NX-SDK-Python-toepassing ontwikkelen, debug en implementeren in Nexus 3000/9000 switches

Inhoud

Inleiding

Dit document biedt een planning voor de ontwikkeling van Python-toepassingen met Cisco NX-Software Development Kit (SDK) met het gebruik van Cisco NX-OS op de Nexus 3000- en Nexus 9000-platforms.

Voorwaarden

Vereisten

Er zijn geen specifieke vereisten van toepassing op dit document.

Gebruikte componenten

De informatie in dit document is gebaseerd op de volgende software- en hardware-versies:

- Dit document gebruikt NX-SDK v1.0.0 en NX-SDK v1.5.0
- NX-SDK v1.0.0 kan op het Nexus 9000-platform worden gebruikt vanaf NX-OS release 7.0(3)I6(1) en het Nexus 3000-platform vanaf NX-OS release 7.0(3)I7(1)
- NX-SDK v1.5.0 kan zowel op het Nexus 9000-platform als op het Nexus 3000-platform worden gebruikt vanaf NX-OS release 7.0(3)I7(3)

De informatie in dit document is gebaseerd op de apparaten in een specifieke laboratoriumomgeving. Alle apparaten die in dit document worden beschreven, hadden een opgeschoonde (standaard)configuratie. Als uw netwerk live is, moet u de potentiële impact van elke opdracht begrijpen.

Achtergrondinformatie

Cisco NX-SDK maakt de ontwikkeling mogelijk van aangepaste toepassingen die anders in Cisco NX-OS op Nexus 9000- en Nexus 3000-platforms kunnen draaien. NX-SDK biedt klanten de mogelijkheid om hun eigen CLI-opdrachten en -uitgangen te maken, aangepaste syslogs te genereren in reactie op specifieke gebeurtenissen, op een stream toegesneden telemetrie en nog veel meer.

NX-SDK heeft een C++ API, die in andere talen wordt vertaald met behulp van een Vereenvoudigde Wrapper en Interface Generator (SWIG). Dit stelt de klant in staat om NX-SDK te gebruiken in elke taal naar keuze. Dit document laat de implementatie van gemeenschappelijke NX-SDK-functies in Python zien, en biedt klanten een mogelijkheid om hun eigen NX-SDK-Python-toepassingen te ontwikkelen.

Een Python-toepassing met NX-SDK ontwikkelen

NX-SDK inschakelen

Om een NX-SDK-toepassing te kunnen uitvoeren, moet de NX-SDK-functie eerst op het apparaat zijn ingeschakeld:

```
switch(config)# feature nxsdk
```

Een Python-bestand maken

Een Python-bestand kan met behulp van de NX-OS Bash-shell worden gemaakt en bewerkt. Om de schaal van Bash te kunnen gebruiken, moet deze eerst op de voorziening zijn ingeschakeld:

```
switch(config)# feature bash-shell
```

Voer de schaal van het Bash in en gebruik de vi teksteditor om het Python-bestand te maken en te bewerken:

```
switch(config)# run bash
bash-4.2$ vi /isan/bin/nxsdk-app.py
```

Opmerking: Best practice is het maken van Python-bestanden in de **/isan/bin/** folder. Python-bestanden hebben uitvoervergunningen nodig om te kunnen uitvoeren. Plaats Python-bestanden niet in de directory**/flitser** of in een van de subdirectories.

Opmerking: Het is niet vereist om Python-bestanden te maken en te bewerken via NX-OS. De developer kan de toepassing maken vanuit hun lokale omgeving en de voltooide bestanden naar het apparaat overdragen via een protocol voor bestandsoverdracht naar keuze. Mogelijk is het voor de ontwikkelaar echter efficiënter om het script te debug en in te stellen met behulp van de NX-OS-hulpprogramma's.

Componenten voor NX-SDK implementeren

Het wordt aanbevolen voor ontwikkelaars om hun NX-SDK Python-toepassing te maken uit de aangepaste CliPyApp-sjabloon op de Cisco DevNet NX-SDK GitHub. Deze secties van dit document verwijzen naar de benodigde onderdelen met het gebruik van hun namen in deze sjabloon.

Er zijn vier belangrijke componenten nodig voor een NX-SDK-Python-toepassing:

1. NX-SDK moet in de applicatie worden geïmporteerd via de **import-nx_sdk_py** statement.
2. Een functie (meestal **sdkThread**) die de NX-SDK-toepassing start en verschillende opties met betrekking tot de toepassing wijzigt.
3. De creatie van aangepaste CLI opdrachten zowel als de definitie van de aangepaste CLI-opdrachtsyntaxis binnen de **sdkThread-**functie.
4. Een class met de naam **pyCmdHandler** met een methode met de naam **postCliCb**, die aangepaste CLI-opdrachten verwerkt die door de NX-SDK-toepassing zijn toegevoegd.

SDK-functie

De sdkThread-functie initialiseert, voegt functionaliteit toe aan, en begint de NX-SDK-toepassing. De functie vereist niet dat er parameters in worden opgenomen. Alle Python NX-SDK toepassingen vereisen dat drie methoden van de nx_sdk_py bibliotheek worden genoemd:

1. **nx_sdk_py.NxSdk.getSdkInst (len(sys.argv), sys.argv)** geeft of een SDK-voorbeeldobject terug, of geeft geen van deze middelen terug. Als een SDK-voorbeeldobject wordt teruggegeven, is de NX-SDK-toepassing geregistreerd met de NX-OS-infrastructuur. Als Geen wordt teruggegeven, dan gebeuren er fouten op het moment dat dit registratieproces plaatsvindt en verschijnt een Error log entry in het syslog van het apparaat. Deze methode is hier gedocumenteerd.

   Opmerking: Vanaf NX-SDK v1.5.0 kan een derde Booleaanse parameter worden doorgegeven in de **NxSDK.getSdkInst-**methode, die Advanced Exceptions toestaat wanneer True en Advanced Exceptions blokkeert wanneer er sprake is van fraude. Deze methode is hier gedocumenteerd.

1. **sdk.startEventLoop()** methode, waarbij **sdk** het SDK-voorbeeldobject is dat is teruggegeven door de **nx_sdk_py.NxSdk.getSdkInst()** methode. Deze methode start de NX-SDK-toepassing en stelt de toepassing in staat om te communiceren met de NX-OS-infrastructuur. Deze methode is hier gedocumenteerd.
2. **nx_sdk_py.NxSdk.__zwemg_kill_**methode, waarbij **sdk** het SDK-invoegobject is dat **wordt** geretourneerd door de eerder genoemde **nx_sdk_py.NxSdk.getInst()** methode. Deze methode aan het eind van de **sdkThread-**functie geplaatst maakt het mogelijk om de NX-SDK-toepassing op vreedzame wijze te verlaten.

Enkele veelgebruikte methoden zijn:

- **sdk.getTracer()**, waar **sdk** het SDK-voorbeeldobject is dat is teruggegeven door de **nx_sdk_py.NxSdk.getSdkInst()** methode. Deze methode retourneert een NxTrace object dat kan worden gebruikt om aangepaste syslogs te genereren, evenals log gebeurtenissen en fouten naar de eventgeschiedenis van de toepassing. Aangepaste symbolen worden weergegeven in de syslog van het apparaat (zichtbaar via de opdracht **logbestand voor** show **registreren**) terwijl de gebeurtenissen die zijn vastgelegd op de geschiedenis van de gebeurtenis in de toepassing zichtbaar zijn via de **opdrachten voor** de **gebeurtenis-geschiedenis van <application-name> nxsdk** of **de** opdrachten voor **fouten uit de afgelopen gebeurtenissen tonen**. Deze methode is hier gedocumenteerd. Het NxTrace-object dat door deze methode is geretourneerd en de bijbehorende methoden zijn hier gedocumenteerd. Bijvoorbeeld, kunt u de gebeurtenis historie van een toepassing zien die **Transceiver_DOM.py** door de **show gebeurtenissen van de gebeurtenis-geschiedenis van Transceiver_DOM.py nxsdk** en **de** opdrachten van de **gebeurtenis-geschiedenis van Transceiver_DOM.py** nxsdk **tonen**.
- **sdk.getCliParser()**, waar **sdk** het SDK instantie object is dat terugkomt door de **nx_sdk_py.NxSdk.getSdkInst()** methode. Deze methode retourneert een NxCliParser object, dat kan worden gebruikt om de CLI-opdrachten uit te voeren die al bestaan via Python, en om aangepaste CLI-opdrachten te maken. Deze methode is hier gedocumenteerd. Het object NxCliParser dat volgens deze methode is geretourneerd, en de bijbehorende methoden zijn hier gedocumenteerd.
- **cliP.execShowCmd("cmd", return_type)**, waarbij **cliP** het object NxCliParser is dat teruggegeven is door de **methode sdk.getCliParser()**, **cmd** is de toonopdracht die u ingesloten aanhaltes wilt uitvoeren, en **return_type is de gegevensindeling die moet worden uitgevoerd.** Het gegevensformaat kan of **R_TEXT** zijn, **R_JSON**, of **R_XML**. Deze methode is hier gedocumenteerd.

  Opmerking: De **R_JSON** en **R_XML** gegevensformaten werken alleen als de opdracht uitvoer in die formaten ondersteunt. In NX-OS kunt u controleren of een opdracht uitvoer in een bepaald gegevensformaat ondersteunt door de uitvoer naar het gevraagde gegevensformaat te leiden. Als de leiding betekenisvolle uitvoer teruggeeft, wordt dat gegevensformaat ondersteund. Als u bijvoorbeeld **hoofdapplicatie** uitvoert**, wordt hoofdadreslijst dynamisch | json** in NX-OS retourneert JSON output, dan wordt de R_JSON data format ook ondersteund in NX-SDK.

- **cliP.execConfigCmd(cmd_bestandsnaam)**, waarbij **cliP** het NxCliParser-object is dat is teruggegeven door de **methode sdk.getCliParser()**, en **cmd_filename** het absolute filepad is naar een bestand dat opdrachten bevat die van regels zijn gescheiden. Deze methode retourneert een string die het succes aangeeft van commando uitvoering - als "SUCCESS" in de string is, dan worden alle opdrachten succesvol uitgevoerd. Anders bevat de string de

uitzondering die beschrijft waarom de opdrachten niet werden uitgevoerd. Deze methode is hier gedocumenteerd.

Sommige optionele methoden die behulpzaam kunnen zijn zijn:

- **sdk.setAppDesc('Description string')**, waarbij **sdk** het SDK instantie object is dat terugkomt door de **nx_sdk_py.NxSdk.getSdkInst()** methode. Deze methode stelt de beschrijving van de NX-SDK-toepassing in. De beschrijving wordt weergegeven in het NX-OS contextgevoelige Help-menu dat via een vraagteken op de CLI wordt benaderd. Deze methode is hier gedocumenteerd. Bijvoorbeeld, een toepassing genaamd **Transceiver_DOM.py**, verschijnt een toepassingsbeschrijving van alle interfaces met DOM-Geschikt transceivers opgenomen in de NX-OS context-gevoelige hulp als volgt:

```
 N9K-C93180LC-EX# show Tra?
track Tracking information
Transceiver_DOM.py Returns all interfaces with DOM-capable transceivers inserted
```

- **sdk.getAppName()**, waar **sdk** het SDK instantie object is dat terugkomt door de **nx_sdk_py.NxSdk.getSdkAgainst()** methode. Deze methode geeft de naam van de Python-toepassing terug. Deze methode is hier gedocumenteerd.

Aangepaste CLI-opdrachten maken

In een Python-toepassing met het gebruik van NX-SDK, worden de aangepaste CLI-opdrachten gecreëerd en gedefinieerd binnen de sdkThread-functie. Er zijn twee soorten opdrachten: Opdrachten **tonen** en opdrachten **Config**

1. Opdrachten **tonen** informatie over het apparaat, de configuratie of de omgeving.
2. Opdrachten **Config** wijzigen de configuratie van het apparaat, waardoor de reactie van het apparaat op het omliggende netwerk wordt gewijzigd.

Deze twee methodes staan de creatie van show opdrachten toe respectievelijk configuratieopdrachten:

- **cliP.newShowCmd("cmd_name", "syntax")**, waarbij **cliP** het object NxCliParser is dat door de **methode sdk.getCliParser()** is geretourneerd, **cmd_name** is een unieke naam voor de opdrachtinterne syntaxis van NX-SDK beschrijft welke trefwoorden en parameters in de opdracht kunnen worden gebruikt. Deze methode retourneert een NxCliCmd object, dat hier gedocumenteerd is. Deze methode is hier gedocumenteerd.

  Opmerking: Deze opdracht is een subklasse van **cliP.newCliCmd("cmd_type", "cmd_name", "syntax")** waarin **cmd_type** ofwel **CONF_CMD** of **SHOW_CMD** (afhankelijk van het type opdracht dat wordt uitgevoerd) is **cmd_name een unieke naam voor de opdracht intern op de aangepaste NX-SDK-toepassing en de syntax beschrijft welke trefwoorden en parameters in de opdracht kunnen worden gebruikt.** Daarom kan de API-documentatie voor deze opdracht behulpzamer zijn voor referentie.

- **cliP.newConfigCmd("cmd_name", "syntax")**, waarbij **cliP** het object NxCliParser is dat door de **methode sdk.getCliParser()** is geretourneerd, **cmd_name** een unieke naam is voor de opdrachtinterne syntaxis van NX-SDK, en beschrijft welke trefwoorden en parameters in de opdracht kunnen worden gebruikt. Deze methode retourneert een NxCliCmd object, dat hier is gedocumenteerd. Deze methode is hier gedocumenteerd.

Opmerking: Deze opdracht is een subklasse van **cliP.newCliCmd("cmd_type", "cmd_name", "syntax")** waarin **cmd_type** ofwel **CONF_CMD** of **SHOW_CMD** is (dit hangt af van het type opdracht dat is geconfigureerd), **cmd_name is een unieke naam voor de opdracht intern aan de aangepaste NX-SDK-toepassing en de syntaxis beschrijft welke zoekwoorden en parameters in de opdracht kunnen worden gebruikt.** Daarom kan de API-documentatie voor deze opdracht behulpzamer zijn voor referentie.

Beide typen opdrachten hebben twee verschillende onderdelen: parameters en trefwoorden:

1. **De parameters** zijn waarden die worden gebruikt om de resultaten van de opdracht te wijzigen. Bijvoorbeeld, in het commando **tonen ip route 192.168.1.0**, is er een **route** sleutelwoord gevolgd door een parameter die een IP adres accepteert, die specificeert dat alleen routes die het opgegeven IP adres omvatten moeten worden getoond.

2. **Trefwoorden** veranderen de resultaten van de opdracht alleen door hun aanwezigheid. Bijvoorbeeld, in de opdracht **toon mac adres-table dynamiek**, is er een **dynamisch** sleutelwoord, dat specificeert dat slechts dynamisch-geleerde adressen van MAC moeten worden weergegeven.

Beide componenten worden gedefinieerd in de syntaxis van een NX-SDK-opdracht wanneer deze wordt gecreëerd. Methoden voor het object NxCliCmd bestaan om de specifieke implementatie van beide componenten te wijzigen.

- **nx_cmd.updateParam("<parameter>", "help_str", type)**, waarbij **nx_cmd** het NxCliCmd-object is dat terugkomt door de **cliP.newShowCmd()** of de **cliP.newConfigCmd()** methoden, **<parameter De naam van de opdrachtparameter die kan worden gewijzigd en ingesloten door hoekhaakjes (<>), help_str stelt de Help-string van de aangepaste opdracht in en wordt weergegeven in het NX-OS contextgevoelige Help-menu dat via een vraagteken op de CLI wordt benaderd, en het type is het type van de parameter.** Er zijn hier aanvullende optionele parameters voor deze methode beschikbaar en gedocumenteerd. Dit zijn geldige typen voor parameters die in het type argument kunnen worden gespecificeerd:

  P_INTEGER - specificeert elk integer**P_STRING** - specificeert elke string**P_INTERFACE** - specificeert elke netwerkinterface**P_IP_ADDR** - specificeert elk IP-adres**P_MAC_ADDR** - specificeert elk MAC-adres**P_VRF** - specificeert elke instantie voor Virtual Routing and Forwarding (VRF)
- **nx_cmd.updateKeyword("sleutelwoord", "help_str", is_key)**, waar **nx_cmd** het object NxCliCmd is dat **cliP.newShowCmd()** of de **cliP.newConfigCmd()** methodes, **sleutelwoordis De naam van het commando sleutelwoord dat u wilt wijzigen, stelt help_str de help-string van het aangepaste commando in en wordt weergegeven in het NX-OS context-gevoelige Help menu benaderd via een vraagteken op de CLI, en is_key is een optionele Booleaanse waarde die naar Faal standaard wordt weergegeven.** Als **is_key** Waar is, dan overschrijft de unieke configuratie die door de opdracht met het gebruik van dit sleutelwoord wordt gecreëerd andere unieke configuratie niet die door de opdracht wordt gecreëerd. Als **is_key** False is, dan is de configuratie gemaakt door de opdracht met het gebruik van dit sleutelwoord overschrijft de andere configuratie gemaakt door de opdracht. Deze methode is hier gedocumenteerd.

Om codevoorbeelden van algemeen gebruikte opdrachtcomponenten te bekijken, bekijkt u de sectie Aangepaste CLI-opdrachtvoorbeelden van dit document.

Nadat de aangepaste CLI-opdrachten zijn gemaakt, moet een object uit de klasse van de

pyCmdHandler die later in dit document wordt beschreven, worden gemaakt en ingesteld als het CLI callback handler-object voor het object NxCliParser. Dit wordt als volgt aangetoond:

```
 cmd_handler = pyCmdHandler()
cliP.setCmdHandler(cmd_handler)
```

Vervolgens moet het object NxCliParser worden toegevoegd aan de parserboom van NX-OS CLI, zodat de aangepaste CLI-opdrachten voor de gebruiker zichtbaar zijn. Dit gebeurt met de opdracht **cliP.addToParseTree()**, waarbij **cliP** het NxCliParser-object is dat teruggegeven is door de **sdk.getCliParser()** methode.

sdkThrefunctie-voorbeeld

Hier is een voorbeeld van een typische sdkThread-functie met het gebruik van de eerder genoemde functies. Deze functie (onder andere binnen een standaard aangepaste NX-SDK Python-toepassing) gebruikt globale variabelen, die worden geconcretiseerd bij de uitvoering van scripts.

```
cliP = ""
sdk = ""
event_hdlr = ""
tmsg = ""

def sdkThread():
    global cliP, sdk, event_hdlr, tmsg

    sdk = nx_sdk_py.NxSdk.getSdkInst(len(sys.argv), sys.argv)
    if not sdk:
       return

    sdk.setAppDesc("Returns all interfaces with DOM-capable transceivers inserted")

    tmsg = sdk.getTracer()
    tmsg.event("[{}] Started service".format(sdk.getAppName()))

    cliP = sdk.getCliParser()

    nxcmd = cliP.newShowCmd("show_port_bw_util_cmd", "port bw utilization [<port>]")
    nxcmd.updateKeyword("port", "Port Information")
    nxcmd.updateKeyword("bw", "Port Bandwidth Information")
    nxcmd.updateKeyword("utilization", "Port BW utilization in (%)")
    nxcmd.updateParam("<port>", "Optional Filter Port Ex) Ethernet1/1", nx_sdk_py.P_INTERFACE)

    nxcmd1 = cliP.newConfigCmd("port_bw_threshold_cmd", "port bw threshold <threshold>")
    nxcmd1.updateKeyword("threshold", "Port BW Threshold in (%)")

    int_attr = nx_sdk_py.cli_param_type_integer_attr()
    int_attr.min_val = 1;
    int_attr.max_val = 100;
    nxcmd1.updateParam("<threshold>", "Threshold Limit. Default 50%", nx_sdk_py.P_INTEGER,
int_attr, len(int_attr))

    mycmd = pyCmdHandler()
    cliP.setCmdHandler(mycmd)

    cliP.addToParseTree()

    sdk.startEventLoop()
```

```
    # If sdk.stopEventLoop() is called or application is removed from VSH...
    tmsg.event("Service Quitting...!")

    nx_sdk_py.NxSdk.__swig_destroy__(sdk)
```

CPDhanler-klasse

De categorie **pyCmdHandler** wordt geërfd van de **klasse NxCmdHandler** binnen de bibliotheek nx_sdk_py. De **postCliCb (zelf, klimd)** methode die binnen de klasse **pyCmdHandler** is gedefinieerd wordt geroepen wanneer CLI opdrachten heeft die van een NX-SDK toepassing afkomstig zijn. Zodoende definieert de **postCliCb (zelf, klimd)** methode waarbij u definieert hoe de aangepaste CLI opdrachten die binnen de sdkThread-functie zijn gedefinieerd, zich op het apparaat gedragen.

De **postCliCb (zelf, klik)** functie retourneert een Booleaanse waarde. Als **True** wordt teruggegeven, dan wordt aangenomen dat de opdracht met succes is uitgevoerd. **False** moet worden teruggegeven als de opdracht om geen enkele reden succesvol is uitgevoerd.

De **geklikte** parameter gebruikt de unieke naam die voor de opdracht was gedefinieerd toen deze in de sdkThread-functie werd gemaakt. Bijvoorbeeld, als u een nieuwe **show** opdracht maakt met een unieke naam van **show_xcvr_dom**, dan wordt aanbevolen om naar deze opdracht te verwijzen door dezelfde naam in de **postCliCb (zelf, klik)** functie nadat u hebt gecontroleerd om te zien of de naam van het aangeklikte argument **show_xcvr_dom** bevat. Hier wordt aangetoond:

```
def sdkThread():
    <snip>
    sh_xcvr_dom = cliP.newShowCmd("show_xcvr_dom", "dom")
    sh_xcvr_dom.updateKeyword("dom", "Show all interfaces with transceivers that are DOM-
capable")
    </snip>

class pyCmdHandler(nx_sdk_py.NxCmdHandler):
    def postCliCb(self, clicmd):
        if "show_xcvr_dom" in clicmd.getCmdName():
            get_dom_capable_interfaces()
```

Als er een opdracht wordt gemaakt die parameters gebruikt, dan moet u deze parameters waarschijnlijk gebruiken op een bepaald punt in de **postCliCb (zelf, klik)** functie. Dit kan worden gedaan met de methode klimd**.getParamValue("<parameter>")**, waarbij **<parameter>** de naam is van de opdrachtparameter die u de waarde van ingesloten door hoekhaakjes wilt hebben (**<>**). Deze methode is hier gedocumenteerd. De waarde die door deze functie wordt geretourneerd, moet echter worden geconverteerd naar het gewenste type. Dit kan gebeuren met de volgende methoden:

- **nx_sdk_py.leegte_to_int** converteert een waarde naar een integertype.
- **nx_sdk_py.leegte_to_string** converteert een waarde naar een string type.

De **postCliCb (zelf, klimd)** functie (of enige daaropvolgende functies) zal ook gewoonlijk zijn waar de show-opdrachtoutput wordt afgedrukt op de console. Dit gebeurt met de methode **climd.printConsole()**.

> Opmerking: Als de applicatie een fout tegenkomt, als er een uitzondering niet wordt verwerkt of als er iets anders opeens buiten komt, wordt de output van de functie Click**.printConsole()** helemaal niet weergegeven. Om deze reden is de beste praktijk wanneer je je Python-

toepassing debug-berichten naar de syslog loggen met het gebruik van een NxTrace-object dat is teruggegeven door de methode **sdk.getTracer()** of gedrukte uitspraken gebruiken en de toepassing uitvoeren via de binaire binaire structuur van de Bash shell**/bin/python**.

Voorbeeld van QDhandler-klasse

De volgende code dient als voorbeeld voor de hierboven beschreven klasse van de **pyCmdHandler**. Deze code is afgeleid uit het ip_Move.py-bestand in de ip-beweging NX-SDK-toepassing die hier beschikbaar is. Het doel van deze toepassing is om de beweging van een door gebruiker bepaald IP adres over interfaces van een Nexus apparaat te volgen. Om dit te doen, vindt de code het adres van MAC van het IP adres ingangssignaal door de **<ip>** parameter binnen het ARP cache van het apparaat, en verifieert dan welk VLAN dat het adres van MAC zich in het gebruiken van de MAC adrestabel van het apparaat bevindt. Wanneer u deze MAC en VLAN gebruikt, toont het **bevel van het showsysteem interne l2fm l2dbg macdb adres <mac> VLAN <VLAN>** een lijst van SNMP interface-indexen waarmee deze combinatie onlangs is geassocieerd. De code gebruikt dan het bevel van de **show interface snmp-ifindex** om recente SNMP interface indexen in mens-leesbare interfacenamen te vertalen.

```
class pyCmdHandler(nx_sdk_py.NxCmdHandler):
    def postCliCb(self, clicmd):
        global cli_parser

        if "show_ip_movement" in clicmd.getCmdName():
            target_ip = nx_sdk_py.void_to_string(clicmd.getParamValue("<ip>"))

            target_mac = get_mac_from_arp(cli_parser, clicmd, target_ip)
            mac_vlan = ""
            if target_mac:
                mac_vlan = get_vlan_from_cam(cli_parser, clicmd, target_mac)
                if mac_vlan:
                    find_mac_movement(cli_parser, clicmd, target_mac, mac_vlan)
                else:
                    print("No entires in MAC address table")
                    clicmd.printConsole("No entries in MAC address table for
{}".format(target_mac))
            else:
                clicmd.printConsole("No entries in ARP table for {}".format(target_ip))
        return True

def get_mac_from_arp(cli_parser, clicmd, target_ip):
    exec_cmd = "show ip arp {}".format(target_ip)
    arp_cmd = cli_parser.execShowCmd(exec_cmd, nx_sdk_py.R_JSON)
    if arp_cmd:
        try:
            arp_json = json.loads(arp_cmd)
        except ValueError as exc:
            return None
        count = int(arp_json["TABLE_vrf"]["ROW_vrf"]["cnt-total"])
        if count:
            intf = arp_json["TABLE_vrf"]["ROW_vrf"]["TABLE_adj"]["ROW_adj"]
            if intf.get("ip-addr-out") == target_ip:
                target_mac = intf["mac"]
                clicmd.printConsole("{} is currently present in ARP table, MAC address
{}\n".format(target_ip, target_mac))
                return target_mac
            else:
                return None
        else:
```

```
            return None
    else:
        return None

def get_vlan_from_cam(cli_parser, clicmd, target_mac):
    exec_cmd = "show mac address-table address {}".format(target_mac)
    mac_cmd = cli_parser.execShowCmd(exec_cmd, nx_sdk_py.R_JSON)
    if mac_cmd:
        try:
            cam_json = json.loads(mac_cmd)
        except ValueError as exc:
            return None
        mac_entry = cam_json["TABLE_mac_address"]["ROW_mac_address"]
        if mac_entry:
            if mac_entry["disp_mac_addr"] == target_mac:
                egress_intf = mac_entry["disp_port"]
                mac_vlan = mac_entry["disp_vlan"]
                clicmd.printConsole("{} is currently present in MAC address table on interface
{}, VLAN {}\n".format(target_mac, egress_intf, mac_vlan))
                return mac_vlan
            else:
                return None
        else:
            return None
    else:
        return None

def find_mac_movement(cli_parser, clicmd, target_mac, mac_vlan):
    exec_cmd = "show system internal l2fm l2dbg macdb address {} vlan {}".format(target_mac,
mac_vlan)
    l2fm_cmd = cli_parser.execShowCmd(exec_cmd)
    if l2fm_cmd:
        event_re = re.compile(r"^\s+(\w{3}) (\w{3}) (\d+) (\d{2}):(\d{2}):(\d{2}) (\d{4})
(0x\S{8}) (\d+)\s+(\S+) (\d+)\s+(\d+)\s+(\d+)")
        unique_interfaces = []
        l2fm_events = l2fm_cmd.splitlines()
        for line in l2fm_events:
            res = re.search(event_re, line)
            if res:
                day_name = res.group(1)
                month = res.group(2)
                day = res.group(3)
                hour = res.group(4)
                minute = res.group(5)
                second = res.group(6)
                year = res.group(7)
                if_index = res.group(8)
                db = res.group(9)
                event = res.group(10)
                src=res.group(11)
                slot = res.group(12)
                fe = res.group(13)
                if "MAC_NOTIF_AM_MOVE" in event:
                    timestamp = "{} {} {} {}:{}:{} {}".format(day_name, month, day, hour,
minute, second, year)
                    intf_dict = {"if_index": if_index, "timestamp": timestamp}
                    unique_interfaces.append(intf_dict)
        if not unique_interfaces:
            clicmd.printConsole("No entries for {} in L2FM L2DBG\n".format(target_mac))
        if len(unique_interfaces) == 1:
            clicmd.printConsole("{} has not been moving between
interfaces\n".format(target_mac))
        if len(unique_interfaces) > 1:
                clicmd.printConsole("{} has been moving between the following interfaces, from
```

```
most recent to least recent:\n".format(target_mac))
                unique_interfaces = get_snmp_intf_index(unique_interfaces)
                clicmd.printConsole("\t{} - {} (Current interface)\n".format(unique_interfaces[-
1]["timestamp"], unique_interfaces[-1]["intf_name"]))
                for intf in unique_interfaces[-2::-1]:
                    clicmd.printConsole("\t{} - {}\n".format(intf["timestamp"],
intf["intf_name"]))
def get_snmp_intf_index(if_index_dict_list): global cli_parser snmp_ifindex =
cli_parser.execShowCmd("show interface snmp-ifindex", nx_sdk_py.R_JSON) snmp_ifindex_json =
json.loads(snmp_ifindex) snmp_ifindex_list =
snmp_ifindex_json["TABLE_interface"]["ROW_interface"] for index_dict in if_index_dict_list:
index = index_dict["if_index"] for ifindex_json in snmp_ifindex_list: if index ==
ifindex_json["snmp-ifindex"]: index_dict["intf_name"] = ifindex_json["interface"] return
if_index_dict_list
```

Aangepaste CLI-bronbelastingvoorbeelden

Deze sectie toont een paar voorbeelden van de syntax parameter die gebruikt wordt wanneer u aangepaste CLI-opdrachten maakt met de **cliP.newShowCmd()** of de **cliP.newConfigCmd()** methoden, waarbij **cliP** het NxCliParser-object is dat door **sdk.get**Parser() is geretourneerd methode.

> Opmerking: Ondersteuning voor syntax met openings- en sluitparenthesen ("(" en ")") wordt geïntroduceerd in NX-SDK v1.5.0, opgenomen in NX-OS release 7.0(3)I7(3). Aangenomen wordt dat de gebruiker NX-SDK v1.5.0 gebruikt wanneer hij een van deze voorbeelden volgt, zoals de syntax van open en sluitparenthesen.

Enkelvoudig toetsenwoord

Deze show opdracht neemt één sleutelwoord mac en voegt een helper string toe van shows alle verkeerd geprogrammeerde MAC adressen op dit apparaat aan het sleutelwoord.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed", "mac")
nx_cmd.updateKeyword("mac", "Shows all misprogrammed MAC addresses on this device")
```

enkele parameter

Deze show opdracht heeft één parameter **<mac>**. De insluiten hoekhaakjes rond het woord kunnen betekenen dat dit een parameter is. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de parameter als MAC-adres te definiëren, dat de eindgebruikersinvoer van een ander type, zoals een string, integer of IP-adres, voorkomt.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed_mac", "<mac>")
nx_cmd.updateParam("<mac>", "MAC address to check for misprogramming", nx_sdk_py.P_MAC_ADDR)
```

Optioneel sleutelwoord

Deze show opdracht kan optioneel één sleutelwoord nemen **[mac]**. De ingesloten haakjes rond het woord kunnen betekenen dat dit trefwoord optioneel is. Een helper string van shows alle verkeerd

geprogrammeerde MAC adressen op dit apparaat wordt toegevoegd aan het sleutelwoord.

```
 nx_cmd = cliP.newShowCmd(  "show_misprogrammed_mac"  ,   "[mac]"  )
nx_cmd.updateKeyword( "mac" , "Shows all misprogrammed MAC addresses on this device" )
```

Optionele parameter

Deze show opdracht kan optioneel één parameter nemen **[<mac>]**. De insluithaakjes rond het woord **<** Mac **>** geven aan dat deze parameter optioneel is. De insluiten hoekhaakjes rond het woord kunnen betekenen dat dit een parameter is. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de parameter als MAC-adres te definiëren, dat de eindgebruikersinvoer van een ander type, zoals een string, integer of IP-adres, voorkomt.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed_mac", "[<mac>]")
nx_cmd.updateParam("<mac>", "MAC address to check for misprogramming", nx_sdk_py.P_MAC_ADDR)
```

Enkelvoudig sleutelwoord en parameter

Deze show opdracht heeft één sleutelwoordenlijst direct gevolgd door de parameter **<mac-adres>**. De insluiten hoekhaakjes rond het woord mac-adres geven aan dat dit een parameter is. Een helper string van het adres van MAC van de Controle voor misprogrammering wordt toegevoegd aan het sleutelwoord. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de parameter als MAC-adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP-adres, voorkomt.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed", "mac <mac-address>")
nx_cmd.updateKeyword("mac", "Check MAC address for misprogramming")
nx_cmd.updateParam("<mac-address>", "MAC address to check for misprogramming",
nx_sdk_py.P_MAC_ADDR)
```

Meerdere sleutelwoorden en parameters

Deze show opdracht kan één van twee sleutelwoorden nemen, die beiden twee verschillende parameters hebben die hen volgen. Het eerste sleutelsleutelwoord mac heeft een parameter van **<mac-adres>**, en het tweede sleutelsleutelwoord ip heeft een parameter van **<ip-adres>**. De insluiten hoekhaakjes rond de woorden adres en ip-adres geven aan dat ze parameters zijn. Een helper string van het adres van MAC van het Controle voor misprogrammering wordt toegevoegd aan het hoofdtrefwoord. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<mac-adres>** parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<mac-adres>** parameter als MAC-adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert. Een helper string van het IP-adres controleren voor fouillering wordt toegevoegd aan het IP-trefwoord. Een helper string van IP adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<ip-adres>** parameter. De **nx_sdk_py.P_IP_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<ip-adres>** parameter als IP adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed", "(mac <mac-address> | ip <ip-address>)")
nx_cmd.updateKeyword("mac", "Check MAC address for misprogramming")
nx_cmd.updateParam("<mac-address>", "MAC address to check for misprogramming",
nx_sdk_py.P_MAC_ADDR)
nx_cmd.updateKeyword("ip", "Check IP address for misprogramming")
nx_cmd.updateParam("<ip-address>", "IP address to check for misprogramming",
nx_sdk_py.P_IP_ADDR)
```

Meerdere sleutelwoorden en parameters met optioneel sleutelwoord

Deze show opdracht kan één van twee sleutelwoorden nemen, die beiden twee verschillende parameters hebben die hen volgen. Het eerste sleutelsleutelwoord mac heeft een parameter van **<mac-adres>**, en het tweede sleutelsleutelwoord ip heeft een parameter van **<ip-adres>**. De insluiten hoekhaakjes rond de woorden adres en ip-adres geven aan dat ze parameters zijn. Een helper string van het adres van MAC van het Controle voor misprogrammering wordt toegevoegd aan het hoofdtrefwoord. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<mac-adres>** parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<mac-adres>** parameter als MAC-adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert. Een helper string van het IP-adres controleren voor fouillering wordt toegevoegd aan het IP-trefwoord. Een helper string van IP adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<ip-adres>** parameter. De **nx_sdk_py.P_IP_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<ip-adres>** parameter als IP adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert. Deze show opdracht kan optioneel een sleutelwoord nemen [duidelijk]. Een helper string Clears adressen die zijn gedetecteerd om verkeerd geprogrammeerd te worden, wordt toegevoegd aan dit optionele sleutelwoord.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed", "(mac <mac-address> | ip <ip-address>) [clear]")
nx_cmd.updateKeyword("mac", "Check MAC address for misprogramming")
nx_cmd.updateParam("<mac-address>", "MAC address to check for misprogramming",
nx_sdk_py.P_MAC_ADDR)
nx_cmd.updateKeyword("ip", "Check IP address for misprogramming")
nx_cmd.updateParam("<ip-address>", "IP address to check for misprogramming",
nx_sdk_py.P_IP_ADDR)
nx_cmd.updateKeyword("clear", "Clears addresses detected to be misprogrammed")
```

Meerdere sleutelwoorden en parameters met optionele parameter

Deze show opdracht kan één van twee sleutelwoorden nemen, die beiden twee verschillende parameters hebben die hen volgen. Het eerste sleutelsleutelwoord mac heeft een parameter van **<mac-adres>**, en het tweede sleutelsleutelwoord ip heeft een parameter van **<ip-adres>**. De insluiten hoekhaakjes rond de woorden adres en ip-adres geven aan dat ze parameters zijn. Een helper string van het adres van MAC van de Controle voor misprogrammeur wordt toegevoegd aan het hoofdtrefwoord. Een helper string van het MAC-adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<mac-adres>** parameter. De **nx_sdk_py.P_MAC_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<mac-adres>** parameter als MAC-adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert. Een helper string van het IP-adres controleren voor fouillering wordt toegevoegd aan het IP-trefwoord. Een helper string van IP adres om te controleren op verkeerde programmering wordt toegevoegd aan de **<ip-adres>** parameter. De **nx_sdk_py.P_IP_ADDR** parameter in de **nx_cmd.updateParam()** methode wordt gebruikt om het type van de **<ip-adres>** parameter als IP adres te definiëren, dat de eindgebruiker input van een ander type, zoals een string, integer of IP adres, verhindert. Deze show opdracht kan optioneel

een parameter **[<module>]** nemen. Een helperstring Alleen heldere adressen op gespecificeerde module worden toegevoegd aan deze optionele parameter.

```
 nx_cmd = cliP.newShowCmd("show_misprogrammed", "(mac <mac-address> | ip <ip-address>)
[<module>]")
nx_cmd.updateKeyword("mac", "Check MAC address for misprogramming")
nx_cmd.updateParam("<mac-address>", "MAC address to check for misprogramming",
nx_sdk_py.P_MAC_ADDR)
nx_cmd.updateKeyword("ip", "Check IP address for misprogramming")
nx_cmd.updateParam("<ip-address>", "IP address to check for misprogramming",
nx_sdk_py.P_IP_ADDR)
nx_cmd.updateParam("<module>", "Clears addresses detected to be misprogrammed",
nx_sdk_py.P_INTEGER)
```

Debug een Python-toepassing met NX-SDK

Zodra een NX-SDK Python-toepassing is gecreëerd, zal deze vaak moeten worden gezuiverd. NX-SDK informeert u voor het geval er syntaxisfouten in uw code zijn, maar omdat de Python NX-SDK-bibliotheek SWIG gebruikt om C++-bibliotheken naar Python-bibliotheken te vertalen, leiden alle uitzonderingen die op het moment van de uitvoering van de code worden aangetroffen, tot een op dit gebied vergelijkbaar toepassingskern:

```
 terminate called after throwing an instance of 'Swig::DirectorMethodException'
what(): SWIG director method error. Error detected when calling 'NxCmdHandler.postCliCb'
Aborted (core dumped)
```

Vanwege de dubbelzinnige aard van deze foutmelding is de beste methode om Python-toepassingen te debug-toepassingen te gebruiken het loggen van debug-berichten naar de syslog met behulp van een NxTrace-object dat is teruggegeven door de methode **sdk.getTracer()**. Dit wordt als volgt aangetoond:

```
#! /isan/bin/python

tracer = 0

def evt_thread():
    <snip>
    tracer = sdk.getTracer()
    tracer.event("[NXSDK-APP][INFO] Started service")
<snip>
class pyCmdHandler(nx_sdk_py.NxCmdHandler):
    def postCliCb(self, clicmd):
        global tracer
        tracer.event("[NXSDK-APP][DEBUG] Received command: {}".format(clicmd))
        if "show_test_command" in clicmd.getCmdName():
            tracer.event("[NXSDK-APP][DEBUG] `show_test_command` recognized")
```

Als loggen geen boodschap naar syslog is, is een alternatieve methode om printstatements te gebruiken en de toepassing uit te voeren via het binaire binaire toetsenbord **van** shell**/isan/bin/python**. De output van deze print statements zal echter alleen zichtbaar zijn wanneer ze op deze manier worden uitgevoerd - de toepassing door de VSH shell geeft geen output. Hier wordt een voorbeeld weergegeven van het gebruik van afdrukverklaringen:

```
#! /isan/bin/python
```

```
tracer = 0

def evt_thread():
    <snip>
    print("[NXSDK-APP][INFO] Started service")
<snip>
class pyCmdHandler(nx_sdk_py.NxCmdHandler):
    def postCliCb(self, clicmd):
        print("[NXSDK-APP][DEBUG] Received command: {}".format(clicmd))
        if "show_test_command" in clicmd.getCmdName():
            print("[NXSDK-APP][DEBUG] `show_test_command` recognized")
```

Een Python-toepassing implementeren met NX-SDK

Zodra een Python-toepassing volledig is getest in de Bash-schelp en klaar is voor gebruik, moet de toepassing via VSH in productie worden gebracht. Dit laat de toepassing toe om door te blijven wanneer het apparaat opnieuw wordt geladen of wanneer de systeemomschakeling in een dubbel controlescenario plaatsvindt. Om een toepassing door VSH te kunnen implementeren moet u een RPM-pakket maken met het gebruik van een NX-SDK- en ENXOS SDK-werkomgeving. Cisco DevNet biedt een Docker-afbeelding die geschikt is voor het maken van eenvoudige RPM-pakketten.

> Opmerking: Raadpleeg de installatiedocumentatie van Docker voor informatie over het installeren van Docker op uw specifieke besturingssysteem.

Op een Docker-Geschikt host trekt u de afbeeldingsversie van uw keuze door de **docker pull-dockercisco/nxsdk:<tag>** opdracht, waarbij **<tag>** de tag is van de afbeeldingsversie van uw keuze. U kunt de beschikbare afbeeldingsversies en de bijbehorende tags hier bekijken. Dit wordt aangetoond met de tag **v1**:

```
docker pull dockercisco/nxsdk:v1
```

Start een container met de naam **nxsdk** van deze afbeelding en hecht er aan. Als het label van uw keuze anders is, kunt u **v1** voor uw tag vervangen:

```
docker run -it --name nxsdk dockercisco/nxsdk:v1 /bin/bash
```

Update naar de laatste versie van NX-SDK en navigeer naar de **NX-SDK** directory en trek de laatste bestanden uit de put:

```
cd /NX-SDK/
git pull
```

Als u een oudere versie van NX-SDK wilt gebruiken, kunt u de NX-SDK-tak klonen met het gebruik van de **respectievelijke** versietag met de **Gigabit kloon-b v<versie>** **https://github.com/CiscoDevNet/NX-SDK.git-**opdracht, waarbij **<versie>**de versie is van NX-SDK die u nodig hebt. Dit wordt hier met NX-SDK v1.0.0 aangetoond:

```
cd /
rm -rf /NX-SDK
git clone -b v1.0.0 https://github.com/CiscoDevNet/NX-SDK.git
```

Breng uw Python-applicatie over op de Docker-container. Er zijn een paar verschillende manieren om dit te doen.

- Afsluiten van de Docker-container (die de container stopt en vereist dat u deze opnieuw start), de Python-toepassing overbrengen naar de Docker-host, en dan de **docker cp-**opdracht gebruiken om de applicatie van de host naar de container te kopiëren. Dit wordt hier aangetoond, in de veronderstelling dat de Python-toepassing is overgebracht naar de Docker host op **/app/python_app.py**.

```
 root@2dcbe841742a:~# exit
[root@localhost ~]# docker cp /app/python_app.py nxsdk:/root/
[root@localhost ~]# docker start nxsdk
nxsdk
[root@localhost ~]# docker attach nxsdk
root@2dcbe841742a:/# ls /root/
python_app.py
```

- Kopieer de inhoud van de Python-toepassing naar het klembord van het systeem en plak de inhoud vervolgens in een bestand dat met behulp van vim in de Docker-container is gemaakt.

Daarna, gebruik het **rpm_gen.py** script dat in **NX-SDK/scripts/scripts/**is geplaatst om een RPM-pakket te maken vanuit de Python-toepassing. Dit script heeft een benodigde argument en twee vereiste switches:

- De naam van de Python-toepassing. Een Python-toepassing in een bestand met de naam **python_app.py** zou bijvoorbeeld resulteren in een argument of **python_app.py**. Deze bestandsnaam zal later worden gebruikt als de toepassingsnaam voor NX-SDK en zal ook worden gebruikt door NX-OS om naar opdrachten te verwijzen die door deze toepassing zijn gemaakt.

  Opmerking: De bestandsnaam hoeft geen bestandsuitbreidingen te bevatten, zoals **.py**. In dit voorbeeld, als de bestandsnaam **python_app** was in plaats van **python_app.py**, zou het RPM-pakket zonder probleem gegenereerd worden.

- De -s switch neemt een argument voor het absolute filepath dat naar de locatie van de bovengenoemde bestandsnaam leidt. Als bijvoorbeeld **python_app.py** zich in **/root/**bevindt, dan zou het juiste argument **-s /root/**.
- De -u schakelaar geeft aan dat de bron bestandsnaam hetzelfde is als de uitvoerbare bestandsnaam.

Gebruik van het **rpm_gen.py** script blijkt hier.

```
root@7bfd1714dd2f:~# python /NX-SDK/scripts/rpm_gen.py test_python_app -s /root/ -u
##############################################################################################################
####
Generating rpm package...
<snip>
RPM package has been built
##############################################################################################################
####

SPEC file: /NX-SDK/rpm/SPECS/test_python_app.spec
RPM file : /NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_64.rpm
```

Het pad naar het RPM-pakket wordt aangegeven in de laatste regel van het script dat **rpm_gen.py** wordt uitgevoerd. Dit bestand moet van de Docker-container naar de host worden gekopieerd zodat het kan worden overgebracht naar het Nexus-apparaat waarop u de toepassing wilt uitvoeren. Nadat u de Docker-container hebt verlaten, kan dit eenvoudig worden gedaan met de

docker-cp <container>:<container_filepath> <host_filepath>opdracht, waar **<container>** de naam van de NX-SDK Docker-container (in dit geval nxsdk) is, **<container_filepath>** het volledige bestandspakket in de container **/NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_64.rpm**) en **<host_filepath> is het volledige bestand op onze Docker-host waarop het RPM-pakket moet worden overgebracht (in dit geval /root/).** Deze opdracht is hier aangetoond:

```
root@7bfd1714dd2f:/# exit
[root@localhost ~]# docker cp nxsdk:/NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_64.rpm /root/
[root@localhost ~]# ls /root/
anaconda-ks.cfg             test_python_app-1.0-1.0.0.x86_64.rpm
```

Breng dit RPM-pakket over naar het Nexus-apparaat met behulp van uw favoriete methode voor het overdragen van bestanden. Nadat het RPM-pakket op het apparaat is aangebracht, moet dit op dezelfde manier worden geïnstalleerd en geactiveerd als een SMU. Dit wordt als volgt aangetoond, in de veronderstelling dat het RPM - pakket werd overgebracht naar de flitser van het apparaat.

```
N9K-C93180LC-EX# install add bootflash:test_python_app-1.0-1.0.0.x86_64.rpm
[####################] 100%
Install operation 27 completed successfully at Tue May  8 06:40:13 2018
N9K-C93180LC-EX# install activate test_python_app-1.0-1.0.0.x86_64
[####################] 100%
Install operation 28 completed successfully at Tue May  8 06:40:20 2018
```

> Opmerking: Wanneer u het RPM-pakket installeert met de **installatieopdracht**, neemt u het opslagapparaat en de exacte bestandsnaam van de verpakking op. Wanneer u het RPM-pakket na de installatie activeert, neemt u het opslagapparaat en de bestandsnaam niet op - gebruik de naam van de verpakking zelf. U kunt de pakketnaam controleren met de **show installeert de inactieve** opdracht.

Nadat het RPM-pakket is geactiveerd, kunt u de toepassing met NX-SDK starten met de **nxsdk-service <application-name>**configuratieopdracht, waarbij **<application-name>** de naam van de Python filename is (en vervolgens de toepassing) die is gedefinieerd toen rpm_gen.py script eerder werd gebruikt. Dit wordt als volgt aangetoond:

```
N9K-C93180LC-EX# conf
Enter configuration commands, one per line. End with CNTL/Z.
N9K-C93180LC-EX(config)# nxsdk service-name test_python_app
% This could take some time. "show nxsdk internal service" to check if your App is Started &
Runnning
```

U kunt verifiëren dat de toepassing omhoog is en met de **tonen nxsdk interne** dienstopdracht is begonnen te lopen:

```
N9K-C93180LC-EX# show nxsdk internal service

NXSDK Started/Temp unavailabe/Max services : 1/0/32
NXSDK Default App Path         : /isan/bin/nxsdk
NXSDK Supported Versions       : 1.0

Service-name              Base App        Started(PID)      Version    RPM Package
------------------------- --------------- ----------------- ---------- ------------------------
test_python_app           nxsdk_app4      VSH(23195)        1.0        test_python_app-1.0-
1.0.0.x86_64
```

U kunt ook controleren of de aangepaste CLI-opdrachten die door deze toepassing zijn gemaakt,

in NX-OS toegankelijk zijn:

```
N9K-C93180LC-EX# show test?
  test_python_app    Nexus Sdk Application
```

Gerelateerde informatie

- NX-SDK Gigabit-hub
- Cisco Nexus 9000 Series NX-OS programmeerbaarheidsgids, release 7.x
- Cisco Nexus 3000 Series netwerkmodule voor NX-OS programmeerbaarheid, release 7.x
- Cisco Nexus 3500 Series netwerkmodule voor NX-OS programmeerbaarheid, release 7.x
- Netwerkprogrammeerbaarheid en -automatisering met Cisco Nexus 9000 Series-switches - White Paper
- Programmeerbaarheid en automatisering met Cisco Open NX-OS (PDF)
- Technische ondersteuning en documentatie – Cisco Systems