在Nexus 3000/9000交換機中開發、調試和部署NX-SDK Python應用程式

目錄

簡介

本文檔提供在Nexus 3000和Nexus 9000平台上使用Cisco NX-OS的Cisco NX-Software Development Kit(SDK)開發Python應用程式的工作流程。

必要條件

需求

本文件沒有特定需求。

採用元件

本文中的資訊係根據以下軟體和硬體版本：

- 本檔案使用NX-SDK v1.0.0和NX-SDK v1.5.0
- 從NX-OS版本7.0(3)I6(1)開始的Nexus 9000平台和從NX-OS版本7.0(3)I7(1)開始的Nexus 3000平台均可使用NX-SDK v1.0.0
- 從NX-OS版本7.0(3)I7(3)開始，NX-SDK v1.5.0可在Nexus 9000平台和Nexus 3000平台上使用

本文中的資訊是根據特定實驗室環境內的裝置所建立。文中使用到的所有裝置皆從已清除（預設）的組態來啟動。如果您的網路正在作用，請確保您已瞭解任何指令可能造成的影響。

背景資訊

Cisco NX-SDK允許開發可在Nexus 9000和Nexus 3000平台上的Cisco NX-OS本地運行的自定義應用程式。NX-SDK使客戶能夠建立自己的CLI命令和輸出、生成自定義的系統日誌以響應特定事件、流定製遙測等。

NX-SDK有一個C++ API，使用簡化包裝和介面生成器(SWIG)將其翻譯為其他語言。 這樣，客戶就可以使用他們選擇的任何語言使用NX-SDK。本文檔演示了在Python中實施常用的NX-SDK功能，並為客戶開發自己的NX-SDK Python應用程式提供工作流。

使用NX-SDK開發Python應用程式

啟用NX-SDK

要運行任何NX-SDK應用程式，必須首先在裝置上啟用NX-SDK功能：

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

建立Python檔案

可以使用NX-OS Bash shell建立和編輯Python檔案。要使用Bash shell，必須首先在裝置上啟用它：

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

輸入Bash shell並使用vi文本編輯器建立和編輯Python檔案：

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

> **附註**：最佳實踐是在**/isan/bin/目錄中建立Python**檔案。Python檔案需要執行許可權才能運行 — 請勿將Python檔案放在**/bootflash目錄**或其任何子目錄中。

> **附註**：不需要通過NX-OS建立和編輯Python檔案。開發人員可以使用他們的本地環境建立應用程式，並使用他們選擇的檔案傳輸協定將完成的檔案傳輸至裝置。但是，開發人員使用NX-OS實用程式調試和排除指令碼故障可能更加有效。

實施NX-SDK元件

建議開發人員從 Cisco DevNet NX-SDK GitHub上的customCliPyApp模板開始建立其NX-SDK Python應用程式。本文檔的這些部分引用了在此模板中使用其名稱的必要元件。

NX-SDK Python應用程式需要四個主要元件：

1. NX-SDK必須通過**import nx**_sdk_py語句**匯入到應用程式。**
2. 啟動NX-SDK應用程式並修改與應用程式相關的各種選項的函式(通常命名為**sdkThread**)。
3. 在sdkThread函式中建立自定義CLI命令以及定義自定義CLI命**令語**法。
4. 一個名為**pyCmdHandler**的類，其方法名為**postCliCb**，用於處理由NX-SDK應用程式新增的自定義CLI命令。

sdkThread函式

sdkThread函式可初始化NX-SDK應用程式、向NX-SDK新增功能並啟動NX-SDK應用程式。該函式不要求傳遞任何引數。所有Python NX-SDK應用程式都需要從nx_sdk_py庫中呼叫三種方法：

1. **nx_sdk_py.NxSdk.getSdkInst(len(sys.argv), sys.argv)返回SDK例項對**象，或返回None。如果返回SDK例項對象，則NX-SDK應用程式已成功註冊到NX-OS基礎架構。如果返回None，則在此註冊過程中會出現錯誤，並且裝置的系統日誌中出現錯誤日誌條目。此方法的文檔載於此處。

   附註：從NX-SDK v1.5.0開始，第三個布林引數可以傳遞到**NxSdk.getSdkInst**方法，該方法在為True時啟用高級異常，在為False時禁用高級異常。此方法的文檔如下。

1. **sdk.startEventLoop()**方法，其中**sdk**是由**nx_sdk_py.NxSdk.getSdkInst()方法返回的SDK例項對**象。此方法啟動NX-SDK應用程式，並允許其與NX-OS基礎設施互動。此方法的文檔如下。
2. **nx_sdk_py.NxSdk.__swig_destroy__(sdk)方**法，其中**sdk**是由前面說明的**nx_sdk_py.NxSdk.getSdkInst()方法返回的SDK例項**對象。放置在**sdkThread**函式末尾的此方法允許NX-SDK應用程式正常退出。

一些常用的方法包括：

- **sdk.getTracer()**，其中**sdk**是由**nx_sdk_py.NxSdk.getSdkInst()方法返回的SDK例項對**象。此方法返回一個NxTrace對象，該對象可用於生成自定義系統日誌，以及將事件和錯誤記錄到應用程式的事件歷史記錄中。自定義系統日誌將顯示在裝置的系統日誌中(通過**show logging logfile**命令可見)，而記錄到應用程式事件歷史記錄的事件可通過**show <application-name> nxsdk event-history events**或**show <application-name> nxsdk event-history errors**命令可見。此方法的文檔載於此處。在此記錄此方法返回的NxTrace對象及其相關方法。例如，可以通過**show Transceiver_DOM.py nxsdk event-history events**和**show Transceiver_DOM.py nxsdk event-history errors**命令檢視名為**Transceiver_DOM.py**的應用程式的事件歷史記錄。
- **sdk.getCliParser()**，其中**sdk**是由**nx_sdk_py.NxSdk.getSdkInst()方法返回的SDK例項對**象。此方法返回一個NxCliParser對象，該對象可用於執行已通過Python存在的CLI命令以及建立自定義CLI命令。此方法的文檔載於此處。在此記錄此方法返回的NxCliParser對象及其相關方法如下。

- **cliP.execShowCmd("cmd", return_type)**，其中**cliP**是通過**sdk.getCliParser()方法返回的NxCliParser對象，cmd**是要執行且封裝在引號中的show命令，而**return_type**是要輸出的資料格式。資料格式可以是**R_TEXT**、**R_JSON**或**R_XML**。此方法的文檔如下。

  附註：**R_JSON**和**R_XML**資料格式僅在命令支援以這些格式輸出時才有效。在NX-OS中，可以通過將輸出管道化為請求的資料格式，驗證命令是否支援特定資料格式的輸出。如果piped命令返回有意義的輸出，則支援該資料格式。例如，如果運行**show mac address-table dynamic | NX**-OS中的json返回JSON輸出，NX-SDK也支援R_JSON資料格式。

- **cliP.execConfigCmd(cmd_filename)**，其中**cliP**是由**sdk.getCliParser()方法返回的NxCliParser對象，cmd_filename**是包含以行分隔的命令的檔案的絕對檔案路徑。此方法返回一個字串，該字串表示命令執行成功 — 如果字串中包含「SUCCESS」，則成功執行所有命令。否則，字串包含描述命令無法執行的原因的異常。此方法的文檔載於此處。

一些有用的可選方法有：

- **sdk.setAppDesc('description string')**，其中**sdk**是由**nx_sdk_py.NxSdk.getSdkInst()方法返回的SDK例項對**象。此方法設定NX-SDK應用程式的說明。描述顯示在NX-OS上下文相關幫助選單中，可通過CLI上的問號進行訪問。此方法的文檔載於此處。例如，名為**Transceiver_DOM.py**的應用程式在NX-OS上下文相關幫助中顯示了Returns all interfaces with DOM-capable transceiver inserted，如下所示：

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

- **sdk.getAppName()**，其中**sdk**是由**nx_sdk_py.NxSdk.getSdkinst()方法返回的SDK例項對**象。此方法返回Python應用程式的名稱。此方法的文檔載於此處。

建立自定義CLI命令

在使用NX-SDK的Python應用程式中，可在sdkThread函式中建立和定義自定義CLI命令。有兩種型別的命令：**Show**命令和**Config命**令。

1. **Show命**令顯示有關裝置、其配置或其環境的資訊。
2. **Config**命令更改裝置的配置，從而修改裝置對周圍網路的反應。

這兩種方法分別允許建立show命令和config命令：

- **cliP.newShowCmd("cmd_name", "syntax")**，其中**cliP**是由**sdk.getCliParser()**方法返回的NxCliParser對象，**cmd_name**是自定義NX-SDK應用程式內部命令的唯一名稱，並且**syntax**描述命令中可以使用哪些關鍵字和引數。此方法返回一個NxCliCmd對象，該對象記錄在此處。此方法的文檔載於此處。

  附註：此命令是cliP.newCliCmd("cmd_type"、"cmd_name"、"syntax")的子類，其中cmd_type 是CONF_CMD 或**SHOW_CMD**（取決於所配置的命令型別），**cmd_name**是自定義NX-SDK應用程式內部命令的唯一名稱，並且**syntax**說明了命令中可以使用哪些關鍵字和引數。因此，此命令的API檔案可能更有助於參考。

- **cliP.newConfigCmd("cmd_name", "syntax")**，其中**cliP**是由**sdk.getCliParser()**方法返回的NxCliParser對象，**cmd_name**是自定義NX-SDK應用程式內部命令的唯一名稱，並且**syntax**描述命令中可以使用哪些關鍵字和引數。此方法返回NxCliCmd對象，該對象記錄在此處。此方法

的文檔如下。

附註：此命令是cliP.newCliCmd("cmd_type"、"cmd_name"、"syntax")的子類，其中cmd_type是CONF_CMD或**SHOW_CMD**（這取決於配置的命令型別），**cmd_name**是自定義NX-SDK應用程式內部命令的唯一名稱，並且**syntax**描述了可在命令中使用的關鍵字和引數。因此，此命令的API檔案可能更有助於參考。

兩種型別的命令都具有兩個不同的元件：引數和關鍵字：

1. **引數**是用於更改命令結果的值。例如，在命令**show ip route 192.168.1.0**中，有一個**route**關鍵字後跟一個接受IP地址的引數，該引數指定只顯示包含所提供IP地址的路由。

2. **關鍵字**僅通過命令存在狀態更改命令的結果。例如，在**show mac address-table dynamic**命令中，有一個**dynamic**關鍵字，該關鍵字指定僅顯示動態學習的MAC地址。

這兩個元件在建立時，均在NX-SDK命令的語法中定義。NxCliCmd對象的方法可用於修改這兩個元件的特定實現。

- **nx_cmd.updateParam（"<parameter>", "help_str"，型別），其中nx_cmd是cliP.newShowCmd()或cliP.newConfigCmd()方法返回的NxCliCmd對象，** <parameter>是可以修改的命令引數的名稱，用尖括弧(<>),**help_str設定自定義命令的幫助字串，並顯示在NX-OS上下文相關幫助選單中（可通過CLI上的問號訪問），type是type。引數。**此方法的其它可選引數可在此處提供並記錄。以下是可以在型別引數中指定的引數的有效型別：

  P_INTEGER — 指定任意整數**P_STRING — 指**定任意字串**P_INTERFACE — 指**定任何網路介面**P_IP_ADDR — 指**定任何IP地址**P_MAC_ADDR — 指**定任何MAC地址**P_VRF -**指定任何虛擬路由和轉發(VRF)例項
- **nx_cmd.updateKeyword("keyword", "help_str", is_key)，其中nx_cmd是cliP.newShowCmd()或cliP.newConfigCmd()方法返回的NxCliCmd對象，keyword**是要修改的命令關鍵字的名稱，**help_str設定自定義命令的幫助字串，顯示在通過CLI上的問號訪問的NX-OS上下文相關幫助選單中，而is_key為可選布林值變成False。**如果**is_key**為True，則使用此關鍵字的命令建立的唯一配置不會覆蓋該命令建立的其他唯一配置。如果**is_key**為False，則使用此關鍵字由命令建立的配置將覆蓋由命令建立的其它配置。此方法的文檔如下。

要檢視常用命令元件的代碼示例，請檢視本文檔的自定義CLI命令示例部分。

建立自定義CLI命令後，需要建立來自本文檔後面所述的**pyCmdHandler**類的對象，並將其設定為NxCliParser對象的CLI回撥處理程式對象。這一點表現如下：

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

然後，需要將NxCliParser對象新增到NX-OS CLI解析器樹中，以便使用者能夠看到自定義CLI命令。這通過**cliP.addToParseTree()**命令完成，其中**cliP**是由**sdk.getCliParser()**方法返回的NxCliParser對象。

sdkThread函式示例

以下是使用前面介紹的函式的典型sdkThread函式的示例。此函式（典型自定義NX-SDK Python應用程式中的函式等）使用全域性變數，這些變數在指令碼執行時例項化。

```
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)
```

pyCmdHandler類

pyCmdHandler類繼承自nx_sdk_py庫中的**NxCmdHandler**類。每次從NX-SDK應用程式發出的CLI命令時，都會呼叫**pyCmdHandler**類中定義的**postCliCb(self， clicmd)**方法。因此，**postCliCb(self， clicmd)**方法可用於定義在sdkThread函式中定義的自定義CLI命令在裝置上的行為。

postCliCb(self， clicmd)函式返回布林值。如果返回**True**，則假定命令已成功執行。**如**果命令因任何原因未成功執行，應返回false。

clicmd引數使用在sdkThread函式中建立命令時為該命令定義的唯一名稱。例如，如果使用**show_xcvr_dom**這一唯一名稱建立一個新的**show**命令，則建議在檢查了clicmd引數的名稱是否包含**show_xcvr_dom**後，在**postCliCb(self， clicmd)**函式中用相同的名稱引用此命令。這裡展示的是：

```
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()
```

如果建立了使用引數的命令，則在**postCliCb(self， clicmd)函式中的某個點最可能需要使用這些參數**。可以使用**clicmd.getParamValue("<parameter>")**方法完成此操作，其中**<parameter>**是要獲取用尖括弧(<>)括起來的值的命令引數的名稱。 此方法的文檔載[於此處](#)。但是，此函式返回的值需要轉換為所需的型別。可以使用以下方法完成此操作：

- **nx_sdk_py.void_to_int將值**轉換為整數型別。
- **nx_sdk_py.void_to_string將值**轉換為字串型別。

postCliCb(self， clicmd)函式（或任何後續函式）通常也是在將show命令輸出列印到控制檯的地方。這使用**clicmd.printConsole()方法完**成。

> **附註**：如果應用程式遇到錯誤、未處理的異常或者突然退出，則根本不顯示**clicmd.printConsole()**函式的輸出。因此，調試Python應用程式的最佳做法是使用由**sdk.getTracer()方法返回的NxTrace對象將調試消息記錄到系統日誌，或使用列印語句並通過Bash shell的/isan/bin/python** binary執行應用程式。

pyCmdHandler類示例

以下代碼用作上述**pyCmdHandler類**的示例。此代碼取自此處提供的[ip-movement NX-SDK應用程式中的ip_move.py檔案](#)。此應用程式的目的是跟蹤使用者定義IP地址在Nexus裝置介面間的移動。為此，代碼通過裝置ARP快取中的**<ip>引數查詢輸入的IP地址的MAC地址，然後使用裝置的MAC地址表驗證MAC地址所在的VLAN。**使用此MAC和VLAN，**show system internal l2fm l2dbg macdb address <mac> vlan <vlan>**命令會顯示最近與此組合關聯的SNMP介面索引清單。然後，代碼使用**show interface snmp-ifindex**命令將最近的SNMP介面索引轉換為易於識別的介面名稱。

```
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
```

自定義CLI命令語法示例

本節顯示使用**cliP.newShowCmd()**或**cliP.newConfigCmd()**方法建立自定義CLI命令時使用的語法引數的一些示例，其中**cliP**是通過**sdk.getCliParser()**方法返回的NxCliParser對象。

> **附註**：NX-SDK v1.5.0(包含在NX-OS版本7.0(3)I7(3)中)引入了對帶左括弧和右括弧("（"和"）")的語法的支援。 假設使用者遵循這些給定示例中的任何一個（包括使用左括弧和右括弧的語法）時，會使用NX-SDK v1.5.0。

單個關鍵字

此show命令採用單個關鍵字mac並將幫助字串Show all misprogrammed MAC address on this device新增到該關鍵字。

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

單引數

此show命令採用單個參**數<mac>**。詞mac的括弧表示這是一個引數。向引數新增用於檢查誤程式設計的MAC地址幫助字串。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將引

數的型別定義為MAC地址，這可以防止終端使用者輸入其他型別，如字串、整數或IP地址。

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

可選關鍵字

此show命令可以選擇採用單個關**鍵字[mac]**。mac一詞的括弧表示此關鍵字是可選的。幫助字串Show all misprogrammed MAC address on this device is added to the keyword。

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

可選引數

此show命令可以選擇採用單**個引數[<mac>]**。單詞<;mac >；的括弧表示此引數為可選引數。詞mac的括弧表示這是一個引數。向引數新增用於檢查誤程式設計的MAC地址幫助字串。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將引數的型別定義為MAC地址，這可以防止終端使用者輸入其他型別，如字串、整數或IP地址。

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

單個關鍵字和引數

此show命令採用單關鍵字mac，後跟引數**<mac-address>**。mac-address一詞的括括弧表示這是一個引數。將檢查MAC地址是否誤程式設計的幫助字串新增到關鍵字。向引數新增用於檢查誤程式設計的MAC地址幫助字串。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將引數的型別定義為MAC地址，這可以防止終端使用者輸入其他型別（如字串、整數或IP地址）。

```
 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)
```

多個關鍵字和引數

此show命令可採用兩個關鍵字之一，這兩個關鍵字後面都有兩個不同的引數。第一個關鍵字mac的引數**為<mac-address>**，第二個關鍵字ip的引數**為<ip-address>**。mac-address和ip-address兩詞旁的尖括弧表示它們是引數。在mac關鍵字中新增一個包含檢查MAC地址誤程式設計的幫助字串。用於檢查誤程式設計的MAC地址幫助字串被新增到**<mac-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將**<mac-address>**引數的型別定義為MAC地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。用於錯誤程式設計的檢查IP地址的幫助字串被新增到ip關鍵字。用於檢查誤程式設計的IP地址幫助字串將新增到**<ip-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_IP_ADDR.**引數用於將**<ip-address>**引數的型別定義為IP地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。

```
 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)
```

包含可選關鍵字的多個關鍵字和引數

此show命令可採用兩個關鍵字之一，這兩個關鍵字後面都有兩個不同的引數。第一個關鍵字mac的引數**<mac-address>**，第二個關鍵字ip的引數**為<ip-address>**。mac-address和ip-address兩詞旁的尖括弧表示它們是引數。在mac關鍵字中新增一個包含檢查MAC地址誤程式設計的幫助字串。用於檢查誤程式設計的MAC地址幫助字串被新增到**<mac-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將**<mac-address>**引數的型別定義為MAC地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。用於錯誤程式設計的檢查IP地址的幫助字串被新增到ip關鍵字。用於檢查誤程式設計的IP地址幫助字串將新增到**<ip-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_IP_ADDR.**引數用於將**<ip-address>**引數的型別定義為IP地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。此show命令可能會選用關鍵字[clear]。在此可選關鍵字中新增一個幫助字串Clears addresses detected to misprogrammed to this optional keyword。

```
 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")
```

包含可選引數的多個關鍵字和引數

此show命令可採用兩個關鍵字之一，這兩個關鍵字後面都有兩個不同的引數。第一個關鍵字mac的引數**<mac-address>**，第二個關鍵字ip的引數**為<ip-address>**。mac-address和ip-address兩詞旁的尖括弧表示它們是引數。將包含檢查MAC地址是否存在錯誤程式設計的幫助字串新增到mac關鍵字。用於檢查誤程式設計的MAC地址幫助字串被新增到**<mac-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_MAC_ADDR.**引數用於將**<mac-address>**引數的型別定義為MAC地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。用於錯誤程式設計的檢查IP地址的幫助字串被新增到ip關鍵字。用於檢查誤程式設計的IP地址幫助字串將新增到**<ip-address>參**數中。**nx_cmd.updateParam()**方法中的**nx_sdk_py.P_IP_ADDR.**引數用於將**<ip-address>**引數的型別定義為IP地址，這將阻止終端使用者輸入其他型別（如字串、整數或IP地址）。此show命令可能會選擇採**用引數[<module>]**。將只清除指定模組上的地址的幫助字串新增到此可選引數中。

```
 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)
```

使用NX-SDK調試Python應用程式

建立NX-SDK Python應用程式後，通常需要對其進行調試。NX-SDK會在代碼中存在語法錯誤時通知您，但由於Python NX-SDK庫利用SWIG將C++庫轉換為Python庫，因此代碼執行時遇到的任何異常都會導致類似於以下內容的應用程式核心轉儲：

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

由於此錯誤消息的性質不明確，調試Python應用程式的最佳做法是使用由**sdk.getTracer()方法返回的NxTrace對象將調試消息記錄到系統日**志。這一點表現如下：

```
#! /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")
```

如果無法將調試消息記錄到系統日誌，另一種方法是使用print語句並通過Bash shell的**/isan/bin/python二進位制檔案執行應用程**序。但是，只有以這種方式執行時，才能看到這些print語句的輸出 — 通過VSH shell運行應用程式不會產生任何輸出。此處顯示了使用列印語句的示例：

```
#! /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")
```

使用NX-SDK部署Python應用程式

一旦Python應用程式在Bash外殼中經過完全測試並準備好進行部署，則該應用程式應通過VSH安裝到生產環境中。這樣，當裝置重新載入或在雙Supervisor場景中發生系統切換時，該應用程式可以持續運行。為了通過VSH部署應用程式，您需要使用NX-SDK和ENXOS SDK生成環境建立RPM包。Cisco DevNet提供一個Docker映像，可以輕鬆建立RPM包。

附註：要獲得幫助，以便將Docker安裝在特定作業系統上，請參閱Docker的安裝文檔。

在支援Docker的主機上，使用**docker pull dockercisco/nxsdk:<tag>** 命令獲取您選擇的映像版本，其中**<tag>** 是您選擇的映像版本的標籤。您可以在此處檢視可用的影象版本及其相應的標籤。以下通過**v1**標籤進行演示：

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

從此映像啟動名為**nxsdk**的容器，並將其附加到該容器。如果您選擇的標籤不同，請用**v1**代替您的標籤：

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

更新到最新版本的NX-SDK並導航到**NX-SDK**目錄，然後從git提取最新檔案：

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

如果您需要使用較舊版本的NX-SDK，則可以使用**git clone -b v<version>** https://github.com/CiscoDevNet/NX-SDK.git命令使用相應的版本標籤來克隆NX-SDK分支，其中<version>是您需要的NX-SDK的版本。這一點在NX-SDK v1.0.0中進行了演示：

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

接下來，將您的Python應用程式傳輸到Docker容器。有幾種不同的方法可以做到這一點。

- 退出Docker容器（停止該容器並要求您再次啟動它），將Python應用程式傳輸到Docker主機，然後使用**docker cp**命令將應用程式從主機複製到容器。此處演示了這一點，假設已將Python應用程式傳輸到**/app/python_app.py**上的Docker主機。

```
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
```

- 將Python應用程式的內容複製到系統剪貼簿中，然後使用vim將內容貼上到Docker容器中建立的檔案中。

接下來，使用**/NX-SDK/scripts/**中的**rpm_gen.py**指令碼，以便從Python應用程式建立RPM包。此指令碼有一個必需的引數和兩個必需的開關：

- Python應用程式的檔名。例如，名為**python_app.py**的檔案中的Python應用程式將導致引數**python_app.py**。此檔名稍後將用作NX-SDK的應用程式名稱，並且NX-OS也將使用此檔名引用此應用程式建立的命令。

  附註：檔名無需包含任何副檔名，例如**.py**。在本例中，如果檔名是**python_app**而不是**python_app.py**，則生成RPM包時不會出現問題。

- -s開關接受一個絕對檔案路徑的引數，該引數指向上述檔名的位置。例如，如果**python_app.py**位於**/root/**中，則正確的引數為**-s /root/**。
- -u開關指示源檔名與執行檔名相同。

此處演示了**rpm_gen.py** 指令碼的用法。

```
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
```

rpm_gen.py指令碼輸出的最後一行中指示**了RPM程式包的**檔案路徑。必須將此檔案從Docker容器複製到主機上，以便將其傳輸到要在其上運行應用程式的Nexus裝置。退出Docker容器後，可以使用docker cp <container>:<container_filepath> <host_filepath> 命令輕鬆完成此操作，其中<container>是NX-SDK Docker容器的名稱（在本例中為nxsdk），<container_filepath>是容器中RPM包的完整檔案路徑(在本例中為**/NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_4.rpm**)和<host_filepath ath>是Docker主機上要將RPM包傳輸到的完整檔案路徑(在本例中為**/root/**)。 此命令如下所示：

```
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
```

使用首選的檔案傳輸方法將此RPM軟體包傳輸到Nexus裝置。RPM軟體包在裝置上後，必須像安裝SMU一樣安裝並啟用它。假設已將RPM軟體包轉移至裝置的bootflash ，演示如下。

```
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
```

> **附註**：使用**install add**命令安裝RPM軟體包時，請包含儲存裝置和軟體包的準確檔名。安裝後啟用RPM軟體包時，不要包含儲存裝置和檔名 — 使用軟體包本身的名稱。您可以使用**show install inactive**命令驗證軟體包名稱。

啟用RPM軟體包後，您可以使用**nxsdk service <application-name>**配置命令使用NX-SDK啟動應用程式，其中**<application-name>**是早期使用rpm_gen.py指令碼時定義的Python檔名（隨後是應用程式）的名稱。這一點表現如下：

```
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
```

您可以使用**show nxsdk internal service**命令驗證應用程式是否已啟動並已開始運行：

```
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
```

您還可以驗證此應用程式建立的自定義CLI命令是否可在NX-OS中訪問：

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

相關資訊

- NX-SDK GitHub
- Cisco Nexus 9000系列NX-OS可程式設計性指南7.x版
- Cisco Nexus 3000系列NX-OS可程式設計性指南7.x版
- Cisco Nexus 3500系列NX-OS可程式設計性指南7.x版
- Cisco Nexus 9000系列交換機的網路可程式設計性和自動化白皮書
- Cisco Open NX-OS的可程式設計性和自動化(PDF)
- 技術支援與文件 - Cisco Systems