XMC 8.5 Workshop
Python implementation

Markus Nikulski
Sr. Corporate System Engineer

October 2020

Extreme

Python Script vs Workflow

single Python Script

- all in one script
  - easy searching
  - easy local data sharing
- Script exist as file
- Script output exist as file

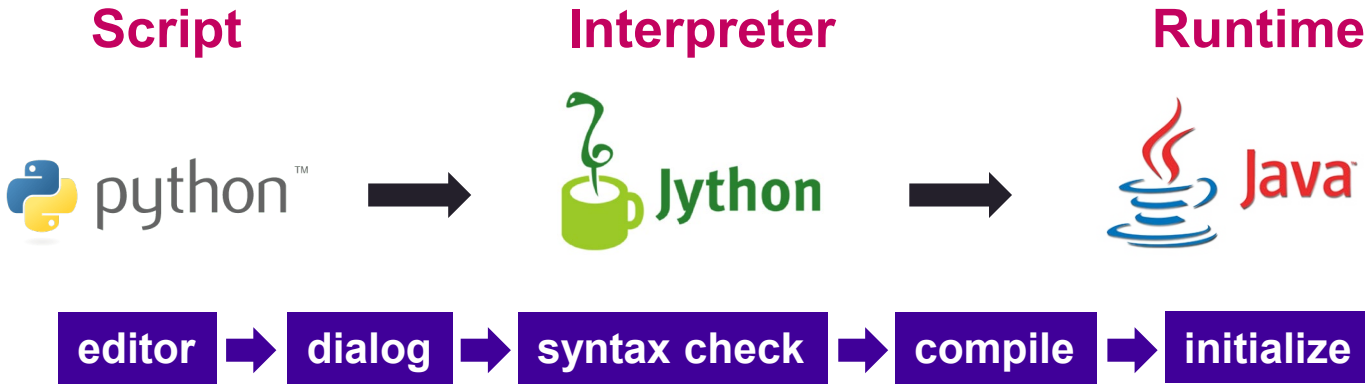
based on Python 2.7.x

Workflow

- chain multiple small Python scripts
- data exchange between scripts
- e-mail support
- event / syslog support
- HTTP(s) GET/PUT support
- Shell support (BASH)
- Dashboard
- Advanced execution (NBI/Alarm/NAC)
- requires XMC advance license

https://emc.extremenetworks.com/
https://emc.extremenetworks.com/content/oneview/docs/tasks/docs/c_workflows.html

How XMC executes Python code



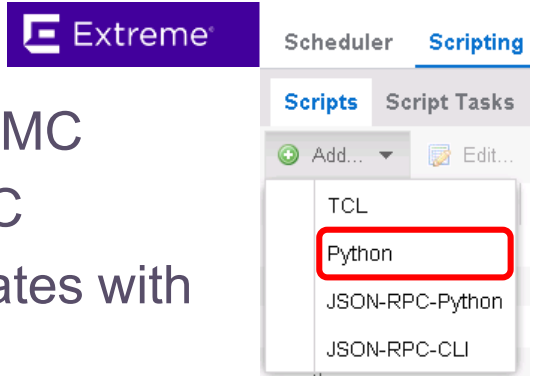
1. Process metadata	user dialog
2. replace metadata with user input	update variables
3. Syntax validation	
4. Inject extensions (internal APIs)	emc_vars / emc_cli / emc_nbi / emc_results
5. Compile to a Java class	
6. Execute in Java class	running in WildFly (non commercial JBOSS)

Python debugger can not be used

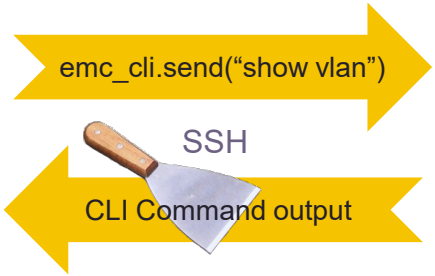
Management Center Python Script

Python Script

- Python script entered in XMC
- Python script runs on XMC
- Python scripts communicates with devices via CLI

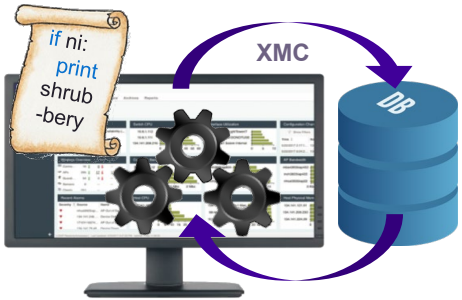


XMC



XOS
VSP
ERS
SLX
VDX
3rd party

XMC Python API: emc_vars



XMC DB populated via network discovery

XOS
VSP
ERS
SLX
VDX
3rd party

```
myVar = emc_vars[key]
```

Where **key**:

serverIP	server IP address
serverVersion	server version
serverName	server host name
time	current date at server (yyyy-MM-dd)
date	current time at server (HH:mm:ss z)
username	EMC user name
userDomain	EMC user domain name
auditLogEnabled	true/false if audit log is supported
scriptTimeout	max script timeout in secs
scriptOwner	scripts owner
deviceName	DNS name of selected device
deviceIP	IP address of the selected device
deviceId	device DB ID
deviceLogin	login user for the selected device
devicePwd	logon password for the selected device
deviceSoftwareVer	software image version number on the device
deviceType	device type of the selected device
deviceSysOid	device system object id
deviceVR	device virtual router name
cliPort	telnet/ssh port
isExos	true/false. Is this device an EXOS device?
family	device family name
vendor	vendor name
deviceASN	AS number of the selected device port selected ports
vrName	selected port(s) VR name ports all device ports
accessPorts	all ports which have config role access
interSwitchPorts	all ports which have config role interswitch
managementPorts	all ports which have config role management

emc_vars

```
print emc_vars["deviceIP"]

newVar = emc_vars["deviceIP"]

emc_vars["deviceIP"] = "1.2.3.4"
```

build in advance when the script is executed

depends on the context (Device, NAC, Alarm,…)

never write in **emc_vars**, always work with a copy

```
print emc_vars


import json
print json.dumps( emc_vars, sort_keys=True, indent=4 )
```

expose the **emc_vars** content

XMC Python API: emc_vars

Edit Script: Test (Python)

Overview | Content | Description | Run-Time Settings | Permissions

```
1  ##################################################################
2  # global variables
3  #@MetaDataStart
4  #@VariableFieldLabel (description   = "VLAN ID <101 - 199>",
5  #                     type          = string,
6  #                     required      = yes,
7  #                     readOnly      = no
8  #                     )
9  set var input_vlan_id 101
10 # @VariableFieldLabel (description  = "Action",
11 #                      type         = String,
12 #                      required     = yes,
13 #                      readOnly     = no
14 #                      validValues  =[add,delete])
15 set var vlan_action add
16 #@MetaDataEnd
17
18 vlan_id = int( emc_vars['input_vlan_id'] )
19
20 # validate user input
21 if vlan_id < 101 or vlan_id > 199:
22     raise SystemExit("ERROR: VLAN-ID "+ str(vlan_id) +"is out of range")
23 else:
24     print "INFO: "
25
```

Run Script: Test

1. Device Selection | 2. Device Settings | 3. Run-Time Settings | 4. Verify Run Script | 5. Results

These parameters (if any) will be passed to the script during execution. If no parameters are shown, just skip to the next step.

Overview | Description

Default

VLAN ID <101 - 199>: 123

Action: add

Results

Date and Time: 2018-02-06T11:45:40.359
EMC User: root
EMC User Domain:
IP: 172.16.10.56
INFO: add VLAN-ID 123

vlan_id = emc_vars['input_vlan_id']

You will get the default value, not the user input!

vlan_id = input_vlan_id

build-in Python classes

Using device CLI (SSH / telnet)

send command

```
result = emc_cli.send("show telnet sessions")
```

handle errors (transport only)

```
result.isSuccess()
result.getError()
```

true/false
string

get returned data

```
cli_lines = result.getOutput()
```

string
multi line

```
OOB-R11# show telnet sessions
Session/Unit  Host
------------  ------------------
1/1           192.168.1.30
OOB-R11#
```

```
show telnet sessions
Session/Unit  Host
------------  ------------------
1/1           192.168.1.30
OOB-R11#
```

send command

CLI prompt

Using device CLI (SSH / telnet)

```
emc_cli.send('disable telnet')
```

just execute
not recommended

```
cli_results = emc_cli.send('show telnet’)

result = cli_results.getOutput()
```

get a object back
recommended

```
output = emc_cli.send('show telnet').getOutput()
```

get a object back
not recommended

Using device CLI (SSH / telnet)

```
def sendCmd(cmd):

    cli_results = emc_cli.send( cmd )

    if cli_results.isSuccess() is False:
        print 'CLI-ERROR: ' + cli_results.getError()
        return False

    return cli_results.getOutput().splitlines()[1:-1]
```

send command

is the transport okay?

catch error message

separate each line
ignore first and last line

```
resultLines = sendCmd("show telnet sessions")

for line in resultLines:
    print "> " + line
```

```
> Session/Unit  Host
> ------------  ------------------
> 1/1           192.168.1.30
```

Using device CLI (SSH / telnet)

```
def sendConfigCmds(cmds):
    for cmd in cmds:
        cli_results = emc_cli.send( cmd )

        if cli_results.isSuccess() is False:
            print 'CLI-ERROR: ' + cli_results.getError()
            return False

    return cli_results.getOutput().splitlines()[1:-1]
```

loop through the list
send command

is the transport okay?

provide **last** result

```
config = '''
enable
configure terminal
username <NAME> <NAME> rw
'''

config = config.replace('<NAME>', userName)

sendConfigCmds( config.splitlines() )
```

multi line string

replace string with string

separate each line
and call the function

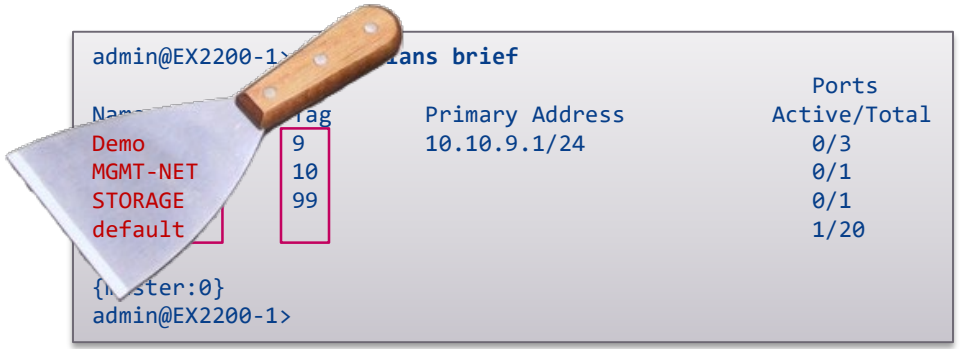
catch CLI output using split by spaces

```
telnet@MLXe-DC1> show lldp neighbors
Total number of LLDP neighbors on all ports: 6
Lcl Port  Chassis ID      Port ID          Port Description          System Name
1/1       c4f5.7ca7.67e5  HundredGigabit~                            BLEAF4
1/2       0024.387c.b400  0024.387c.b401   100GigabitEthernet1/2    MLXe-DC2
2/1       001f.123f.c400  ge-0/0/22        ge-0/0/22.0              EX4200-VCF
2/2       8418.88a8.1040  ge-0/0/22        ge-0/0/22.0              EX2200-1
2/3       0004.969e.a076  23               38 !                     x460-G2
3/20      0cc4.7ace.57ee  0cc4.7ace.57f0   dp0o3                    localhost
telnet@MLXe-DC1>
```

```
def getLldpNeighborList():
  lldp_list = {}
  cmds = []

  cmds.append( 'show lldp neighbors' )

  cli_result = sendConfigCmds( cmds )

  if cli_result:
    for line in cli_result:
      if any(i.isdigit() for i in line):                    # is a number in string contained?
        text_block = line.split()                           # separate text in blocks
        if text_block and text_block[0] != text_block[-1]:  # at-least two columns in table
          lldp_list[ text_block[0] ] = text_block[-1]       # take first and last block
  else:
    print "ERROR: unable to fetch LLDP Neighbor list“

  return lldp_list
```


catch CLI output using REGEX

CLI scraping



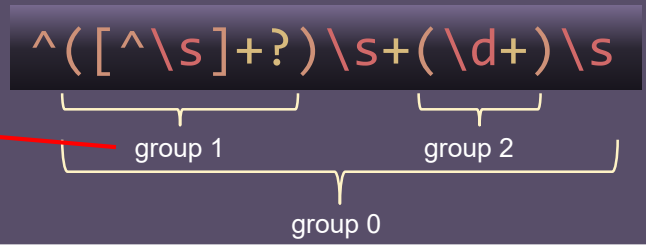

```python
def getVlanList():
  regex = re.compile(r"^([^\s]+?)\s+(\d+)\s")
  vlans = {}
  cmds  = []

  cmds.append( 'show vlans brief' )

  cli_result = sendConfigCmds( cmds )

  for line in cli_result:
      result = regex.search( line )
      if result:
        vlanName = str( result.group(1) )
        vlanId   = int( result.group(2) )
        vlans[vlanName] = vlanId

  return vlans
```

Regular Expression (REGEX)

Anchor
- `^` begin of the string

Quantifier
- `+` one or more
- `+?` one or more (not so greedy)

matching Operator
- `\s` space or tab
- `\d` digit (number)
- `[^\s]` negate (anything except space)



XMC Python API: emc_cli

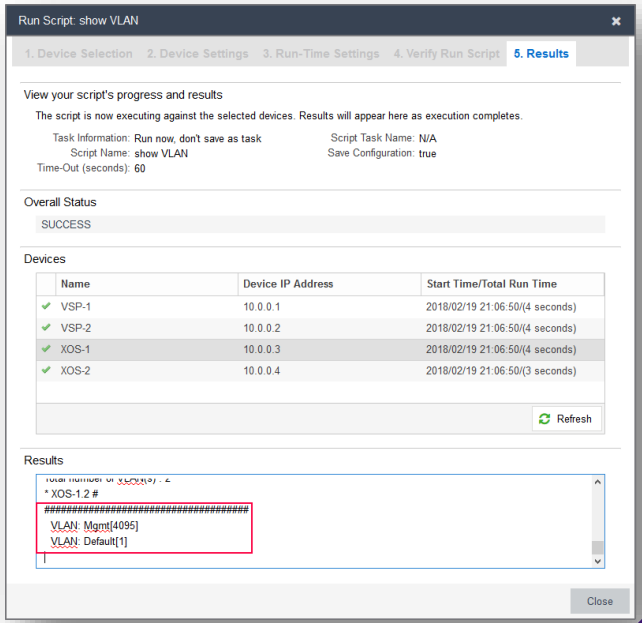
- Example to get all VLANs from device

collect VLAN information
from VOSS and EXOS via CLI

```python
def sendConfigCmds(cmds):
  for cmd in cmds:

    cli_results = emc_cli.send( cmd )

    if cli_results.isSuccess() is True:
      return cli_results.getOutput().splitlines()
    else
      print 'CLI-ERROR: ' + cli_results.getError()
      return False
```

```python
def getVlanList():
  import re

  voss_re  = re.compile(r"^(\d+)\s+([^\s]+?)\s")
  exos_re  = re.compile(r"^([^\s]+?)\s+(\d+)\s")
  vlanId   = 0
  vlanName = ''
  vlans    = {}
  cmds     = []

  if emc_vars["family"] == "VSP Series":
    cmds.append('enable')
    cmds.append('show vlan basic')
  if emc_vars["family"] == "Summit Series":
    cmds.append('show vlan')

  cli_result = sendConfigCmds( cmds )

  if cli_result:
    for line in cli_result:
      if emc_vars["family"] == "VSP Series":
        result = voss_re.search(line)
        if result:
          vlanId   = int( result.group(1) )
          vlanName = str( result.group(2) )

      if emc_vars["family"] == "Summit Series":
        result = exos_re.search(line)
        if result:
          vlanId   = int( result.group(2) )
          vlanName = str( result.group(1) )

      if result:
        vlans[vlanName] = vlanId

  return vlans
```

```python
vlans = getVlanList()

print '####################################'

for vlanName,vlanId in vlans.iteritems():
  print " VLAN: " + vlanName + "[" + str(vlanId) + "]"
```



Platform

CLI commands

Regular expression

Device CLI handling

Using device CLI (SSH / telnet)

don't wait for the prompt

```
emc_cli.send("reboot", False )
```

change the default timeout (30)

```
emc_cli.setSessionTimeout(60)
```

get current prompt recognition

```
emc_cli.getCliRule()
```

change prompt recognition

```
emc_cli.setCliRule("Avaya (Rapid City)")
```

close CLI session

```
emc_cli.close()
```

Please be aware that only up to **520 kB** CLI output can be handled.
In example **show tech** will not work for some platforms

Using device CLI

change device session context

```
emc_cli.send("show vlan brief")
```

open the session
use the device context prompt detection

```
emc_cli.close()
emc_cli.setCliRule(None)
emc_cli.setSSHEnabled(None)
```

close the session
release the prompt detection
release session type

```
emc_cli.setIpAddress("20.0.20.32")
emc_cli.setCliRule("Avaya (SynOptics)")

emc_cli.send("show vlan brief")
```

change the device context
only if really needed!

Python script result handling

Workflows script **emc_results**

```
emc_results.put("key", "value")
```

Write back to workflow variables
Not for large data!

```
emc_results.setOutput( "text" )
```

Overwrite all activity output
including device CLI output

```
status = emc_results.Status
```

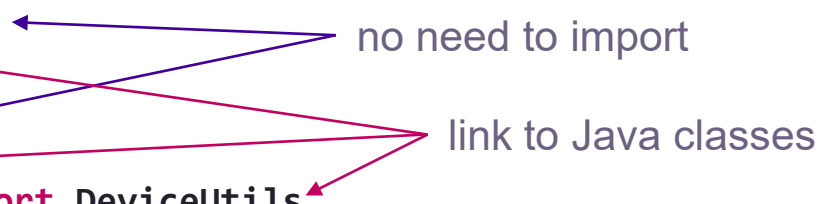
catch status object

```
emc_results.setStatus( status.ERROR )
emc_results.setStatus( status.CANCELLED )
emc_results.setStatus( status.SUCCESS )
emc_results.setStatus( status.COMPLETED )
```

Overwrite the result

System Python Scripts/Workflows (unsupported)

```
from xmclib import emc_vars
from xmclib import logger
from xmclib import cli
from device import api
from device.deviceutils import DeviceUtils

try:
    family = api.get_device_family()
    if family and family != DeviceUtils.UNKNOWN_FAMILY:
        emc_results.put("deviceFamily", family)
    else:
        emc_results.setStatus(emc_results.Status.ERROR)
except Exception as e:
    cli.process_exception(e, emc_results)
```



The use require inside knowledge need you don’t have.
Guessing and reverse engineering isn’t a good choice.

Debugging

Debugging

dump **emc_vars** dictionary

```
import json

print json.dumps(emc_vars, sort_keys=True, indent=4)
```

```
Script Name: Dump emc_vars
Date and Time: 2019-11-19T11:58:20.446
XMC User: mnikulski
XMC User Domain:
IP: 20.0.20.41
{
    "STATUS": "1",
    "abort_on_error": "true",
    "accessPorts": "",
    "auditLogEnabled": "",
    "date": "11/19/2019 11:58:19 AM",
    "deviceASN": "4261418025",
    "deviceCliType": "SSH",
    "deviceIP": "20.0.20.41",
    "deviceId": "244",
    "deviceLogin": "rwa",
    "deviceName": "VSP4450-1",
    "devicePwd": "rwa",
    "deviceSoftwareVer": "8.0.0.0",
    "deviceSysOid": "1.3.6.1.4.1.2272.206",
    "deviceType": "VSP-4450GSX-PWR+",
    "family": "VSP Series",
    "interSwitchPorts": "",
```

Debugging

write your own log file

```
1   import os
2   import time
3   import logging
4
5   loggFileName  = emc_vars['deviceIP'].replace('.','_') + '-' + time.strftime("%Y%m%d-%H%M%S") + '.txt'
6   loggDirectory = str( os.path.abspath( os.path.join( emc_vars['jboss.server.log.dir'] + '/' )))
7
8   logging.basicConfig(level    = logging.DEBUG,
9                       format   = '%(asctime)s,%(msecs)-3d %(levelname)-8s [%(filename)s:%(lineno)d] %(message)s',
10                      filename = loggDirectory + loggFileName,
11                      filemode = 'w')
12
13  logging.debug('This message should go to the log file')
14  logging.info('So should this')
15  logging.warning('And this, too')
16  logging.error('this is not good')
17  logging.critical('even worse')
```

overwrite the file
'w+' will append

10-0-8-11_20191112-134855.txt

```
2019-11-13:12:16:53,470 DEBUG    [test.py:13] This message should go to the log file
2019-11-13:12:16:53,470 INFO     [test.py:14] So should this
2019-11-13:12:16:53,470 WARNING  [test.py:15] And this, too
2019-11-13:12:16:53,471 ERROR    [test.py:16] this is not good
2019-11-13:12:16:53,471 CRITICAL [test.py:17] even worse
```

tips & tricks

tip & tricks

will not work under XMC

```
import sys

if emc_vars["family"] == 'Summit Series':
    print "INFO: detect EXOS switch"
else:
    print "WARNING: device family '%s' is not supported" % emc_vars["family"]
    sys.exit(1)
```

Recommended under XMC

```
#####################################################
def main():

    if emc_vars["family"] == 'Summit Series':
        print "INFO: detect EXOS switch"
    else:
        print "WARNING: device family '%s' is not supported" % emc_vars["family"]
        return
#####################################################

main()
```

tip & tricks

Using own **Python classes**

(for experts only)

is part of the advance section

Next Presentation

Use the following link to advance to the next PDF in the Workflow education presentation.

WWW.EXTREMENETWORKS.COM