Delete BGP Neighbor

In this use case, you delete a BGP neighbor using YANG models.

1. Using standard YANG tools, get the configuration using the NETCONF <get-config> operation in YANG format.

```
<get-config>
    <source>
      <running/>
    </source>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <bgp xmlns="http://openconfig.net/yang/bgp">
 </get-config>

router bgp 1000
 bgp confederation peers
  65002
 !
 bgp confederation identifier 102
 bgp router-id 1.1.1.1
 bgp graceful-restart restart-time 30
 bgp graceful-restart stalepath-time 30
 bgp graceful-restart
 address-family ipv4 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv4 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
!router bgp 1000
 bgp confederation peers
  65002
 !
```

```
 bgp confederation identifier 102
 bgp router-id 1.1.1.1
 bgp graceful-restart restart-time 30
 bgp graceful-restart stalepath-time 30
 bgp graceful-restart
 address-family ipv4 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv4 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 unicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
 address-family ipv6 multicast
  distance bgp 200 20 200
  maximum-paths ebgp 30
  maximum-paths ibgp 30
 !
!
```

2. Change the configuration <edit-config> operation.

```
<edit-config>
    <target>
      <candidate/>
    </target>
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <bgp xmlns="http://openconfig.net/yang/bgp">
        <global>
          <config>
            <as xc:operation="delete">1000</as>
          </config>
        </global>
      </bgp>
    </config>
  </edit-config>
```

3. Send the <edit-config> request through NETCONF SSH to the router.
4. Verify that the configuration changes is successful and the BGP neighbor is deleted.

Note BGP configuration can be fetched using gRPC GetConfig operation:

```
{
  "bgp:bgp": [
 null
  ]
}
```

Delete BGP configuration use gRPC DeleteConfig:

```
{
    "bgp:bgp": {
        "global": {
            "config": {
                "as": [
                    null
                ]
            }
        }
    }
}
```

- Request for AAA Access Details, on page 3
- Using NETCONF with Flexible CLI Configuration Groups, on page 4

Request for AAA Access Details

In this use case, you use a Calvados model to view AAA access details.

Note If any user on XR is deleted, the local database checks whether there is a first user on Calvados VM.

- If there is a first user, no syncing occurs.
- If there is no first user, then the first user on XR (based on the order of creation) is synced to Calvados VM.

Prerequisites

- Ensure that the user is added to the Calvados environment. This is because even if the user is added to the XR environment and has `root-lr` permissions, access to Calvados models is denied.
- Establish a NETCONF or gRPC connection between the router and the client application.

Note The gRPC YANG path or JSON data is based on YANG module name and not YANG namespace.

1. Using standard YANG tools, send a request to the router from the client using the NETCONF <get> operation.

```
[ Request ]
<get>
  <filter type="subtree">
    <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
      <privileged-access xmlns="http://www.cisco.com/calvados/aaa_show"/>
    </aaa>
  </filter>
</get>
```

2. Verify the response sent by the router to the client.

```
[ Response ]
<?xml version="1.0" encoding="UTF-8"?><data
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
   <privileged-access xmlns="http://www.cisco.com/calvados/aaa_show">
    <shell-access>None</shell-access>
    <first-user>root</first-user>
    <first-user-change>No</first-user-change>
    <current-disaster-recovery-user>root</current-disaster-recovery-user>
   </privileged-access>
  </aaa>
 </data>
```

Note To accomplish this task using gRPC GetOper request:

```
{
    "tailf-aaa:aaa": {
        "aaa_show:privileged-access": [
            null
        ]
    }
}
```

gRPC GetOper response:

```
{
 "tailf-aaa:aaa": {
  "aaa_show:privileged-access": {
   "shell-access": "None",
   "first-user": "root",
   "first-user-change": "No",
   "current-disaster-recovery-user": "root"
  }
 }
}
```

Using NETCONF with Flexible CLI Configuration Groups

If you want to use NETCONF protocol with flexible CLI conifguration groups, you need to use the inherited configuration. To transition to NETCONF and YANG based configuration from a CLI configuration which includes flexible CLI configuration groups, use the following steps. Using these steps, you can retrieve all the configuration on the device which can be used in further NETCONF operations.

1. Send a NETCONF **get-config** request with source as `<running-inheritance/>` .

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
          <running-inheritance xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-group-cfg"/>

        </source>
     </get-config>
</rpc>
##
```

This operation returns all the configuration present on the device (inherited or expanded) in the following format:

```
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
      ….
    </data>
</rpc-reply
```

2. To apply the configuration to another device, send a NETCONF **edit-config** request in the following format:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
       <candidate/>
      </target>
      <config>
      </config>
  </edit-config>
</rpc>
```