Développez, débuggez et déployez l'application de python NX-SDK dans des Commutateurs du Nexus 3000/9000

Contenu

Introduction
Conditions préalables
Conditions requises
Composants utilisés
Informations générales
Développez une application de python avec NX-SDK
Enable NX-SDK
Créez un fichier de python
Composants de la mise en place NX-SDK
Créez les commandes faites sur commande CLI
classe de pyCmdHandler
Exemples faits sur commande de syntaxe de commande CLI
Mot clé simple
Paramètre simple
Mot clé facultatif
Paramètre optionnel
Mot clé et paramètre simples
Plusieurs mots clé et paramètres
Plusieurs mots clé et paramètres avec le mot clé facultatif
Plusieurs mots clé et paramètres avec le paramètre optionnel
Débuggez une application de python avec NX-SDK
Déployez une application de python avec NX-SDK
Informations connexes

Introduction

Ce document fournit un processus pour le Développement d'applications de python en kit de développement de NX-logiciel de Cisco (SDK) en Cisco NX-OS d'utilisation sur les Plateformes du Nexus 3000 et du Nexus 9000.

Conditions préalables

Conditions requises

Aucune spécification déterminée n'est requise pour ce document.

Composants utilisés

Les informations contenues dans ce document sont basées sur les versions de matériel et de logiciel suivantes :

- Ce document utilise NX-SDK v1.0.0 et NX-SDK v1.5.0
- NX-SDK v1.0.0 peut être utilisé sur la plate-forme du Nexus 9000 à partir de la version NX-OS 7.0(3)I6(1) et la plate-forme du Nexus 3000 à partir de la version NX-OS 7.0(3)I7(1)
- NX-SDK v1.5.0 peut être utilisé sur la plate-forme du Nexus 9000 et la plate-forme du Nexus 3000 à partir de la version NX-OS 7.0(3)I7(3)

Les informations contenues dans ce document ont été créées à partir des périphériques d'un environnement de laboratoire spécifique. Tous les périphériques utilisés dans ce document ont démarré avec une configuration effacée (par défaut). Si votre réseau est opérationnel, assurez-vous que vous comprenez l'effet potentiel de toute commande.

Informations générales

Cisco NX-SDK tient compte du développement des applications personnalisées qui peuvent fonctionner à la façon des indigènes dans le Cisco NX-OS sur des Plateformes du Nexus 9000 et du Nexus 3000. NX-SDK offre la capacité pour que les clients créent leurs propres commandes CLI et les sorties, génèrent des Syslog personnalisés en réponse aux événements spécifiques, télémétrie travaillée par flot, et beaucoup davantage.

NX-SDK a le courant alternatif ++ API, qui est traduit dans d'autres langages avec l'utilisation du wrapper et du générateur simplifiés d'interface (GORGÉE). Ceci tient compte pour que le client utilise NX-SDK en n'importe quel langage de leur choix. Ce document explique l'implémentation des fonctions communes NX-SDK dans le python, aussi bien que fournit un processus pour que les clients développent leurs propres applications de python NX-SDK.

Développez une application de python avec NX-SDK

Enable NX-SDK

Pour que n'importe quelle application NX-SDK s'exécute, la caractéristique NX-SDK doit d'abord être activée sur le périphérique :

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

Créez un fichier de python

Un fichier de python peut être créé et édité avec l'utilisation du shell de coup NX-OS. Pour que le shell de coup soit utilisé, il doit d'abord être activé sur le périphérique :

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

Écrivez le shell de coup et employez l'éditeur de texte vi afin de créer et éditer le fichier de python :

```
switch(config)# run bash
```

```
bash-4.2$ vi /isan/bin/nxsdk-app.py
```

Note: La pratique recommandée est de créer des fichiers de python dans le répertoire de **/isan/bin/**. Des autorisations d'exécution du besoin de fichiers de python afin de s'exécuter - ne placez les fichiers de python dans le répertoire de **/bootflash** ou aucun de ses sous-répertoires.

Note: On ne l'exige pas pour créer et éditer des fichiers de python par NX-OS. Le développeur peut créer l'application utilisant leur environnement local et transférer les fichiers terminés vers le périphérique utilisant un protocole de transfert de fichiers de leur choix. Cependant, il pourrait être plus efficace pour que le développeur mette au point et de dépanne leur script utilisant des utilitaires NX-OS.

Composants de la mise en place NX-SDK

Il est recommandé pour que des développeurs commencent à créer leur application de python NX-SDK du modèle de customCliPyApp sur Cisco DevNet NX-SDK GitHub. Ces sections de ce document se rapportent aux composants nécessaires avec l'utilisation de leurs noms dans ce modèle.

Il y a quatre composants importants nécessaires pour une application de python NX-SDK :

1. NX-SDK doit être importé dans l'application par l'intermédiaire de la déclaration **nx_sdk_py d'importation**.
2. Une fonction (**sdkThread** en général Désigné) cette commence l'application NX-SDK et modifie de diverses options liées à l'application.
3. La création des commandes faites sur commande CLI aussi bien que la définition de la syntaxe de commande faite sur commande CLI dans le **sdkThread** fonctionnent.
4. Une classe a nommé le **pyCmdHandler** avec une méthode nommée le **postCliCb**, qui traite des commandes faites sur commande CLI ajoutées par l'application NX-SDK.

fonction de sdkThread

La fonction de sdkThread initialise, ajoute la fonctionnalité à, et commence l'application NX-SDK. La fonction n'exige d'aucun paramètre d'être passé dans elle. Toutes les applications du python NX-SDK exigent de trois méthodes de la bibliothèque nx_sdk_py de s'appeler :

1. **nx_sdk_py.NxSdk.getSdkInst (len(sys.argv), sys.argv) renvoie un** objet d'**exemple** SDK, ou n'en renvoie aucun. **Si un objet d'exemple SDK est retourné, alors l'application NX-SDK s'est avec succès inscrite à l'infrastructure NX-OS. Si aucun n'est retourné, alors les erreurs se produisent au moment de cette procédure d'enregistrement, et une entrée de journal des erreurs apparaît dans le Syslog du périphérique. Cette méthode est documentée ici.**

   Note: À partir de NX-SDK v1.5.0, un troisième paramètre booléen peut être passé dans la

méthode NxSdk.getSdkInst, qui active des exceptions avancées si vraie et désactive des exceptions avancées si fausse. Cette méthode est documentée ici.

1. méthode sdk.startEventLoop (), où les sdkis l'objet d'exemple SDK sont retournés par la méthode nx_sdk_py.NxSdk.getSdkInst (). Cette méthode commence l'application NX-SDK et lui permet pour interagir avec l'infrastructure NX-OS. Cette méthode est documentée ici.
2. méthode de __swig_destroy__(sdk) nx_sdk_py.NxSdk., où le sdk est l'objet d'exemple SDK retourné par la méthode nx_sdk_py.NxSdk.getSdkInst () expliquée précédemment. Cette méthode placée à l'extrémité des functionallows de sdkThread pour la sortie gracieuse de l'application NX-SDK.

Quelques méthodes utilisées généralement incluent :

- sdk.getTracer (), où les sdkis l'objet d'exemple SDK sont retournés par la méthode nx_sdk_py.NxSdk.getSdkInst (). Cette méthode renvoie un objet de NxTrace qui peut être utilisé pour générer des Syslog faits sur commande, aussi bien que se connecte des événements et des erreurs à l'historique de l'événement d'application. Les Syslog faits sur commande apparaîtront dans le Syslog du périphérique (visible par l'intermédiaire de la commande de show logging logfile) tandis que les événements connectés à l'historique de l'événement d'application sont visibles par l'intermédiaire des événements d'événement-historique de nxsdk de <application-name> d'exposition ou afficheront des commandes d'erreurs d'événement-historique de nxsdk de <application-name>. Cette méthode est documentée ici. Le NxTrace objectreturned par cette méthode et ses méthodes associées sont documentés ici. Par exemple, la vue youcan l'historique d'événement d'une application nommée des événements de nxsdkevent-historique de l'exposition Transceiver_DOM.py de throughthe Transceiver_DOM.py et des erreurs d'événement-historique de nxsdk de l'exposition Transceiver_DOM.py commande.
- sdk.getCliParser (), où les sdkis l'objet d'exemple SDK sont retournés par la méthode nx_sdk_py.NxSdk.getSdkInst (). Cette méthode renvoie un objet de NxCliParser, qui peut être utilisé pour exécuter le CLI commande qui déjà existent par l'intermédiaire du python aussi bien que créent des commandes faites sur commande CLI. Cette méthode est documentée ici. Le NxCliParserobject retourné par cette méthode et ses méthodes associées sont documentés ici.
- cliP.execShowCmd (« cmd », return_type), où les cliPis l'objet de NxCliParser retourné par la méthode sdk.getCliParser (), cmd est la commande show que vous souhaitez exécuter encapsulé dans les guillemets, et return_type est le format des données à sortir. Le format des données peut être R_TEXT, R_JSON, ou R_XML. Cette méthode est documentée ici.

  Note: Les formats des données R_JSON et R_XML fonctionnent seulement si la commande prend en charge la sortie dans ces formats. Dans NX-OS, vous pouvez vérifier si une commande prend en charge la sortie dans un format des données particulier en sifflant la sortie au format des données demandé. Si la commande sifflée renvoie la sortie signicative, alors ce format des données est pris en charge. Par exemple, si vous exécutez show mac address-table dynamic | le json dans NX-OS renvoie la sortie JSON, puis le format des données R_JSON est aussi bien pris en charge dans NX-SDK.

- cliP.execConfigCmd(cmd_filename), où les cliPis l'objet de NxCliParser sont retournés par la méthode sdk.getCliParser (), et cmd_filenameis que le filepath absolu à un fichier qui contient des commandes a séparés par des lignes. Cette méthode renvoie une chaîne qui indique le succès de l'exécution de la commande - si le « SUCCÈS » est dans la chaîne, alors toutes les

commandes obtiennent avec succès exécuté. Autrement, la chaîne contient l'exception qui décrit pourquoi les commandes n'ont pas exécuté. Cette méthode est documentée ici.

Quelques méthodes facultatives qui peuvent être utiles sont :

- sdk.setAppDesc (« chaîne de description "), où les sdkis l'objet d'exemple SDK sont retournés par la méthode nx_sdk_py.NxSdk.getSdkInst (). Cette méthode place la description de l'application NX-SDK. La description est affichée dans le menu Help contextuel NX-OS accédé à par l'intermédiaire d'un point d'interrogation sur le CLI. Cette méthode est documentée ici. Par exemple, Transceiver_DOM.py nommé par anapplication, une description d'application des retours toutes les interfaces avec les émetteurs-récepteurs DOM-capables insérés apparaît dans l'aide contextuelle NX-OS comme suit :

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

- sdk.getAppName (), où les sdkis l'objet d'exemple SDK sont retournés par la méthode nx_sdk_py.NxSdk.getSdkinst (). Cette méthode renvoie le nom de l'application de python. Cette méthode est documentée ici.

Créez les commandes faites sur commande CLI

Dans une application de python avec l'utilisation de NX-SDK, des commandes faites sur commande CLI sont créées et définies dans la fonction de sdkThread. Il y a deux types de commandes : Commandes de commandes show, et de config.

1. Informations d'affichage de commandes show sur le périphérique, sa configuration, ou son environnement.
2. Les commandes de config changent la configuration de périphérique, qui modifie comment le périphérique réagit au réseau environnant.

Ces deux méthodes permettent la création des commandes show et des commandes de config respectivement :

- cliP.newShowCmd (« cmd_name », « syntaxe »), où les cliPis l'objet de NxCliParser sont revenus par la méthode, des cmd_nameis un nom unique pour la commande interne à l'application de la coutume NX-SDK, et des syntaxdescribes sdk.getCliParser () quels mots clé et paramètres peuvent être utilisés dans la commande. Cette méthode renvoie un objet de NxCliCmd, qui est documenté ici. Cette méthode est documentée ici.

  Note: Ce commande est sous-classe de cliP.newCliCmd (« cmd_type », « cmd_name », la « syntaxe ») où le cmd_type est CONF_CMD ou SHOW_CMD (selon le type de commande étant configurée), le cmd_name est un nom unique pour la commande interne à l'application de la coutume NX-SDK, et des syntaxdescribes quels mots clé et paramètres peuvent être utilisés dans la commande. Pour cette raison, l'APIDOCUMENTATION pour cette commande pourrait être plus utile pour la référence.

- cliP.newConfigCmd (« cmd_name », « syntaxe »), où les cliPis l'objet de NxCliParser sont revenus par la méthode, des cmd_nameis un nom unique pour la commande interne à l'application de la coutume NX-SDK, et des syntaxdescribes sdk.getCliParser () quels mots clé et paramètres peuvent être utilisés dans la commande. Cette méthode renvoie un objet de

NxCliCmd, qui est documenté ici. Cette méthode est documentée ici.

Note: Ce commande est sous-classe de **cliP.newCliCmd (« cmd_type », « cmd_name », la « syntaxe ») où** le cmd_type **est** CONF_CMD **ou** SHOW_CMD **(**il dépend du type de commande qui est configurée), le cmd_name **est un** nom unique pour la commande interne à l'application de la coutume NX-SDK, et des syntaxdescribes quels mots clé et paramètres peuvent être utilisés dans la commande. Pour cette raison, l'APIDOCUMENTATION pour cette commande pourrait être plus utile pour la référence.

Les deux types de commandes ont deux composants différents : Paramètres et mots clé :

1. **Les paramètres** sont des valeurs utilisées pour changer les résultats de la commande. Par exemple, dans le **show ip route 192.168.1.0 de** commande, il y a un mot clé d'**artère** suivi d'un paramètre qui reçoit une adresse IP, qui spécifie que seulement des artères qui incluent l'adresse IP fournie devraient être affichées.

2. **Les mots clé** changent les résultats de la commande par leur seule présence. Par exemple, dans la commande **show mac address-table dynamic**, il y a un mot clé **dynamique**, qui spécifie que seulement des adresses MAC dynamique-instruites doivent être affichées.

Les deux composants sont définis dans la syntaxe d'une commande NX-SDK quand elle est créée. Les méthodes pour l'objet de NxCliCmd existent pour modifier l'implémentation spécifique les des deux les composants.

- **nx_cmd.updateParam (« <parameter> », « help_str », type), où le** nx_cmd **est l'**objet de NxCliCmd retourné par le cliP.newShowCmd **() ou** les méthodes cliP.newConfigCmd **()**, <parameter> **est le** nom du paramètre de commande que qui peut être modifié inclus par les crochets angulaires (**<>**), des help_strsets l'aide-chaîne de la commande faite sur commande et est affiché dans le menu Help contextuel NX-OS accédé à par l'intermédiaire d'un point d'interrogation sur le CLI, et type est le **type du** paramètre. Les paramètres optionnels supplémentaires pour cette méthode sont disponibles et documentés ici. Types valides de Theseare pour les paramètres qui peuvent être spécifiés dans l'argument de type :

  P_INTEGER - spécifie n'importe quel entier**P_STRING** - spécifie n'importe quelle chaîne**P_INTERFACE** - spécifie n'importe quelle interface réseau**P_IP_ADDR** - spécifie n'importe quelle adresse IP**P_MAC_ADDR** - spécifie n'importe quelle adresse MAC**P_VRF** - spécifie n'importe quel exemple de Virtual Routing and Forwarding (VRF)
- **nx_cmd.updateKeyword (« mot clé », « help_str », is_key), où le** nx_cmd **est l'**objet de NxCliCmd retourné par le cliP.newShowCmd **() ou** les méthodes cliP.newConfigCmd **()**, mot clé **est le nom du** mot clé de commande que vous souhaitez modifier, des help_strsets l'aide-chaîne de la commande faite sur commande et est affiché en menu Help contextuel NX-OS accédé à par l'intermédiaire d'un point d'interrogation sur le CLI, et valeur booléenne **facultative** isan d'is_key cette des par défaut à faux. Si l'isTrue d'is_key**,** alors seule configuration créée par la commande avec l'utilisation de ce mot clé ne remplace pas l'autre seule configuration qui est créée par la commande. Si l'isFalse d'is_key, alors configuration créée par la commande avec l'utilisation de ce mot clé remplace l'autre configuration créée par la commande. Cette méthode est documentée ici.

Afin de visualiser des exemples de code des composants utilisés généralement de commande, visualisez la section d'exemples de commande CLI de coutume de ce document.

Après que des commandes faites sur commande CLI aient été créées, un objet de la classe de **pyCmdHandler** décrite plus tard dans ce document doit être créé et placé comme objet de gestionnaire de rappel CLI pour l'objet de NxCliParser. Ceci est expliqué comme suit :

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

Puis, l'objet de NxCliParser doit être ajouté à l'arborescence de programme d'analyse syntaxique NX-OS CLI de sorte que les commandes faites sur commande CLI soient visibles à l'utilisateur. Ceci est fait avec la **commande cliP.addToParseTree ()**, où les cliPis l'objet de NxCliParser sont retournés par la **méthode** sdk.getCliParser **()**.

exemple de fonction de sdkThread

Voici un exemple d'une fonction typique de sdkThread avec l'utilisation des fonctions expliquées précédemment. Cette fonction (notamment dans une application typique de python de coutume NX-SDK) utilise les variables globales, qui sont instanciées sur l'exécution de script.

```
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)
```

classe de pyCmdHandler

La classe de **pyCmdHandler** est héritée de la classe de **NxCmdHandler** dans la bibliothèque nx_sdk_py. La méthode de **postCliCb (individu, clicmd)** définie dans la classe de **pyCmdHandler** s'appelle toutes les fois que les commandes CLI qui proviennent d'une application NX-SDK. Ainsi, la méthode de **postCliCb (individu, clicmd)** est où vous définissez comment les commandes CLI de coutume définies dans la fonction de sdkThread se comportent sur le périphérique.

La fonction de **postCliCb (individu, clicmd)** renvoie une valeur booléenne. Si **vrai** est retourné, alors on le présume que la commande a été exécutée avec succès. **Faux** devrait être retourné si la commande n'exécutait pas avec succès pour une raison quelconque.

Le paramètre de **clicmd** utilise le nom unique qui a été défini pour la commande quand il a été créé dans la fonction de sdkThread. Par exemple, si vous créez une nouvelle **commande show** avec un nom unique de **show_xcvr_dom**, puis il est recommandé pour se rapporter à cette commande par le même nom dans la fonction de **postCliCb (individu, clicmd)** après que vous ayez vérifié pour voir si le nom de l'argument de clicmd contient le **show_xcvr_dom**. On l'explique ici :

```
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()
```

Si une commande qui utilise des paramètres est créée, alors vous devrez très probablement utiliser ces paramètres à un moment de la fonction de **postCliCb (individu, clicmd)**. Ceci peut être fait avec le **clicmd.getParamValue (« méthode de <parameter>")**, où le <parameter> est le nom du paramètre de commande que vous souhaitez obtenir la valeur d'inclus par les crochets angulaires (**<>**). Cette méthode est documentée ici. Cependant, la valeur est retournée par cette fonction doit être convertie en type du lequel vous avez besoin. Ceci peut être fait avec ces méthodes :

- **nx_sdk_py.void_to_int convertit** une valeur en type d'entier.
- **nx_sdk_py.void_to_string convertit** une valeur en type de chaîne.

La fonction de **postCliCb (individu, clicmd)** (ou toutes fonctions ultérieures) sera également typiquement où la sortie de commande show est imprimée à la console. Ceci est fait avec la **méthode clicmd.printConsole ()**.

> **Note**: Si l'application rencontre une erreur, une exception unhandled, ou autrement soudainement des sorties, alors la sortie de la **fonction clicmd.printConsole ()** ne sera pas affichée du tout. **Pour cette raison, la pratique recommandée quand vous mettez au point votre application de python est aux messages de débogage de log au Syslog avec l'utilisation d'un objet de NxTrace retourné par la méthode sdk.getTracer ()**, ou aux

déclarations d'impression d'utilisation et exécute l'application par l'intermédiaire de la **binaire de** /isan/bin/python du shell de coup.

exemple de classe de pyCmdHandler

Les services suivants de code comme exemple pour la classe de **pyCmdHandler** décrite ci-dessus. Ce code est pris à partir du fichier ip_move.py dans l'application du l'IP-mouvement NX-SDK disponible ici. Le but de cette application est de dépister le mouvement d'une adresse IP définie par l'utilisateur à travers des interfaces d'un périphérique de Nexus. Pour faire ceci, le code trouve l'adresse MAC de l'entrée d'adresse IP par le **paramètre de** <ip> dans le cache de l'ARP du périphérique, puis la vérifie qui VLAN que l'adresse MAC réside en utilisant la table de l'adresse MAC du périphérique. Utilisant ces MAC et VLAN, la **commande interne de <vlan> de VLAN de <mac> d'adresse de macdb du** show system **l2fm l2dbg** affiche une liste d'index d'interface SNMP que cette combinaison a été récemment associée avec. Le code utilise alors la **commande SNMP-ifindex d'interface d'**exposition de traduire les index récents d'interface SNMP en noms humain-lisibles d'interface.

```
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
```

Exemples faits sur commande de syntaxe de commande CLI

Cette section présente quelques exemples du paramètre de syntaxe utilisé quand vous créez des commandes faites sur commande CLI avec le **cliP.newShowCmd () ou les méthodes** cliP.newConfigCmd **()**, où le clip **est l'**objet de NxCliParser retourné par la **méthode** sdk.getCliParser **().**

> **Note**: Le soutien de la syntaxe avec l'ouverture et les parenthèses fermantes ("(» et ") ») est introduit dans NX-SDK v1.5.0, inclus dans la version NX-OS 7.0(3)I7(3). On le suppose que l'utilisateur utilise NX-SDK v1.5.0 quand ils suivent l'un de ces exemples donnés qui incluent la syntaxe utilisant l'ouverture et les parenthèses fermantes.

Mot clé simple

Cette commande show prend un MAC simple de mot clé et ajoute une chaîne des expositions toute d'aide les adresses MAC misprogrammed sur ce périphérique au mot clé.

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

Paramètre simple

Cette commande show prend un **<mac>** simple de paramètre. Les chevrons entourants autour du MAC de mot signifient que c'est un paramètre. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé pour définir le type du paramètre comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP.

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

Mot clé facultatif

Cette commande show peut sur option prendre un mot clé simple **[MAC]**. Les crochets entourants autour du MAC de mot signifient que ce mot clé est facultatif. Une chaîne des expositions toute d'aide les adresses MAC misprogrammed sur ce périphérique est ajoutée au mot clé.

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

Paramètre optionnel

Cette commande show peut sur option prendre un paramètre simple **[<mac>]**. Les crochets entourants autour du mot < MAC > signifient que ce paramètre est facultatif. Les chevrons entourants autour du MAC de mot signifient que c'est un paramètre. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé pour définir le type du paramètre comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP.

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

Mot clé et paramètre simples

Cette commande show prend un MAC simple de mot clé immédiatement suivi du paramètre **< mac-address >**. Les chevrons entourants autour du mac-address de mot signifient que c'est un paramètre. Une chaîne d'aide de l'adresse MAC de contrôle pour l'erreur de programmation est ajoutée au mot clé. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé afin de définir le type du paramètre comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP.

```
 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)
```

Plusieurs mots clé et paramètres

Cette commande show peut prendre un de deux mots clé, qui ont deux paramètres différents les suivant. Le premier MAC de mot clé a un paramètre de **< mac-address >**, et le deuxième IP de mot clé a un paramètre de **<ip address>**. Les chevrons entourants autour du mac-address et de l'IP address de mots signifient qu'ils sont des paramètres. Une chaîne d'aide de l'adresse MAC de contrôle pour l'erreur de programmation est ajoutée au mot clé de MAC. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre de **< mac-address >**. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé pour définir le type du **paramètre de** < mac-address **>** comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP. **Une chaîne d'aide de l'adresse IP de contrôle pour misprograming est ajoutée au mot clé d'IP. Une chaîne d'aide de l'adresse IP pour vérifier l'erreur de programmation est ajoutée au paramètre de <ip address>**. **. Le paramètre nx_sdk_py P_IP_ADDR** dans la **méthode** nx_cmd.updateParam **()** est utilisé pour définir le type du **paramètre de <ip address>** comme adresse IP, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP.

```
 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)
```

Plusieurs mots clé et paramètres avec le mot clé facultatif

Cette commande show peut prendre un de deux mots clé, qui ont deux paramètres différents les suivant. Le premier MAC de mot clé a un paramètre de **< mac-address >**, et le deuxième IP de mot clé a un paramètre de **<ip address>**. Les chevrons entourants autour du mac-address et de l'IP address de mots signifient qu'ils sont des paramètres. Une chaîne d'aide de l'adresse MAC de contrôle pour l'erreur de programmation est ajoutée au mot clé de MAC. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre de **< mac-address >**. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé pour définir le type du **paramètre de** < mac-address **>** comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP. **Une chaîne d'aide de l'adresse IP de contrôle pour misprograming est ajoutée au mot clé d'IP. Une chaîne d'aide de l'adresse IP pour vérifier l'erreur de programmation est ajoutée au paramètre de <ip address>**. **. Le paramètre nx_sdk_py P_IP_ADDR** dans la **méthode** nx_cmd.updateParam **()** est utilisé pour définir le type du **paramètre de <ip address>** comme adresse IP, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP. **Cette commande show pourrait sur option prendre un mot clé [clair]. Une aide que la chaîne efface des adresses détectées pour misprogrammed est ajoutée à ce mot clé facultatif.**

```
 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")
```

Plusieurs mots clé et paramètres avec le paramètre optionnel

Cette commande show peut prendre un de deux mots clé, qui ont deux paramètres différents les suivant. Le premier MAC de mot clé a un paramètre de **< mac-address >**, et le deuxième IP de mot clé a un paramètre de **<ip address>**. Les chevrons entourants autour du mac-address et de l'IP address de mots signifient qu'ils sont des paramètres. Une chaîne d'aide de l'adresse MAC de contrôle pour des misprogrammingis a ajouté au mot clé de MAC. Une chaîne d'aide de l'adresse MAC pour vérifier l'erreur de programmation est ajoutée au paramètre de **< mac-address >**. **. Le** paramètre **nx_sdk_py P_MAC_ADDR** dans la **méthode nx_cmd.updateParam ()** est utilisé pour définir le type du **paramètre de** < mac-address **>** comme adresse MAC, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP. **Une chaîne d'aide de l'adresse IP de contrôle pour misprograming est ajoutée au mot clé d'IP. Une chaîne d'aide de l'adresse IP pour vérifier l'erreur de programmation est ajoutée au paramètre de <ip address>**. **. Le paramètre nx_sdk_py P_IP_ADDR** dans la **méthode** nx_cmd.updateParam **()** est utilisé pour définir le type du **paramètre de <ip address>** comme adresse IP, qui empêche l'entrée d'utilisateur final d'un autre type, tel qu'une chaîne, un entier, ou une adresse IP. **Cette commande show pourrait sur option prendre un paramètre [<module>].** Des adresses claires d'une chaîne d'**aide** seulement sur le module spécifié est ajoutées à ce paramètre optionnel.

```
 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)
```

Débuggez une application de python avec NX-SDK

Une fois qu'une application de python NX-SDK a été créée, elle devra souvent être mise au point. NX-SDK vous informe au cas où il y aurait toutes les erreurs de syntaxe en votre code, mais parce que la bibliothèque du python NX-SDK utilise la GORGÉE pour traduire des bibliothèques de C++ aux bibliothèques de python, toutes exceptions produites au moment du résultat d'exécution de code dans un vidage de mémoire d'application semblable à ceci :

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

En raison de la nature ambiguë de ce message d'erreur, la pratique recommandée de mettre au point des applications de python est de se connecter des messages de débogage au Syslog avec l'utilisation d'un objet de NxTrace retourné par la **méthode sdk.getTracer ()**. **Ceci est expliqué comme suit :**

```
#! /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")
```

Si se connecter des messages de débogage au Syslog n'est pas une option, une approche alternative est d'utiliser des déclarations d'impression et d'exécuter l'application par l'intermédiaire de la binaire de **/isan/bin/python du** shell de coup. Cependant, la sortie de ces déclarations d'impression sera seulement visible quand exécuté de cette manière - exécuter l'application par le shell VSH ne produit aucune sortie. Un exemple d'utiliser des déclarations d'impression est affiché ici :

```
#! /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")
```

Déployez une application de python avec NX-SDK

Une fois qu'une application de python a été entièrement testée dans le shell de coup et est prête pour le déploiement, l'application devrait être installée dans la production par VSH. Ceci permet à l'application pour persister quand les recharges de périphérique ou quand le system switchover se produit dans un scénario de double-superviseur. Afin de déployer une application par VSH, vous devez créer un paquet rpm avec l'utilisation environnement NX-SDK et ENXOS SDK de construction. Cisco DevNet fournit une image de docker qui tient compte de la création facile de paquet rpm.

> **Note**: Pour l'assistance afin d'installer le docker sur votre système d'exploitation spécifique, référez-vous à la documentation relative à l'installation du docker.

Sur un hôte Docker-capable, tirez la version d'image de votre choix avec la **traction dockercisco/nxsdk de docker :** commande de **<tag>**, où le **<tag>** est la balise de la version d'image de votre choix. Vous pouvez visualiser les versions d'image disponibles et leurs étiquettes correspondantes ici. Ceci est expliqué avec la balise **v1** ici :

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

Mettez en marche un conteneur nommé **nxsdk** à partir de cette image et reliez à elle. Si la balise de votre choix est **v1** différent et de remplacement pour votre balise :

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

La mise à jour à la dernière version de NX-SDK et naviguent vers le répertoire **NX-SDK**, puis tirent les derniers fichiers du git :

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

Si vous devez utiliser une version plus ancienne de NX-SDK, vous pouvez copier le branchement NX-SDK avec l'utilisation de la balise respective de version avec le **clone de git -** commande de **https://github.com/CiscoDevNet/NX-SDK.git de** **v<version> b**, où le **<version>** est la version de NX-SDK vous avez besoin. Ceci est expliqué ici avec NX-SDK v1.0.0 :

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

Ensuite, transférez votre application de python dans le conteneur de docker. Il y a quelques différentes manières de faire ceci.

- Quittez le conteneur de docker (qui arrête le conteneur et exige de vous de le commencer une fois de plus), transférez l'application de python vers l'hôte de docker, puis employez la

commande de **cp de docker** afin de copier l'application de l'hôte dans le conteneur. Ceci est expliqué ici, dans la supposition que l'application de python a été transférée vers l'hôte de docker chez **/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
```

- Copiez le contenu de la demande de python dans votre presse-papier de système, puis collez le contenu dans un fichier créé dans le conteneur de docker avec l'utilisation du score.

Ensuite, utilisez le **script rpm_gen.py** situé dans /NX-SDK/scripts/ **afin de** créer un paquet rpm de l'application de python. **Ce script a un argument exigé, et deux ont exigé des Commutateurs :**

- Le nom du fichier de l'application de python. Par exemple, une application de python dans un fichier nommé **python_app.py** aurait comme conséquence un argument de python_app.py**. Ce** nom du fichier plus tard sera utilisé comme nom d'application pour NX-SDK, et également utilisé par NX-OS afin de se rapporter à des commandes créées par cette application.

  Note: Le nom du fichier n'a pas besoin de ne contenir aucune extension de fichier, telle que **.py**. Dans cet exemple, si le nom du fichier était **python_app** au lieu de **python_app.py, le** paquet rpm serait généré sans question.

- - Le commutateur s prend un argument pour le filepath absolu que cela mène à où le nom du fichier mentionné ci-dessus se trouve. Par exemple, si **python_app.py se trouve** dans /root/**, puis l'**argument correct serait - s **/root/.**
- - Le commutateur u indique que le nom du fichier de source est identique que le nom du fichier exécutable.

L'utilisation du **script rpm_gen.py** est expliquée ici.

```
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
```

Le filepath au paquet rpm est indiqué dans la ligne finale de la sortie de **script rpm_gen.py**. Ce fichier doit être copié hors fonction du conteneur de docker sur l'hôte de sorte qu'il puisse être transféré vers le périphérique de Nexus sur lequel vous souhaitez exécuter l'application. Après que vous quittiez le conteneur de docker, il peut être fait facilement avec le **<container> de** dockercp **: la commande de <host_filepath> de <container_filepath>**, où le <container> **est le** nom du conteneur de docker NX-SDK (dans ce cas, nxsdk), <container_filepath> **est le** plein filepath du paquet rpm à l'intérieur du conteneur (dans ce cas, /NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_64.rpm**),** et le <host_filepath> **est le** plein filepath sur notre hôte de docker vers où le

paquet rpm doit être transféré (dans ce cas, /root/). Cette commande est expliquée ici :

```
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
```

Transférez ce paquet rpm vers le périphérique de Nexus avec l'utilisation de votre méthode préférée du transfert de fichiers. Une fois que le paquet rpm est sur le périphérique, il doit être installé et lancé pareillement à un SMU. Ceci est expliqué comme suit, dans la supposition que le paquet rpm a été transféré vers le bootflash du périphérique.

```
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
```

> **Note**: Quand vous installez le paquet rpm avec la commande d'**install add**, incluez le périphérique de stockage et le nom du fichier précis du module. Quand vous lancez le paquet rpm après installation, n'incluez pas le périphérique de stockage et le nom du fichier - utilisez le nom du module lui-même. Vous pouvez vérifier le nom du paquet avec la commande de **show install inactive**.

Une fois que le paquet rpm est lancé, vous pouvez commencer l'application avec NX-SDK avec la commande de configuration de **<application-name> de service de nxsdk**, où le **<application-name>** est le nom du nom du fichier de python (et, ultérieurement, de l'application) qui a été défini quand le script rpm_gen.py a été utilisé plus tôt. Ceci est expliqué comme suit :

```
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
```

Vous pouvez vérifier que l'application est en hausse et a commencé à s'exécuter avec la commande **interne de service de nxsdk d'exposition** :

```
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
```

Vous pouvez également vérifier que les commandes faites sur commande CLI créées par cette application sont accessibles dans NX-OS :

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

Informations connexes

- NX-SDK GitHub
- Guide de programmabilité de la gamme 9000 NX-OS de Cisco Nexus, version 7.x
- Guide de programmabilité de la gamme 3000 NX-OS de Cisco Nexus, version 7.x
- Guide de programmabilité de la gamme 3500 NX-OS de Cisco Nexus, version 7.x
- La programmabilité et l'automatisation de réseau avec la gamme 9000 de Cisco Nexus commute le livre blanc
- Programmabilité et automatisation avec Cisco NX-OS ouvert (PDF)
- **Support et documentation techniques - Cisco Systems**