nornir_pyez

Release 0.0.4

Knox Hutchinson

Feb 27, 2023

CONTENTS

CHAPTER

ONE

INSTALL

You can install nornir_pyez with

```
pip install nornir-pyez==0.2.8
```

CHAPTER

TWO

GUIDE

2.1 quickstart

1) Install Nornir_PyEZ:

```
pip install nornir-pyez==0.2.1
```

2) Import the Task you care about, such as collecting facts:

```
from nornir_pyez.plugins.tasks import pyez_facts
```

3) Use in a script:

```
from nornir_pyez.plugins.tasks import pyez_facts
from nornir import InitNornir
from rich import print
import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    pyez_facts
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

See Contacts for any issues

2.2 help

If you have any issues, please raise an issue at https://github.com/DataKnox/nornir_pyez

2.3 contact

- https://github.com/DataKnox/nornir_pyez
- https://dataknox.dev/
- https://youtube.com/c/dataknox
- https://twitter.com/data_knox

2.4 tasks

Here you will find a list of available methods and their corresponding documentation

2.4.1 pyez_facts

1. Import the Task you care about, such as collecting facts:

```
from nornir_pyez.plugins.tasks import pyez_facts
```

2. Use in a script:

```
from nornir_pyez.plugins.tasks import pyez_facts
from nornir import InitNornir
from rich import print
import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    pyez_facts
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

Output:

```
{'2RE': False,
'HOME': '/var/home/knox',
'RE0': {'last_reboot_reason': '0x1:power cycle/failure',
```

(continues on next page)

(continued from previous page)

```
        'mastership_state': 'master',
        'model': 'RE-SRX300',
        'status': 'OK',
        'up_time': '1 day, 26 minutes, 46 seconds'},
'RE1': None,
'RE_hw_mi': False,
'current_re': ['master',
               'node',
               'fwdd',
               'member',
               'pfem',
               'backup',
               'fpc0',
               're0',
               'fpc0.pic0'],
'domain': None,
'fqdn': 'Srx',
'hostname': 'Srx',
'hostname_info': {'re0': 'Srx'},
'ifd_style': 'CLASSIC',
'junos_info': {'re0': {'object': junos.version_info(major=(19, 3), type=R, minor=1,
build=8),
                       'text': '19.3R1.8'}},
'master': 'RE0',
'model': 'SRX300',
'model_info': {'re0': 'SRX300'},
'personality': 'SRX_BRANCH',
're_info': {'default': {'0': {'last_reboot_reason': '0x1:power cycle/failure',
                              'mastership_state': 'master',
                              'model': 'RE-SRX300',
                              'status': 'OK'},
                        'default': {'last_reboot_reason': '0x1:power '
                                                          'cycle/failure',
                                    'mastership_state': 'master',
                                    'model': 'RE-SRX300',
                                    'status': 'OK'}}},
're_master': {'default': '0'},
'serialnumber': 'CV3216AF0510',
'srx_cluster': False,
'srx_cluster_id': None,
'srx_cluster_redundancy_group': None,
'switch_style': 'VLAN_L2NG',
'vc_capable': False,
'vc_fabric': None,
'vc_master': None,
'vc_mode': None,
'version': '19.3R1.8',
'version_RE0': '19.3R1.8',
'version_RE1': None,
'version_info': junos.version_info(major=(19, 3), type=R, minor=1, build=8),
'virtual': False}
```

See contacts for support

2.4.2 pyez_get_config

1. Import pyez_get_config:

```
from nornir_pyez.plugins.tasks import pyez_get_config
```

2. This function can be sent naked in order to get the entire running config. The response is returned as a Dict:

```
from nornir_pyez.plugins.tasks import pyez_get_config
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_get_config
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

3. This function can be provided with parameters database and filter_xml , just as you would with PyEZ:

```
from nornir_pyez.plugins.tasks import pyez_get_config
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")
# Can use either an XPath or a Subtree
xpath = 'interfaces/interface'
xml = '<interfaces></interfaces>'
database = 'committed'

response = nr.run(
    task=pyez_get_config, filter_xml=xpath, database=database
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.3 pyez_checksum

Use this task to calculate the checksum (md5, sha or sha256) of a file in the remote Juniper device.

Example:

```
from nornir_pyez.plugins.tasks import pyez_checksum
import os
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result

script_dir = os.path.dirname(os.path.realpath(__file__))
filename = "<your_filename>"
path = "<your_path>"

nr = InitNornir(config_file=f"{script_dir}/config.yml"

response = netbox_inventory.run(
    task=pyez_checksum, filepath=f"{path}/{filename}", calc="sha256",
)
print_result(response)
for key in response.keys():
    remote_hash = response[key][0].result

return remote_hash
```

2.4.4 pyez_cmd

Use this task to execute single command on your devices. You can just display the whole result or parse it.

Example:

```
 from nornir_pyez.plugins.tasks import pyez_scp
 import os
 from nornir import InitNornir
 from nornir_utils.plugins.functions import print_result
 from rich import print

 script_dir = os.path.dirname(os.path.realpath(__file__))

 command = "show system information"

 nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = task.run(
     task=pyez_cmd,
     command=command
 )

 # Print the whole result
 print_result(response)

 # Or parse it (and do whatever you want)
```

(continues on next page)

(continued from previous page)

```
 for key in response.keys():
     result = response[key][1].result
```

2.4.5 pyez_int_terse

This is the equivalent to running "show interfaces terse" and receiving the result as a Dict

Example:

```
from nornir_pyez.plugins.tasks import pyez_int_terse
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_int_terse
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.6 pyez_rollback

Use this task to rollback to a previous configuration or delete a pending config in the candidate datastore and unlock for next config.

Example:

```
from nornir_pyez.plugins.tasks import pyez_rollback
import os
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_rollback,
    rollback_number=2
)

print_result(response)
```

2.4.7 pyez_route_info

This is the equivalent to running "show route" and receiving the result as a Dict. You can now add the following variable to filter the route :

> task: Task, all: bool = False, best: bool = False, brief: bool = False, detail: bool = False, exact: bool = False, hidden: bool = False, localization: bool = False, # get-fib-localization-information martians: bool = False, # get-route-martians private: bool = False, instance_name: str = "all", # get-instance-information protocol: str = "all", table: str = "all", rib_sharding: str = "main", destination: str = ""

Example:

```
from nornir_pyez.plugins.tasks import pyez_route_info
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_route_info
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.8 pyez_config

1. Begin by importing your method:

```
from nornir_pyez.plugins.tasks import pyez_config
```

2. Now you will need to decide which serialization that you want to send the data in with. 'text' is the default. Payload must be passed as a string A text payload example:

```
payload = """interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 3.3.3.3/32;
            }
        }
    }
}
"""
```

Take notice of the XML payload's ability to use attributes like operation="replace" This will remove and replace this subtree of the candidate config. An XML payload example:

```
xml_payload = """
<configuration>
        <interfaces>
                <interface>
                    <name>lo0</name>
                    <unit>
                        <name>0</name>
                        <family operation="replace">
                            <inet>
                                <address>
                                    <name>3.3.3.3/32</name>
                                </address>
                            </inet>
                        </family>
                    </unit>
                </interface>
            </interfaces>
</configuration>
"""
```

Note JSON is also a valid payload with data_format='json' set

3. If you want to load several configuration files, you will need to use the first_loading flag. This flag solve the problem of the lock on the Junos OS database when loading configuration. In your code you will need to set this flag as False at your second and more calls of the pyez_config task.

first_loading = True for conf_file in conf_files:

config_response = task.run(
task=pyez_config, template_path=conf_file, template_vars={},
first_loading=first_loading,

) first_loading = False

4. Next you need to decide if you want to commit the changes now, or create a new task to view the diff

Example of NO commit now:

```
send_result = task.run(
    task=pyez_config, payload=xml_payload, data_format='xml')
```

Example of commit now:

```
send_result = task.run(
    task=pyez_config, payload=xml_payload, data_format='xml', commit_now=True)
```

Full example:

```
from nornir_pyez.plugins.tasks import pyez_config
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")
```

(continues on next page)

(continued from previous page)

```
payload = """interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 3.3.3.3/32;
            }
        }
    }
}
"""

response = nr.run(
    task=pyez_config, payload=payload
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

Now supporting Templates The PyEZ Library uses a template like so. First let's explore the Jinja2 template:

```
set system name-server {{ dns_server }}
set system ntp server {{ ntp_server }}
```

We can retrieve this as arbitrary data from hosts or groups.yml:

```
---
junos_group:
    username: 'knox'
    password: 'juniper1'
    platform: junos
    data:
        dns_server: '10.10.10.189'
        ntp_server: 'time.google.com'
```

The official PyEZ method is typically written like so:

```
cu.load(template_path=CONFIG_FILE, template_vars=CONFIG_DATA, format='set', merge=True)
```

However the load method is replaced by pyez_config. Here is a sample script:

```
from nornir_pyez.plugins.tasks import pyez_config, pyez_diff, pyez_commit
import os
from nornir import InitNornir
from nornir.core.task import Task, Result
from nornir_utils.plugins.functions import print_result
from nornir_utils.plugins.tasks.data import load_yaml
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))
```

(continues on next page)

(continued from previous page)

```
nr = InitNornir(config_file=f"{script_dir}/config.yml")

def template_config(task):
    # retrieve data from groups.yml
    data = {}
    data['dns_server'] = task.host['dns_server']
    data['ntp_server'] = task.host['ntp_server']
    print(data)
    response = task.run(
        task=pyez_config, template_path='junos.j2', template_vars=data, data_format='set
↪')
    if response:
        diff = task.run(pyez_diff)
    if diff:
        task.run(task=pyez_commit)


response = nr.run(
    task=template_config)
print_result(response)
```

2.4.9 pyez_diff

Use this task to return a diff between the candidate datastore and committed datastore

Example:

```
from nornir_pyez.plugins.tasks import pyez_diff
import os
from nornir import InitNornir
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_diff
)

print_result(response)
```

2.4.10 pyez_commit

Use this task to commit the candidate datastore to the committed datastore. You can add an optionnal comment. Note this performs a commit check first and performs a Rollback upon failure

Example:

```
from nornir_pyez.plugins.tasks import pyez_diff
import os
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = nr.run(
    task=pyez_commit, comment="Your comment"
)

print_result(response)
```

2.4.11 pyez_rpc

This task is used to run any ad-hoc RPC using PyEZ

Example:

```
from nornir_pyez.plugins.tasks import pyez_rpc
import os
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yml")

extras = {
    "level-extra": "detail",
    "interface-name": "ge-0/0/0"
}


response = nr.run(
    task=pyez_rpc, func='get-interface-information', extras=extras)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

Of note: the func param takes a string that is the actual RPC name to be run. You can find this by typing your command on the Juniper CLI and then piping it to "display xml rpc". Try it out:

show interfaces ge-0/0/0 detail | display xml rpc

The results will look like:

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.2R1/junos"> <rpc>

<get-interface-information>
<level-extra>detail</level-extra> <interface-name>ge-0/0/0</interface-name>

</get-interface-information>

</rpc> <cli>

<banner></banner>

</cli> </rpc-reply>

You will want to use everything contained within the RPC tags. If there are additional items to specify, like level-extra and interface-name in this case, you can create a dictionary to contain them. These keys and values get unpacked at runtime by passing them in with the extras key:

response = nr.run(task=pyez_rpc, func='get-interface-information', extras=extras)

2.4.12 pyez_scp

Use this task to scp files on your devices (unfortunately there is no way to check if the copy was successfull or not, you will have to check manually)

Example:

```
from nornir_pyez.plugins.tasks import pyez_scp
import os
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from rich import print

script_dir = os.path.dirname(os.path.realpath(__file__))
filename = "<your_filename>"

nr = InitNornir(config_file=f"{script_dir}/config.yml")

response = task.run(
    task=pyez_scp,
    file=f"{script_dir}./files/{filename}",
    path=path,
)

print_result(response)
```

2.4.13 pyez_sec_ike

This is the equivalent to running "show security ike security-associations" and receiving the result as a Dict

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_ike
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

response = firewall.run(
    task=pyez_sec_ike
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.14 pyez_sec_ipsec

This is the equivalent to running "show security ipsec security-associations" and receiving the result as a Dict

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_ipsec
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

response = firewall.run(
    task=pyez_sec_ipsec
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.15 pyez_sec_nat_dest

This is equivalent to running "show security nat destination rule" on a Juniper SRX. Execution of this function will send the RPC call over the NETCONF API on your firewall, and handle the XML-to-JSON translation of the returned data.

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_nat_dest
from nornir import InitNornir

import os

# create an object that stores the path of working directory (pwd) of your script
script_dir = os.path.dirname(os.path.realpath(__file__))

# instantiate Nornir as an object named nr, point to your config file with
nr = InitNornir(config_file=f"{script_dir}/config.yaml")

# filter for a network device with the name of "katy"
firewall = nr.filter(name="katy")

# create the nornir task, storing the output our RPC function in an object named
↪"response"
response = firewall.run(
    task=pyez_sec_nat_dest
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
for dev in response:
    print(response[dev].result)
```

If you would like to specify a rule by it's name, pass the argument when your task is created. If this is omitted from the task, the assumption is that you want all rules returned.

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_nat_dest
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

response = firewall.run(
    task=pyez_sec_nat_dest,
    rule="r1"
)
```

(continues on next page)

(continued from previous page)

```
for dev in response:
    print(response[dev].result)
```

2.4.16 pyez_sec_nat_src

This is equivalent to running "show security nat source rule" on a Juniper SRX. Execution of this function will send the RPC call over the NETCONF API on your firewall, and handle the XML-to-JSON translation of the returned data.

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_nat_src
from nornir import InitNornir

import os

# create an object that stores the path of working directory (pwd) of your script
script_dir = os.path.dirname(os.path.realpath(__file__))

# instantiate Nornir as an object named nr, point to your config file with
nr = InitNornir(config_file=f"{script_dir}/config.yaml")

# filter for a network device with the name of "katy"
firewall = nr.filter(name="katy")

# create the nornir task, storing the output our RPC function in an object named
↪"response"
response = firewall.run(
    task=pyez_sec_nat_src
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
for dev in response:
    print(response[dev].result)
```

If you would like to specify a rule by it's name, pass the argument when your task is created. If this is omitted from the task, the assumption is that you want all rules returned.

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_nat_src
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

response = firewall.run(
```

(continues on next page)

(continued from previous page)

```
    task=pyez_sec_nat_src,
    rule="r1"
)

for dev in response:
    print(response[dev].result)
```

2.4.17 pyez_sec_policy

This is the equivalent to running "show security policies" and receiving the result as a Dict

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_policy
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

response = firewall.run(
    task=pyez_sec_policy
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

2.4.18 pyez_sec_zones

This is the equivalent to running "show security zones" and receiving the result as a Dict

Example:

```
from nornir_pyez.plugins.tasks import pyez_sec_zones
from nornir import InitNornir

import os

script_dir = os.path.dirname(os.path.realpath(__file__))

nr = InitNornir(config_file=f"{script_dir}/config.yaml")

firewall = nr.filter(name="katy")

```

(continues on next page)

(continued from previous page)

```
response = firewall.run(
    task=pyez_sec_zones
)

# response is an AggregatedResult, which behaves like a list
# there is a response object for each device in inventory
devices = []
for dev in response:
    print(response[dev].result)
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search