noc_doc Documentation

Release 0.1

A

October 17, 2016

Оглавление

Contents:

Overview

1.1 Overview

NOC is the scalable, high-performance and open-source OSS system for ISP, service and content providers.

1.1.1 Quick links

-
- Source
- StackOverflow
- Wiki

1.1.2 Key Features

Major features are:

- Telecom-specific modules
  - Network Inventory
  - IP Address Management
  - VLAN Management
  - Service Activation
  - Configuration Management
  - Fault Management
  - DNS Provisioning
  - Performance Management
  - Peering Management
  - Knowledge Base
- Vendor-agnostic
  - Over 50 supported vendors, from CPEs to core MPLS routers
- Battle-proven

    - Used by small and large companies worldwide
- Leverages mature and proven open-source technologies »
    - Python
    - PostgreSQL
    - MongoDB
    - Django
    - ExtJS
- Integration »
    - REST/JSON API
    - Python API
    - Established professional community »
    - DevTeam and Community
    - BSD License

1.1.3 Limitations

- No supporting Windows

1.1.4 Microservices

Activator

Add API for running scripts on devices

Settings: * Thread * Count Service Name: Ping

Pinging devices and RTT

Идея микросервисов в разделении части функционала на слабосвязанные модули, которые общаются между собой при помощи API. Это облегчает масштабирование и поддержку. На данный момент в NOC реализованы следующие сервисы:

Installation

2.1 Manual Installation

There are following components you need to install before running Moira microservices:

1. golang_ version 1.5 or higher
2. redis_ database version 2.8 or higher
3. python version 2.7
4. web server e.g. nginx

2.1.1 Установка башни

Установка вручную

Докер контейнер

:current_tarball:‘z‘

2.1.2 Настройка башни

Основной интерфей доступен по порту 8888 и адресу, на которой запущена башня.

На скриншоте видим основное меню и панель инструментов. Пункты меню:

1. Environments - основная рабочая область. В ней создаются Окружения и настраиваются их параметры. Также из него выполняются действия над ними. Для выпонения настроек в последующих пунктах необходимо выбрать окружение.
2. Datacenters - Здесь настраиваются Датацентры. Каждая нода (хост), относится к какому-либо датацентру. Из настроек доступно только указание прокси сервера
3. Pools - здесь создаются Пулы. Часть сервисов HOKa позволяют работать в связке. Связок может быть несколько и по ним можно распределять оборудование (делается в настройках HOKa)
4. Nodes - здесь создаются ноды (хосты). На них будут устанавливаться компоненты HOKa. Имя ноды (хоста) соответствует его hostname.
5. Services - в данном меню происходит настройка сервисов HOKa (количество, пути и пр.) и место их размещения.

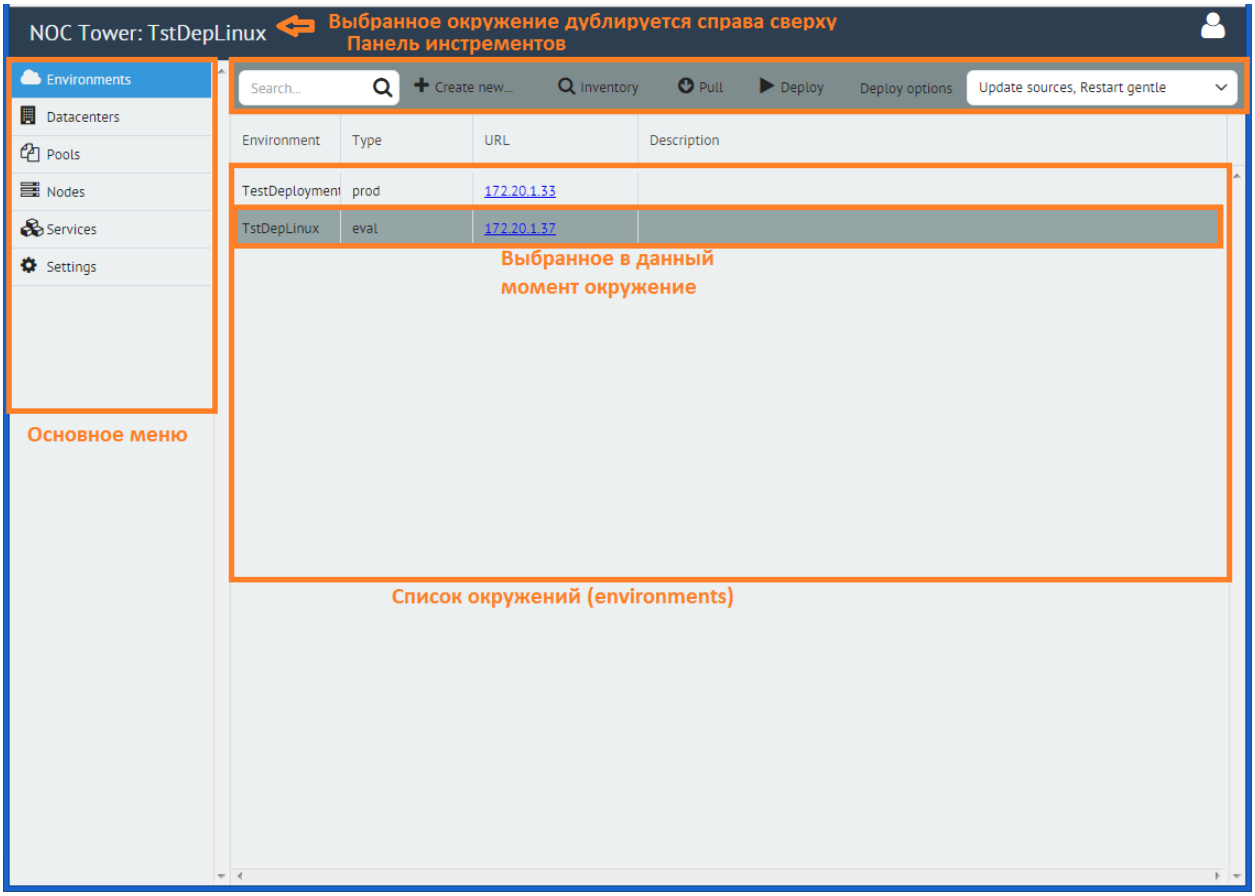

Рис. 2.1: Основной экран Tower

6. Settings - указывается адрес репозитория НОКа (совпадает с хостом, на котором установлена башня). Для завершения настройки в этой вкладке необходимо нажать Save.

Tip: Все настройки, выполняемые в пунктах меню относятся к выбранному окружению. Т.е. Ноды, добавленные в одном окружении будут недоступны для другого.

На панели инструкментов доступны действия:

- Create new создать новое окружение
- Inventory - показывает текстовый вариант всех введённых настроек для окружения
- Pull - подтянуть последние изменения из репозитария НОКа
- Deploy - развернуть НОК согласно настройкам.
- Deploy options - позволяет выбрать операции, которые выполняются во время деплоя. По умолчанию необходимо выбрать Install Everything (Установить всё)

Создаём новое окружение, начинаем заполнять поля (кнопка Create New)

На экране создания окружения заполняем поле имени (используем только буквенно-цифровые имена) и Host (указываем либо IP либо hostname) по этому адресу будет развёрнут веб-интерфейс.

Attention: После добавления окружения необходимо нажать кнопку Pull

Далее:

1. Создаём датацентр. В нём,в слуае необходимости, заполняем поле Proxy.
2. Добавляем ноды, в настройках указываем тип (FreeBSD, Linux), IP адрес (в формате IP), и имя пользователя, из под которого будет проходить развёртывание компонентов.
3. Переходим в раздел Settings. Он разделён на несколько подразделов. Первым идёт Global - в нём сосредоточены основные компоненты НОКа и стороннее ПО, необходимо для функционирования. Дополнительные разделы соответствуют числу Пулов, созданных в разделе Pools. Поначалу Pool только 1 - default. В пулах находятся сервисы, работающие в связке с оборудованием.

Note: Подробнее о назначении каждого из сервисов можно почитать тут!

Note: Более подробно все настройки этого раздела освещены в руководстве!

Начинаем заполнять раздел. Необходимо пройти по всем сервисам и проставить их количество, которое будет установлено. На данный момент есть подводный камень. Если просто ставить цифры и перескакивать с сервиса на сервис - то настройки сохранить не удастся. Получится нечто такое:

В такой ситуации необходимо обновить окно браузера и повторить процедуру. По окончании нажимаем Save, после обновляем окно браузера. Должно получиться как-то так:

Последним шагом переходим в меню Settings и нажимем кнопку Save.

Последним шагом возвращаемся в меню Environments, проверяем что в Deploy options выставлено Install Everythong, нажимаем кнопку Deploy и ждём.

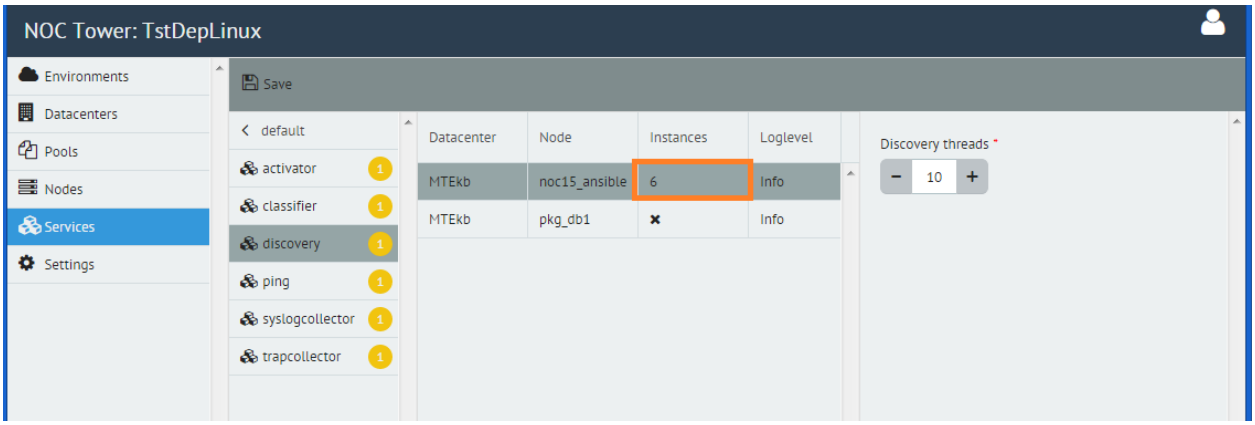

Рис. 2.2: Ошибка при настройке сервисов

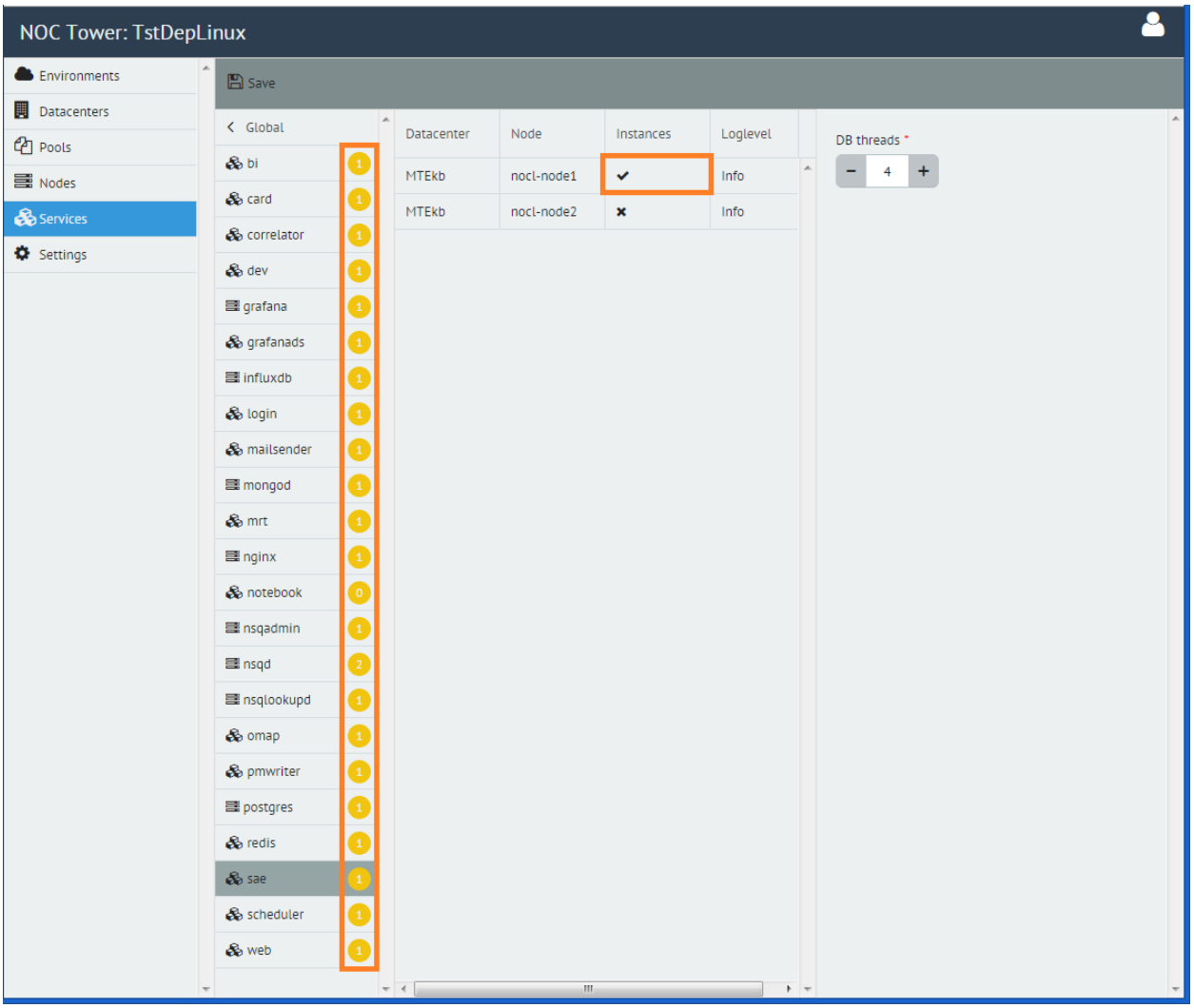

Рис. 2.3: Сервисы успешно настроены

2.1.3 Деплой НОКа

Сама по себе процедура деплоя предельно автоматизирована. Но может понадобиться отследить какие-то моменты. Саму процедуру Деплоя проводит Ansible. Это система управления конфигурацией, работающая на основе Playbook (рецептов). Сама башня выполняет настроечную функцию, а по кнопке деплой запускается Ансибль и по Playbook, выполняет действия, которые при другом раскладе пришлось выполнять вручную.

Лог действий он выводит в консоль. Жёлтые - это внесённые изменения, Зелёные - это прошло успешно, Красные - сбой. Если сбой критический, то деплой остановится, если нет - продолжится. При возникновении ошибок, рекомандуется поискать решение в FAQ или спросить в чатике.

ГЛАВА 3

Users's Guide

Documentation

4.1 Developers Guide

Оглавление

4.1.1 Разработка профиля для оборудования

Введение

Любое устройство, с которым взаимодействует NOC, требует соответствущего профиля. В нём описываются особенности работы с оборудованием: * команды, в случае CLI или SNMP OID'ы в случае работы по SNMP * форматы данных * поддерживаемые оборудованием технологии и возможности

Профиль Это компонент NOC'а, предназанченная для обеспечение взаимодействия между компонентами.

Формально, можно описать Профиль как посредника. Он принимает необработанный поток данных от оборудования и преобразует его в данные, которые передаются NOC'у для последующей обработки и сохранения в БД. Профили жёстко привязаны к конкретному програмному обеспечению, используемому на оборудовании. Хотя, имеется возможность учёта версии ПО (она передаётся профилю при вызове) и другие параметры при вызове тех или иных команд, при существенных изменениях рекомендуется написать другой профиль, т.к. большое количество ветвлений усложняет поддержку.

Структура и особенности работы профиля

- Пишутся на языке программирования Python;
- Подгружаются автоматически, при старте системы;
- Не хранят состояния после завершения
- Ограничение на время работы профиля устанавливается таймаутом !!!!
- Ограничение на использование модулей не устанавливается
- Вызываются с заданным периодом (в зависимости от назначения)
- Профиль передаёт информацию в сторону NOC'а, реализуя один из доступных интерфейсов.

Note: Интерфейс описывает формат и состав данных, которые необходимо передать в сторону NOC'а.

Взаимодействие с NOC'ом

Профиль взаимодействут с NOC'ом 2 путями:

1. NOC предоставляет методы для взаимодействия с оборудованием (подключение по CLI, запрос информации по SNMP) и для часто встречающихся операций (н-р преобразование MAK адресов, IP адресов). Практически любой из них можно переопределить в своём профиле, под особенности определённого оборудования. Например под уникальное представление MAC адреса или IP префикса. Преимущество этого подхода - при обращении к оборудованию NOC уже будет знать об этих особенностях.

В разделе Взаимодействие с оборудованием будут рассмотрены основные методы. Полный перечень доступен в Приложении Базовый класс скрипта. Для описания параметров взаимодействия с оборудованием (например, таймауты, настройки telnet подключения, и др.) существует профиль оборудования. Он находится в модуле __init__.py и является наследников класса noc.core.profile.base.BaseProfile . Полный перечень доступных для изменения параметров доступен в Приложении Базовый класс профиля .

1. Для передачи результатов применяются интерфейсы.

   Интерфейс - это специальная сущность NOC'а, предназанченная для обеспечение взаимодействия между компонентами.

При описании интерфейса указывается формат и структура данных, которые необходимо передавать. Также указывается обязательность/необязательность определённых полей. Проще всего, его можно представить в виде канала, с одной стороны которого 1 компонент ПО, а с другого 2. Передача осуществляется в одну сторону. Если данные не прошли проверку - возникает исключение.

В качестве примера возьмём интерфейс noc.sa.interfaces.igetversion.IGetVersion

class noc.sa.interfaces.igetversion.IGetVersion

Returns Parameters in dict

Vendor StringParameter:

Platform StringParameter

Version StringParameter

Attributes DictParameter

Return type DictParameter

Как можно увидедеть, в нём указывается тип и структура данных. Для успешной передачи необходимо соответствовать заданным требованиям. В данном случае, для успешной передачи нам необходимо сформировать словарь с ключами vendor, version, platform, их значения, это текстовые поля (StringParameter()).Также, есть необязательный параметр - словарь attributes. Перечень ключей, доступных для передачи не задан. Это означает, что разработчик может самостоятельно выбрать что в нём передавать. В итоге, нам необходимо передать следующую структуру:

```
{
"vendor": "Cisco",
"version": "12.4(5)",
"platform": "IOS",
"attributes":
        {
        "image": "image.bin",
        "type": "type1",
        "count": 2
        }
}
```

Схема вызова профиля

Работа с профилем происходит путём вызова модулей. В случае Python, модуль представляет собой файл. Имена модулей являются зарезервированными для выполнения конкретных задач и реализации интерфейсов. Эти методы должны содержать класс Script(BaseScript) и, в нём, метод execute. Для модулей, которые не зарезервированными метода execute() может не быть. Например, для сбора информации о версии ПО, платформы профиль должен содержать модуль get_version (т.е. в папке должен лежать файл get_version.py). При выполнении задачи сбора версии, NOC обращается к профилю и пытается вызвать метод execute() в модуле get_version. В случае успеха, выполняются перечисленные там действия и, через оператор return этого модуля, возвращается результат.

> Attention: Профиль считывается при старте NOC'а и кэшируется. Поэтому, для того чтобы, NOC восприняла изменения необходим перезапуск. Исключением является использование отладки через ./noc script. См. Отладка .

Состав профиля

Профиль состоит из файлов (в терминологии Python - Модулей, набор файлов в одной директории - это Пакет), называемых скриптами, написанными на ЯП Python (в терминологии Python они называются модулями). Часть имён файлов являются зарезервированными и вызываются при работе конкретных задач. Запросов вызова скрипта имеет вид <folder1>.<folder2>.<script_name>.py.

Note: Термины Пакет, Модуль, Метод это термины языка программирования Python. Рекомендуется изучить литературу по нему.

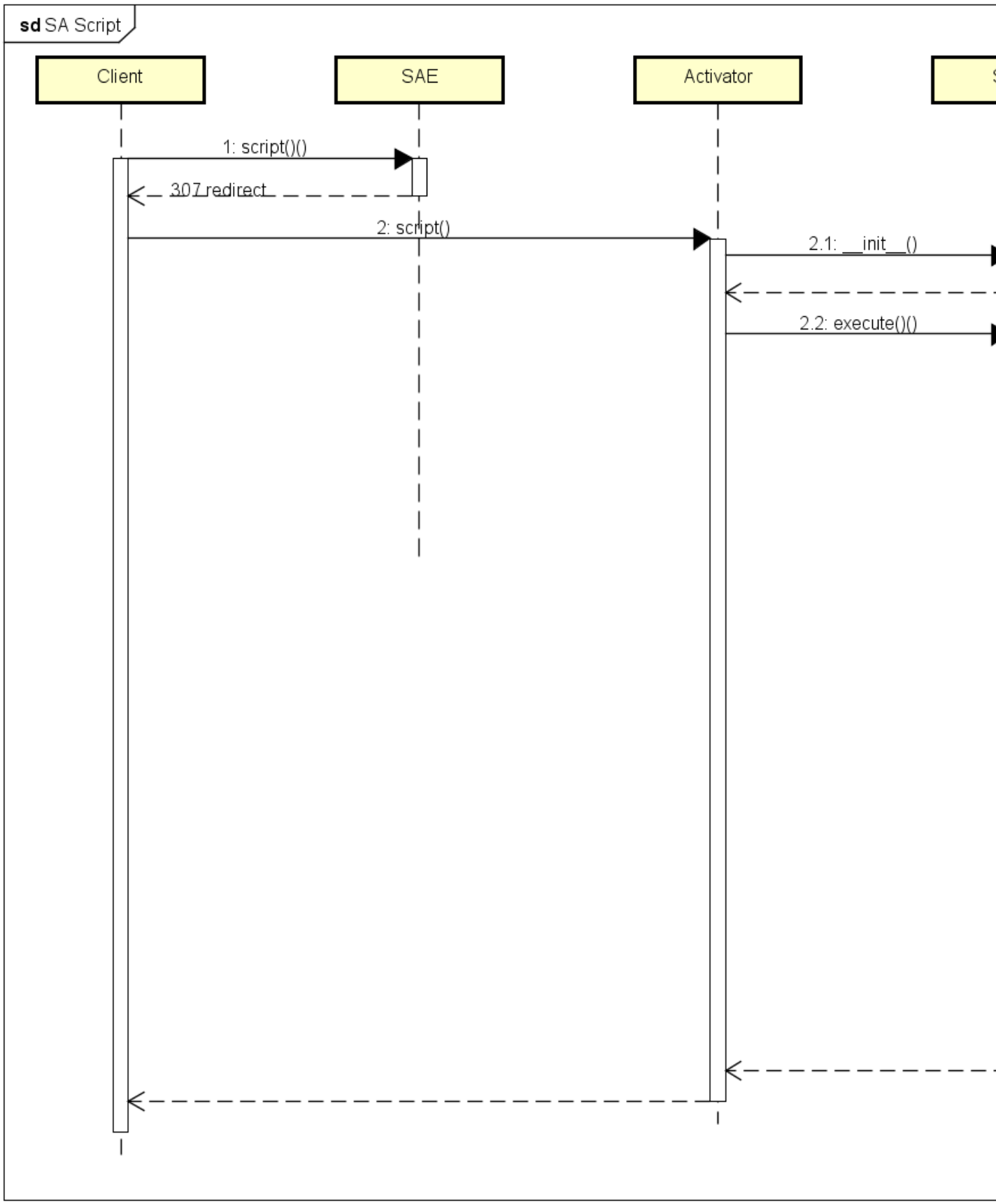

Рис. 4.1: Схема вызова профиля

В NOC принято соглашение по которому часть имени <folder1>, именуется как производитель оборудования, а часть имени <folder2> как название ПО, которое установлено на оборудовании. Н-р Cisco.IOS, Juniper.JunOS, Dlink.DxS.

> Attention: Имена являются регистрозависимыми!

Инициализация профиля осуществляется в файле __init.py__. В нём, путём наследования класса noc.core.profile.base.BaseProfile происходит переопределение настроек работы с оборудованием по умолчанию. Также в него выносятся методы, которые используются в нескольких скриптах.

Реализация интерфейсов и работы с оборудованием осуществляется в файлах скриптов. В них, путём наследования класса noc.core.script.base.BaseScript реализуется логика работы с оборудованием и нормализация полученных данных для передачи в NOC. Базовый набор скриптов для взаимодействия с оборудованием состоит из:

- get_version Реализует интерфейс noc.sa.interfaces.igetversion.IGetVersion . Запрашивает с оборудования платформу, версию ПО, и дополнительные аттрибуты (например, имя файла образа ПО, серийный номер...)
- get_capabilities Реализует интерфейс noc.sa.interfaces.igetcapabilities.IGetCapabilities . Производит опрос оборудования на предмет поддерживаемых протоколов (SNMP, LLDP, CDP). Данная информация используется при вызове скриптов и внутри, для принятия решения, по какому протоколу работать.
- get_interfaces Ралузует интерфейс noc.sa.interfaces.igetinterfaces.IGetInterfaces . Запрашивает список интерфейсов с оборудования.

Для построения :term:топологии потребуются скрипты:

- get_chassis_id (реализует интерфейс IGetChassisid).
- get_fqdn (реализует интерфейс IGetFqdn)
- get_<method>_neighnbors (реализует интерфейс соответствующего метода)
  - get_cdp_neighnbors (реализует интерфейс IGetCDPNeighbors)
  - get_lldp_neighbors (реализует интерфейс IGetLLDPNeighbors)
  - get_udld_neighbors (реализует интерфейс IGetUDLDNeighbors)
- get_mac_address_table (реализует интерфейс IGetConfig)
- get_arp (реализует интерфейс IGetArp)

Для сбора конфигурации

- get_config (реализует интерфейс igetconfig)

Для сбора состава оборудования

- get_inventory (реализует интерфейс igetinventory)

По необходимости, в профиле может быть добавлено любое количество файлов скриптов.

Взаимодействие с оборудованием

Для взаимодействия с оборудованием базовый класс noc.core.script.base.BaseScript предоставляет следующие методы.

CLI

BaseScript.cli(cmd, command_submit=None, bulk_lines=None, list_re=None, cached=False, file=None, ignore_errors=False, nowait=False, obj_parser=None, cmd_next=None, cmd_stop=None)

Execute CLI command and return result. Initiate cli session when necessary

Parameters

- cmd (str) – CLI command to execute
- command_submit (str) – Set in Profile, submit enter command
- bulk_lines – Not use
- list_re – Format output
- cached (bool) – Cache result
- file (file) – Read cli from file
- ignore_errors (bool) – Ignore syntax error in commands
- nowait (bool) – Not use

Type re.MatchObject

Returns if list_re is None, return a string

Return type str

Returns dict : if list_re is regular expression object, return a list of dicts (group name -> value), one dict per matched line

Return type dict

Метод позволяет выполнять команды на оборудовании. Возвращает вывод запрошенной команды в виде строки с текстом к которой, в дальнейшем, возможно применять любые методы для работы с текстовыми строками в Python.

SNMP

SNMP.get(oids, cached=False, version=None)

Perform SNMP GET request

Parameters

- oid (string) – string or list of oids
- cached (bool) – True if get results can be cached during session

Returns eigther result scalar or dict of name -> value

SNMP.getnext(oid, community_suffix=None, filter=None, cached=False, only_first=False, bulk=None, max_repetitions=None, version=None)

Методы позволяют выполнять SNMP запросы к оборудованию путём вызова метода с передачей ему OID'а. Для облегчения работы по SNMP, можно использовать:

SNMP.get_table(oid, community_suffix=None, cached=False)

GETNEXT wrapper. Returns a hash of <index> -> <value>

SNMP.get_tables(oids, community_suffix=None, bulk=False, min_index=None, max_index=None, cached=False)

Query list of SNMP tables referenced by oids and yields tuples of (key, value1, ..., valueN)

Parameters

- oids – List of OIDs
- community_suffix – Optional suffix to be added to community
- bulk – Use BULKGETNEXT if true
- min_index – Not use
- max_index – Not use
- cached – Optional parameter. If True - getting value will be cached

Returns

Полный перечень доступных методов смотрите в приложении.

HTTP

HTTP.get(path, headers=None, json=False)
Perform HTTP GET request

Parameters

- path – URI
- headers – Dict of additional headers
- json – Decode json if set to True

HTTP.post(path, data, headers=None, json=False)
Perform HTTP GET request

Parameters

- path – URI
- headers – Dict of additional headers
- json – Decode json if set to True

Выполняет, соответственно, GET и POST запрос к оборудованию. В рельзультате возвращает ответ в виде JSON.

Написание своего профиля

Для полноценной реализации профиля необходимо:

1. Создание структуры каталогов.
2. Создание профиля - __init__.py и прописывание параметров работы с оборудованием.
3. Реализация скриптов get_version и get_capabilities.
4. Реализация необходимого функционала.

Отсутствие того или иного скрипта в профиле ограничивает возможности системы во взаимодействии с оборудованием. Например, отсутствие скрипта получения интерфейсов не позволит построить топологию, а, в отсутствие скрипта get_cdp_neighnbors, не удастся построить топологию по протоколу CDP, но не помешает посмтроить топологию по lldp.

Файл профиля

В файле профиля - __init__.py необходимо прописать

```
from noc.core.profile.base import BaseProfile
import re


class Profile(BaseProfile):
    name = "Huawei.VRP"
    pattern_more = [
        (r"^ ---- More ----", " "),
        (r"[Cc]ontinue?\S+", "y\n\r"),
        (r"[Cc]onfirm?\S+", "y\n\r"),
        (r" [Aa]re you sure?\S+", "y\n\r"),
        (r"^Delete flash:", "y\n\r"),
        (r"^Squeeze flash:", "y\n\r")
    ]
    pattern_prompt = r"^[<#\[](?P<hostname>[a-zA-Z0-9-_\.\[/`\s]+)(?:-[a-zA-Z0-9/]+)*[>#\]]"
    pattern_syntax_error = r"(Error: |% Wrong parameter found at|% Unrecognized command found at|Error:Too many parameters
    command_more = " "
    config_volatile = ["^%.*?$"]
    command_disable_pager = "screen-length 0 temporary"
    command_enter_config = "system-view"
    command_leave_config = "return"
    command_save_config = "save"
    command_exit = "quit"
```

Аттрибуты описывают работу с оборудованием методу cli. Они указывают:

- pattern_prompt - описывает строку приглашения на оборудовании.
- pattern_syntax_error - строки, которые выводит оборудование в случае ошибки в команде
- command_more - комманда (или клавиша), которую необходимо передать оборудованию для продолжения постраничного вывода
- config_volatile - список строк, которые могут меняться в конфигурации, при заходе на оборудование (например изменение времени)
- command_disable_pager - комманда, которую необходимо передать оборудованию для отключения постраничного вывода информации
- command_enter_config - комманда, которую необходимо передать оборудованию для входа в режим настройки
- command_leave_config - комманда, которую необходимо передать оборудованию для выхода из режима настройки
- command_save_config - комманда, которую необходимо передать оборудованию для сохранения конфигурации
- command_exit - комманда, которую необходимо передать оборудованию для завершения сеанса

Note: Полный перчень методов и аттрибутов, доступных для переопределения смотрите в Приложении 1

Note: Проверить правильность заданных настроек можно выполнив ./noc script <profile> login!!

Скрипты

В первую очередь необходимо реализовать скрипты get_version и get_capabilities , поскольку информация от них используется для работы остальных скриптов. Скрипт начинается с области импорта. В ней, мы импортируем базовый класс (строка 1) скрипта и интерфейс, который собираемся реализовывать (строка 2). Здесь же можно импортировать дополнительные, необходимые нам модули. Например, модуль поддержки регулярных выражений (строка 3)

```
from noc.core.script.base import BaseScript
from noc.sa.interfaces.igetversion import IGetVersion
import re
```

После импорта необходимых модулей мы объявляем класс Scrip, наследую его от базового класса (BaseScript). После указываем полное имя скрипта, интерфейс и есть ли необходимость кэшировать результат выполнения.

```
class Script(BaseScript):
    name = "Huawei.VRP.get_version"
    cache = True
    interface = IGetVersion
```

Первый метод класса обязательно должен быть execute(). При запуске скрипта, с него начинается исполнение, остальные вызовы делаются из него.

```
def execute(self):
    v = ""
```

И, в методе execute() идёт обращение к методам работы с оборудованием, получение информации и, в конце, результат передаётся через оператор return.

```
1  ## Python modules
2  import re
3  ## NOC modules
4  from noc.core.script.base import BaseScript
5  from noc.sa.interfaces.igetversion import IGetVersion
6
7
8  class Script(BaseScript):
9      name = "Huawei.VRP.get_version"
10     cache = True
11     interface = IGetVersion
12
13
14     def execute(self):
15         v = ""
16         if self.has_snmp():
17             # Trying SNMP
18             try:
19                 # SNMPv2-MIB::sysDescr.0
20                 v = self.snmp.get("1.3.6.1.2.1.1.1.0", cached=True)
21             except self.snmp.TimeOutError:
22                 pass
23         if v == "":
```

```
            # Trying CLI
            try:
                v = self.cli("display version", cached=True)
            except self.CLISyntaxError:
                raise self.NotSupportedError()
        rx = self.find_re([
            self.rx_ver,
            self.rx_ver_snmp,
            self.rx_ver_snmp2,
            self.rx_ver_snmp3,
            self.rx_ver_snmp4,
            self.rx_ver_snmp5
        ], v)
        match = rx.search(v)
        r = {
            "vendor": "Huawei",
            "platform": platform,
            "version": match.group("version")
        }
        if "image" in match.groupdict():
            image = match.group("image")
            r["attributes"] = {"image": image}
        return r
```

Скрипт get_capabilities отличается от остальных скриптов. Его предназначение - определять поддержку оборудованием того или иного функционала. В дальнейшем подобная информация используется для оптимизации опроса оборудования. Например, если оборудование не поддерживает SNMP (например он отключён, или в настройках указан наверный SNMP Community) то скрипты, которые требуют рабочего SNMP не выполняются. Также отличием является то, что он относится к категории модульных скриптов. И он наследует не класс noc.core.script.base.BaseScript а класс вышестоящего скрипта noc.sa.profiles.Generic.get_capabilities. Также, в нём используется специальная конструкция - декоратор .Это позволяет обрабатывать ошибки, при вводе комманд, как стандартную ситуацию и делать вывод о недоступности функционала.

Рассмотрим пример. В стройках 2, 3 мы импортируем модули. В отличие от остальных скриптом, импортируются noc.sa.profiles.Generic.get_capabilities и noc.sa.profiles.Generic.get_capabilities. Строки с интерфейсом нет, т.к. она определена в вышестоящем скрипте - Generic.get_capabilities. По этой же причине отсутствует метод exetcute(), он вызывается из вышестоящего скрипта.

```
## NOC modules
from noc.sa.profiles.Generic.get_capabilities import Script as BaseScript
from noc.sa.profiles.Generic.get_capabilities import false_on_cli_error


class Script(BaseScript):
    name = "Huawei.VRP.get_capabilities"

    @false_on_cli_error
    def has_stp(self):
        """
        Check box has STP enabled
        """
        try:
            r = self.cli("display stp global | include Enabled")
            return "Enabled" in r
        except self.CLISyntaxError:
            try:
                r = self.cli("display stp | include disabled")
```

```
            return "Protocol Status" not in r
        except self.CLISyntaxError:
            r = self.cli("display stp")
            return "Protocol Status" not in r

```

Полный перчень проверяемых возможностей можно посмотреть в скрипте Generic.get_capabilities

class noc.sa.profiles.Generic.get_capabilities.Script(service, credentials, args=None, capabilities=None, version=None, parent=None, timeout=None, name=None, collect_beef=False)

CHECK_SNMP_GET = {}

SNMP_CAPS = {0: 'SNMP | v1', 1: 'SNMP | v2', 3: 'SNMP | v3'}

SNMP_VERSIONS = (1, 0)

cache = True

check_snmp_get(oid, version=None)
Check SNMP GET response to oid

check_snmp_getnext(oid, bulk=False, only_first=True)
Check SNMP response to GETNEXT/BULK

execute()

execute_platform(caps)
Method to be overriden in subclasses. :param caps: Dict of capabilities, can be modified

get_snmp_versions()
Get SNMP version

Returns Working SNMP versions set or empty set

has_cdp()
Returns True when CDP is enabled

has_ipv6()
Returns True when IPv6 ND is enabled

has_lldp()
Returns True when LLDP is enabled

has_oam()
Returns True when OAM is enabled

has_snmp()
Check basic SNMP support

has_snmp_bulk()

has_snmp_ifmib()
Check IF-MIB support

Return bool

has_snmp_ifmib_hc()
Check IF-MIB 64 bit counters

Return bool

has_stp()
Returns True when STP is enabled

has_udld()
Returns True when UDLD is enabled

interface
alias of IGetCapabilities

name = 'Generic.get_capabilities'

requires = []

noc.sa.profiles.Generic.get_capabilities.false_on_cli_error(f)

После отработки скрипта get_capabilities становится возможно пользоваться данными проверок. Для этого используются методы BaseScript.has_capability() , BaseScript.has_snmp() .

Все особенности работы с тем или иным оборудованием сосредоточены внутри профиля. Чем больше информации сможет собрать профиль (в рамках потребляемого NOC'ом), тем больше будет знать NOC.

Note: Происходящее внутри профиля, целиком возложена на разработчика. И после запуска NOC'ом не контроллируется.

Отладка

Для отладки профиля используется инструмент ./noc script. Он позволяет запускать скрипты из профиля в режиме отладки. Делается это следующим образом:

./noc script --debug <имя_скрипта> <имя_объекта> <параметры>, где

- <имя_скрипта> - полное имя скрипта (в формате <папка1>.<папка2>.<имя_скрипта>
- <имя_объекта> - имя Объекта (из меню Объекты -> Список объектов)
- <параметры> - параметры (не обязательно, только если используются)

Для удобства, параметры, можно передавать в файлике формата JSON, это не требует добавление объекта в систему.

./noc script --debug <имя_скрипта> <путь_к_файлу_json>

```
{
  "scheme": "telnet",
  "address": "192.168.1.1",
  "port": 23,
  "profile": "Cisco.IOS",
  "credentials": {
    "user": "login", "password": "pass", "super_password": "", "snmp_ro": "public", "snmp_rw": "private"
 },
 "caps": {
   "SNMP": true,
   "SNMP | IF-MIB": true,
   "SNMP | Bulk": true,
   "SNMP | IF-MIB | HC": true
 }
}
```

Примеры скриптов

- Huawei.VRP.get_version

```
# -*- coding: utf-8 -*-
##----------------------------------------------------------------------
## Copyright (C) 2007-2016 The NOC Project
## See LICENSE for details
##----------------------------------------------------------------------

## Python modules
import re
## NOC modules
from noc.core.script.base import BaseScript
from noc.sa.interfaces.igetversion import IGetVersion


class Script(BaseScript):
    name = "Huawei.VRP.get_version"
    cache = True
    interface = IGetVersion

    rx_ver = re.compile(
        r"^VRP.+Software, Version (?P<version>[^ ,]+),? .*?\n"
        r"\s*(?:Quidway|Huawei) (?P<platform>(?:NetEngine\s+|MultiserviceEngine\s+)?\S+)[^\n]+uptime",
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )
    rx_ver_snmp = re.compile(
        r"Versatile Routing Platform Software.*?"
        r"Version (?P<version>[^ ,]+),? .*?\n"
        r"\s*(?:Quidway|Huawei) (?P<platform>(?:NetEngine\s+)?"
        r"[^ \t\n\r\f\v\-]+)[^\n]+",
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )
    rx_ver_snmp2 = re.compile(
        r"(?P<platform>(?:\S+\s+)?S\d+(?:[A-Z]+-[A-Z]+)?(?:\d+\S+)?)"
        r"\s+Huawei\sVersatile\sRouting\sPlatform"
        r"\sSoftware.*Version\s(?P<version>\d+\.\d+)\s"
        r"\(S\d+\s(?P<image>\S+)+\).*",
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )
    rx_ver_snmp3 = re.compile(
        r"^\s*VRP.+Software, Version (?P<version>\S+)\s+"
        r"\((?P<platform>S\S+|CX\d+) (?P<image>[^)]+)",
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )

    rx_ver_snmp4 = re.compile(
        r"Huawei Versatile Routing Platform Software.*?"
        r"Version (?P<version>\S+) .*?"
        r"\s*(?:Quidway|Huawei) (?P<platform>(?:NetEngine\s+|MultiserviceEngine\s+)?\S+)[^\n]\d",
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )

    rx_ver_snmp5 = re.compile(
        r"Huawei Versatile Routing Platform.*?"
        r"Version (?P<version>\S+) .*?"
        r"\s*(?:Quidway|Huawei) (?P<platform>[A-Z0-9]+)\s",
```

```
        re.MULTILINE | re.DOTALL | re.IGNORECASE
    )

    def execute(self):
        v = ""
        if self.has_snmp():
            # Trying SNMP
            try:
                # SNMPv2-MIB::sysDescr.0
                v = self.snmp.get("1.3.6.1.2.1.1.1.0", cached=True)
            except self.snmp.TimeOutError:
                pass
        if v == "":
            # Trying CLI
            try:
                v = self.cli("display version", cached=True)
            except self.CLISyntaxError:
                raise self.NotSupportedError()
        rx = self.find_re([
            self.rx_ver,
            self.rx_ver_snmp,
            self.rx_ver_snmp2,
            self.rx_ver_snmp3,
            self.rx_ver_snmp4,
            self.rx_ver_snmp5
        ], v)
        match = rx.search(v)
        platform = match.group("platform")
        # Convert NetEngine to NE
        if platform.lower().startswith("netengine"):
            n, p = platform.split(" ", 1)
            platform = "NE%s" % p.strip().upper()
        elif platform.lower().startswith("multiserviceengine"):
            n, p = platform.split(" ", 1)
            platform = "ME%s" % p.strip().upper()
        r = {
            "vendor": "Huawei",
            "platform": platform,
            "version": match.group("version")
        }
        if "image" in match.groupdict():
            image = match.group("image")
            r["attributes"] = {"image": image}
        return r
```

- Huawei.VRP.get_capabilities

```
# -*- coding: utf-8 -*-
##----------------------------------------------------------------------
## Huawei.VRP.get_capabilities
##----------------------------------------------------------------------
## Copyright (C) 2007-2016 The NOC Project
## See LICENSE for details
##----------------------------------------------------------------------

## NOC modules
from noc.sa.profiles.Generic.get_capabilities import Script as BaseScript
from noc.sa.profiles.Generic.get_capabilities import false_on_cli_error
```

```
class Script(BaseScript):
    name = "Huawei.VRP.get_capabilities"

    @false_on_cli_error
    def has_stp(self):
        """
        Check box has STP enabled
        """
        try:
            r = self.cli("display stp global | include Enabled")
            return "Enabled" in r
        except self.CLISyntaxError:
            try:
                r = self.cli("display stp | include disabled")
                return "Protocol Status" not in r
            except self.CLISyntaxError:
                r = self.cli("display stp")
                return "Protocol Status" not in r

    @false_on_cli_error
    def has_lldp(self):
        """
        Check box has LLDP enabled
        """
        r = self.cli("display lldp local")
        return "Global LLDP is not enabled" not in r

    @false_on_cli_error
    def has_bfd(self):
        """
        Check box has BFD enabled
        """
        r = self.cli("display bfd configuration all")
        return not "Please enable BFD in global mode first" in r

    @false_on_cli_error
    def has_udld(self):
        """
        Check box has UDLD enabled
        """
        r = self.cli("display dldp")
        return "Global DLDP is not enabled" not in r \
        and "DLDP global status : disable" not in r
```

Приложение

Базовый класс профиля

class noc.core.profile.base.BaseProfile

Equipment profile. Contains all equipment personality and specific

can_strip_hostname_to = None

Device can strip long hostname in various modes i.e my.very.long.hostname# converts to my.very.long.hos(config)# In this case set can_strip_hostname_to = 16 None by default

cleaned_config(cfg)

Clean up config. Wipe out volatile strings before returning result

Parameters cfg (str) – Configuration

Returns Clean up configuration

Return type str

cleaned_input(input)

Preprocessor to clean up and normalize input from device. Delete ASCII sequences by default. Can be overriden to achieve desired behavior

ecma48.strip_control_sequences(s)

Normal text leaved untouched

```
>>> strip_control_sequences("Lorem Ipsum")
'Lorem Ipsum'
```

CR,LF and ESC survive from C0 set

```
>>> repr(strip_control_sequences("".join([chr(i) for i in range(32)])))
"'\\t\\n\\r'"
```

C1 set stripped (ESC+[survive)

```
>>> strip_control_sequences("".join(["\x1b"+chr(i) for i in range(64,96)]))
'\x1b['
```

CSI without P and I stripped

```
>>> strip_control_sequences("\x1b[@\x1b[a\x1b[~")
''
```

CSI with I stripped

```
>>> strip_control_sequences("\x1b[ @\x1b[/~")
''
```

CSI with P and I stripped >>> strip_control_sequences(“\x1b[0 @\x1b[0;7/~”) ‘’

Cleaned stream

```
>>> strip_control_sequences("L\x1b[@or\x1b[/~em\x1b[0 @ Ips\x1b[0;7/~um")
'Lorem Ipsum'
```

Incomplete CSI passed

```
>>> strip_control_sequences("\x1b[")
'\x1b['
```

Incomplete C1 passed

```
>>> strip_control_sequences('\x1b')
'\x1b'
```

Single backspace

```
>>> strip_control_sequences('1234')
'124'
```

Triple backspace

```
>>> strip_control_sequences('1234')
'4'
```

Backspaces followed with spaces

```
>>> strip_control_sequences('    test')
' test'
```

ASCII mess

```
>>> strip_control_sequences('[2J[?7l[3;23r[?6l[24;27H[?25h[24;27H[?6l[1;24r[?7l[2J[24;27H[1;24r[24;27H[2J[?7l[1;24r[
'switch# '
```

Parameters input (str) – Input text for clean

Returns Text with strip control Sequences

Return type str

classmethod cmp_version(v1, v2)

Compare two versions. Must return:

<0 , if v1<v2 0 , if v1==v2

>0 , if v1>v2

None , if v1 and v2 cannot be compared

Default implementation compares a versions in format N1. .. .NM

command_disable_pager = None

Sequence to disable pager

command_enter_config = None

Sequence to enter configuration mode

command_exit = None

Sequence to gracefully close session

command_leave_config = None

Sequence to leave configuration mode

command_more = '\n'

Sequence to be send to list forward pager If pattern_more is string and is matched

command_save_config = None

Sequence to save configuration

command_submit = '\n'

Sequence to be send at the end of all CLI commands

command_super = None

Sequence to enable priveleged mode

config_volatile = None
Volatile strings: A list of strings can be changed over time, which can be sweeped out of config safely or None Strings are regexpes, compiled with re.DOTALL|re.MULTILINE

convert_interface_name(s)
Normalize interface name

Returns Normalize interface name

Return type str

convert_interface_name_cisco(s)

```
>>> Profile().convert_interface_name_cisco("Gi0")
'Gi 0'
>>> Profile().convert_interface_name_cisco("GigabitEthernet0")
'Gi 0'
>>> Profile().convert_interface_name_cisco("Gi 0")
'Gi 0'
>>> Profile().convert_interface_name_cisco("tengigabitethernet 1/0/1")
'Te 1/0/1'
>>> Profile().convert_interface_name_cisco("tengigabitethernet 1/0/1.5")
'Te 1/0/1.5'
>>> Profile().convert_interface_name_cisco("Se 0/1/0:0")
'Se 0/1/0:0'
>>> Profile().convert_interface_name_cisco("Se 0/1/0:0.10")
'Se 0/1/0:0.10'
>>> Profile().convert_interface_name_cisco("ATM1/1/ima0")
'At 1/1/ima0'
>>> Profile().convert_interface_name_cisco("Port-channel5B")
'Po 5B'
```

convert_mac(mac)
Leave 00:11:22:33:44:55 style MAC-address untouched

Parameters mac (str) –

Returns MAC-address HH:HH:HH:HH:HH:HH

Return type str

```
>>> Profile().convert_mac_to_colon("00:11:22:33:44:55")
'00:11:22:33:44:55'
>>> Profile().convert_mac_to_colon("00:11:22:33:44:55")
'0011:2233:4455'
```

convert_mac_to_cisco(mac)
Convert 00:11:22:33:44:55 style MAC-address to 0011.2233.4455

Parameters mac (str) – HH:HH:HH:HH:HH:HH

Return str MAC-address HHHH.HHHH.HHHH

```
>>> Profile().convert_mac_to_cisco("00:11:22:33:44:55")
'0011.2233.4455'
```

convert_mac_to_colon(mac)
Leave 00:11:22:33:44:55 style MAC-address untouched

Parameters mac (str) –

Returns MAC-address HH:HH:HH:HH:HH:HH

Return type str

```
>>> Profile().convert_mac_to_colon("00:11:22:33:44:55")
'00:11:22:33:44:55'
>>> Profile().convert_mac_to_colon("00:11:22:33:44:55")
'0011:2233:4455'
```

convert_mac_to_dashed(mac)

Convert 00:11:22:33:44:55 style MAC-address to 00-11-22-33-44-55

Parameters mac (str) – MAC-address HH:HH:HH:HH:HH:HH

Returns MAC-address HH-HH-HH-HH-HH-HH

Return type str

```
>>> Profile().convert_mac_to_dashed("00:11:22:33:44:55")
'00-11-22-33-44-55'
```

convert_mac_to_huawei(mac)

Convert 00:11:22:33:44:55 style MAC-address to 0011-2233-4455

Returns MAC-address HHHH-HHHH-HHHH

Return type str

```
>>> Profile().convert_mac_to_huawei("00:11:22:33:44:55")
'0011-2233-4455'
```

convert_prefix(prefix)

Convert ip prefix to the format accepted by router's CLI

Parameters prefix (str) – IP Prefix

Returns IP MASK notation

Return type str

```
>>> Profile().convert_prefix("192.168.2.0/24")
'192.168.2.0/24'
>>> Profile().convert_prefix("192.168.2.0 255.255.255.0")
'192.168.2.0 255.255.255.0'
```

default_parser = None

Default config parser name. Full path to BaseParser subclass i.e noc.cm.parsers.Cisco.IOS.switch.IOSSwitchParser Can be overriden in get_parser method

generate_prefix_list(name, pl)

Generate prefix list: name - name of prefix list pl - is a list of (prefix, min_len, max_len) Strict - should tested prefix be exactly matched or should be more specific as well Can be overriden to achieve desired behavior

Not implemented in Base Class

get_interface_names(name)

Return possible alternative interface names, i.e. for LLDP discovery Local method Can be overriden to achieve desired behavior

Parameters name (str) – Interface Name

Returns List Alternative interface names

Return type list

classmethod get_interface_type(name)
Return IGetInterface-compatible interface type

Parameters name – Normalized interface name

Returns None

get_linecard(interface_name)
Returns linecard number related to interface >>> Profile().get_linecard("Gi 4/15") 4 >>> Profile().get_linecard("Lo") >>> Profile().get_linecard("ge-1/1/0") 1

classmethod get_parser(vendor, platform, version)
Returns full path to BaseParser instance to be used as config parser. None means no parser for particular platform

classmethod initialize()
Called once by profile loader

max_scripts = None
Upper concurrent scripts limit, if set

name = None
Profile name in form <vendor>.<system>

password_submit = None

Sequence to submit password. Use "

" if None

pattern_more = '^—MORE—'
Optional[regexp]: Regular expression to catch pager (Used in command results) If pattern_more is string, send command_more If pattern_more is a list of (pattern,command) send appropriative command

pattern_operation_error = None
Optional[regexp]: Regular expression to catch the CLI commands errors in cli output. If CLI output matches pattern_syntax_error and not matches pattern_syntax_error, then CLIOperationError exception raised

pattern_password = '[Pp]ass[Ww]ord: ?'
Optional[regexp]: Regulal expression to catch password prompt (Telnet/SSH sessions)

pattern_prompt = '^\\S*[>#]'
Optional[regexp]: Regular expression to catch command prompt (CLI Sessions)

pattern_syntax_error = None
Optional[regexp]: Regular expression to catch the syntax errors in cli output. If CLI output matches pattern_syntax_error, then CLISyntaxError exception raised

pattern_unpriveleged_prompt = None
Optional[regexp]: Regular expression to catch unpriveleged mode command prompt (CLI Session)

pattern_username = '([Uu]ser ?[Nn]ame|[Ll]ogin): ?'
List[str]: Regular expression to catch user name prompt. Usually during telnet sessions)

requires_netmask_conversion = False
Does the equipment supports bitlength netmasks or netmask should be converted to traditional formats

rogue_chars = ['\r']
List of chars to be stripped out of input stream before checking any regular expressions (when Action.CLEAN_INPUT==True)

root_interface(name)
Returns root interface >>> Profile().root_interface("Gi 0/1") 'Gi 0/1' >>> Profile().root_interface("Gi 0/1.15") 'Gi 0/1'

setup_script = None
Callable accepting script instance to set up additional script attributes and methods. Use Profile.add_script_method() to add methods

setup_session = None
Callable accepting script instance to set up session.

shutdown_session = None
Callable accepting script instance to finaly close session

supported_schemes = []
A list of supported access schemes. Access schemes constants are defined in noc.sa.protocols.sae_pb2 (TELNET, SSH, HTTP, etc) @todo: Deprecated

telnet_naws = '\xff\xff\xff\xff'
Telnet NAWS negotiation

telnet_send_on_connect = None
String to send just after telnet connect is established

telnet_slow_send_password = False
Password sending mode for telnet False - send password at once True - send password by characters

username_submit = None

Sequence to submit username. Use "

" if None

Базовый класс скрипта

class noc.core.script.base.BaseScript(service, credentials, args=None, capabilities=None, version=None, parent=None, timeout=None, name=None, collect_beef=False)
Bases: object

Service Activation script base class

exception CLIOperationError
Bases: noc.core.script.base.ScriptError

Operational CLI error

Raise CLIOperationError

exception BaseScript.CLISyntaxError
Bases: noc.core.script.base.ScriptError

Syntax error

Raise CLISyntaxError

exception BaseScript.NotSupportedError
Bases: noc.core.script.base.ScriptError

Feature is not supported

Raise NotSupportedError

exception BaseScript.ScriptError
Bases: exceptions.Exception

Script error

Raise ScriptError

BaseScript.TIMEOUT = 120
Default script timeout

exception BaseScript.UnexpectedResultError
Bases: noc.core.script.base.ScriptError

Unexpected result

Raise UnexpectedResultError

BaseScript.base_logger = <logging.Logger object>

BaseScript.cache = False
Enable call cache If True, script result will be cached and reused during lifetime of parent script

BaseScript.cached()
Return cached context managed. All nested CLI and SNMP GET/GETNEXT calls will be cached.

Usage:

with self.cached(): self.cli(".....) self.scripts.script()

BaseScript.clean_input(args)
Cleanup input parameters against interface

Parameters args – Arguments for cleaning method

Returns Cleaned input

Return type str

BaseScript.clean_output(result)
Clean script result against interface

Parameters result (str) – Output from device

Returns Cleaned output

Return type str

BaseScript.cleaned_config(config)
Clean up config from all unnecessary trash

Parameters config (str) – Configuration for clean

Returns Clean up config from all unnecessary trash

Return type str

BaseScript.cli(cmd, command_submit=None, bulk_lines=None, list_re=None, cached=False, file=None, ignore_errors=False, nowait=False, obj_parser=None, cmd_next=None, cmd_stop=None)
Execute CLI command and return result. Initiate cli session when necessary

Parameters

- cmd (str) – CLI command to execute
- command_submit (str) – Set in Profile, submit enter command
- bulk_lines – Not use

- list_re – Format output
- cached (bool) – Cache result
- file (file) – Read cli from file
- ignore_errors (bool) – Ignore syntax error in commands
- nowait (bool) – Not use

Type re.MatchObject

Returns if list_re is None, return a string

Return type str

Returns dict : if list_re is regular expression object, return a list of dicts (group name -> value), one dict per matched line

Return type dict

BaseScript.cli_protocols = {'beef': 'noc.core.script.cli.beef.BeefCLI', 'ssh': 'noc.core.script.cli.ssh.SSHCLI', 'telnet': 'no

BaseScript.close_cli_stream()

classmethod BaseScript.compile_match_filter(*args, **kwargs)
Compile arguments into version check function

Returns Returns callable accepting self and version hash arguments

BaseScript.configure()

Returns Returns configuration context

Return type object

BaseScript.enter_config()
Enter configuration mote

Returns

BaseScript.execute(**kwargs)
Default script behavior: Pass through _execute_chain and call appropriative handler

BaseScript.expand_interface_range(s)
Convert interface range expression to a list of interfaces "Gi 1/1-3,Gi 1/7" -> ["Gi 1/1", "Gi 1/2", "Gi 1/3", "Gi 1/7"] "1:1-3" -> ["1:1", "1:2", "1:3"] "1:1-1:3" -> ["1:1", "1:2", "1:3"]

Parameters s (str) – Comma-separated list

Returns ["port", ..]

Return type list

BaseScript.expand_rangelist(s)
Expand expressions like "1,2,5-7" to [1, 2, 5, 6, 7]

Parameters s (str) – Comma-separated list

Returns [1, 2, 5, 6, 7]

Return type list

BaseScript.find_re(iter, s)
Find first matching regular expression or raise Unexpected result error

Parameters

- iter – Iterable objecth for search

- s (re) –

Returns

Return type re.match

BaseScript.get_cache(key1, key2)
Get cached result or raise KeyError

Parameters

- str (key2) – Cache key1
- str – Cache key2

Returns Cache result

Return type object

BaseScript.get_cli_stream()

BaseScript.get_timeout()

Returns TIMEOUT for script

Return type int

BaseScript.has_capability(capability)
Shech whether equipment supports capability

Parameters capability (str) – Capability name

Returns Check result

Return type bool

BaseScript.has_oid(oid)
Check object responses to oid

Returns Check result

Return type bool

BaseScript.has_snmp()
Check whether equipment has SNMP enabled

Returns Check result

Return type bool

BaseScript.has_snmp_bulk()
Check whether equipment supports SNMP BULK

Returns Check result

Return type bool

BaseScript.has_snmp_v1()
Check whether equipment supports SNMP v1

Returns Check result

Return type bool

BaseScript.has_snmp_v2c()
Check whether equipment supports SNMP v2c

Returns Check result

Return type bool

BaseScript.has_snmp_v3()
Check whether equipment supports SNMP v3

Returns Check result

Return type bool

BaseScript.hex_to_bin(s)
Convert hexadecimal string to boolean string. All non-hexadecimal characters are ignored

Parameters s – Input string

Returns Boolean string

Return type basestring

BaseScript.hexbin = {'a': '1010', 'c': '1100', 'b': '1011', 'e': '1110', 'd': '1101', 'f': '1111', '1': '0001', '0': '0000', '3': '001

BaseScript.hexstring_to_mac(s)
Convert a 6-octet string to MAC address

Parameters s (str) – 6-octet string

Returns MAC Address

Return type MAC

BaseScript.ignored_exceptions(iterable)
Context manager to silently ignore specified exceptions

Param Iterable object

Returns

Return type object

BaseScript.interface = None
Implemented interface

BaseScript.leave_config()
Leave configuration mode

Returns

BaseScript.macs_to_ranges(macs)
Converts list of macs to rangea

Parameters macs – Iterable yielding mac addresses

Returns [(from, to), ..]

classmethod BaseScript.match(*args, **kwargs)
execute method decorator

classmethod BaseScript.match_lines(rx, s)

BaseScript.match_version(*args, **kwargs)
inline version for BaseScript.match

BaseScript.motd
Return message of the day

Returns Message of the day

Return type str

BaseScript.name = None
Script name in form of <vendor>.<system>.<name>

BaseScript.pop_prompt_pattern()

BaseScript.push_prompt_pattern(pattern)

BaseScript.re_match(rx, s, flags=0)
Match s against regular expression rx using re.match Raise UnexpectedResultError if regular expression is not matched. Returns match object. rx can be string or compiled regular expression

Returns re match object

Return type object

BaseScript.re_search(rx, s, flags=0)
Match s against regular expression rx using re.search Raise UnexpectedResultError if regular expression is not matched. Returns match object. rx can be string or compiled regular expression

Returns re match object

Return type re.match

BaseScript.requires = []
Scripts required by generic script. For common scripts - empty list For generics - list of pairs (script_name, interface)

BaseScript.root
Get root script

:return :rtype: str

BaseScript.run()
Run script

Returns Result return execute() of method

BaseScript.rx_detect_sep = <_sre.SRE_Pattern object>

BaseScript.save_config(immediately=False)
Save current config

Returns

BaseScript.schedule_to_save()

BaseScript.set_cache(key1, key2, value)
Set cached result

Parameters

- key2 (key1,) – Cache key
- value – Value for write to cache

BaseScript.set_motd(motd)
Set _motd Attrinute - Message of The day

Parameters motd (str) –

Returns

BaseScript.strip_first_lines(text, lines=1)
Strip first lines

Parameters

- text (str) – Text
- lines (int) – Number of lines for strip

Returns Text with stripped first N lines

Return type str

class noc.core.script.base.ScriptsHub(script)

Bases: object

Object representing Script.scripts structure. Returns initialized child script which can be used ans callable

Интерфейсы NOCa

Профиль передаёт данные в сторону основной системы через интерфейс обмена данными. Интерфейс описывает формат и набор данных, который должен вернуть скрипт его реализующий. Существуют следующие интерфейсы для реализации профилем:

class noc.sa.interfaces.igetversion.IGetVersion

Returns Parameters in dict

Vendor StringParameter:

Platform StringParameter

Version StringParameter

Attributes DictParameter

Return type DictParameter

class noc.sa.interfaces.igetcapabilities.IGetCapabilities

Common usage scenarios

•Capabilities

- SNMP = True|False
- LLDP = True|False
- CDP = True|False

List of available Capabilities in Inventory → Setup → Capabilities.

Returns Dictionary of capabilities. Capabilities: True|False

Return type DictParameter

class noc.sa.interfaces.igetinterfaces.IGetInterfaces

Bases: noc.sa.interfaces.base.Interface

IGetInterfaces.

Common usage scenarios

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
  - name = physical port name

        - type = “physical”
        - mac = interface mac address
        - subinterfaces:
            - name = interface name (same as physical name for most platforms)
            - enabled_afi = [”IPv4”, “IPv6”]
            - ipv4_addresses = list of IPv4 addresses
            - ipv6_addresses = list of IPv6 addresses

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
    - name = physical port name
    - type = “physical”
    - mac = interface mac address
    - subinterfaces:
        - name = interface name (same as physical name for most platforms)
        - enabled_afi = [”IPv4”]
        - ip_unnumbered_subinterface = subinterface name to borrow an address

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
    - name = physical port name
    - type = “physical”
    - mac = interface mac address
    - subinterfaces:
        - name = interface name (same as physical name for most platforms)
        - enabled_afi = [”BRIDGE”]
        - untagged_vlan = VLANID

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
    - name = physical port name
    - type = “physical”
    - mac = interface mac address

        - subinterfaces:
            - name = interface name (same as physical name for most platforms)
            - enabled_afi = ["BRIDGE"]
            - tagged_vlans = VLANS (list)

- forwarding_instance:
    - forwarding_instance = "default"
    - type = "ip"
    - interfaces:
        - name = physical port name
        - type = "physical"
        - mac = interface mac address
        - subinterfaces:
            - name = interface name.VLAN1 (for most platforms)
            - vlan_ids = [VLAN1]
            - enabled_afi = ["IPv4"]
            - ipv4_addresses = [list of VLAN1 addresses]
            - name = interface name.VLAN2 (for most platforms)
            - vlan_ids = [VLAN2]
            - enabled_afi = ["IPv4"]
            - ipv4_addresses = [list of VLAN2 addresses]

- forwarding_instance:
    - forwarding_instance = "default"
    - type = "ip"
    - interfaces:
        - name = if1
        - type = "aggregated"
        - subinterfaces:
            - name = interface name (same as parent for most platforms)
            - enabled_afi = ["IPv4"]
            - ipv4_addresses = list of IPv4 addresses
        - name = if2
        - type = "physical"
        - enabled_protoocols = ["LACP"]
        - aggregated_interface = "if1"
        - name = if3
        - type = "physical"

        * enabled_protoocols = [”LACP”]
        * aggregated_interface = “if1”

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
    * name = if1
    * type = “aggregated”
    * subinterfaces:
        · name = interface name (same as parent for most platforms)
        · enabled_afi = [”BRIDGE”]
        · tagged_vlans = list of tagged vlans
    * name = if2
    * type = “physical”
    * enabled_protocols = [”LACP”]
    * aggregated_interface = “if1”
    * name = if3
    * type = “physical”
    * aggregated_interface = “if1”

•forwarding_instance:

- forwarding_instance = “default”
- type = “ip”
- interfaces:
    * name = interface name
    * type = “physical”
    * subinterfaces:
        · name = interface name.VLAN1 (for most platforms)
        · vlan_ids = [VLAN1]
        · enabled_afi = [”IPv4”]
        · ipv4_addresses = List of VLAN1 addresses
- forwarding_instance = “VRF1”
- type = “vrf”

@todo RD * interfaces:

- name = interface name
- type = “physical”
- subinterfaces:

* name = interface name.VLAN2 (for most platforms)
* vlan_ids = [VLAN2]
* enabled_afi = ["IPv4"]
* ipv4_addresses = List of VLAN2 addresses in VRF1

•forwarding_instance:
•forwarding_instance = "default"
•type = "ip"
•interfaces:

- name = physical port name
- type = "physical"
- subinterfaces:
  * name = interface name (same as physical name for most platforms)
  * enabled_afi = ["IPv4", "IPv6", "ATM"]
  * ipv4_addresses = list of IPv4 addresses
  * ipv6_addresses = list of IPv6 addresses
  * vpi = port vpi
  * vci = port vci

•forwarding_instance:
•forwarding_instance = "default"
•type = "ip"
•interfaces:

- name = physical port name
- type = "physical"
- subinterfaces:
  * name = interface name (same as physical name for most platforms)
  * enabled_afi = ["BRIDGE", "ATM"]
  * untagged_vlan = untagged vlan, if any
  * tagged_vlans = list of tagged vlans, if any
  * vpi = port vpi
  * vci = port vci

Returns

Return type DictParameter

class noc.sa.interfaces.igetchassisid.IGetChassisID

Returns Dictionary with first_chassis_mac and last_chassis_mac. May equal.

Return type DictListParameter

class noc.sa.interfaces.igetlldpneighbors.IGetLLDPNeighbors

LLDP neighbor information Rules:
•local_inteface must be filled with interface name (will be cleaned automatically)

local_interface_id depens upon how the box advertises own interfaces:

•If interfaces advertised with macAddress(3) LldpPortIdSubtype,

local_interface_id must be set to interface MAC address (will be cleaned automatically)

•If interface advertised with networkAddress(4) LldpPortIdSubtype,
local_interface_id must be set to interface IP address

•If interfaces advertised with interfaceName(5) LldpPortIdSubtype,
local_interface_id must be left empty or ommited.

•If interfaces advertised with local(7) LldpPortIdSubtype, local_interface_id must be set to local identifier

Remote port handling solely depends upon remote_port_subtype:

•For macAddress(3) - convert to common normalized form
•For networkAddress(4) - return as IP address
•For interfaceName(5) - return untouched
•For local(7) - convert to integer and return untouched

class noc.sa.interfaces.igetcdpneighbors.IGetCDPNeighbors

Common usage scenarios

•device_id = Local device id: FQDN or serial number
•neighbors = Remote device id: FQDN or serial number

– device_id = Remote device id: FQDN or serial number
– local_interface = Local interface
– remote_interface = Remote interface
– remote_ip = Remote IP

Returns Dictionary

Return type dict

Типы данных, применяемые в интерфейсах NOCa

В интерфейсах применяются следующие типы данных:

class noc.sa.interfaces.base.BooleanParameter(required=True, default=None)

```
>>> BooleanParameter().clean(True)
True
>>> BooleanParameter().clean(False)
False
>>> BooleanParameter().clean("True")
True
>>> BooleanParameter().clean("yes")
True
>>> BooleanParameter().clean(1)
True
>>> BooleanParameter().clean(0)
False
>>> BooleanParameter().clean([])
Traceback (most recent call last):
...
InterfaceTypeError: BooleanParameter: [].
>>> BooleanParameter(default=False).clean(None)
False
```

```
>>> BooleanParameter(default=True).clean(None)
True
```

class noc.sa.interfaces.base.DictListParameter(required=True, default=None, attrs=None, convert=False)

```
>>> DictListParameter().clean([{"1": 2},{"2":3, "4":1}])
[{'1': 2}, {'2': 3, '4': 1}]
>>> DictListParameter(attrs={"i":IntParameter(),"s":StringParameter()}).clean([{"i":10,"s":"ten"},{"i":"5","s":"five"}])
[{'i': 10, 's': 'ten'}, {'i': 5, 's': 'five'}]
>>> DictListParameter(attrs={"i":IntParameter(),"s":StringParameter()},convert=True).clean({"i":"10","s":"ten"})
[{'i': 10, 's': 'ten'}]
```

class noc.sa.interfaces.base.DictParameter(required=True, default=None, attrs=None, truncate=False)

```
>>> DictParameter(attrs={"i":IntParameter(),"s":StringParameter()}).clean({"i":10,"s":"ten"})
{'i': 10, 's': 'ten'}
>>> DictParameter(attrs={"i":IntParameter(),"s":StringParameter()}).clean({"i":"10","x":"ten"})
Traceback (most recent call last):
  ...
InterfaceTypeError: DictParameter: {'i': '10', 'x': 'ten'}
```

class noc.sa.interfaces.base.DocumentParameter(document, required=True)

Document reference parameter

class noc.sa.interfaces.base.FloatParameter(required=True, default=None, min_value=None, max_value=None)

```
>>> FloatParameter().clean(1.2)
1.2
>>> FloatParameter().clean("1.2")
1.2
>>> FloatParameter().clean("not a number")
Traceback (most recent call last):
  ...
InterfaceTypeError: FloatParameter: 'not a number'
>>> FloatParameter(min_value=10).clean(5)
Traceback (most recent call last):
  ...
InterfaceTypeError: FloatParameter: 5
>>> FloatParameter(max_value=7).clean(10)
Traceback (most recent call last):
  ...
InterfaceTypeError: FloatParameter: 10
>>> FloatParameter(max_value=10,default=7).clean(5)
5
>>> FloatParameter(max_value=10,default=7).clean(None)
7
>>> FloatParameter(max_value=10,default=15)
Traceback (most recent call last):
  ...
InterfaceTypeError: FloatParameter: 15
```

class noc.sa.interfaces.base.GeoPointParameter(required=True, default=None)

```
>>> GeoPointParameter().clean([180, 90])
[180, 90]
>>> GeoPointParameter().clean([75.5, "90"])
[75.5, 90]
>>> GeoPointParameter().clean("[180, 85.5]")
[180, 85.5]
>>> GeoPointParameter().clean([1])
Traceback (most recent call last):
...
InterfaceTypeError: GeoPointParameter: [1]
```

class noc.sa.interfaces.base.IPv4Parameter(required=True, default=None, choices=None)

```
>>> IPv4Parameter().clean("192.168.0.1")
'192.168.0.1'
>>> IPv4Parameter().clean("192.168.0.256")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPvParameter: '192.168.0.256'
```

class noc.sa.interfaces.base.IPv4PrefixParameter(required=True, default=None, choices=None)

```
>>> IPv4PrefixParameter().clean("192.168.0.0/16")
'192.168.0.0/16'
>>> IPv4PrefixParameter().clean("192.168.0.256")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4PrefixParameter: '192.168.0.256'
>>> IPv4PrefixParameter().clean("192.168.0.0/33")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4PrefixParameter: '192.168.0.0/33'
>>> IPv4PrefixParameter().clean("192.168.0.0/-5")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4PrefixParameter: '192.168.0.0/-5'
```

class noc.sa.interfaces.base.IPv6Parameter(required=True, default=None, choices=None)

```
>>> IPv6Parameter().clean("::")
'::'
>>> IPv6Parameter().clean("::1")
'::1'
>>> IPv6Parameter().clean("2001:db8::1")
'2001:db8::1'
>>> IPv6Parameter().clean("2001:db8::")
'2001:db8::'
>>> IPv6Parameter().clean("::ffff:192.168.0.1")
'::ffff:192.168.0.1'
```

```
>>> IPv6Parameter().clean('g::')
Traceback (most recent call last):
...
InterfaceTypeError: IPv6Parameter: 'g::'.
>>> IPv6Parameter().clean("0:00:0:0:0::1")
'::1'
>>> IPv6Parameter().clean("::ffff:c0a8:1")
'::ffff:192.168.0.1'
>>> IPv6Parameter().clean("2001:db8:0:7:0:0:0:1")
'2001:db8:0:7::1'
```

class noc.sa.interfaces.base.IPv6PrefixParameter(required=True, default=None, choices=None)

```
>>> IPv6PrefixParameter().clean("::/128")
'::/128'
>>> IPv6PrefixParameter().clean("2001:db8::/32")
'2001:db8::/32'
>>> IPv6PrefixParameter().clean("2001:db8::/129")
Traceback (most recent call last):
...
InterfaceTypeError: IPv6PrefixParameter: '2001:db8::/129'
>>> IPv6PrefixParameter().clean("2001:db8::/g")
Traceback (most recent call last):
...
InterfaceTypeError: IPv6PrefixParameter: '2001:db8::/g'
>>> IPv6PrefixParameter().clean("2001:db8::")
Traceback (most recent call last):
...
InterfaceTypeError: IPv6PrefixParameter: '2001:db8::'
```

class noc.sa.interfaces.base.InstanceOfParameter(cls, required=True, default=None)

```
>>> class C: pass
>>> class X: pass
>>> class CC(C): pass
>>> InstanceOfParameter(cls=C).clean(C()) and "Ok"
'Ok'
>>> InstanceOfParameter(cls=C).clean(CC()) and "Ok"
'Ok'
>>> InstanceOfParameter(cls=C).clean(1) and "Ok"
Traceback (most recent call last):
...
InterfaceTypeError: InstanceOfParameter: 1
>>> InstanceOfParameter(cls="C").clean(C()) and "Ok"
'Ok'
>>> InstanceOfParameter(cls="C").clean(1) and "Ok"
Traceback (most recent call last):
...
InterfaceTypeError: InstanceOfParameter: 1
```

class noc.sa.interfaces.base.IntParameter(required=True, default=None, min_value=None, max_value=None)

```
>>> IntParameter().clean(1)
1
>>> IntParameter().clean("1")
1
>>> IntParameter().clean("not a number")
Traceback (most recent call last):
  ...
InterfaceTypeError: IntParameter: 'not a number'
>>> IntParameter(min_value=10).clean(5)
Traceback (most recent call last):
  ...
InterfaceTypeError: IntParameter: 5
>>> IntParameter(max_value=7).clean(10)
Traceback (most recent call last):
  ...
InterfaceTypeError: IntParameter: 10
>>> IntParameter(max_value=10, default=7).clean(5)
5
>>> IntParameter(max_value=10, default=7).clean(None)
7
>>> IntParameter(max_value=10, default=15)
Traceback (most recent call last):
  ...
InterfaceTypeError: IntParameter: 15
>>> IntParameter().clean(None)
Traceback (most recent call last):
  ...
InterfaceTypeError: IntParameter: None
None
```

class noc.sa.interfaces.base.ListOfParameter(element, required=True, default=None, convert=False)

```
>>> ListOfParameter(element=IntParameter()).clean([1,2,3])
[1, 2, 3]
>>> ListOfParameter(element=IntParameter()).clean([1,2,"3"])
[1, 2, 3]
>>> ListOfParameter(element=IntParameter()).clean([1,2,"x"])
Traceback (most recent call last):
...
InterfaceTypeError: IntParameter: 'x'
>>> ListOfParameter(element=StringParameter()).clean([1,2,3,"x"])
['1', '2', '3', 'x']
>>> ListOfParameter(element=StringParameter(),default=[]).clean(None)
[]
>>> ListOfParameter(element=StringParameter(),default=[1,2,3]).clean(None)
['1', '2', '3']
>>> ListOfParameter(element=[StringParameter(), IntParameter()]).clean([("a",1), ("b", "2")])
[['a', 1], ['b', 2]]
>>> ListOfParameter(element=[StringParameter(), IntParameter()]).clean([("a",1), ("b", "x")])
Traceback (most recent call last):
...
InterfaceTypeError: IntParameter: 'x'
```

class noc.sa.interfaces.base.MACAddressParameter(required=True, default=None, accept_bin=True)

```
>>> MACAddressParameter().clean("1234.5678.9ABC")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("1234.5678.9abc")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("0112.3456.789a.bc")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("1234.5678.9abc.def0")
Traceback (most recent call last):
  ...
InterfaceTypeError: MACAddressParameter: '1234.5678.9ABC.DEF0'
>>> MACAddressParameter().clean("12:34:56:78:9A:BC")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("12-34-56-78-9A-BC")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("0:13:46:50:87:5")
'00:13:46:50:87:05'
>>> MACAddressParameter().clean("123456-789abc")
'12:34:56:78:9A:BC'
>>> MACAddressParameter().clean("12-34-56-78-9A-BC-DE")
Traceback (most recent call last):
  ...
InterfaceTypeError: MACAddressParameter: '12:34:56:78:9A:BC:DE'
>>> MACAddressParameter().clean("AB-CD-EF-GH-HJ-KL")
Traceback (most recent call last):
  ...
InterfaceTypeError: MACAddressParameter: 'AB:CD:EF:GH:HJ:KL'
>>> MACAddressParameter().clean("aabb-ccdd-eeff")
'AA:BB:CC:DD:EE:FF'
>>> MACAddressParameter().clean("aabbccddeeff")
'AA:BB:CC:DD:EE:FF'
>>> MACAddressParameter().clean("AABBCCDDEEFF")
'AA:BB:CC:DD:EE:FF'
>>> MACAddressParameter().clean("\xa8\xf9K\x80\xb4\xc0")
'A8:F9:4B:80:B4:C0'
>>> MACAddressParameter(accept_bin=False).clean("\xa8\xf9K\x80\xb4\xc0")
Traceback (most recent call last):
  ...
InterfaceTypeError: MACAddressParameter: '¨ùK´À'.
```

class noc.sa.interfaces.base.ModelParameter(model, required=True)

Model reference parameter

class noc.sa.interfaces.base.NoneParameter(required=True)

```
>>> NoneParameter().clean(None)
>>> NoneParameter().clean("None")
Traceback (most recent call last):
  ...
InterfaceTypeError: NoneParameter: 'None'
```

class noc.sa.interfaces.base.OIDParameter(required=True, default=None)

```
>>> OIDParameter().clean("1.3.6.1.2.1.1.1.0")
'1.3.6.1.2.1.1.1.0'
```

```
>>> OIDParameter(default="1.3.6.1.2.1.1.1.0").clean(None)
'1.3.6.1.2.1.1.1.0'
>>> OIDParameter().clean("1.3.6.1.2.1.1.X.0")
Traceback (most recent call last):
  ...
InterfaceTypeError: OIDParameter: '1.3.6.1.2.1.1.X.0'
```

class noc.sa.interfaces.base.ORParameter(left, right)

```
>>> ORParameter(IntParameter(),IPv4Parameter()).clean(10)
10
>>> ORParameter(IntParameter(),IPv4Parameter()).clean("192.168.1.1")
'192.168.1.1'
>>> ORParameter(IntParameter(),IPv4Parameter()).clean("xxx")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4Parameter: 'xxx'
>>> (IntParameter()|IPv4Parameter()).clean(10)
10
>>> (IntParameter()|IPv4Parameter()).clean("192.168.1.1")
'192.168.1.1'
>>> (IntParameter()|IPv4Parameter()).clean("xxx")
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4Parameter: 'xxx'
>>> (IntParameter()|IPv4Parameter()).clean(None)
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4Parameter: None.
>>> (IntParameter(required=False)|IPv4Parameter(required=False)).clean(None)
>>> (IntParameter(required=False)|IPv4Parameter()).clean(None)
Traceback (most recent call last):
  ...
InterfaceTypeError: IPv4Parameter: None.
```

class noc.sa.interfaces.base.Parameter(required=True, default=None)

Abstract parameter

clean(value)

Input parameter normalization

Parameters value (Arbitrary python type) – Input parameter

Returns Normalized value

form_clean(value)

Clean up form field

Parameters value (Arbitrary python type) – Input parameter

Returns Normalized value

get_form_field(label=None)

Get appropriative form field

raise_error(value, msg='')

Raise InterfaceTypeError

Parameters

- value (Arbitrary python type) – Value where error detected
- msg (String) – Optional message

:raises InterfaceTypeError

script_clean_input(profile, value)
Clean up script input parameters. Can be overloaded to handle profile specific.

Parameters

- profile (Profile instance) – Profile
- value (Arbitrary python type) – Input parameter

Returns Normalized value

script_clean_result(profile, value)
Clean up script result parameters. Can be overloaded to handle profile specific.

Parameters

- profile (Profile instance) – Profile
- value (Arbitrary python type) – Input parameter

Returns Normalized value

class noc.sa.interfaces.base.PyExpParameter(required=True, default=None, choices=None)
Check python expression >>> PyExpParameter().clean("(a + 3) * 7") '(a + 3) * 7' >>> PyExpParameter().clean("a =!= b") #doctest: +IGNORE_EXCEPTION_DETAIL Traceback (most recent call last):
...
InterfaceTypeError: REParameter: 'a =!= b'

class noc.sa.interfaces.base.REParameter(required=True, default=None, choices=None)
Check Regular Expression >>> REParameter().clean(".+?") ".+?" >>> REParameter().clean("+") #doctest: +IGNORE_EXCEPTION_DETAIL Traceback (most recent call last):
...
InterfaceTypeError: REParameter: '+'

class noc.sa.interfaces.base.REStringParameter(regexp, required=True, default=None)

```
>>> REStringParameter("ex+p").clean("exp")
'exp'
>>> REStringParameter("ex+p").clean("exxp")
'exxp'
>>> REStringParameter("ex+p").clean("regexp 1")
'regexp 1'
>>> REStringParameter("ex+p").clean("ex")
Traceback (most recent call last):
  ...
InterfaceTypeError: REStringParameter: 'ex'
>>> REStringParameter("ex+p",default="exxp").clean("regexp 1")
'regexp 1'
>>> REStringParameter("ex+p",default="exxp").clean(None)
'exxp'
```

class noc.sa.interfaces.base.StringListParameter(required=True, default=None, convert=False)

```
>>> StringListParameter().clean(["1","2","3"])
['1', '2', '3']
>>> StringListParameter().clean(["1",2,"3"])
['1', '2', '3']
```

class noc.sa.interfaces.base.StringParameter(required=True, default=None, choices=None)

```
>>> StringParameter().clean("Test")
'Test'
>>> StringParameter().clean(10)
'10'
>>> StringParameter().clean(None)
'None'
>>> StringParameter(default="test").clean("no test")
'no test'
>>> StringParameter(default="test").clean(None)
'test'
>>> StringParameter(choices=["1","2"]).clean("1")
'1'
>>> StringParameter(choices=["1","2"]).clean("3")
Traceback (most recent call last):
...
InterfaceTypeError: StringParameter: '3'.
```

class noc.sa.interfaces.base.SubclassOfParameter(cls, required=True, default=None)

```
>>> class C: pass
>>> class C1(C): pass
>>> class C2(C1): pass
>>> class C3(C1): pass
>>> SubclassOfParameter(cls=C).clean(C2) and "Ok"
'Ok'
>>> SubclassOfParameter(cls=C).clean(1)
Traceback (most recent call last):
...
InterfaceTypeError: SubclassOfParameter: 1
>>> SubclassOfParameter(cls="C").clean(C2) and "Ok"
'Ok'
>>> SubclassOfParameter(cls=C).clean(1)
Traceback (most recent call last):
...
InterfaceTypeError: SubclassOfParameter: 1
>>> SubclassOfParameter(cls=C2).clean(C3)
Traceback (most recent call last):
...
InterfaceTypeError: SubclassOfParameter: <class base.C3>
>>> SubclassOfParameter(cls="C",required=False).clean(None)
```

class noc.sa.interfaces.base.TagsParameter(required=True, default=None)

```
>>> TagsParameter().clean([1, 2, "tags"])
[u'1', u'2', u'tags']
```

```
>>> TagsParameter().clean([1, 2, "tags "])
[u'1', u'2', u'tags']
>>> TagsParameter().clean("1,2,tags")
[u'1', u'2', u'tags']
>>> TagsParameter().clean("1 , 2,  tags")
[u'1', u'2', u'tags']
```

class noc.sa.interfaces.base.VLANIDListParameter(required=True, default=None)

```
>>> VLANIDListParameter().clean(["1","2","3"])
[1, 2, 3]
>>> VLANIDListParameter().clean([1,2,3])
[1, 2, 3]
```

class noc.sa.interfaces.base.VLANIDParameter(required=True, default=None)

```
>>> VLANIDParameter().clean(10)
10
>>> VLANIDParameter().clean(5000)
Traceback (most recent call last):
  ...
InterfaceTypeError: VLANIDParameter: 5000
>>> VLANIDParameter().clean(0)
Traceback (most recent call last):
  ...
InterfaceTypeError: VLANIDParameter: 0
```

class noc.sa.interfaces.base.VLANStackParameter(required=True, default=None)

```
>>> VLANStackParameter().clean(10)
[10]
>>> VLANStackParameter().clean([10])
[10]
>>> VLANStackParameter().clean([10, "20"])
[10, 20]
>>> VLANStackParameter().clean([10, 0])
[10, 0]
```

noc.sa.interfaces.base.iparam(**params)

Function parameters decorator. Usage:

@iparam(mac=MACAddressParameter(), count=IntParameter(default=3)) def iparam_test(mac, count):

return (mac, count)

iparam_test(mac="1:2:3:4:5:6", count="7") ("01:02:03:04:05:06", 7)

4.1.2 HTTP Web API

Введение

1

Glossary

5.1 Глоссарий

Python Язык программирования общего назначения

Модуль Термин Python, обозначающий коллекции функция и методов. В простом случае представляет собой файл с расширением .py

Пакет Термин Python, набор модулей. В простом случае представляет собой папку с файлами модулей.

Профиль (SA Profile, Профиль оборудования) Компонент НОКа обеспечивающий описание работы с оборудованием. Профили находится в папке sa/profiles/<VendorName>/<SystemName>

Интерфейс Интерфейс - это компонент Совы, предназанченная для обеспечение взаимодействия между компонентами.

Объект (объект управления) Основная единица с которой работает NOC. Находится в Service Activation –> Management Object

Профиль оборудования (Object Profile)

Профиль интерфейса - (Interface Profile)

Indices and tables

- genindex
- modindex
- search

Discussion and support

Site Telegram

n

Symbols

B

C

D

E

F

G

H

I

L

M

N

O