ned-settings cisco-staros behaviour config-warning-ignore .*not recommended to change the dictionary.*

Some of the other common ones are:

```
ned-settings cisco-staros behaviour config-warning-ignore .*About
to overwrite boot.*
```

```
ned-settings cisco-staros behaviour config-warning-ignore .*Standby
 card not ready.*
```

This last one is required for configuration push to SIs.

5. RCM supports the concept of an SSH IP. An SSH IP is a way to unambiguously track a given Active UP regardless of which VM is serving that function. The NSO-based solution does not use SSH IPs. However, the solution requires configuration of a dummy SSH IP. This is configured as a secondary IP address in the management interface. To avoid any errors in this configuration, the following setting is recommended as part of Day-0.5

```
configure
  redundancy-configuration-module rcm rcm
    nso-ssh-ip context local interface-name local1  mask 255.255.255.224
```

6. Read and write operations from the NSO to the VNFs can take more or less time depending on the latency. These are tunable as shown below. Do this only if you see issues with read or write errors that are truly due to latency. Typically, default settings should suffice.

```
devices global-settings read-timeout 180
```

```
devices global-settings write-timeout 180
```

Limitations and Restrictions

The NSO-based Configuration Management feature has the following limitations in this release:

- Production NSO instance can run only on popular Linux flavors (for example, RedHat, Cisco Linux, Ubuntu, CentOS, and so on).
- Only Day-1 configuration is pushed for UP on RCM notifications. No other configurations are pushed.
- For pushing Day-N configuration change at a later point, you must merge that change with the Day-1 configuration files for it to be automatically pushed on an RCM notification going forward.
- If there are changes to pre-populated configuration files, they aren't pushed automatically. It's required to push them manually for all target devices. The configuration changes only for next auto-push is considered.

  For N:M UPs, the pre-populated configuration files must be preserved on the NSO (both instances if running as an HA pair) if there is at least one VNF using them.
- Only configuration commands are supported. The show CLI commands within configuration files aren't supported.
- Any configuration, to be managed from the NSO, must be understood by the corresponding NED (StarOS NED for CPs, UPs, and RCM NED for RCMs). Currently, not all StarOS configuration commands are supported—only the most used configurations in CUPS field deployments are supported. Support for any missing commands requires a newer NED.

- Support of native StarOS CLI is not 100%. While majority of the supported CLI commands are acceptable in native StarOS CLI format, there are some cases where the NSO accepts only a variant of the corresponding StarOS CLI. Such CLIs are documented in Appendix A: Incompatible StarOS Native Command Syntax, on page 45. You can use the "dry-run" functionality of the configuration MOP to detect any errors due to incompatible/unsupported CLI prior to performing a configuration push.
- A configuration push may fail if the NSO handling the request goes down during the operation. This is applicable for both manual and automated configuration push. It is also applicable for both NSO HA and standalone NSO deployments. Operator intervention may be required depending on the exact nature of the failure.
- The Day-0.5 configuration change workflow for the N:M redundancy scenario is not fully functional in this release. For this release, the workaround is to:
  1. Remove the UP from the redundancy group (making it Standby first, if it was Active)
  2. UP boots up with Day-0 and current Day-0.5 configuration
  3. Make the Day-0.5 configuration change on the UP and save it persistently as the boot configuration
  4. Add the UP back to the redundancy group. The UP registers with the RCM and the remaining configurations are pushed by NSO automatically
- In this release, the Day-N configuration push in the N:M redundancy scenario requires a prior extra step if active-charging configuration changes are pushed using the rcm-upf MOP type. It is required to manually delete the file */flash/mobility_production.cfg* on all the UPs in that redundancy group prior to invoking the MOP.
- The standard StarOS CLI NED stores certain sensitive configuration data as cleartext locally. Access to this can be restricted by using the NACM rules on the NSO. If there are additional concerns with this, contact your Cisco Account representative for a version of the NED that encrypts this sensitive data locally. Note that this encryption is specific to the NED and NSO. StarOS encrypts sensitive data on its own—the two encryptions are separate. When NSO encrypts the sensitive data locally, it decrypts it prior to transmitting it to StarOS device (it is sent over SSH, so it is encrypted in transit, but is received by StarOS CLI as cleartext).
- For the N:M redundancy scenario, RCM supports a concept of an SSH IP. The NSO-based solution does not use the SSH IP. However, for the FCS (3.0.0 and 21.25), there is a requirement to specify an SSH IP for the solution. Any address, including private, non-routable address will suffice. This address is configured as a secondary address on the management interface of the UPs. Also refer to the Prerequisites for Configuration Push, on page 42 section for a note on the SSH IP configuration requirements for the UP.

Troubleshooting

The following options are available for troubleshooting purposes:

1. Use the dashboard output for the task-id for available details. For example:

   ```
   mobility-mop:action mop-automation-status task-id 12d5fc33-2f9e-44e3-81e3-14043d4ee39d
   ```

2. In case of failures, some alarms may be raised. These can be viewed using the

   `show alarms`

CLI command.

3. Detailed logs can be viewed by examining */var/log/ncs/ncs-java-vm.log* file. However, this is oriented towards developer debugging.

Appendix A: Incompatible StarOS Native Command Syntax

This section identifies the commands that are already supported in the NED (tagged in Bold below) but not compatible with StarOS native command syntax.

Mode	Command	Comments
context xxx/ggsn-service yyy	**ip qos-dscp qci 9 af31 gtpc af41**	It accepts "ip qos-dscp qci 9 af31" and "ip qos-dscp gtpc af41" separately. When it pushes to device, it pushes as combined.
context xxx/apn yyy	apn fnetcoriolis **default max-contexts** **default cc-roaming** **default ipv6 address alloc-method** exit	It accepts "no max-contents", "no cc-roaming", and "no ipv6 addres alloc-method" and generates "default xxx" towards the device.
context xxx/crypto map yyy (ikev2-pv4)/payload zzz match ipv4	crypto map ipsec_tunnel ikev2-ipv4 keepalive interval 4 timeout 1 num-retry 4 payload mypaiload match ipv4 **default lifetime** exit	It accepts "no lifetime" and generates "default lifetime" towards the device.
active-charging service xxx/credit-control group yyy/	credit-control group cc-m2mpt quota validity-time 600 diameter reauth-blacklisted-content content-based-rar **default diameter send-ccru on-rar always** **default diameter mscc-per-ccr-update** exit	It accepts "no xxx" and generates "default xxx" towards the device.

Mode	Command	Comments
context xxx/ikev2-ikesa transform-set yyy	ikev2-ikesa transform-set transformset_li **default encryption** **default group** **default hmac** **default lifetime** **default prf** exit	It accepts "no xxx" and generates "default xxx" towards the device.
global config mode	**end** and **#exit**	rload does not accept these commands.
global config mode	snmp trap enable	Equivalent is: no snmp trap suppress
active-charging service xxx/ruledef yyy Or any command that has "!" in it	active-charging service ecs ruledef rd-webredirect-apn-sl2sfr **bearer 3gpp imsi !range imsi-pool imsi-NOREDIRECT** exit	rload does not recognize the "!" character unless it is enclosed in double-quotes or escaped with backslash.
active-charging service xxx/ruledef yyy Any URL with %NN in it	ruledef rd1 www url = http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewSoftware**%3f**id=362713555&amp exit	The %3f must be escaped with a "\"

Mode	Command	Comments
context xxx/gtpp group yyy	gtpp group sgw **no gtpp attribute node-id** **no gtpp attribute losdv** **no gtpp trigger time-limit** **no gtpp trigger tariff-time-change** **no gtpp trigger serving-node-change-limit** **no gtpp trigger inter-plmn-sgsn-change** **no gtpp trigger qos-change** **no gtpp trigger rat-change** **no gtpp trigger ms-timezone-change** **no gtpp trigger uli-change**	StarOS does not handle these commands as a default setting, whereas NED does.
context xxx/ims-auth-service yyy	p-cscf table 1 row-precedence 1 ipv4-address 209.165.200.227 secondary **ipv4-address** 209.165.200.229	The highlighted keyword must be changed to ipv4-address only for the secondary IP address.
context xxx/ims-auth-service yyy	default signaling-flag	Use "no signaling-flag" instead.
context xxx/ims-auth-service yyy	default traffic-policy general-pdp-context no-matching-gates direction downlink	Use "no traffic-policy general-pdp-context no-matching-gates direction downlink" instead.
context xxx/ims-auth-service yyy	p-cscf table 1 row-precedence 1 ipv6-address 2001:860:ffff:feb6::a secondary **ipv6-address** 2001:860:ffff:feb4::9	The highlighted keyword must be changed to ipv6-address only for the secondary IP address.
context xxx/hexdump-module	default file rotation volume time-stamp monitor- subscriber-file-name	Equivalent is: no file rotation volume no file time-samp no file monitor-subscriber-file-name
context xxx/hexdump-module	default file rotation volume	Equivalent is: no file rotation volume
context xxx/hexdump-module	default file time-stamp	Equivalent is: no file time-stamp

Mode	Command	Comments
context xxx/hexdump-module	default subscriber-file-name	Equivalent is no subscriber-file-name
context xxx/hexdump-module	default hexdump transfer-mode	Equivalent is: no hexdump transfer-mode
context xxx/hexdump-module	default hexdump push-interval	Equivalent is: no hexdump push-interval
context xxx/edr-module active-charging-service	default cdr transfer-mode push via transfer-mode	Equivalent is: no cdr transfer-mode push
context xxx/session-event-module	default event transfer-mode push via transfer-mode	Equivalent is: no event transfer-mode push
context xxx/router bgp NNN	context gy router bgp 64650 neighbor 209.165.200.226 remote-as 15557 **no neighbor 209.165.200.226 capability graceful-restart**	StarOS default setting is to have graceful-restart capability. NED handles these commands the reverse way. **Note** This CLI is supported from StarOS CLI NED version 5.50 onwards.

Appendix B: Example Configurations for N:M Deployment with RCM

Host-specific Configuration-UP

The following are examples of host-specific configurations for two Active UPs. These are pushed to the respective UPs.

First Active UP

```
config
context EPC2
interface loop1_up1 loopback
ip address 209.165.200.225 255.255.255.224

interface loop2_up1 loopback
ip address 209.165.200.226 255.255.255.224

interface loop3_up1 loopback
ip address 209.165.200.227 255.255.255.224
```

```
interface loop4_up1 loopback
ip address 209.165.200.228 255.255.255.224

interface loop5_up1 loopback
ip address 209.165.200.229 255.255.255.224

exit
exit

context EPC2
sx-service sx_up1
instance-type userplane
bind ipv4-address 209.165.200.229
exit

exit
exit

context EPC2
gtpu-service pgw-gtpu_up1
bind ipv4-address 209.165.200.226
exit
gtpu-service saegw-sxu_up1
bind ipv4-address 209.165.200.227
exit
gtpu-service sgw-engress-gtpu_up1
bind ipv4-address 209.165.200.228
exit
gtpu-service sgw-ingress-gtpu_up1
bind ipv4-address 209.165.200.225

exit
exit
config
context EPC2
user-plane-service up_up1
associate gtpu-service pgw-gtpu_up1 pgw-ingress
associate gtpu-service sgw-ingress-gtpu_up1 sgw-ingress
associate gtpu-service sgw-engress-gtpu_up1 sgw-egress
associate gtpu-service saegw-sxu_up1 cp-tunnel
associate sx-service sx_up1
associate fast-path service
associate control-plane-group g1
exit

exit
exit
```

Second Active UP

```
config
context EPC2
interface loop1_up2 loopback
ip address 209.165.200.230 255.255.255.224

interface loop2_up2 loopback
ip address 209.165.200.231 255.255.255.224

interface loop3_up2 loopback
ip address 209.165.200.232 255.255.255.224

interface loop4_up2 loopback
ip address 209.165.200.233 255.255.255.224
```

```
interface loop5_up2 loopback
ip address 209.165.200.234 255.255.255.224

exit
exit

context EPC2
sx-service sx_up2
instance-type userplane
bind ipv4-address 209.165.200.234
exit

exit
exit

context EPC2
gtpu-service pgw-gtpu_up2
bind ipv4-address 209.165.200.231
exit
gtpu-service saegw-sxu_up2
bind ipv4-address 209.165.200.232
exit
gtpu-service sgw-engress-gtpu_up2
bind ipv4-address 209.165.200.233
exit
gtpu-service sgw-ingress-gtpu_up2
bind ipv4-address 209.165.200.230

exit
exit

context EPC2
user-plane-service up_up2
associate gtpu-service pgw-gtpu_up2 pgw-ingress
associate gtpu-service sgw-ingress-gtpu_up2 sgw-ingress
associate gtpu-service sgw-engress-gtpu_up2 sgw-egress
associate gtpu-service saegw-sxu_up2 cp-tunnel
associate sx-service sx_up2
associate fast-path service
associate control-plane-group g1
exit

exit
exit
```

Host-specific Configuration-RCM

The following are examples of host-specific configurations for RCM. This is pushed to the RCM.

First Active RCM

```
config
control-plane-group g1
 redundancy-group 1
  host Active1
   peer-node-id ipv4-address 209.165.200.240
  exit
 exit
exit
context EPC2
 interface-loopback loop1_up1
  redundancy-group 1
```

```
  host Active1
   ipv4-address 209.165.200.224/27
  exit
 exit
exit
interface-loopback loop2_up1
 redundancy-group 1
  host Active1
   ipv4-address 209.165.201.0/27
  exit
 exit
exit
interface-loopback loop3_up1
 redundancy-group 1
  host Active1
   ipv4-address 209.165.202.128/27
  exit
 exit
exit
interface-loopback loop4_up1
 redundancy-group 1
  host Active1
   ipv4-address 192.0.2.0/24
  exit
 exit
exit
interface-loopback loop5_up1
 redundancy-group 1
  host Active1
   ipv4-address 198.51.100.0/24
  exit
 exit
exit
user-plane-service up_up1
 redundancy-group 1
  host Active1
   associate control-plane-group g1
   associate fast-path service
   associate sx-service sx_up1
   associate gtpu-service pgw-gtpu_up1 pgw-ingress
   associate gtpu-service saegw-sxu_up1 cp-tunnel
   associate gtpu-service sgw-engress-gtpu_up1 sgw-egress
   associate gtpu-service sgw-ingress-gtpu_up1 sgw-ingress
  exit
 exit
exit
gtpu-service pgw-gtpu_up1
 redundancy-group 1
  host Active1
   bind ipv4-address 209.165.201.0
  exit
 exit
exit
gtpu-service saegw-sxu_up1
 redundancy-group 1
  host Active1
   bind ipv4-address 209.165.202.128
  exit
 exit
exit
gtpu-service sgw-engress-gtpu_up1
 redundancy-group 1
  host Active1
   bind ipv4-address 192.0.2.0
```

```
  exit
 exit
exit
gtpu-service sgw-ingress-gtpu_up1
 redundancy-group 1
  host Active1
   bind ipv4-address 198.51.100.123
  exit
 exit
exit
sx-service sx_up1
 redundancy-group 1
  host Active1
   bind ipv4-address 198.51.100.0
   instance-type userplane
  exit
 exit
exit
exit
```

Second Active RCM

```
config
control-plane-group g1
 redundancy-group 1
  host Active2
   peer-node-id ipv4-address 209.165.200.240
  exit
 exit
exit
context EPC2
 interface-loopback loop1_up2
  redundancy-group 1
   host Active2
    ipv4-address 209.165.200.224/27
   exit
  exit
 exit
 interface-loopback loop2_up2
  redundancy-group 1
   host Active2
    ipv4-address 209.165.201.0/27
   exit
  exit
 exit
 interface-loopback loop3_up2
  redundancy-group 1
   host Active2
    ipv4-address 209.165.202.128/27
   exit
  exit
 exit
 interface-loopback loop4_up2
  redundancy-group 1
   host Active2
    ipv4-address 192.0.2.0/24
   exit
  exit
 exit
 interface-loopback loop5_up2
  redundancy-group 1
   host Active2
    ipv4-address 198.51.100.0/24
   exit
```

```
 exit
exit
user-plane-service up_up2
 redundancy-group 1
  host Active2
   associate control-plane-group g1
   associate fast-path service
   associate sx-service sx_up2
   associate gtpu-service pgw-gtpu_up2 pgw-ingress
   associate gtpu-service saegw-sxu_up2 cp-tunnel
   associate gtpu-service sgw-engress-gtpu_up2 sgw-egress
   associate gtpu-service sgw-ingress-gtpu_up2 sgw-ingress
  exit
 exit
exit
gtpu-service pgw-gtpu_up2
 redundancy-group 1
  host Active2
   bind ipv4-address 209.165.201.0
  exit
 exit
exit
gtpu-service saegw-sxu_up2
 redundancy-group 1
  host Active2
   bind ipv4-address 209.165.202.128
  exit
 exit
exit
gtpu-service sgw-engress-gtpu_up2
 redundancy-group 1
  host Active2
   bind ipv4-address 192.0.2.0
  exit
 exit
exit
gtpu-service sgw-ingress-gtpu_up2
 redundancy-group 1
  host Active2
   bind ipv4-address 198.51.100.123
  exit
 exit
exit
sx-service sx_up2
 redundancy-group 1
  host Active2
   bind ipv4-address 198.51.100.0
   instance-type userplane
  exit
 exit
exit
exit
```

Common Configuration

The following is an example of a common configuration. This is pushed to all Active UPs and all Standby UPs.

```
config
 active-charging service ACS
 idle-timeout udp 60
 statistics-collection ruledef all
 host-pool IPv6_VoLTE_Phone_Host_7
```

```
ip 209.165.200.224/27
ip 64:ff9b::d3f6:6b00/120
ip 2001:e60:6000::/46
ip 2001:e60:6004::/46
ip range 209.165.200.225 to 209.165.200.234
ip range 64:ff9b::e00:4f12 to 64:ff9b::e00:4f14
ip range 64:ff9b::3d6e:ff52 to 64:ff9b::3d6e:ff59
ip range 64:ff9b::d3f6:682c to 64:ff9b::d3f6:683e
exit
port-map M_learning_Port
port range 1 to 9500
port range 10001 to 30000
exit
port-map OTM_Advertisement_port
port 90
port 9090
exit
ruledef ICMP
ip protocol = 1
exit
ruledef ICMPv6
ip protocol = icmpv6
exit
ruledef IPv6_VoLTE_Phone_1
udp either-port range port-map M_learning_Port
ip server-ip-address range host-pool IPv6_VoLTE_Phone_Host_7
exit
ruledef RD-allTraffic
ip any-match = TRUE
exit
ruledef RD_Charge
ip server-ip-address = 209.165.201.0/27
exit
ruledef catchall
ip any-match = TRUE
exit
ruledef googles
icmpv6 any-match = TRUE
exit
ruledef qci1
tcp any-match = TRUE
exit
ruledef route-ims-ipv6-nexthop
ip uplink = TRUE
ip version = ipv6
exit
ruledef optIn
ip any-match = TRUE
exit
group-of-ruledefs GoR_FOTA
add-ruledef priority 1 ruledef FOTA_SAMSUNG
add-ruledef priority 2 ruledef FOTA_LG
add-ruledef priority 3 ruledef FOTA_LG_2
add-ruledef priority 5 ruledef FOTA_PANTECH_2
add-ruledef priority 8 ruledef IOS_OTA_Update
add-ruledef priority 9 ruledef GOTA_google
add-ruledef priority 10 ruledef FOTA_Hybrid_Egg
add-ruledef priority 11 ruledef FOTA_SAMSUNG_2
add-ruledef priority 12 ruledef FOTA_SAMSUNG_3
add-ruledef priority 13 ruledef FOTA_SAMSUNG_4
add-ruledef priority 15 ruledef FOTA_SAMSUNG_5
add-ruledef priority 16 ruledef FOTA_LG_4
add-ruledef priority 17 ruledef FOTA_LG_5
add-ruledef priority 18 ruledef FOTA_HUAWEI_Egg
```

```
 add-ruledef priority 20 ruledef KTF_DMS_FOTA
 add-ruledef priority 21 ruledef FOTA_Nlabs
 add-ruledef priority 22 ruledef FOTA_LTE_Beam
 add-ruledef priority 23 ruledef FOTA_S_Mobile
 add-ruledef priority 24 ruledef FOTA_Giga_Genie
 add-ruledef priority 100 ruledef SAMSUNG_SKT_issue
 add-ruledef priority 104 ruledef new_FOTA_Pantech
 add-ruledef priority 106 ruledef new_FOTA_KTtech
 add-ruledef priority 107 ruledef new_IOS_OTA_Log
 add-ruledef priority 114 ruledef new_FOTA_LG_3
 add-ruledef priority 200 ruledef IoT_FOTA_mexus
 add-ruledef priority 201 ruledef IoT_FOTA_acnt
 add-ruledef priority 202 ruledef IoT_FOTA_amtel
 exit
 packet-filter qci1
 ip protocol = 1
 ip remote-port = 1001
 priority 1
 exit
 packet-filter subscriber-pools
 exit
 charging-action CA-nothing
 content-id 5
 exit
 charging-action CA_Chargeable_2
 content-id 1
 billing-action egcdr
 exit
charging-action CA_Charge
exit
 charging-action DSI
 billing-action egcdr
 flow action discard
 tft packet-filter permit_all
 exit
 charging-action ca11
 service-identifier 22
 billing-action egcdr
 cca charging credit
 flow action discard
 flow limit-for-bandwidth id 4
 exit
 charging-action catchall
 content-id 10
 billing-action egcdr
 cca charging credit rating-group 10 preemptively-request
 exit
 charging-action qci1
 billing-action egcdr
 cca charging credit rating-group 1 preemptively-request
 qos-class-identifier 1
 tft packet-filter qci1
 exit
 bandwidth-policy bw-policy
 flow limit-for-bandwidth id 2 group-id 2
 flow limit-for-bandwidth id 4 group-id 4
 flow limit-for-bandwidth id 10 group-id 12
 flow limit-for-bandwidth id 562 group-id 562
 group-id 2 direction downlink peak-data-rate 225280 peak-burst-size 2253 violate-action
discard
 group-id 4 direction uplink peak-data-rate 450560 peak-burst-size 4506 violate-action
discard
 group-id 10 direction uplink peak-data-rate 1153434 peak-burst-size 11534 violate-action
discard
```

```
 group-id 11 direction uplink peak-data-rate 10000 peak-burst-size 10000 violate-action
discard
 exit
 rulebase 5G-DF
 tcp packets-out-of-order timeout 30000
 no retransmissions-counted
 edr sn-charge-volume count-dropped-units
 bandwidth default-policy bw-policy
 exit
 rulebase RB-allTraffic
 action priority 100 ruledef RD-allTraffic charging-action CA_Charge
 egcdr threshold interval 3600
 egcdr threshold volume total 4000000
 exit
 rulebase RB_Charge
 action priority 10 ruledef RD_Charge charging-action CA_Charge
 exit
 rulebase cisco
 billing-records egcdr
 action priority 12 ruledef catchall charging-action catchall monitoring-key 123
 egcdr threshold interval 120
 egcdr threshold volume total 1000000
 exit
 rulebase cisco_dynamic
 action priority 11 dynamic-only ruledef qci1 charging-action qci1
 action priority 10000 ruledef catchall charging-action catchall
 egcdr threshold interval 120
 egcdr threshold volume total 100000
 exit
 rulebase P2P
 transactional-rule-matching
 dynamic-rule order first-if-tied
 tethering-detection application ip-ttl value 62
 flow end-condition timeout normal-end-signaling session-end charging-edr flow-edr
 billing-records egcdr
 edr transaction-complete http charging-edr http-edr
 flow control-handshaking charge-to-application all-packets
 egcdr threshold interval 3600
 egcdr threshold volume total 4000000000
 no cca quota retry-time
 cca diameter requested-service-unit sub-avp volume cc-total-octets 5000
 p2p dynamic-flow-detection
 no tft-notify-ue-def-bearer
 exit
 rulebase default
 exit
 rulebase wap_adult
      transactional-rule-matching
      tcp mss 1320 limit-if-present
      flow end-condition handoff timeout normal-end-signaling session-end charging-edr
flow-edr
      billing-records egcdr radius
      action priority 28 ruledef catchall charging-action CA_Chargeable_2
      action priority 29 ruledef catchall charging-action CA_Chargeable_2
      edr transaction-complete http charging-edr http-edr
     flow control-handshaking charge-to-application  mid-session-packets tear-down-packets

      egcdr threshold volume total 3000000
#exit
 service-scheme ss1
 exit
 credit-control group DCCA_grp1
 diameter origin endpoint Gy
 diameter peer-select peer minid-Gy
```

```
 pending-traffic-treatment noquota buffer
 pending-traffic-treatment quota-exhausted buffer
 pending-traffic-treatment validity-expired pass
 exit
 credit-control group default
 pending-traffic-treatment noquota pass
 pending-traffic-treatment quota-exhausted buffer
 exit
 policy-control charging-rule-base-name active-charging-rulebase
 policy-control burst-size auto-readjust duration 3
 exit
 context ecs
 apn cisco.com
 ip context-name ecs
 exit
 apn starent.com
 ip context-name ecs
 exit
end
```

Standby Configuration (Active1 + Active2)

```
config
context EPC2
interface loop1_up1 loopback
ip address 198.51.100.123 255.255.255.224

interface loop2_up1 loopback
ip address 209.165.201.0 255.255.255.224

interface loop3_up1 loopback
ip address 209.165.202.128 255.255.255.224

interface loop4_up1 loopback
ip address 192.0.2.0 255.255.255.224

interface loop5_up1 loopback
ip address 198.51.100.0 255.255.255.224

exit
exit

context EPC2
sx-service sx_up1
instance-type userplane
bind ipv4-address 198.51.100.0
exit

exit
exit

context EPC2
gtpu-service pgw-gtpu_up1
bind ipv4-address 209.165.201.0
exit
gtpu-service saegw-sxu_up1
bind ipv4-address 209.165.202.128
exit
gtpu-service sgw-engress-gtpu_up1
bind ipv4-address 192.0.2.0
exit
gtpu-service sgw-ingress-gtpu_up1
bind ipv4-address 198.51.100.0
```

```
exit
exit

context EPC2
user-plane-service up_up1
associate gtpu-service pgw-gtpu_up1 pgw-ingress
associate gtpu-service sgw-ingress-gtpu_up1 sgw-ingress
associate gtpu-service sgw-engress-gtpu_up1 sgw-egress
associate gtpu-service saegw-sxu_up1 cp-tunnel
associate sx-service sx_up1
associate fast-path service
associate control-plane-group g1
exit

exit
exit

config
context EPC2
interface loop1_up2 loopback
ip address 209.165.200.230 255.255.255.224

interface loop2_up2 loopback
ip address 209.165.200.231 255.255.255.224

interface loop3_up2 loopback
ip address 209.165.200.232 255.255.255.224

interface loop4_up2 loopback
ip address 209.165.200.233 255.255.255.224

interface loop5_up2 loopback
ip address 209.165.200.234 255.255.255.224

exit
exit

context EPC2
sx-service sx_up2
instance-type userplane
bind ipv4-address 209.165.200.234
exit

exit
exit

context EPC2
gtpu-service pgw-gtpu_up2
bind ipv4-address 209.165.200.231
exit
gtpu-service saegw-sxu_up2
bind ipv4-address 209.165.200.232
exit
gtpu-service sgw-engress-gtpu_up2
bind ipv4-address 209.165.200.233
exit
gtpu-service sgw-ingress-gtpu_up2
bind ipv4-address 209.165.200.230

exit
exit

context EPC2
```

```
user-plane-service up_up2
associate gtpu-service pgw-gtpu_up2 pgw-ingress
associate gtpu-service sgw-ingress-gtpu_up2 sgw-ingress
associate gtpu-service sgw-engress-gtpu_up2 sgw-egress
associate gtpu-service saegw-sxu_up2 cp-tunnel
associate sx-service sx_up2
associate fast-path service
associate control-plane-group g1
exit

exit
exit
```