Desenvolva, debugar e distribua o aplicativo do pitão NX-SDK no nexo 3000/9000 de Switches

Índice

Introdução

Este original fornece uns trabalhos para o Desenvolvimento de aplicações do pitão com o jogo do desenvolvimento do NX-software de Cisco (SDK) com o Cisco NX-OS do uso no nexo 3000 e no nexo 9000 Plataformas.

Pré-requisitos

Requisitos

Não existem requisitos específicos para este documento.

Componentes Utilizados

As informações neste documento são baseadas nestas versões de software e hardware:

- Este original utiliza NX-SDK v1.0.0 e NX-SDK v1.5.0
- NX-SDK v1.0.0 pode ser utilizado na plataforma do nexo 9000 que partem da liberação NX-OS 7.0(3)I6(1) e na plataforma do nexo 3000 que parte da liberação NX-OS 7.0(3)I7(1)
- NX-SDK v1.5.0 pode ser utilizado na plataforma do nexo 9000 e na plataforma do nexo 3000 que partem da liberação NX-OS 7.0(3)I7(3)

As informações neste documento foram criadas a partir de dispositivos em um ambiente de laboratório específico. Todos os dispositivos usados neste original começaram com uma configuração cancelada (do padrão). Se sua rede está viva, certifique-se de que você compreende o impacto potencial do comando any.

Informações de fundo

Cisco NX-SDK permite o desenvolvimento dos aplicativos feitos sob encomenda que podem ser executado nativamente no Cisco NX-OS no nexo 9000 e no nexo 3000 Plataformas. NX-SDK oferece a capacidade para que os clientes criem seus próprios comandos CLI e as saídas, gerenciem Syslog personalizados em resposta aos eventos específicos, telemetria costurada córrego, e muito mais.

NX-SDK tem a corrente alternada ++ API, que é traduzida em outras línguas com o uso do envoltório e do gerador simplificados da relação (ENTORNAR). Isto permite o cliente utilizar NX-SDK em toda a língua de sua escolha. Este original demonstra a aplicação de funções comuns NX-SDK no pitão, assim como fornece uns trabalhos para que os clientes desenvolvam seus próprios aplicativos do pitão NX-SDK.

Desenvolva um aplicativo do pitão com NX-SDK

Permita NX-SDK

Para que todo o aplicativo NX-SDK seja executado, a característica NX-SDK deve primeiramente ser permitida no dispositivo:

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

Crie um arquivo do pitão

Um arquivo do pitão pode ser criado e editado com o uso do shell da festança NX-OS. Para que o shell da festança seja usado, deve primeiramente ser permitido no dispositivo:

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

Incorpore o shell da festança e use o editor de texto vi a fim criar e editar o arquivo do pitão:

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

Note: O melhor prática é criar arquivos do pitão no diretório de **/isan/bin/**. Permissões da

execução da necessidade dos arquivos do pitão a fim ser executado - não coloque arquivos do pitão no diretório de **/bootflash** ou em algum de seus sub-diretórios.

Note: Não se exige criar e editar arquivos do pitão com NX-OS. O colaborador pode criar o aplicativo usando seu ambiente local e transferir os arquivos terminados ao dispositivo usando um protocolo de transferência de arquivo de sua escolha. Contudo, pôde ser mais eficiente para que o colaborador debugue e pesquise defeitos seu script usando utilidades NX-OS.

Componentes do implementar NX-SDK

Recomenda-se para que os colaboradores comecem a criar seu aplicativo do pitão NX-SDK do molde do customCliPyApp em Cisco DevNet NX-SDK GitHub. Estas seções deste original referem componentes necessários com o uso de seus nomes dentro deste molde.

Há quatro componentes principais necessários para um aplicativo do pitão NX-SDK:

1. NX-SDK deve ser importado no aplicativo através da indicação **nx_sdk_py da importação**.
2. Uma função (**sdkThread** tipicamente Nomeado) essa começa o aplicativo NX-SDK e altera as várias opções relativas ao aplicativo.
3. A criação de comandos CLI feitos sob encomenda assim como a definição da sintaxe de comando CLI feita sob encomenda dentro do **sdkThread** funcionam.
4. Uma classe nomeada **pyCmdHandler** com um método nomeou o **postCliCb**, que processa os comandos CLI feitos sob encomenda adicionados pelo aplicativo NX-SDK.

função do sdkThread

A função do sdkThread inicializa, adiciona a funcionalidade a, e começa o aplicativo NX-SDK. A função não exige nenhuns parâmetros ser passada nela. Todos os aplicativos do pitão NX-SDK exigem três métodos da biblioteca nx_sdk_py ser chamados:

1. **nx_sdk_py.NxSdk.getSdkInst (len(sys.argv), sys.argv) retorna um** objeto do **exemplo** SDK, ou não retorna nenhuns. **Se um objeto do exemplo SDK é retornado, a seguir o aplicativo NX-SDK registrou-se com sucesso com a infraestrutura NX-OS. Se nenhum é retornado, a seguir os erros ocorrem na altura deste processo de registro, e uma entrada de log de erros aparece no Syslog do dispositivo. Este método é documentado aqui.**

   Note: Partindo de NX-SDK v1.5.0, um terceiro parâmetro booleano pode ser passado no **método NxSdk.getSdkInst**, que permite exceções avançadas quando verdadeiro e desabilita exceções avançadas quando falso. Este método é documentado aqui.

1. **método sdk.startEventLoop ()**, onde os sdkis o objeto do exemplo SDK retornaram pelo **método nx_sdk_py.NxSdk.getSdkInst (). Este método** começa o aplicativo NX-SDK e permite que interaja com a infraestrutura NX-OS. Este método é documentado aqui.

2. método do __swig_destroy__(sdk) nx_sdk_py.NxSdk., onde o sdk é o objeto do exemplo SDK retornado pelo método nx_sdk_py.NxSdk.getSdkInst () explicado previamente. Este método colocado na extremidade dos functionallows do sdkThread para a saída graciosa do aplicativo NX-SDK.

Alguns métodos de uso geral incluem:

- sdk.getTracer (), onde os sdkis o objeto do exemplo SDK retornaram pelo método nx_sdk_py.NxSdk.getSdkInst (). Este método retorna um objeto de NxTrace que possa ser usado para gerar Syslog feitos sob encomenda, assim como registra eventos e erros à história do evento de aplicativo. Os Syslog feitos sob encomenda aparecerão no Syslog do dispositivo (visível através do comando de registro do arquivo histórico da mostra) quando os eventos registrados à história do evento de aplicativo forem visíveis através dos eventos da evento-história do nxsdk do <application-name> da mostra ou mostrarão comandos dos erros da evento-história do nxsdk do <application-name>. Este método é documentado aqui. O NxTrace objectreturned através deste método e seus métodos associados são documentados aqui. Por exemplo, a vista youcan a história do evento de um aplicativo nomeou eventos da nxsdkevent-história da mostra Transceiver_DOM.py do throughthe Transceiver_DOM.py e comandos dos erros da evento-história do nxsdk da mostra Transceiver_DOM.py.
- sdk.getCliParser (), onde os sdkis o objeto do exemplo SDK retornaram pelo método nx_sdk_py.NxSdk.getSdkInst (). Este método retorna um objeto de NxCliParser, que possa ser usado para executar os comandos CLI que já existem através do pitão assim como criam comandos CLI feitos sob encomenda. Este método é documentado aqui. O NxCliParserobject retornado através deste método e seus métodos associados são documentados aqui.
- cliP.execShowCmd ("cmd", return_type), onde os cliPis o objeto de NxCliParser retornado pelo método sdk.getCliParser (), cmd são o comando show que você deseja executar encapsulado na cotação - marcas, e return_type é o formato de dados a ser output. O formato de dados pode ser R_TEXT, R_JSON, ou R_XML. Este método é documentado aqui.

  Note: Os formatos de dados R_JSON e R_XML trabalham somente se o comando apoia a saída naqueles formatos. Em NX-OS, você pode verificar se um comando apoia a saída em um formato de dados particular conduzindo a saída ao formato de dados pedido. Se o comando conduzido retorna a saída significativa, a seguir esse formato de dados está apoiado. Por exemplo, se você executa a tabela de endereço do Mac da mostra dinâmica | o json em NX-OS retorna a saída JSON, a seguir o formato de dados R_JSON é apoiado em NX-SDK também.

- cliP.execConfigCmd(cmd_filename), onde os cliPis o objeto de NxCliParser retornaram pelo método sdk.getCliParser (), e cmd_filenameis que o caminho do arquivo absoluto a um arquivo que contivesse comandos separou por linhas. Este método retorna uma corda que indique o sucesso da execução do comando - se o "SUCESSO" está na corda, a seguir os comandos all obtêm com sucesso executados. Se não, a corda contém a exceção que descreve porque os comandos não executaram. Este método é documentado aqui.

Alguns métodos opcionais que podem ser úteis são:

- sdk.setAppDesc ("description string "), onde os sdkis o objeto do exemplo SDK retornaram pelo método nx_sdk_py.NxSdk.getSdkInst (). Este método ajusta a descrição do aplicativo NX-SDK. A descrição é indicada no menu de ajuda contexto-sensível NX-OS alcançado através de um ponto de interrogação no CLI. Este método é documentado aqui. Por exemplo, o anapplication nomeou Transceiver_DOM.py, uma descrição do aplicativo dos retornos que

todas as relações com os transceptores DOM-capazes introduzidos parece na ajuda contexto-sensível NX-OS como segue:

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

- **sdk.getAppName (), onde os** sdkis o objeto do exemplo SDK retornaram pelo **método nx_sdk_py.NxSdk.getSdkinst (). Este método** retorna o nome do aplicativo do pitão. Este método é documentado aqui.

Crie comandos CLI feitos sob encomenda

Em um aplicativo do pitão com o uso de NX-SDK, os comandos CLI feitos sob encomenda são criados e definidos dentro da função do sdkThread. Há dois tipos de comandos: **Comandos show**, e **comandos config**.

1. Exibir informação dos **comandos show** sobre o dispositivo, sua configuração, ou seu ambiente.
2. **Os comandos config** mudam a configuração de dispositivo, que altera como o dispositivo reage à rede circunvizinha.

Estes dois métodos permitem a criação dos comandos show e dos comandos config respectivamente:

- **cliP.newShowCmd ("cmd_name", "sintaxe"), onde os** cliPis o objeto de NxCliParser retornaram pelo **método**, por cmd_nameis um nome exclusivo para o comando internal ao aplicativo do costume NX-SDK, e por syntaxdescribes sdk.getCliParser **()** que palavras-chaves e parâmetros podem ser usados no comando. Este método retorna um objeto de NxCliCmd, que seja documentado aqui. Este método é documentado aqui.

  Note: Este comando é subclasse de **cliP.newCliCmd ("cmd_type", "cmd_name", a "sintaxe") onde** o cmd_type **é** CONF_CMD **ou** SHOW_CMD **(segundo o** tipo de comando que está sendo configurado), o cmd_name **é um** nome exclusivo para o comando internal ao aplicativo do costume NX-SDK, e syntaxdescribes que palavras-chaves e parâmetros podem ser usados no comando. Devido a isto, o APIDOCUMENTATION para este comando pôde ser mais útil para a referência.

- **cliP.newConfigCmd ("cmd_name", "sintaxe"), onde os** cliPis o objeto de NxCliParser retornaram pelo **método**, por cmd_nameis um nome exclusivo para o comando internal ao aplicativo do costume NX-SDK, e por syntaxdescribes sdk.getCliParser **()** que palavras-chaves e parâmetros podem ser usados no comando. Este método retorna um objeto de NxCliCmd, que seja documentado aqui. Este método é documentado aqui.

  Note: Este comando é subclasse de **cliP.newCliCmd ("cmd_type", "cmd_name", a "sintaxe") onde** o cmd_type **é** CONF_CMD **ou** SHOW_CMD **(**depende do tipo de comando que é configurado), o cmd_name **é um** nome exclusivo para o comando internal ao aplicativo do costume NX-SDK, e syntaxdescribes que palavras-chaves e parâmetros podem ser usados no comando. Devido a isto, o APIDOCUMENTATION para este comando pôde ser mais útil para a referência.

Ambos os tipos de comandos têm dois componentes diferentes: Parâmetros e palavras-chaves:

1. **Os parâmetros** são valores usados para mudar os resultados do comando. Por exemplo, no comando show ip route **192.168.1.0**, há uma palavra-chave da **rota** seguida por um parâmetro que aceite um IP address, que especifique que somente as rotas que incluem o IP address fornecido devem ser mostradas.

2. **As palavras-chaves** mudam os resultados do comando com sua presença apenas. Por exemplo, na **tabela de endereço do** comando show mac **dinâmica**, há uma **palavra-chave dinâmica**, que especifique que somente os endereços dinâmico-instruídos MAC devem ser indicada.

Ambos os componentes estão definidos na sintaxe de um comando NX-SDK quando é criada. Os métodos para o objeto de NxCliCmd existem para alterar a aplicação específica de ambos os componentes.

- **nx_cmd.updateParam ("<parameter>", "help_str", tipo), onde o** nx_cmd **é o** objeto de NxCliCmd retornado pelo cliP.newShowCmd **() ou** os métodos cliP.newConfigCmd **()**, <parameter> **é o** nome do comando parameter que que pode ser alterado incluido pelos suportes angulares (<>), help_strsets a ajuda-corda do comando custom e é indicado no menu de ajuda contexto-sensível NX-OS alcançado através de um ponto de interrogação no CLI, e o tipo é o **tipo do** parâmetro. Os parâmetros opcionais adicionais para este método são disponíveis e documentados aqui. Tipos válidos de Theseare para os parâmetros que podem ser especificados no tipo argumento:

  P_INTEGER - especifica qualquer inteiro**P_STRING** - especifica toda a corda**P_INTERFACE** - especifica toda a interface de rede**P_IP_ADDR** - especifica todo o IP address**P_MAC_ADDR** - especifica todo o MAC address**P_VRF** - especifica todo o exemplo do roteamento virtual e da transmissão (VRF)
- **nx_cmd.updateKeyword ("palavra-chave", "help_str", is_key), onde o** nx_cmd **é o** objeto de NxCliCmd retornado pelo cliP.newShowCmd **() ou** pelos métodos cliP.newConfigCmd **()**, palavra-chave **é o nome do** palavra-chave de comando que você deseja alterar, de help_strsets a ajuda-corda do comando custom e é indicado no menu de ajuda contexto-sensível NX-OS alcançado através de um ponto de interrogação no CLI, e no valor de booleano **opcional** isan do is_key esse padrões a falso. Se o isTrue do is_key**,** então configuração exclusiva criada pelo comando com o uso desta palavra-chave não overwrite a outra configuração exclusiva que está criada pelo comando. Se o isFalse do is_key, então configuração criada pelo comando com o uso desta palavra-chave overwrites a outra configuração criada pelo comando. Este método é documentado aqui.

A fim ver exemplos do código de componentes do comando utilizado comumente, veja a seção feita sob encomenda dos exemplos do comando CLI deste original.

Depois que os comandos CLI feitos sob encomenda foram criados, um objeto da classe do **pyCmdHandler** descrita mais tarde neste original precisa de ser criado e ajustado como o objeto do alimentador da rechamada CLI para o objeto de NxCliParser. Isto é demonstrado como segue:

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

Então, o objeto de NxCliParser precisa de ser adicionado à árvore do parser NX-OS CLI de modo que os comandos CLI feitos sob encomenda sejam visíveis ao usuário. Isto é feito com o **comando cliP.addToParseTree ()**, aonde os cliPis o objeto de NxCliParser retornaram pelo **método** sdk.getCliParser **()**.

exemplo da função do sdkThread

Está aqui um exemplo de uma função típica do sdkThread com o uso das funções explicadas previamente. Esta função (entre outros dentro de um aplicativo típico do pitão do costume NX-SDK) utiliza as variáveis globais, que instantiated na execução do 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 do pyCmdHandler

A classe do **pyCmdHandler** é herdada da classe de **NxCmdHandler** dentro da biblioteca nx_sdk_py. O método do **postCliCb (auto, clicmd)** definido dentro da classe do **pyCmdHandler** é chamado sempre que os comandos CLI que originam de um aplicativo NX-SDK. Assim, o método do **postCliCb (auto, clicmd)** é onde você define como os comandos CLI feitos sob encomenda definidos dentro da função do sdkThread se comportam no dispositivo.

A função do **postCliCb (auto, clicmd)** retorna um valor de booleano. Se **verdadeiro** é retornado, a seguir presume-se que o comando esteve executado com sucesso. **Falso** deve ser retornado se o comando não executou com sucesso por qualquer razão.

O parâmetro do **clicmd** usa o nome exclusivo que foi definido para o comando quando foi criado na função do sdkThread. Por exemplo, se você cria um **comando show** novo com um nome exclusivo do **show_xcvr_dom**, a seguir é recomendado referir este comando pelo mesmo nome na função do **postCliCb (auto, clicmd)** depois que você verificou para ver se o nome do argumento do clicmd contém o **show_xcvr_dom**. Demonstra-se aqui:

```
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()
```

Se um comando que utilize parâmetros é criado, a seguir você precisará muito provavelmente de usar aqueles parâmetros a dada altura da função do **postCliCb (auto, clicmd)**. Isto pode ser feito com o **clicmd.getParamValue (do "método <parameter>")**, onde o <parameter> é o nome do comando parameter que você deseja obter ao valor de incluido pelos suportes angulares (**<>**). Este método é documentado aqui. Contudo, o valor retornou por esta função precisa de ser convertido ao tipo que você precisa. Isto pode ser feito com estes métodos:

- **nx_sdk_py.void_to_int converte** um valor a um tipo do inteiro.
- **nx_sdk_py.void_to_string converte** um valor a um tipo da corda.

A função do **postCliCb (auto, clicmd)** (ou algumas funções subsequentes) igualmente serão tipicamente o lugar onde o show command output (resultado do comando show) é imprimido ao console. Isto é feito com o **método clicmd.printConsole ()**.

> **Note**: Se o aplicativo encontra um erro, uma exceção não manejada, ou de outra maneira de repente umas saídas, a seguir a saída da **função clicmd.printConsole ()** não estará indicada de todo. **Por este motivo, o melhor prática quando você debuga seu aplicativo do pitão é ao log debuga mensagens ao Syslog com o uso de um objeto de NxTrace retornado pelo método sdk.getTracer ()**, ou indicações da cópia do uso e executa o aplicativo através do **binário de** /isan/bin/python do shell da festança.

exemplo da classe do pyCmdHandler

O seguinte código serve como um exemplo para a classe do **pyCmdHandler** descrita acima. Este código é tomado do arquivo ip_move.py no aplicativo do IP-movimento NX-SDK disponível aqui. A

finalidade deste aplicativo é seguir o movimento de um IP address definido pelo utilizador através das relações de um dispositivo do nexo. Para fazer isto, o código encontra o MAC address do IP address entrado com o **parâmetro do** <ip> dentro do cache ARP do dispositivo, a seguir verifica que VLAN que o MAC address reside em usar a tabela do MAC address do dispositivo. Usando estes MAC e VLAN, do **<mac> interno do endereço do macdb do sistema o comando vlan do <vlan> l2fm l2dbg da** mostra indica uma lista de índices de interface snmp que esta combinação tem sido associada recentemente com. O código usa então o **comando do SNMP-ifIndex da relação da** mostra traduzir índices de interface snmp recentes em nomes humano-legíveis da relação.

```
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
```

Exemplos feitos sob encomenda da sintaxe de comando CLI

Esta seção apresenta alguns exemplos do parâmetro da sintaxe usado quando você cria comandos CLI feitos sob encomenda com o **cliP.newShowCmd () ou os métodos** cliP.newConfigCmd **()**, onde o grampo **é o** objeto de NxCliParser retornado pelo **método** sdk.getCliParser **().**

> **Note**: O apoio para a sintaxe com abertura e parênteses de fechamento (“(” e ")”) é introduzido em NX-SDK v1.5.0, incluído na liberação NX-OS 7.0(3)I7(3). Supõe-se que o usuário utiliza NX-SDK v1.5.0 quando segue qualquens um exemplos dados que incluem a sintaxe que utiliza a abertura e que fecha parênteses.

Única palavra-chave

Este comando show toma um único Mac da palavra-chave e adiciona toda do ajudante uma corda das mostras endereços misprogrammed MAC neste dispositivo à palavra-chave.

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

Único parâmetro

Este comando show toma um único **<mac> do** parâmetro. Os suportes de ângulo encerrando em torno do Mac da palavra significam que este é um parâmetro. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado para definir o tipo do parâmetro como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address.

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

Palavras-chave opcionais

Este comando show pode opcionalmente tomar um único **[mac] da** palavra-chave. Os suportes encerrando em torno do Mac da palavra significam que esta palavra-chave é opcional. Toda do ajudante uma corda das mostras endereços misprogrammed MAC neste dispositivo é adicionada à palavra-chave.

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

Parâmetro opcional

Este comando show pode opcionalmente tomar um único **[<mac>] do** parâmetro. Os suportes encerrando em torno da palavra **<** Mac **>** significam que este parâmetro é opcional. Os suportes de ângulo encerrando em torno do Mac da palavra significam que este é um parâmetro. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado para definir o tipo do parâmetro como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address.

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

Únicos palavra-chave e parâmetro

Este comando show toma um único Mac da palavra-chave seguido imediatamente pelo parâmetro **< endereço MAC >**. Os suportes de ângulo encerrando em torno do endereço MAC da palavra significam que este é um parâmetro. Uma corda do ajudante do MAC address da verificação para o erro de programação é adicionada à palavra-chave. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado a fim definir o tipo do parâmetro como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address.

```
 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)
```

Palavras-chaves e parâmetros múltiplos

Este comando show pode tomar uma de duas palavras-chaves, ambo têm dois parâmetros diferentes que seguem os. O primeiro Mac da palavra-chave tem um parâmetro **< do endereço MAC >**, e o segundo IP da palavra-chave tem um parâmetro do **<ip-address>**. Os suportes de ângulo encerrando em torno do endereço MAC e do IP address das palavras significam que são parâmetros. Uma corda do ajudante do MAC address da verificação para o erro de programação é adicionada à palavra-chave do Mac. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro **< do endereço MAC >**. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado para definir o tipo do **parâmetro <** do endereço MAC **>** como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address. **Uma corda do ajudante do IP address da verificação para misprograming é adicionada à palavra-chave IP. Uma corda do ajudante do IP address a verificar para ver se há o erro de programação é adicionada ao parâmetro do <ip-address>**. **. O parâmetro nx_sdk_py P_IP_ADDR** no método nx_cmd.updateParam **()** é usado para definir o tipo do **parâmetro do <ip-address>** como um IP address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address.

```
 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)
```

Palavras-chaves e parâmetros múltiplos com palavras-chave opcionais

Este comando show pode tomar uma de duas palavras-chaves, ambo têm dois parâmetros diferentes que seguem os. O primeiro Mac da palavra-chave tem um parâmetro **< do endereço MAC >**, e o segundo IP da palavra-chave tem um parâmetro do **<ip-address>**. Os suportes de ângulo encerrando em torno do endereço MAC e do IP address das palavras significam que são parâmetros. Uma corda do ajudante do MAC address da verificação para o erro de programação é adicionada à palavra-chave do Mac. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro **< do endereço MAC >**. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado para definir o tipo do **parâmetro <** do endereço MAC **>** como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address. **Uma corda do ajudante do IP address da verificação para misprograming é adicionada à palavra-chave IP. Uma corda do ajudante do IP address a verificar para ver se há o erro de programação é adicionada ao parâmetro do <ip-address>. . O parâmetro nx_sdk_py P_IP_ADDR** no método nx_cmd.updateParam **()** é usado para definir o tipo do **parâmetro do <ip-address>** como um IP address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address. **Este comando show pôde opcionalmente tomar um [clear] da palavra-chave. Os endereços dos espaços livres de uma corda do ajudante detectados para ser misprogrammed sãos a estas palavras-chave opcionais.**

```
 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")
```

Palavras-chaves e parâmetros múltiplos com parâmetro opcional

Este comando show podido tomar uma de duas palavras-chaves, ambo têm dois parâmetros diferentes que seguem os. O primeiro Mac da palavra-chave tem um parâmetro **< do endereço MAC >**, e o segundo IP da palavra-chave tem um parâmetro do **<ip-address>**. Os suportes de ângulo encerrando em torno do endereço MAC e do IP address das palavras significam que são parâmetros. Uma corda do ajudante do MAC address da verificação para os misprogrammingis adicionados à palavra-chave do Mac. Uma corda do ajudante do MAC address a verificar para ver se há o erro de programação é adicionada ao parâmetro **< do endereço MAC >**. **. O** parâmetro **nx_sdk_py P_MAC_ADDR no método nx_cmd.updateParam ()** é usado para definir o tipo do **parâmetro <** do endereço MAC **>** como um MAC address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address. **Uma corda do ajudante do IP address da verificação para misprograming é adicionada à palavra-chave IP. Uma corda do ajudante do IP address a verificar para ver se há o erro de programação é adicionada ao parâmetro do <ip-address>. . O parâmetro nx_sdk_py P_IP_ADDR** no método nx_cmd.updateParam **()** é usado para definir o tipo do **parâmetro do <ip-address>** como um IP address, que impeça a entrada do utilizador final de um outro tipo, tal como uma corda, um inteiro, ou um IP address. **Este comando show pôde opcionalmente tomar um [<module>] do parâmetro.** Os endereços claros de uma corda do **ajudante** somente no módulo especificado são adicionados a este parâmetro opcional.

```
 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)
```

Debugar um aplicativo do pitão com NX-SDK

Uma vez que um aplicativo do pitão NX-SDK foi criado, deverá frequentemente ser debugado. NX-SDK informa-o que caso que há todos os erros de sintaxe em seu código, mas porque a biblioteca do pitão NX-SDK utiliza ENTORNAR para traduzir bibliotecas do C++ às bibliotecas do pitão, algumas exceções encontradas na altura do resultado da execução de código em um dump principal do aplicativo similar a este:

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

Devido à natureza ambígua deste Mensagem de Erro, o melhor prática debugar aplicativos do pitão é registrar debuga mensagens ao Syslog com o uso de um objeto de NxTrace retornado pelo **método sdk.getTracer ()**. **Isto é demonstrado como segue:**

```
#! /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")
```

Se registrando debugar mensagens ao Syslog não é uma opção, um método alternativo é usar indicações da cópia e executar o aplicativo através do binário de **/isan/bin/python do** shell da festança. Contudo, a saída destas indicações da cópia será somente visível quando executada desse modo - executar o aplicativo através do shell VSH não produz nenhuma saída. Um exemplo de utilizar indicações da cópia é mostrado aqui:

```
#! /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")
```

Distribua um aplicativo do pitão com o NX-SDK

Uma vez que um aplicativo do pitão foi testado inteiramente no shell da festança e está pronto para o desenvolvimento, o aplicativo deve ser instalado na produção com VSH. Isto permitir o aplicativo persistir quando os recarregamentos de dispositivo ou quando o switchover do sistema ocorrer em uma encenação do supervisor dual. A fim distribuir um aplicativo com VSH, você precisa de criar um pacote RPM com o uso ambiente de uma construção NX-SDK e ENXOS SDK. Cisco DevNet fornece uma imagem do estivador que permita a criação fácil do pacote RPM.

> **Note**: Para o auxílio a fim instalar o estivador em seu sistema operacional específico, refira a documentação de instalação do estivador.

Em um host Estivador-capaz, puxe a versão da imagem de sua escolha com a **tração dockercisco/nxsdk do estivador:** comando do **<tag>**, onde o **<tag>** é a etiqueta da versão da imagem de sua escolha. Você pode ver as versões da imagem disponíveis e suas etiquetas correspondentes aqui. Isto é demonstrado com a etiqueta **v1** aqui:

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

Ligue um recipiente nomeado **nxsdk d**esta imagem e anexe-lhe. Se a etiqueta de sua escolha é diferente, substitua **v1** para sua etiqueta:

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

A atualização à versão a mais atrasada de NX-SDK e navega ao diretório **NX-SDK**, a seguir puxa os arquivos os mais atrasados do git:

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

Se você precisa de usar uma versão mais velha de NX-SDK, você pode clonar o ramo NX-SDK com o uso da etiqueta da versão respectiva com o **clone do git - o** comando de **https://github.com/CiscoDevNet/NX-SDK.git do v<version> b**, onde o **<version>** é a versão de NX-SDK você precisa. Isto é demonstrado aqui com NX-SDK v1.0.0:

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

Em seguida, transfira seu aplicativo do pitão ao recipiente do estivador. Há algumas maneiras diferentes de fazer isto.

- Retire o recipiente do estivador (que para o recipiente e o exige o começar uma vez mais), transfira o aplicativo do pitão ao host do estivador, a seguir use o comando do **cp do estivador** a fim copiar o aplicativo do host ao recipiente. Isto é demonstrado aqui, sob a suposição que o aplicativo do pitão esteve transferido ao host do estivador em **/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
```

- Copie os índices do aplicativo do pitão em sua prancheta do sistema, a seguir cole os índices em um arquivo criado no recipiente do estivador com o uso do vim.

Em seguida, use o **script rpm_gen.py** situado em /NX-SDK/scripts/ **a fim** criar um pacote RPM do aplicativo do pitão. **Este script tem um argumento requerido, e dois exigiram o Switches:**

- O nome de arquivo do aplicativo do pitão. Por exemplo, um aplicativo do pitão em um arquivo nomeado **python_app.py** conduziria a um argumento de python_app.py**. Este** nome de arquivo será usado mais tarde como o nome do aplicativo para NX-SDK, e igualmente usado por NX-OS a fim referir os comandos criados por este aplicativo.

  Note: O nome de arquivo não precisa de conter nenhuma extensões de arquivo, tal como **.py**. Neste exemplo, se o nome de arquivo era **python_app** em vez de **python_app.py, o** pacote RPM seria gerado sem uma edição.

- - O interruptor s toma um argumento para o caminho do arquivo que absoluto aquele conduz a onde o nome de arquivo acima mencionado é encontrado. Por exemplo, se **python_app.py** é ficado situado em /root/**, a seguir o** argumento correto seria - s **/root/.**
- - O interruptor u indica que o nome do arquivo de origem é o mesmo que o nome de arquivo executável.

O uso do **script rpm_gen.py** é demonstrado aqui.

```
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
```

O caminho do arquivo ao pacote RPM é indicado na linha final da saída do **script rpm_gen.py**. Este arquivo deve ser copiado fora do recipiente do estivador no host de modo que se possa transferir ao dispositivo do nexo que você deseja executar sobre o aplicativo. Depois que você retira o recipiente do estivador, pode ser feito facilmente com o **<container> do** dockercp**: o comando do <host_filepath> do <container_filepath>**, onde o <container> **é o** nome do recipiente do estivador NX-SDK (neste caso, nxsdk), **<container_filepath> é o** caminho do arquivo completo do pacote RPM dentro do recipiente (neste caso, /NX-SDK/rpm/RPMS/test_python_app-1.0-1.0.0.x86_64.rpm**),** e o **<host_filepath> é o** caminho do arquivo completo em nosso host do estivador a onde o pacote RPM deve ser transferida (neste caso, /root/**).** Este comando é demonstrado aqui:

```
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
```

Transfira este pacote RPM ao dispositivo do nexo com o uso de seu método preferido de transferência de arquivo. Uma vez que o pacote RPM está no dispositivo, deve ser instalado e ativado similarmente a um SMU. Isto é demonstrado como segue, sob a suposição que o pacote RPM esteve transferido ao bootflash do dispositivo.

```
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**: Quando você instala o pacote RPM com o **comando add da instalação**, inclua o dispositivo de armazenamento e o nome de arquivo exato do pacote. Quando você ativa o pacote RPM após a instalação, não inclua o dispositivo de armazenamento e o nome de arquivo - use o nome do pacote próprio. Você pode verificar que o nome do pacote com a **mostra instala o** comando **inativo**.

Uma vez que o pacote RPM é ativado, você pode começar o aplicativo com o NX-SDK com o comando configuration do **<application-name> do serviço do nxsdk**, onde o **<application-name>** é o nome do nome de arquivo do pitão (e, subseqüentemente, do aplicativo) que foi definido quando o script rpm_gen.py foi usado mais cedo. Isto é demonstrado como segue:

```
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
```

Você pode verificar que o aplicativo é ascendente e começou ser executado com o **comando service interno do nxsdk da mostra**:

```
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
```

Você pode igualmente verificar que os comandos CLI feitos sob encomenda criados por este aplicativo são acessíveis em NX-OS:

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

Informações Relacionadas

- NX-SDK GitHub
- Guia do Programmability do 9000 Series NX-OS do nexo de Cisco, liberação 7.x
- Guia do Programmability do 3000 Series NX-OS do nexo de Cisco, liberação 7.x
- Guia do Programmability do 3500 Series NX-OS do nexo de Cisco, liberação 7.x
- Programmability e automatização da rede com White Paper dos 9000 Series Switch do nexo de Cisco
- Programmability e automatização com Cisco NX-OS aberto (PDF)
- Suporte técnico & documentação - Cisco Systems