User Manual

Studienarbeit FS-2020

26. Mai 2020

Autoren:

Mike Schmid
mike.schmid@hsr.ch

Janik Schlatter
janik.schlatter@hsr.ch

Supervisors:

Prof. Stettler Beat
beat.stettler@hsr.ch

Baumann Urs
urs.baumann@hsr.ch

Inhaltsverzeichnis

1 Installation

1.1 Installationsvoraussetzungen

NUTS2.0 sollte sich auf jedem Betriebssystem ausführen lassen. Für die Ausführung wird eine Python 3.7 installation (oder aktueller) benötigt. Diese kann unter https://www.python.org/downloads/ heruntergeladen werden.

Um NUTS2.0 ausführen zu können, muss zuerst das Github-Repository geklont werden: https://github.com/EkoGuandor229/Network-Unit-Testing. Danach können die verwendeten Module über das Requirements-File mit dem Befehl 'pip install requirements.txt' installiert werden.

1.2 Ausführen

Das Programm kann zu testzwecken regulär in einer Programmierumgebung wie zum Beispiel PyCharm ausgeführt werden.

Um NUTS2.0 aus einer Konsole zu starten muss zum Root-Ordner NUTS2.0 navigiert werden. Man kann das Programm mit dem Befehl: 'python -m nuts' starten. Wenn man das GUI auslassen und direkt alle Tests ausführen möchte kann man mit dem Befehl: 'python -m nuts -r' starten.

1.2.1 Konfiguration

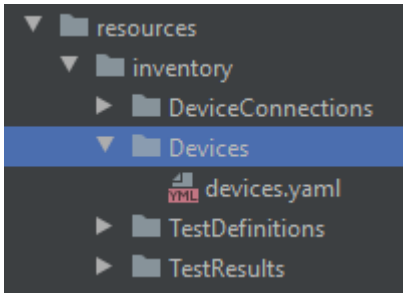
Im File 'Config.yaml' können die Pfäde aller Ordner geändert werden, um beispielsweise das Inventory oder die Resultate zentral in einem Repository zu verwalten. Zusätzlich kann noch bestimmt werden, ob das GUI Per default übersprungen werden soll.

2 Inventar

Um Netzwerktests auszuführen benötigt man zuerst ein Inventar mit Devices und Device Connections. Die Devices sind die Netzwerkgeräte wie zum Beispiel Router oder Switches. Die Device Connections sind die Verbindungen zwischen den Devices.

2.1 Devices

Die Definitionen der Devices sind unter Resources/Inventory/Devices/Devices.yaml abgelegt:



Um neue Devices zu erfassen müssen folgende Informationen im yaml eingegeben werden:

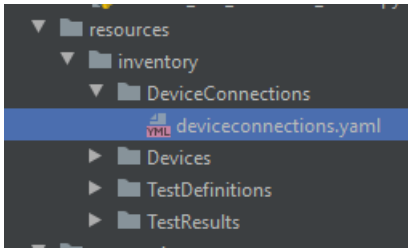
Attribut	Beschreibung
device_id	Eine eindeutige ID für das Device
platform	Das OS welches das Device benutzt
username	Der username welches das Device für das Login benutzt
password	Das Passwort welches das Device für das Login benutzt
hostname	Die Ip Adresse über welche das Device angesprochen werden kann
loopback-addresse	IP-Addresse des Loopback-Interface. Für einige Tests benötigt.

Diese Informationen sollten gemäss folgendem Beispiel dargestellt werden:

```
router01:
  - router01
  - cisco_ios
  - Cisco
  - cisco
  - 10.20.0.31
  - 172.16.255.1
```

2.2 Device Connections

Die Definitionen der Device Connections sind unter Resources/Inventory/DeviceConnections/DeviceConnections.yaml abgelegt



Um Device Connections zu erfassen müssen folgende Informationen im yaml eingegeben werden:

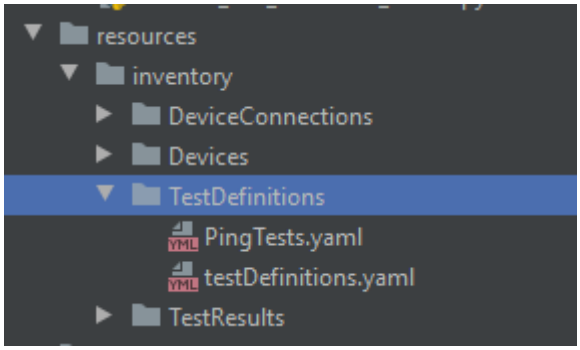
Attribut	Beschreibung
device a	Die ID des ersten Devices
device b	Die ID des zweiten Devices
connection speed	Die Übertragungsrate der Verbindung

Diese Informationen sollten wie folgt dargestellt werden:

```
connection1:
  - router01
  - router02
  - 100Mbit
```

3 Netzwerktests

Die Netzwerktests sind die Tests, welche effektiv auf dem zu testenden Netzwerk ausgeführt werden sollen. Die Testdefinitionen befinden sich unter Resources/Inventory/TestDefinitions:



Es können in diesem Ordner beliebig viele YAML Files abgelegt werden und es werden vonn allen Files die Tests erfasst.

Um die Tests zu erfassen müssen folgende Informationen im yaml eingegeben werden:

Attribut	Beschreibung
test_id	Eine eindeutige ID für den Test
command	Ein Command um den Test zu bestimmen
test_device	Die ID des Devices auf welchem der Test ausgeführt werden soll
target	Das Ziel (Zum Beispiel eine IP im Falle eines Pings)
expected_result	Das erwartete Resultat (Zum Beispiel Success im Falle eines Pings)
test_group	Ein Gruppenname um später die Tests zu Kategorisieren

Diese Informationen sollten wie folgt dargestellt werden:

```
test01:
  - PingRouter1Loopback0
  - Ping
  - router01
  - 172.16.255.1
  - Success
  - connectivity
```

3.1 Commands

Folgende Commands sind bereits implementiert und können mit den jeweiligen expected_results verwendet werden:

3.1.1 Ping

Führt einen Ping-Test auf das spezifizierte Target mit 4 ICMP Packeten aus. In der jetzigen Konfiguration wird eine 100% Erfolgsquote erwartet, damit der Ping-Test als erfolgreich gilt. Wenn andere Werte erwartet werden, muss dafür ein neuer Ping-Test implementiert werden, welche unterschiedliche Erwartungswerte implementiert hat.

Als erwartetes Resultat kann 'Success' oder 'Failure' verwendet werden.

3.1.2 Show Interfaces

Der 'Show Interfaces'-Befehl benötigt kein Target, da die Interfaces des test_device abgefragt werden. Bei der Definition kann somit einfach 'No Target' eingegeben werden. Als erwartetes Resultat wird ein Dictionary mit key:'Interfacename' und value:'True' oder 'False' verwendet werden. Dies sollte in folgender Form dargestellt werden:

```
- {
    'GigabitEthernet1': True,
    'GigabitEthernet2': True,
    'GigabitEthernet3': True,
    'GigabitEthernet4': True,
    'Loopback0': True
  }
```

3.1.3 Traceroute

Führt einen Traceroute vom test_device auf das in target angegebene Ziel aus. Als erwartetes Resultat wird ein Array von IP Adressen angegeben. Diese müssen in der Reihenfolge, in denen die Hops im Traceroute besucht werden, angegeben werden. Für Hops, die keine IP-Addresse anzeigen, kann ein '*' in das Array eingetragen werden.

Das Array bei expected_result kann beispielsweise so aussehen:

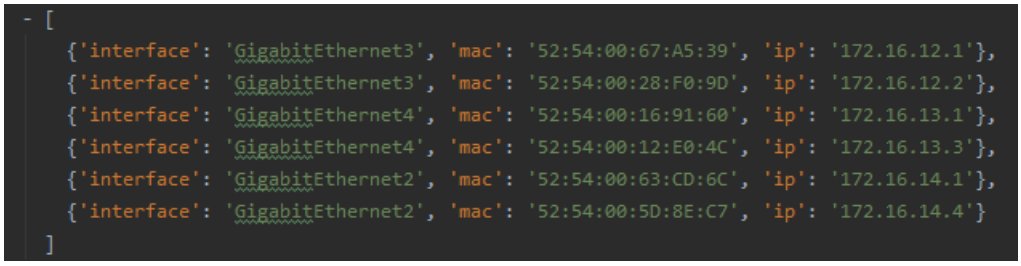
```
- ["172.16.13.1", "172.16.14.4"]
```

3.1.4 Arp Table

Für den Befehl 'Arp Table' wird kein Target benötigt, da der Arp Table des im test_device angegebenen Netzwerkgeräts abgefragt wird. Es kann 'No Target' im 'target'-Feld eingegeben werden. Als erwartetes Resultat wird ein Array von Dictionaries erwartet. In den Dictionaries werden folgende Informationen erwartet:

Parametername	Parameterwert
'interface':	Name des Interface.
'mac':	MAC-Addresse des Nachbargeräts
'ip':	IP-Addresse des Nachbargeräts

Im Bild ist ein Beispiel des Arrays mit den Dictionaries in jeder Zeile angegeben:



3.1.5 Ospf Neighbor

Für den Befehl 'Ospf Neighbor' wird kein Target benötigt, da OSPF Nachbarn des im test_device angegebenen Netzwerkgeräts abgefragt wird. Es kann 'No Target' im 'target'-Feld eingegeben werden. Als erwartetes Resultat wird ein Array mit Dictionaries erwartet. In den Dictionaries werden folgende Informationen benötigt:

Parametername	Parameterwert
'Neighbor-ID':	IP-Addresse des Nachbargeräts
'Priority':	OSPF-Priorität als Ganzzahl
'State':	Status des Nachbargeräts
'Address':	IP-Addresse des Interface, über welches der Nachbar erreichbar ist.
'Interface':	Name des Interface, über welches der Nachbar erreichbar ist.

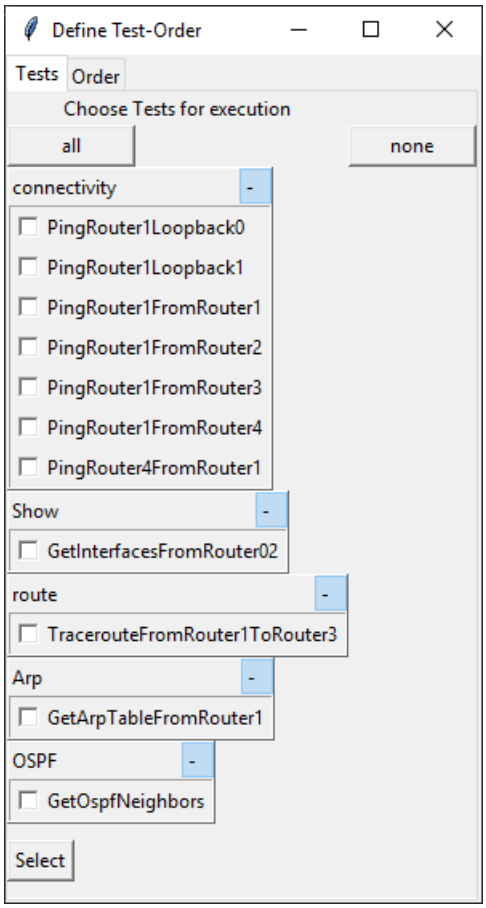
Das folgende Beispiel zeigt das Array von Dictionaries, wie es im YAML dargestellt wird:

4 Durchführung

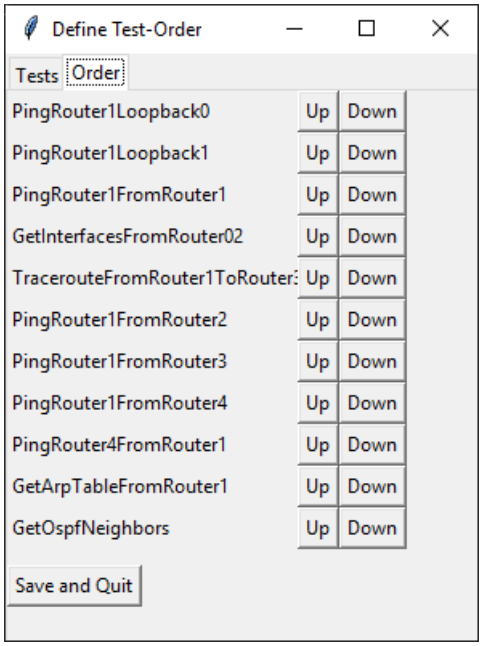
Nachdem dass Inventar erstellt und die Testdefinitionen erfasst wurden, kann man das Programm starten. Falls die Option Skip-GUI aktiviert wurde, werden alle Tests in der Reihenfolge durchgeführt, in der sie in der Testdefinition angegeben wurden. Falls dies nicht aktiviert wurde öffnet sich ein Grafikinterface.

4.1 GUI

Das GUI für die Definition der Test Reihenfolge besteht aus zweit Tabs:



Im ersten Tab werden alle Tests nach Gruppen sortiert angezeigt. Die Gruppierung ist diejenige, die in der Testdefinition angegeben wurde. Hier kann der Benutzer auswählen welche Tests er ausführen möchte. Nachdem die Tests ausgewählt wurden weden mit dem Button 'Select' alle ausgewählten Tests selektiert und man kann danach die Reihenfolge der selektierten Test im zweiten Tab einstellen.

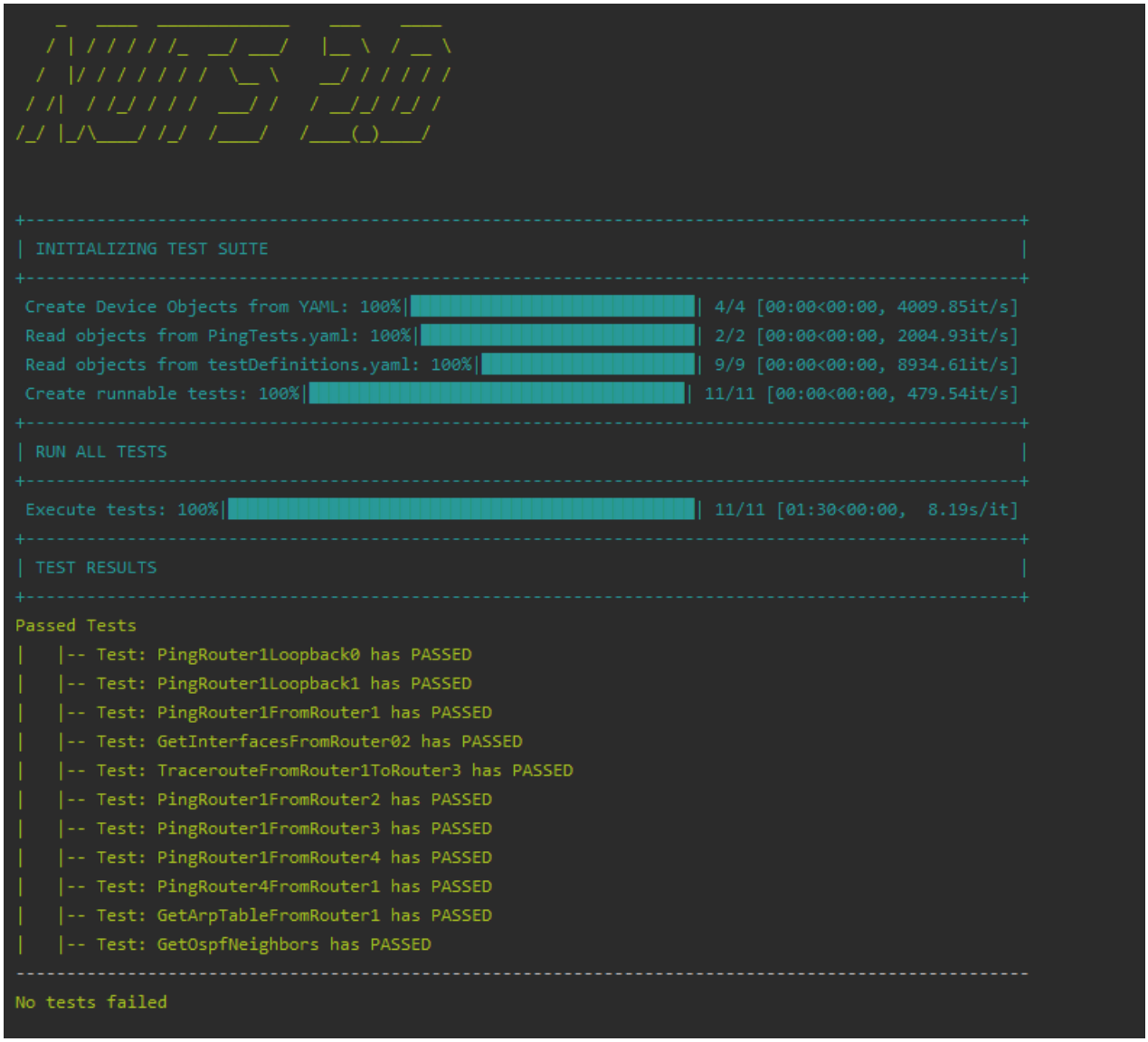


Im zweiten Tab werden alle selektierten Tests angezeigt und der Benutzer kann mit den jeweiligen Buttons die Reihenfolge bestimmen. Nachdem der Benutzer mit der Reihenfolge zufrieden ist, kann mit dem Button 'Save and Quit' das GUI beendet werden. Alle selektierten Tests werden nun in der angegebenen Reihenfolge durchgeführt.

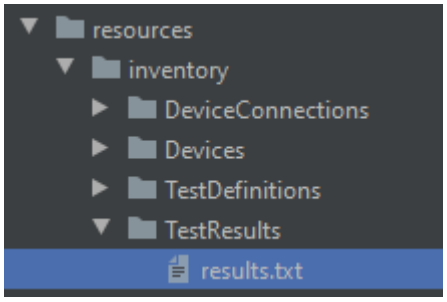
4.2 Test Resultate

Die Resultate der jeweiligen Durchführungen werden in der Konsole angezeigt und zusätzlich noch in einem File gespeichert. Bestandene Tests werden nur mit ihrem Namen und dem Vermerk, dass der Test bestanden ist, angegeben. Nicht bestandene Tests haben den Testnamen, das erwartete Ergebniss und das tatsächliche Ergebniss für einen soll-ist-Vergleich.

Das folgende Bild zeigt eine komplette Durchführung des Programms mit elf bestandenen Tests und null nicht bestandenen Tests:


INITIALIZING TEST SUITE
Create Device Objects from YAML: 100%| 4/4 [00:00<00:00, 4009.85it/s]
Read objects from PingTests.yaml: 100%| 2/2 [00:00<00:00, 2004.93it/s]
Read objects from testDefinitions.yaml: 100%| 9/9 [00:00<00:00, 8934.61it/s]
Create runnable tests: 100%| 11/11 [00:00<00:00, 479.54it/s]
RUN ALL TESTS
Execute tests: 100%| 11/11 [01:30<00:00, 8.19s/it]
TEST RESULTS
Passed Tests
|-- Test: PingRouter1Loopback0 has PASSED
|-- Test: PingRouter1Loopback1 has PASSED
|-- Test: PingRouter1FromRouter1 has PASSED
|-- Test: GetInterfacesFromRouter02 has PASSED
|-- Test: TracerouteFromRouter1ToRouter3 has PASSED
|-- Test: PingRouter1FromRouter2 has PASSED
|-- Test: PingRouter1FromRouter3 has PASSED
|-- Test: PingRouter1FromRouter4 has PASSED
|-- Test: PingRouter4FromRouter1 has PASSED
|-- Test: GetArpTableFromRouter1 has PASSED
|-- Test: GetOspfNeighbors has PASSED
No tests failed

Das File, in dem der Testreport abgespeichert wird, befindet sich unter: 'Resources/Inventory/TestResults/results.txt'.



In dem File wird zuerst der Zeitstempel der Testdurchführung angegeben, danach werden zuerst die bestandenen Tests und am Schluss die nicht bestandenen Tests angezeigt. Bei den nicht bestandenen Tests werden zusätzlich noch das erwartete und das tatsächliche Resultat angezeigt.

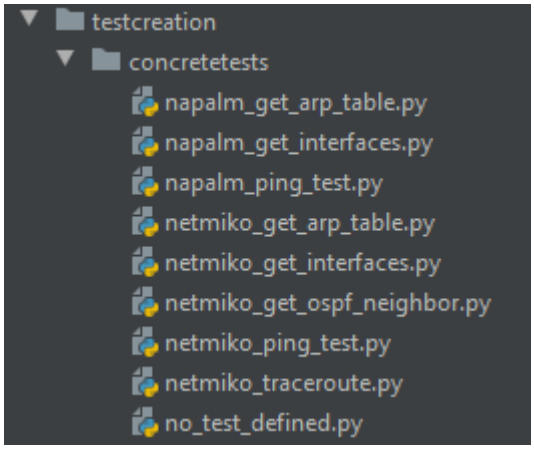
```
New Test Run on 2020-05-13 10:10:54.465216:
  Passed Tests:
    Test: PingRouter1Loopback0 has PASSED
    Test: PingRouter1Loopback1 has PASSED
    Test: PingRouter1FromRouter1 has PASSED
    Test: GetInterfacesFromRouter02 has PASSED
    Test: TracerouteFromRouter1ToRouter3 has PASSED
    Test: PingRouter1FromRouter2 has PASSED
    Test: PingRouter1FromRouter3 has PASSED
    Test: PingRouter1FromRouter4 has PASSED
    Test: PingRouter4FromRouter1 has PASSED

  Failed Tests:
    Test: GetArpTableFromRouter1 has FAILED
      Expected: [{'interface': 'GigabitEthernet3', 'mac': '52:54:00:67:A5:39', 'ip': '172.16.12.1'},
                 {'interface': 'GigabitEthernet3', 'mac': '52:54:00:28:F0:9D', 'ip': '172.16.12.2'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:16:91:60', 'ip': '172.16.13.1'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:12:E0:4C', 'ip': '172.16.13.3'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:63:CD:6C', 'ip': '172.16.14.1'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:5D:8E:C7', 'ip': '172.16.14.1'}]

      Actual:   [{'interface': 'GigabitEthernet3', 'mac': '52:54:00:67:A5:39', 'ip': '172.16.12.1'},
                 {'interface': 'GigabitEthernet3', 'mac': '52:54:00:28:F0:9D', 'ip': '172.16.12.2'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:16:91:60', 'ip': '172.16.13.1'},
                 {'interface': 'GigabitEthernet4', 'mac': '52:54:00:12:E0:4C', 'ip': '172.16.13.3'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:63:CD:6C', 'ip': '172.16.14.1'},
                 {'interface': 'GigabitEthernet2', 'mac': '52:54:00:5D:8E:C7', 'ip': '172.16.14.4'}]

End of Test Run
```

5 Neue Tests hinzufügen

Falls der Benutzer eigene Tests hinzufügen möchte, müssen an folgenden Orten änderungen vorgenommen werden:

5.1 Conctrete Tests

Die konkreten Tests befinden sich unter 'nuts/testcreation/concretetests' In diesem Ordner muss ein neues Python-File angelegt werden. Im File wird der Test als Klasse implementiert. Es ist darauf zu achten, dass das Basisinterface 'NetworkTestStrategyInterface' von der Testklasse implementiert wird, um davon die grundlegenden Funktionalitäten zu erben.



Nach der Erstellung muss der Test geschrieben werden. Dazu wird in der ___init___ Methode die Funktion self.nr. = InitNornir() aufgerufen und darin die benötigten Parameter übergeben.

In der Methode run_test(): wird angegeben, welches Nornir-Plugin mit welchem Task verwendet wird, z.B. task=napalm_get.

In der set_result() Methode wird die Logik für das Parsen des Rückgabewerts des Tests implementiert. Es ist darauf zu achten, dass dabei das Resultat in ein einheitliches Format gebracht wird, so dass man in der evaluate_result() Methode das erwartete Ergebnis möglichst mit einem == zum tatsächlichen Ergebnis vergleichen kann. Falls dies nicht möglich ist, muss im evaluate_result() zusätzlich Logik implementiert werden, um die Resultate zu vergleichen.

Mehr Informationen, welche Begehle mit Nornir ausgeführt werden können, findet man auf https://nornir.readthedocs.io

Informationen zum Napalm-Treiber findet man auf https://napalm.readthedocs.io

Die bereits erstellten Tests können als Vorlage für weitere Testimplementationen verwendet werden.

5.2 Network Test Strategy Factory

Die Network Test Strategy Factory implementiert die Logik, nach der die Tests ausgewählt und instanziert werden. Dafür werden sämtliche Tests in einer test_map gespeichert. Die test_map ist ein Dictionary von Dictionaries und hat als äusseren Key den Befehl, welcher Test ausgeführt werden soll und als inneren Key die Connection, die für den Test verwendet wird. Als Value ist die konkrete Klasse eingetragen, die für den Test instanziert werden soll. Gibt es für eine Kombination aus Command-Connection keinen konkreten Test, muss hier statt der Testklasse ein 'None' angegeben werden. Falls in der Instanzierungslogik ein Test, welcher in der Testdefinition angegeben wurde, nicht existiert, wird stattdessen ein NoTestDefined-Test instanziert, welcher in der Evaluation immer 'nicht bestanden' zurückgibt mit der Anmerkung, dass dieser Test noch nicht erstellt wurde.

Das folgende Bild zeigt die test_map mit den oben beschriebenen Werten.

```
def __init__(self):
    self.test_map = {
        "Ping": {
            "Napalm": NapalmPingTest,
            "Netmiko": NetmikoPingTest,
        },
        "Show Interfaces": {
            "Napalm": NapalmShowInterfaces,
            "Netmiko": NetmikoShowInterfaces
        },
        "Traceroute": {
            "Napalm": None,
            "Netmiko": NetmikoTraceroute
        },
        "Arp Table": {
            "Napalm": NapalmShowArpTables,
            "Netmiko": NetmikoShowArpTables
        },
        "Ospf Neighbor": {
            "Napalm": None,
            "Netmiko": NetmikoShowOspfNeighbor
        }

        # Add more Tests as "Testcommand: {connection_dictionary}
    }
```