

## ข้อมูลชุดคำสั่ง API ที่สามารถใช้งานได้ และตัวอย่างการใช้งาน

การใช้งาน API module ผู้ใช้จะต้องเข้าผ่านเว็บเบราว์เซอร์โดยมีรูปแบบ URL ดังนี้

<http://ControllerIp:5001/api/v1/Command/>

ตัวอย่าง: <http://10.50.34.15:5001/api/v1/device>

1. Controller Ip เป็นหมายเลขไอพีของเครื่องเซิร์ฟเวอร์ที่รันระบบคอนโทรลเลอร์อยู่ (ในแล็บนี้ใช้ไอพี 10.50.34.15)
2. Command เป็นกลุ่มชุดคำสั่งที่สามารถเรียกใช้ได้โดยตัวเลือกชุดคำสั่งมีดังต่อไปนี้
  - device
  - flow
  - link
  - path
  - flow/routing

### 1. รายละเอียดของคำสั่ง “device”

ผู้ใช้งานสามารถเรียกดูข้อมูลผ่านทาง <http://10.50.34.15:5001/api/v1/device> โดยหน้านี้จะแสดงข้อมูลพื้นฐานอุปกรณ์ทั้งหมดในเครือข่าย เช่น ชื่ออุปกรณ์ ชื่อ รายละเอียดต่างๆที่เกี่ยวข้องกับอินเตอร์เฟซ ข้อมูล SSH และสถานการณ์ทำงานของ SNMP CDP และ NetFlow โดยผู้ใช้สามารถเลือกดูข้อมูลอุปกรณ์ที่ต้องการแบบเฉพาะเจาะจงโดยการเพิ่ม /hostname ของอุปกรณ์ที่ต้องการลงไปต่อท้ายชุดคำสั่งข้างต้นได้ ตัวอย่าง <http://10.50.34.15:5001/api/v1/device/R1>

## ตัวอย่างการใช้งาน API device ผ่าน Python

```
import requests

#ทดลองดึงค่า management ip จากอุปกรณ์ทุกตัว
device_info = requests.get("http://10.50.34.15:5001/api/v1/device").json()
for device in device_info['devices']:
    print(device['management_ip'])
```

```
192.168.1.1
192.168.2.1
192.168.3.1
192.168.4.1
192.168.5.1
192.168.6.1
192.168.7.1
192.168.8.1
192.168.9.1
192.168.10.1
192.168.11.1
192.168.12.1
192.168.13.1
192.168.14.1
```

## 2. รายละเอียดของคำสั่ง “flow”

ผู้ใช้งานสามารถเรียกดูข้อมูลผ่านทาง <http://10.50.34.15:5001/api/v1/flow> โดยหน้านี้จะแสดงข้อมูลโฟลว์ที่ได้รับจากอุปกรณ์เครือข่าย เป็นรายละเอียดเกี่ยวกับข้อมูลที่มีการรับส่งผ่านอุปกรณ์เครือข่าย โดยข้อมูลเหล่านี้สามารถนำมาประยุกต์ใช้ในการบังคับเปลี่ยนเส้นทางเพื่อทำ Network Traffic Engineering ได้

## ตัวอย่างการใช้งาน API flow ผ่าน Python

```
import requests

flow_info = requests.get("http://10.50.34.15:5001/api/v1/flow").json()
for flow in flow_info['flows']:
    print(flow)
```

```
{'_id': {'$oid': '6234bb956b3e4f9c9c7de928'}, 'cisco_51': 0, 'direction': 0, 'dst_as': 0, 'dst_mask': 24, 'flow_sampler_id': 0, 'from_ip': '192.168.2.2', 'input_snmp': 2, 'ipv4_dst_addr': '192.168.10.1', 'ipv4_next_hop': '192.168.3.2', 'ipv4_src_addr': '10.50.34.15', 'l4_dst_port': 161, 'l4_src_port': 37786, 'output_snmp': 1, 'protocol': 17, 'src_as': 0, 'src_mask': 0, 'src_to': 0, 'tcp_flags': 16, 'last_switched': {'$date': '1647647504296'}, 'first_switched': {'$date': '1647647504296'}, 'in_bytes': 96, 'in_pkts': 1, 'created_at': {'$date': '1647623061105'}}
{'_id': {'$oid': '6234bb956b3e4f9c9c7de92e'}, 'cisco_51': 0, 'direction': 0, 'dst_as': 0, 'dst_mask': 24, 'flow_sampler_id': 0, 'from_ip': '192.168.2.2', 'input_snmp': 2, 'ipv4_dst_addr': '192.168.9.1', 'ipv4_next_hop': '192.168.3.2', 'ipv4_src_addr': '10.50.34.15', 'l4_dst_port': 161, 'l4_src_port': 52309, 'output_snmp': 1, 'protocol': 17, 'src_as': 0, 'src_mask': 0, 'src_to': 0, 'tcp_flags': 16, 'last_switched': {'$date': '1647647504296'}, 'first_switched': {'$date': '1647647504296'}, 'in_bytes': 96, 'in_pkts': 1, 'created_at': {'$date': '1647623061105'}}
{'_id': {'$oid': '6234bb956b3e4f9c9c7de934'}, 'cisco_51': 0, 'direction': 0, 'dst_as': 0, 'dst_mask': 24, 'flow_sampler_id': 0, 'from_ip': '192.168.2.2', 'input_snmp': 2, 'ipv4_dst_addr': '192.168.11.1', 'ipv4_next_hop': '192.168.3.2', 'ipv4_src_addr': '10.50.34.15', 'l4_dst_port': 161, 'l4_src_port': 48077, 'output_snmp': 1, 'protocol': 17, 'src_as': 0, 'src_mask': 0, 'src_to': 0, 'tcp_flags': 16, 'last_switched': {'$date': '1647647504384'}, 'first_switched': {'$date': '1647647504384'}, 'in_bytes': 96, 'in_pkts': 1, 'created_at': {'$date': '1647623061105'}}
{'_id': {'$oid': '6234bb976b3e4f9c9c7deab7'}, 'cisco_51': 0, 'direction': 0, 'dst_as': 0, 'dst_mask': 24, 'flow_sampler_id': 0, 'from_ip': '192.168.7.2', 'input_snmp': 2, 'ipv4_dst_addr': '192.168.12.1', 'ipv4_next_hop': '192.168.8.2', 'ipv4_src_addr': '10.50.34.15', 'l4_dst_port': 161, 'l4_src_port': 41206, 'output_snmp': 1, 'protocol': 17, 'src_as': 0, 'src_mask': 0, 'src_to': 0, 'tcp_flags': 16, 'last_switched': {'$date': '1647647504384'}, 'first_switched': {'$date': '1647647504384'}, 'in_bytes': 96, 'in_pkts': 1, 'created_at': {'$date': '1647623061105'}}
```

## 3. รายละเอียดของคำสั่ง “link”

ผู้ใช้งานสามารถเรียกดูข้อมูลผ่านทาง <http://10.50.34.15:5001/api/v1/link> โดยหน้านี้จะแสดงข้อมูลการเชื่อมต่อของอุปกรณ์ สามารถรู้ได้ว่าอุปกรณ์กำลังต่อกับอุปกรณ์อื่นใดบ้าง และสามารถแสดงข้อมูลปริมาณการใช้งานของแต่ละลิงก์ได้เช่นกัน

## ตัวอย่างการใช้งาน API link ผ่าน Python

```
import requests

link_info = requests.get("http://10.50.34.15:5001/api/v1/link").json()
for link in link_info['links']:
    print('This link is connect between', link['src_node_hostname'], ' and ', link['dst_node_hostname'])
    print('Connect with Port', link['src_port'], ' and ', link['dst_port'])
```

```
This link is connect between R1.pcn and R2.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R3.pcn and R1.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R2.pcn and R3.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R3.pcn and R4.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R4.pcn and R5.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R5.pcn and R6.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
This link is connect between R6.pcn and R7.pcn
Connect with Port GigabitEthernet0/0 and GigabitEthernet0/1
```

## 4. รายละเอียดของคำสั่ง “path”

ผู้ใช้งานสามารถเรียกดูข้อมูลผ่านทาง <http://10.50.34.15:5001/api/v1/path/srcip,dstip>

ตัวอย่าง <http://10.50.34.15:5001/api/v1/path/192.168.1.1,192.168.10.1> โดยหน้านี้จะ

สามารถแสดงข้อมูลเส้นทางที่เป็นไปได้ทั้งหมด จากไอพีต้นทาง และไอพีปลายทาง

## ตัวอย่างการใช้งาน API path ผ่าน Python

```
import requests

path_info = requests.get("http://10.50.34.15:5001/api/v1/path/192.168.1.1,192.168.10.1").json()
all_path = []
print('There are', len(path_info['paths']), 'possible paths')
for path in path_info['paths']:
    all_path.append(path['path'])
least_hop_path = min(all_path, key=len)
print('Lowest hop use =', len(least_hop_path), 'hops')
print('Path:', least_hop_path)
```

```
There are 2 possible paths
Lowest hop use = 10 hops
Path: ['192.168.1.1', '192.168.2.1', '192.168.3.1', '192.168.4.1', '192.168.5.1', '192.168.6.1', '192.168.7.1', '192.168.8.1', '192.168.9.1', '192.168.10.1']
```

ตัวอย่างทดลองหาเส้นทางจากไอพี 192.168.1.1 ไป 192.168.10.1 และหาเส้นทางที่ใช้ hop น้อยที่สุด

## 5. รายละเอียดของคำสั่ง “flow/routing”

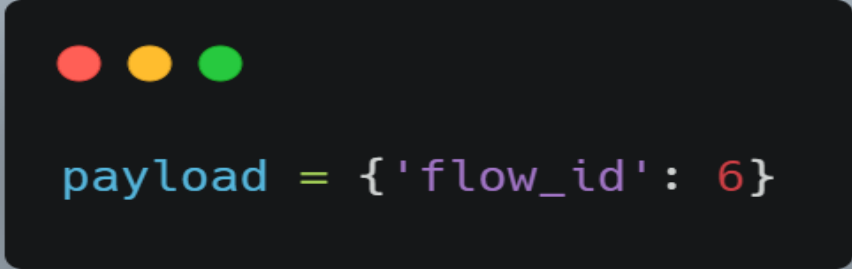
ผู้ใช้สามารถจัดการ policy based routing สำหรับการตั้งค่าเส้นทางโดยเฉพาะตามที่ต้องการได้ โดยผู้ใช้สามารถเรียกดู สร้าง และลบ policy ดังกล่าวผ่าน API นี้ได้โดย method GET POST DELETE ตามลำดับ ผ่าน URL <http://10.50.34.15:5001/api/v1/flow/routing> โดยพารามิเตอร์ที่ใช้สำหรับ DELETE และ POST method มีดังต่อไปนี้

Delete method parameter สำหรับลบ policy routing.

ผู้ใช้สามารถเรียกใช้ API โดยมี payload ดังต่อไปนี้

1. flow\_id: ไอดี policy ที่ต้องการลบ โดยผู้ใช้สามารถดูไอดีผ่าน GET method ของ flow/routing API

## ตัวอย่าง payload สำหรับ DELETE method



```
payload = {'flow_id': 6}
```

## POST method parameter สำหรับสร้าง policy routing.

ผู้ใช้สามารถเรียกใช้ API โดยมี payload ดังต่อไปนี้

1. name: ชื่อ policy สามารถตั้งอะไรก็ได้
2. src\_ip/dst\_ip: ไอพีเครือข่ายต้นทาง และปลายทาง
3. src\_port/dst\_port: พอร์ตต้นทาง และปลายทาง หากเป็นพอร์ตอะไรก็ได้ให้ใส่ any
4. src\_subnet/dst\_subnet: ไวลด์การ์ดต้นทาง และปลายทาง
5. actions: กำหนดอุปกรณ์ที่จะติดตั้ง policy และวิธีการรับมือของอุปกรณ์เหล่านั้นเมื่อมี Flow ตาม Policy ผ่านเข้ามา สามารถติดตั้งบนอุปกรณ์หลายตัวพร้อมกันได้โดยคำสั่งเดียว
  - device\_id: ไอดีของอุปกรณ์ สามารถดูได้จาก API device
  - action: วิธีการส่งข้อมูลตัวอย่างใช้ 2 หมายถึงส่งให้ Next Hop Ip (1 หมายถึง Next Hop Interface)
  - data: ข้อมูล Next hop ต้องสอดคล้องกับ action ถ้า action เป็น 1 ให้ใส่ชื่อ Next Hop Interface เป็นต้น

ตัวอย่าง payload สำหรับ POST method

```
payload = {  
    'name': 'Test Policy',  
    'src_ip': '192.168.1.1',  
    'src_port': 'any',  
    'src_subnet': '0.0.0.255',  
    'dst_ip': '192.168.5.1',  
    'dst_port': '8080',  
    'dst_subnet': '0.0.0.255',  
    'actions': [  
        {'device_id': '62349df46b3',  
         'action': 2,  
         'data': '192.168.2.1'},  
        {'device_id': 'atd54257s3',  
         'action': 2,  
         'data': '192.168.3.1'}  
    ]  
}
```

## ตัวอย่างการใช้งาน API Flow Routing ผ่าน Python

```
import requests

src_net = '192.168.200.0'
src_port = 'any'
src_wildcard = '0.0.0.255'
dst_net = '192.168.201.0'
dst_port = 'any'
dst_wildcard = '0.0.0.255'

for router in requests.get("http://10.50.34.15:5001/api/v1/device").json()['devices']:
    if router['name'] == 'R1.pcn':
        router_id = router['_id']['$oid']

action = [{'device_id': router_id, 'action': 2, 'data': '192.168.2.1'}]
payload = {'name': 'Test_policy', 'src_ip': src_net, 'src_port': src_port, 'src_subnet': src_wildcard,
           'dst_ip': dst_net, 'dst_port': dst_port, \
           'dst_subnet': dst_wildcard, 'actions': action}
requests.post("http://10.50.34.15:5001/api/v1/flow/routing", json=payload)
```

ตัวอย่างการใช้ POST method สำหรับสร้าง policy routing ตัวอย่างหาก Router R1 ได้รับโพล์ที่มีไอพีเครือข่ายต้นทางเป็น 192.168.200.0 ไอพีเครือข่ายปลายทางเป็น 192.168.201.0 พอร์ตต้นทางและปลายทางเป็นอะไรก็ได้ จะส่ง Flow ดังกล่าวไปยัง Next Hop IP ที่เป็น 192.168.2.1

```
import requests

all_policy = requests.get("http://10.50.34.15:5001/api/v1/flow/routing").json()['flows']
for policy in all_policy:
    if policy['name'] == 'Test_policy':
        policy_id = policy['flow_id']

payload = {'flow_id': policy_id}
requests.delete("http://localhost:5001/api/v1/flow/routing", params=payload)
```

ตัวอย่างการใช้ DELETE method โดยการลบ policy ที่มีชื่อว่า Test\_policy โดยดึง flow\_id จาก GET method