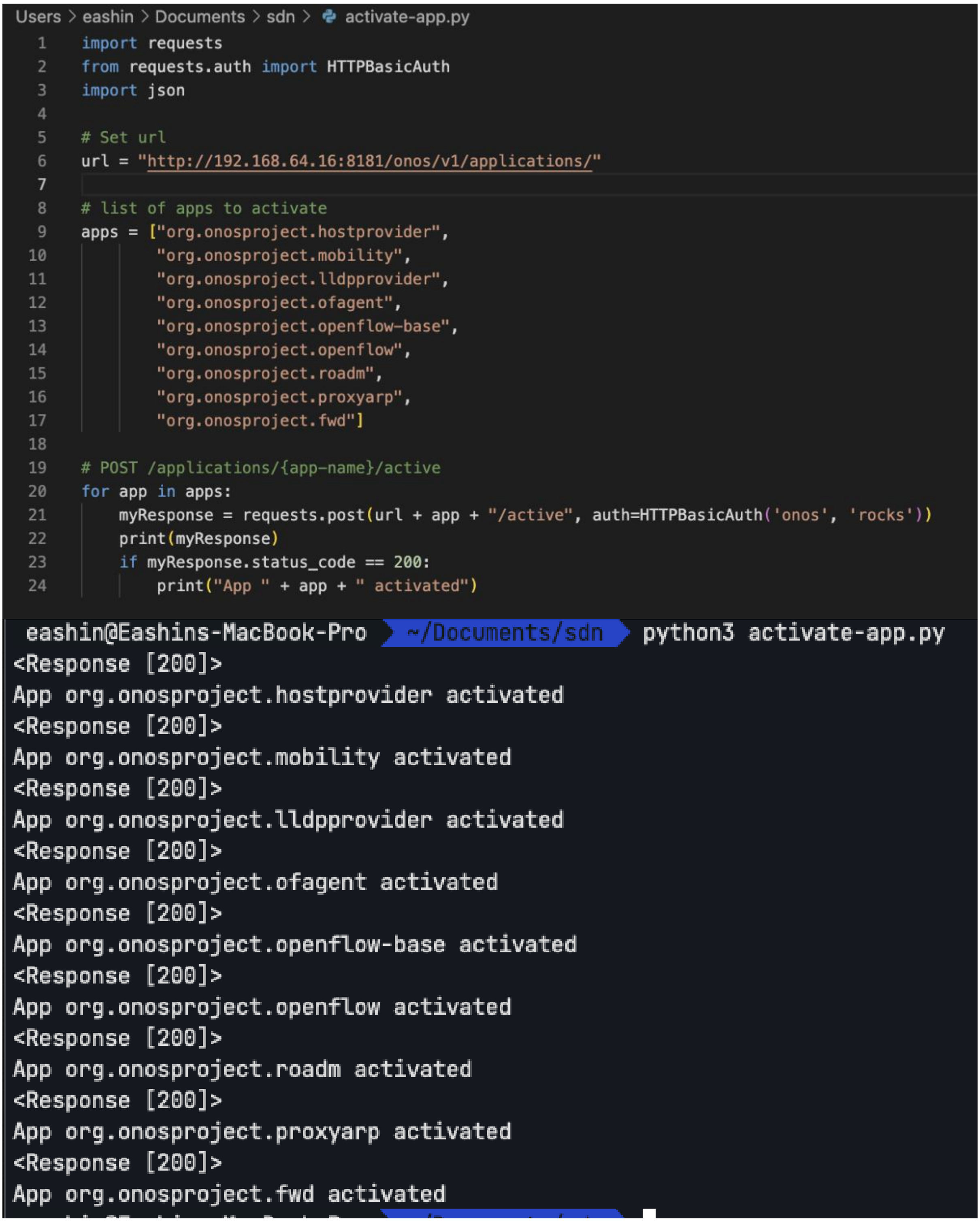
Table of Contents

Demo6: Using ONOS RESTful API to filter, mirror, and forward networking traffic based on OpenFlow capabilities

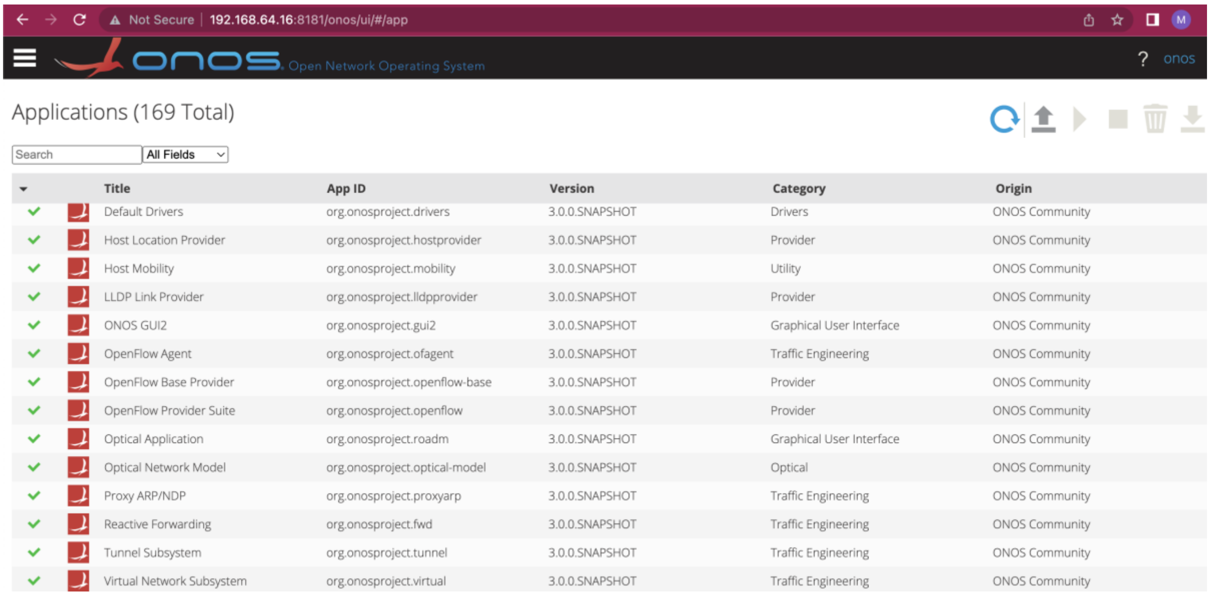
2.2. Pre-Task

- Use the python code created in the previous demonstration to access ONOS RESTful API interface and activate the required ONOS applications.



```python
import requests
from requests.auth import HTTPBasicAuth
import json

# Set url
url = "http://192.168.64.16:8181/onos/v1/applications/"

# list of apps to activate
apps = ["org.onosproject.hostprovider",
        "org.onosproject.mobility",
        "org.onosproject.lldpprovider",
        "org.onosproject.ofagent",
        "org.onosproject.openflow-base",
        "org.onosproject.openflow",
        "org.onosproject.roadm",
        "org.onosproject.proxyarp",
        "org.onosproject.fwd"]

# POST /applications/{app-name}/active
for app in apps:
    myResponse = requests.post(url + app + "/active", auth=HTTPBasicAuth('onos', 'rocks'))
    print(myResponse)
    if myResponse.status_code == 200:
        print("App " + app + " activated")
```

```
eashin@Eashins-MacBook-Pro  ~/Documents/sdn  python3 activate-app.py
<Response [200]>
App org.onosproject.hostprovider activated
<Response [200]>
App org.onosproject.mobility activated
<Response [200]>
App org.onosproject.lldpprovider activated
<Response [200]>
App org.onosproject.ofagent activated
<Response [200]>
App org.onosproject.openflow-base activated
<Response [200]>
App org.onosproject.openflow activated
<Response [200]>
App org.onosproject.roadm activated
<Response [200]>
App org.onosproject.proxyarp activated
<Response [200]>
App org.onosproject.fwd activated
```



Not Secure | 192.168.64.16:8181/onos/ui/#/app

onos. Open Network Operating System

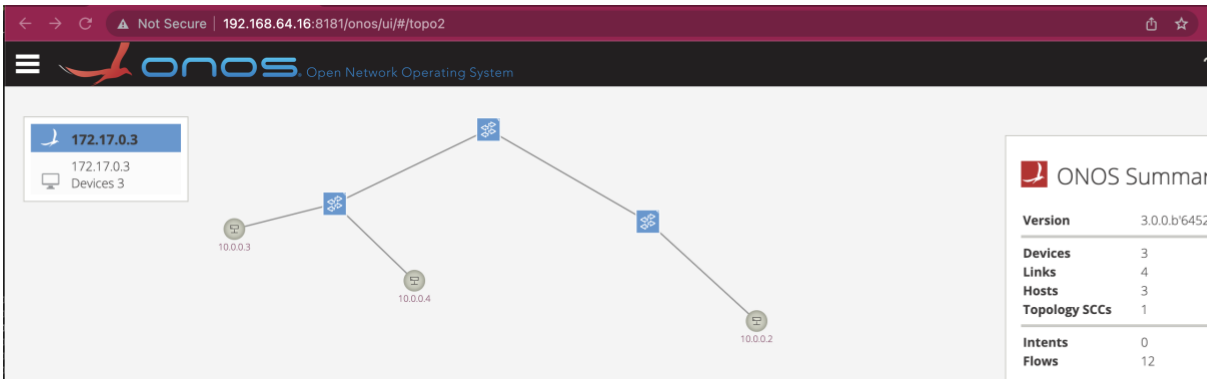
Applications (169 Total)

Search | All Fields

	Title	App ID	Version	Category	Origin
✓	Default Drivers	org.onosproject.drivers	3.0.0.SNAPSHOT	Drivers	ONOS Community
✓	Host Location Provider	org.onosproject.hostprovider	3.0.0.SNAPSHOT	Provider	ONOS Community
✓	Host Mobility	org.onosproject.mobility	3.0.0.SNAPSHOT	Utility	ONOS Community
✓	LLDP Link Provider	org.onosproject.lldpprovider	3.0.0.SNAPSHOT	Provider	ONOS Community
✓	ONOS GUI2	org.onosproject.gui2	3.0.0.SNAPSHOT	Graphical User Interface	ONOS Community
✓	OpenFlow Agent	org.onosproject.ofagent	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community
✓	OpenFlow Base Provider	org.onosproject.openflow-base	3.0.0.SNAPSHOT	Provider	ONOS Community
✓	OpenFlow Provider Suite	org.onosproject.openflow	3.0.0.SNAPSHOT	Provider	ONOS Community
✓	Optical Application	org.onosproject.roadm	3.0.0.SNAPSHOT	Graphical User Interface	ONOS Community
✓	Optical Network Model	org.onosproject.optical-model	3.0.0.SNAPSHOT	Optical	ONOS Community
✓	Proxy ARP/NDP	org.onosproject.proxyarp	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community
✓	Reactive Forwarding	org.onosproject.fwd	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community
✓	Tunnel Subsystem	org.onosproject.tunnel	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community
✓	Virtual Network Subsystem	org.onosproject.virtual	3.0.0.SNAPSHOT	Traffic Engineering	ONOS Community

2.3. Tasks

Similar to the demonstration related to OpenFlow rules in demo 4. Topology created using provided script.



Not Secure | 192.168.64.16:8181/onos/ui/#/topo2

onos. Open Network Operating System

ONOS Summar

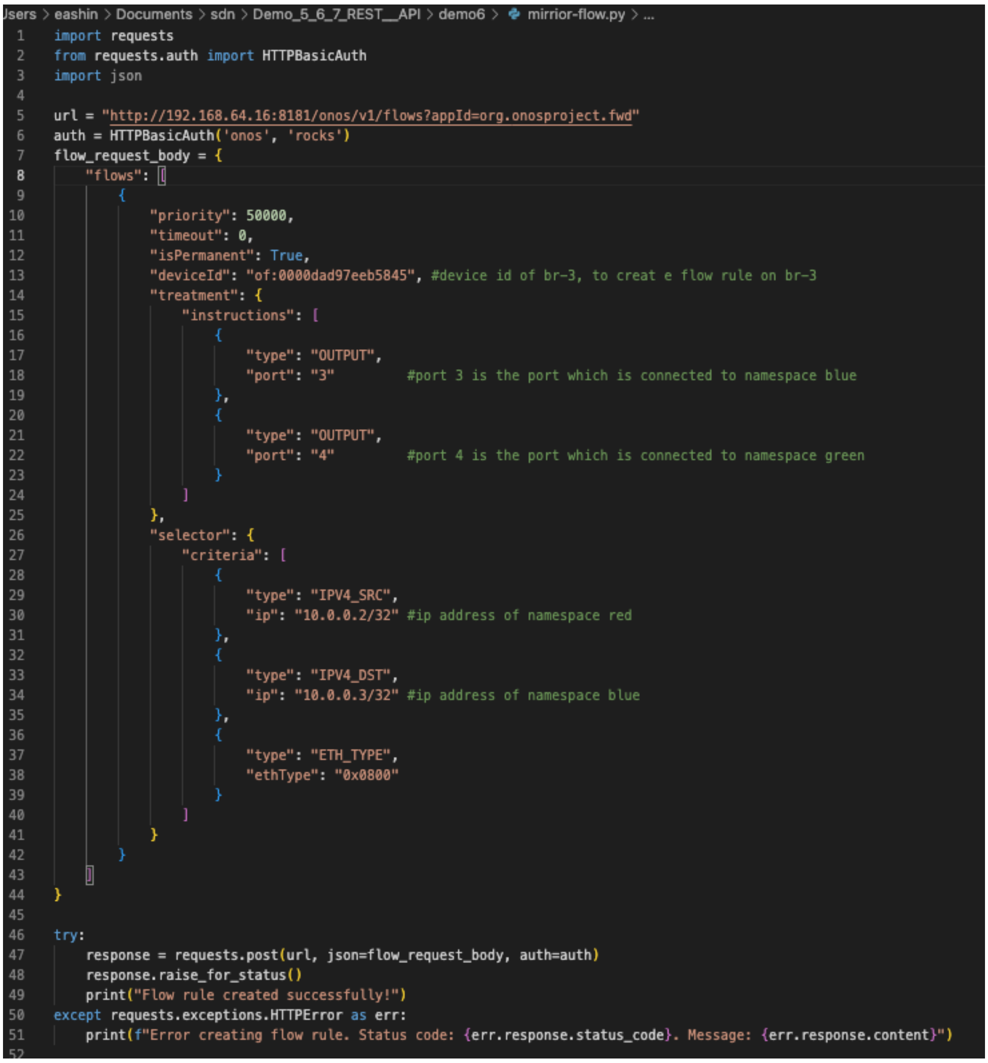
Version	3.0.0.b'6452
Devices	3
Links	4
Hosts	3
Topology SCCs	1
Intents	0
Flows	12

2.3.1. Task 1

- Create a flow rule to mirror (copy) the traffic between the “Red" and the “Blue" namespaces, i.e., consider only traffic from “Red" to “Blue", to also be sent to the “Green" namespace.

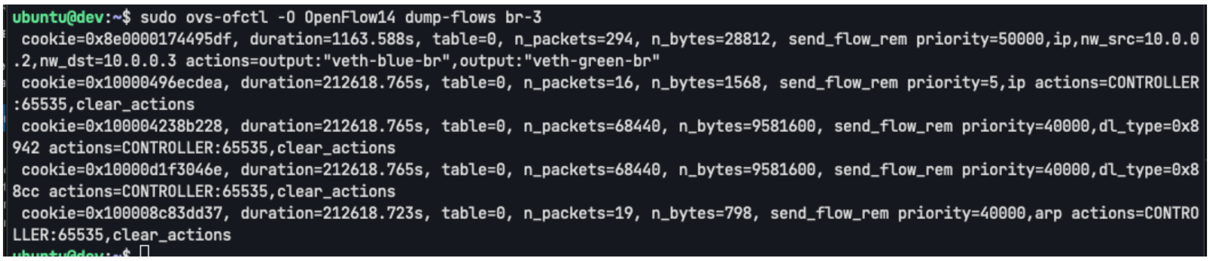
Below is the python program to mirror traffic between red and blue name spaces which also sent to green namespace.

Here we know that Port 3 of br-3 is connecting host blue and Port 4 is connecting Host green. We also know the Ip Addresses and as selector we mention Ethernet Type as traffic type.


Jsers > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > mirrior-flow.py > ...

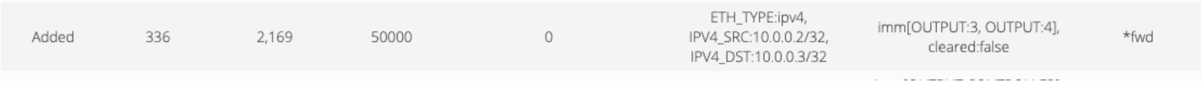
```python
import requests
from requests.auth import HTTPBasicAuth
import json

url = "http://192.168.64.16:8181/onos/v1/flows?appId=org.onosproject.fwd"
auth = HTTPBasicAuth('onos', 'rocks')
flow_request_body = {
    "flows": [
        {
            "priority": 50000,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "treatment": {
                "instructions": [
                    {
                        "type": "OUTPUT",
                        "port": "3"         #port 3 is the port which is connected to namespace blue
                    },
                    {
                        "type": "OUTPUT",
                        "port": "4"         #port 4 is the port which is connected to namespace green
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                        "type": "IPV4_SRC",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.3/32" #ip address of namespace blue
                    },
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    }
                ]
            }
        }
    ]
}

try:
    response = requests.post(url, json=flow_request_body, auth=auth)
    response.raise_for_status()
    print("Flow rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

Check it by using dump-flow as below and confirm that its in ”ADDED” state



```
ubuntu@dev:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x8e0000174495df, duration=1163.588s, table=0, n_packets=294, n_bytes=28812, send_flow_rem priority=50000,ip,nw_src=10.0.0
.2,nw_dst=10.0.0.3 actions=output:"veth-blue-br",output:"veth-green-br"
 cookie=0x10000496ecdea, duration=212618.765s, table=0, n_packets=16, n_bytes=1568, send_flow_rem priority=5,ip actions=CONTROLLER
:65535,clear_actions
 cookie=0x100004238b228, duration=212618.765s, table=0, n_packets=68440, n_bytes=9581600, send_flow_rem priority=40000,dl_type=0x8
942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d1f3046e, duration=212618.765s, table=0, n_packets=68440, n_bytes=9581600, send_flow_rem priority=40000,dl_type=0x8
8cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100008c83dd37, duration=212618.723s, table=0, n_packets=19, n_bytes=798, send_flow_rem priority=40000,arp actions=CONTRO
LLER:65535,clear_actions
ubuntu@dev:~$
```

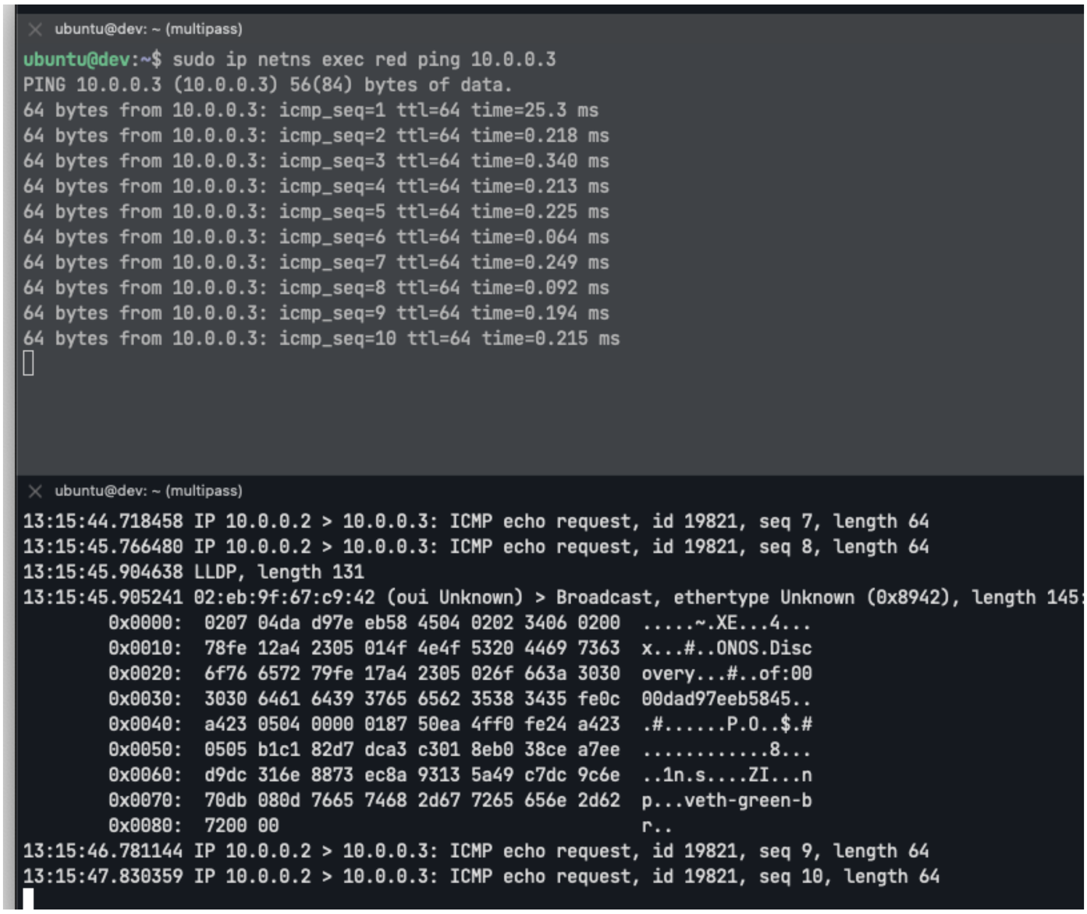
Also, check from ONOS GUI that its in ”ADDED” state



Added	336	2,169	50000	0	ETH_TYPE:ipv4, IPV4_SRC:10.0.0.2/32, IPV4_DST:10.0.0.3/32	imm[OUTPUT:3, OUTPUT:4], cleared:false	*fwd

To test and validate : used ping from red to blue in one terminal

and used ”sudo tcpdump -i veth-green-br” to validate that the traffic mirroring and also send to green namespace. Here we know that veth-green-br is the interface of the name space Green.



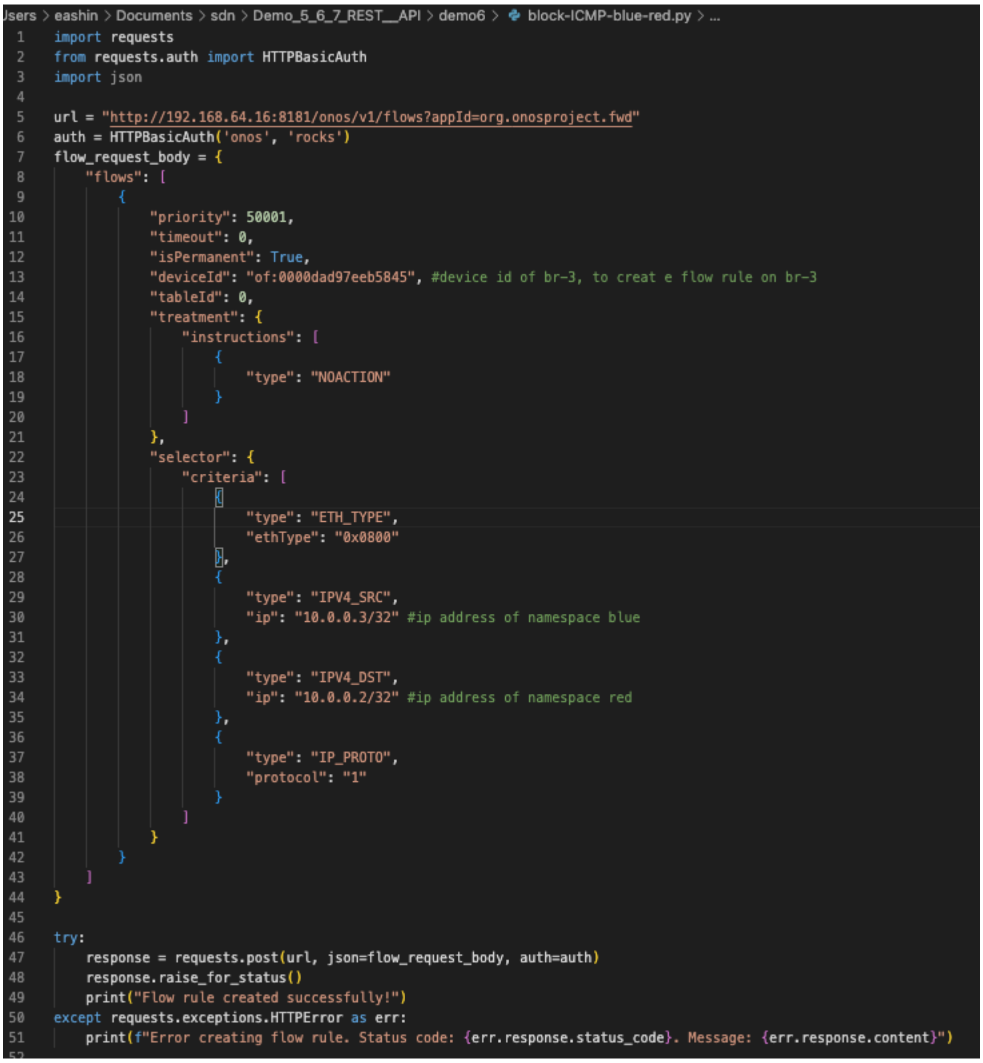
```
ubuntu@dev:~$ sudo ip netns exec red ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=25.3 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.218 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.340 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.213 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.225 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.064 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.249 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.092 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.194 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.215 ms
```

```
13:15:44.718458 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 19821, seq 7, length 64
13:15:45.766480 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 19821, seq 8, length 64
13:15:45.904638 LLDP, length 131
13:15:45.905241 02:eb:9f:67:c9:42 (oui Unknown) > Broadcast, ethertype Unknown (0x8942), length 145:
        0x0000:  0207 04da d97e eb58 4504 0202 3406 0200  .....~.XE...4...
        0x0010:  78fe 12a4 2305 014f 4e4f 5320 4469 7363  x...#..ONOS.Disc
        0x0020:  6f76 6572 79fe 17a4 2305 026f 663a 3030  overy...#..of:00
        0x0030:  3030 6461 6439 3765 6562 3538 3435 fe0c  00dad97eeb5845..
        0x0040:  a423 0504 0000 0187 50ea 4ff0 fe24 a423  .#......P.O..$.#
        0x0050:  0505 b1c1 82d7 dca3 c301 8eb0 38ce a7ee  ............8...
        0x0060:  d9dc 316e 8873 ec8a 9313 5a49 c7dc 9c6e  ..1n.s....ZI...n
        0x0070:  70db 080d 7665 7468 2d67 7265 656e 2d62  p...veth-green-b
        0x0080:  7200 00                                  r..
13:15:46.781144 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 19821, seq 9, length 64
13:15:47.830359 IP 10.0.0.2 > 10.0.0.3: ICMP echo request, id 19821, seq 10, length 64
```

- Create a flow rule to block the ICMP traffic from the “Blue" namespace to the “Red" namespace.

Below is the python code to create flow rule which block ICMP traffic from Blue to red.

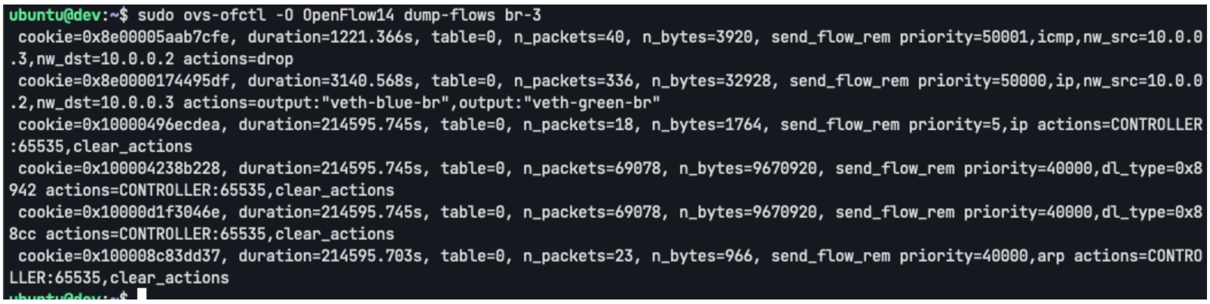
Here we mention in selector “IP_PROTO” type and its “protocol:1” for ICMP and Instructions NO ACTION is equivalent to DROP.



Jsers > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > block-ICMP-blue-red.py > ...

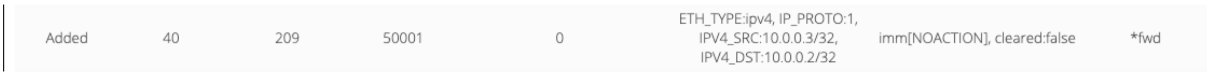
```python
import requests
from requests.auth import HTTPBasicAuth
import json

url = "http://192.168.64.16:8181/onos/v1/flows?appId=org.onosproject.fwd"
auth = HTTPBasicAuth('onos', 'rocks')
flow_request_body = {
    "flows": [
        {
            "priority": 50001,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "tableId": 0,
            "treatment": {
                "instructions": [
                    {
                        "type": "NOACTION"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    },
                    {
                        "type": "IPV4_SRC",
                        "ip": "10.0.0.3/32" #ip address of namespace blue
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    },
                    {
                        "type": "IP_PROTO",
                        "protocol": "1"
                    }
                ]
            }
        }
    ]
}

try:
    response = requests.post(url, json=flow_request_body, auth=auth)
    response.raise_for_status()
    print("Flow rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

Fom the dump-flow we can confirm that its “ADDED” the rule with “priority 50001”



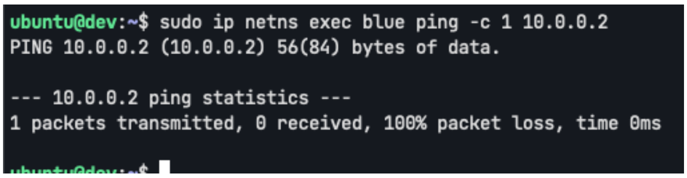
```
ubuntu@dev:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x8e00005aab7cfe, duration=1221.366s, table=0, n_packets=40, n_bytes=3920, send_flow_rem priority=50001,icmp,nw_src=10.0.0
.3,nw_dst=10.0.0.2 actions=drop
 cookie=0x8e0000174495df, duration=3140.568s, table=0, n_packets=336, n_bytes=32928, send_flow_rem priority=50000,ip,nw_src=10.0.0
.2,nw_dst=10.0.0.3 actions=output:"veth-blue-br",output:"veth-green-br"
 cookie=0x10000496ecdea, duration=214595.745s, table=0, n_packets=18, n_bytes=1764, send_flow_rem priority=5,ip actions=CONTROLLER
:65535,clear_actions
 cookie=0x100004238b228, duration=214595.745s, table=0, n_packets=69078, n_bytes=9670920, send_flow_rem priority=40000,dl_type=0x8
942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d1f3046e, duration=214595.745s, table=0, n_packets=69078, n_bytes=9670920, send_flow_rem priority=40000,dl_type=0x8
8cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100008c83dd37, duration=214595.703s, table=0, n_packets=23, n_bytes=966, send_flow_rem priority=40000,arp actions=CONTRO
LLER:65535,clear_actions
ubuntu@dev:~$ 
```

We can check from ONOS GUI that rule Added.



Added	40	209	50001	0	ETH_TYPE:ipv4, IP_PROTO:1, IPV4_SRC:10.0.0.3/32, IPV4_DST:10.0.0.2/32	imm[NOACTION], cleared:false	*fwd

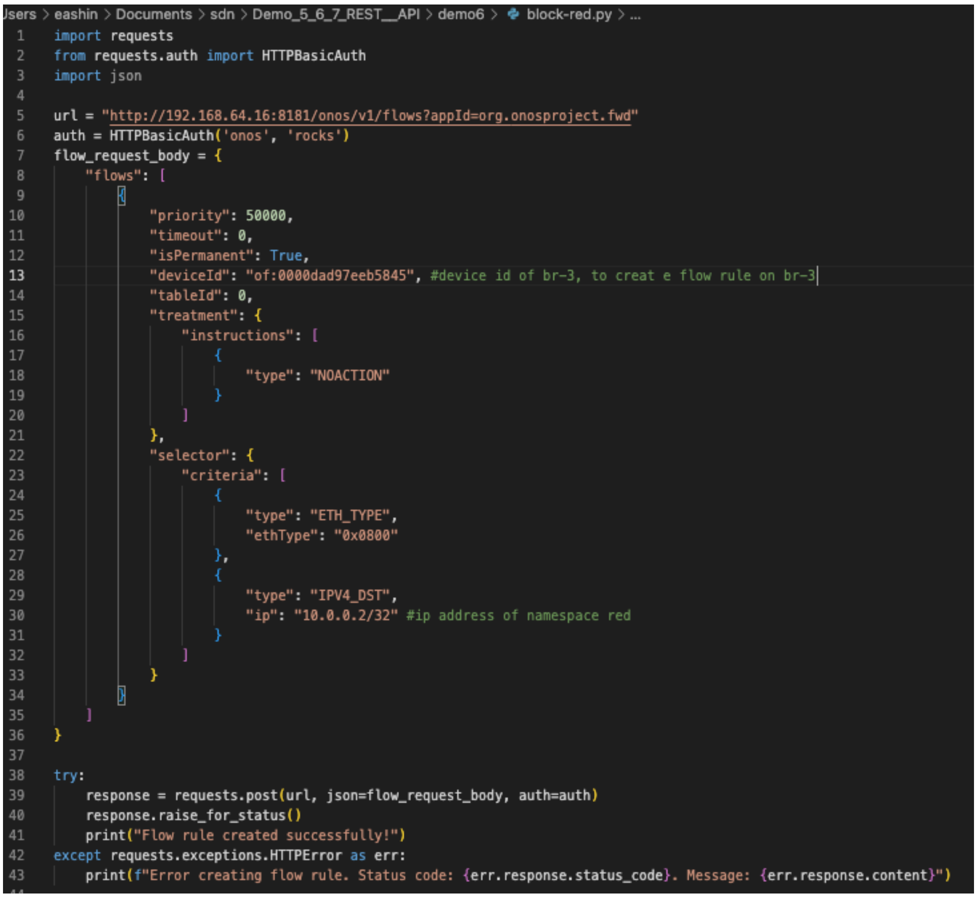
We used ping to test and validate that works as expected.



```
ubuntu@dev:~$ sudo ip netns exec blue ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$
```

- Create a flow rule to block the traffic to the “Red" namespace.

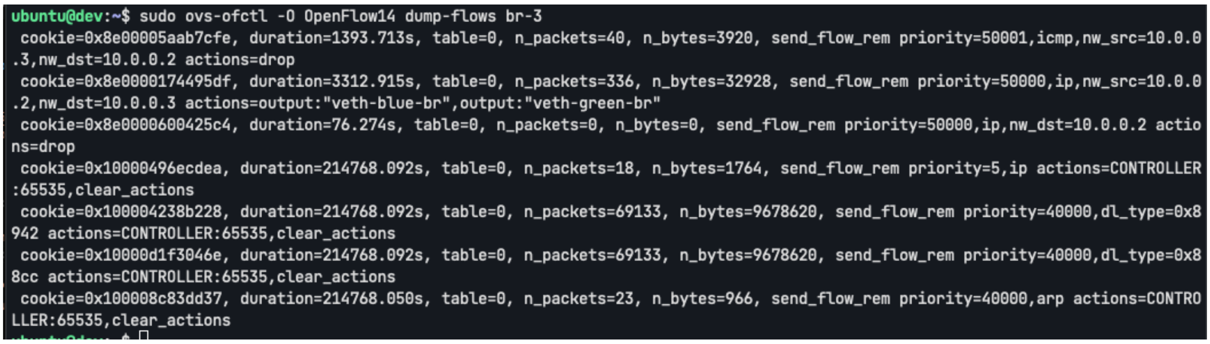
Here below is the python program to block traffic to red namespace.



Jsers > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > block-red.py > ...

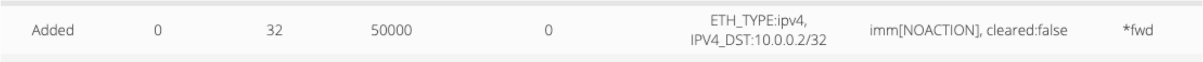
```python
import requests
from requests.auth import HTTPBasicAuth
import json

url = "http://192.168.64.16:8181/onos/v1/flows?appId=org.onosproject.fwd"
auth = HTTPBasicAuth('onos', 'rocks')
flow_request_body = {
    "flows": [
        {
            "priority": 50000,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "tableId": 0,
            "treatment": {
                "instructions": [
                    {
                        "type": "NOACTION"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    }
                ]
            }
        }
    ]
}

try:
    response = requests.post(url, json=flow_request_body, auth=auth)
    response.raise_for_status()
    print("Flow rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

From the dump-flow we can confirm that the rule Added successfully.



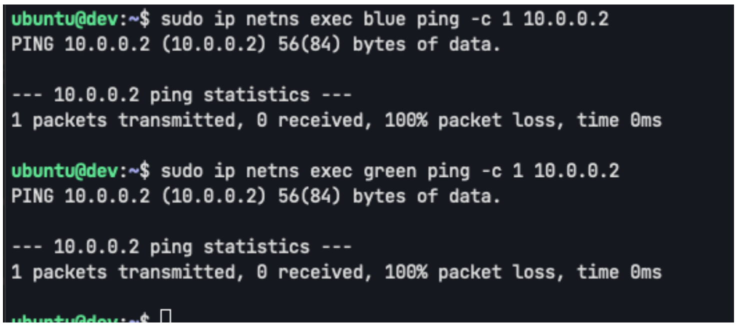
```
ubuntu@dev:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x8e00005aab7cfe, duration=1393.713s, table=0, n_packets=40, n_bytes=3920, send_flow_rem priority=50001,icmp,nw_src=10.0.0
.3,nw_dst=10.0.0.2 actions=drop
 cookie=0x8e0000174495df, duration=3312.915s, table=0, n_packets=336, n_bytes=32928, send_flow_rem priority=50000,ip,nw_src=10.0.0
.2,nw_dst=10.0.0.3 actions=output:"veth-blue-br",output:"veth-green-br"
 cookie=0x8e0000600425c4, duration=76.274s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50000,ip,nw_dst=10.0.0.2 actio
ns=drop
 cookie=0x10000496ecdea, duration=214768.092s, table=0, n_packets=18, n_bytes=1764, send_flow_rem priority=5,ip actions=CONTROLLER
:65535,clear_actions
 cookie=0x100004238b228, duration=214768.092s, table=0, n_packets=69133, n_bytes=9678620, send_flow_rem priority=40000,dl_type=0x8
942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d1f3046e, duration=214768.092s, table=0, n_packets=69133, n_bytes=9678620, send_flow_rem priority=40000,dl_type=0x8
8cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100008c83dd37, duration=214768.050s, table=0, n_packets=23, n_bytes=966, send_flow_rem priority=40000,arp actions=CONTRO
LLER:65535,clear_actions
ubuntu@dev:~$
```

We also checked from ONOS GUI that rule Added.



Added	0	32	50000	0	ETH_TYPE:ipv4, IPV4_DST:10.0.0.2/32	imm[NOACTION], cleared:false	*fwd

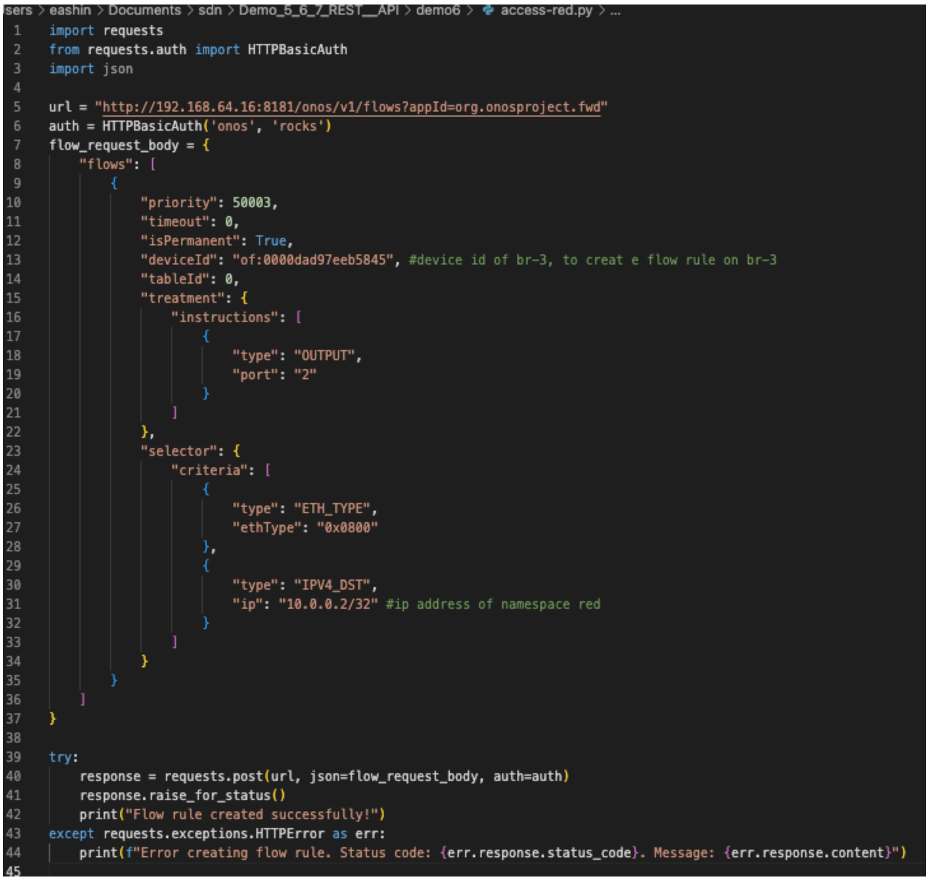
We test and validated using ping that it is working as expected and from both namespaces blue and green traffic blocked to Red namespace.



```
ubuntu@dev:~$ sudo ip netns exec blue ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$ sudo ip netns exec green ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$ 
```

- Create a flow rule to allow total access to the “Red" namespace from the “Green" and the “Blue" namespaces.

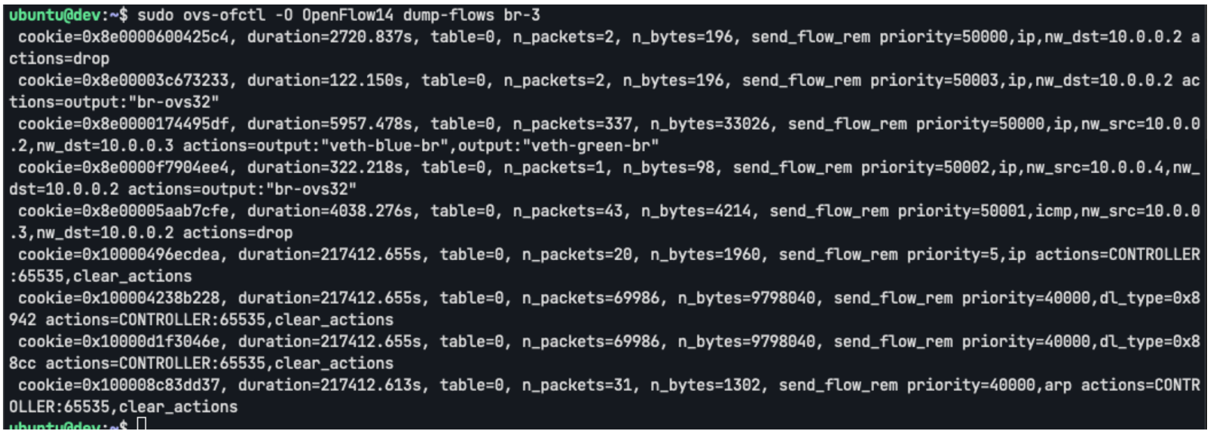
Here is the script to accept again all traffic to Red and this time I used higher priority so that the previous rule should not apply, and I mention Port 2 of the same device of br-3 as it is connecting the Red namespace to this bridge.



sers > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > access-red.py > ...

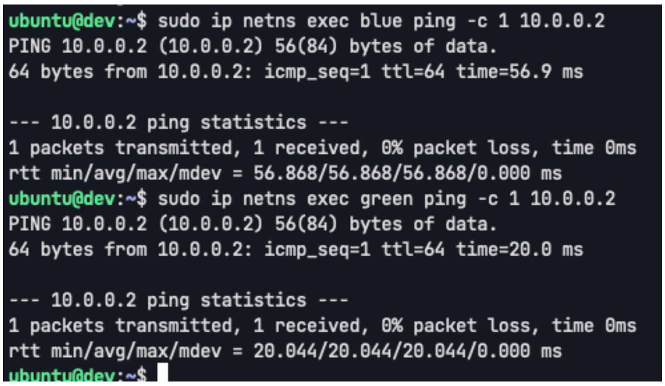
```python
import requests
from requests.auth import HTTPBasicAuth
import json

url = "http://192.168.64.16:8181/onos/v1/flows?appId=org.onosproject.fwd"
auth = HTTPBasicAuth('onos', 'rocks')
flow_request_body = {
    "flows": [
        {
            "priority": 50003,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "tableId": 0,
            "treatment": {
                "instructions": [
                    {
                        "type": "OUTPUT",
                        "port": "2"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    }
                ]
            }
        }
    ]
}

try:
    response = requests.post(url, json=flow_request_body, auth=auth)
    response.raise_for_status()
    print("Flow rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

So, same was as previous we have dump-flow to check its added.



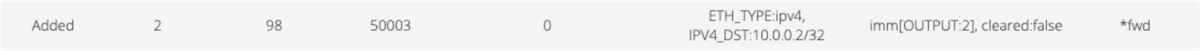
```
ubuntu@dev:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x8e0000600425c4, duration=2720.837s, table=0, n_packets=2, n_bytes=196, send_flow_rem priority=50000,ip,nw_dst=10.0.0.2 a
ctions=drop
 cookie=0x8e00003c673233, duration=122.150s, table=0, n_packets=2, n_bytes=196, send_flow_rem priority=50003,ip,nw_dst=10.0.0.2 ac
tions=output:"br-ovs32"
 cookie=0x8e0000174495df, duration=5957.478s, table=0, n_packets=337, n_bytes=33026, send_flow_rem priority=50000,ip,nw_src=10.0.0
.2,nw_dst=10.0.0.3 actions=output:"veth-blue-br",output:"veth-green-br"
 cookie=0x8e0000f7904ee4, duration=322.218s, table=0, n_packets=1, n_bytes=98, send_flow_rem priority=50002,ip,nw_src=10.0.0.4,nw_
dst=10.0.0.2 actions=output:"br-ovs32"
 cookie=0x8e00005aab7cfe, duration=4038.276s, table=0, n_packets=43, n_bytes=4214, send_flow_rem priority=50001,icmp,nw_src=10.0.0
.3,nw_dst=10.0.0.2 actions=drop
 cookie=0x10000496ecdea, duration=217412.655s, table=0, n_packets=20, n_bytes=1960, send_flow_rem priority=5,ip actions=CONTROLLER
:65535,clear_actions
 cookie=0x100004238b228, duration=217412.655s, table=0, n_packets=69986, n_bytes=9798040, send_flow_rem priority=40000,dl_type=0x8
942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d1f3046e, duration=217412.655s, table=0, n_packets=69986, n_bytes=9798040, send_flow_rem priority=40000,dl_type=0x8
8cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100008c83dd37, duration=217412.613s, table=0, n_packets=31, n_bytes=1302, send_flow_rem priority=40000,arp actions=CONTR
OLLER:65535,clear_actions
ubuntu@dev:~$
```

Also as previous, we ping Red namespace from Blue and Green and its working as expected.



```
ubuntu@dev:~$ sudo ip netns exec blue ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=56.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 56.868/56.868/56.868/0.000 ms
ubuntu@dev:~$ sudo ip netns exec green ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=20.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.044/20.044/20.044/0.000 ms
ubuntu@dev:~$
```

We also checked that from ONOS GUI that 2 packet transmitted.



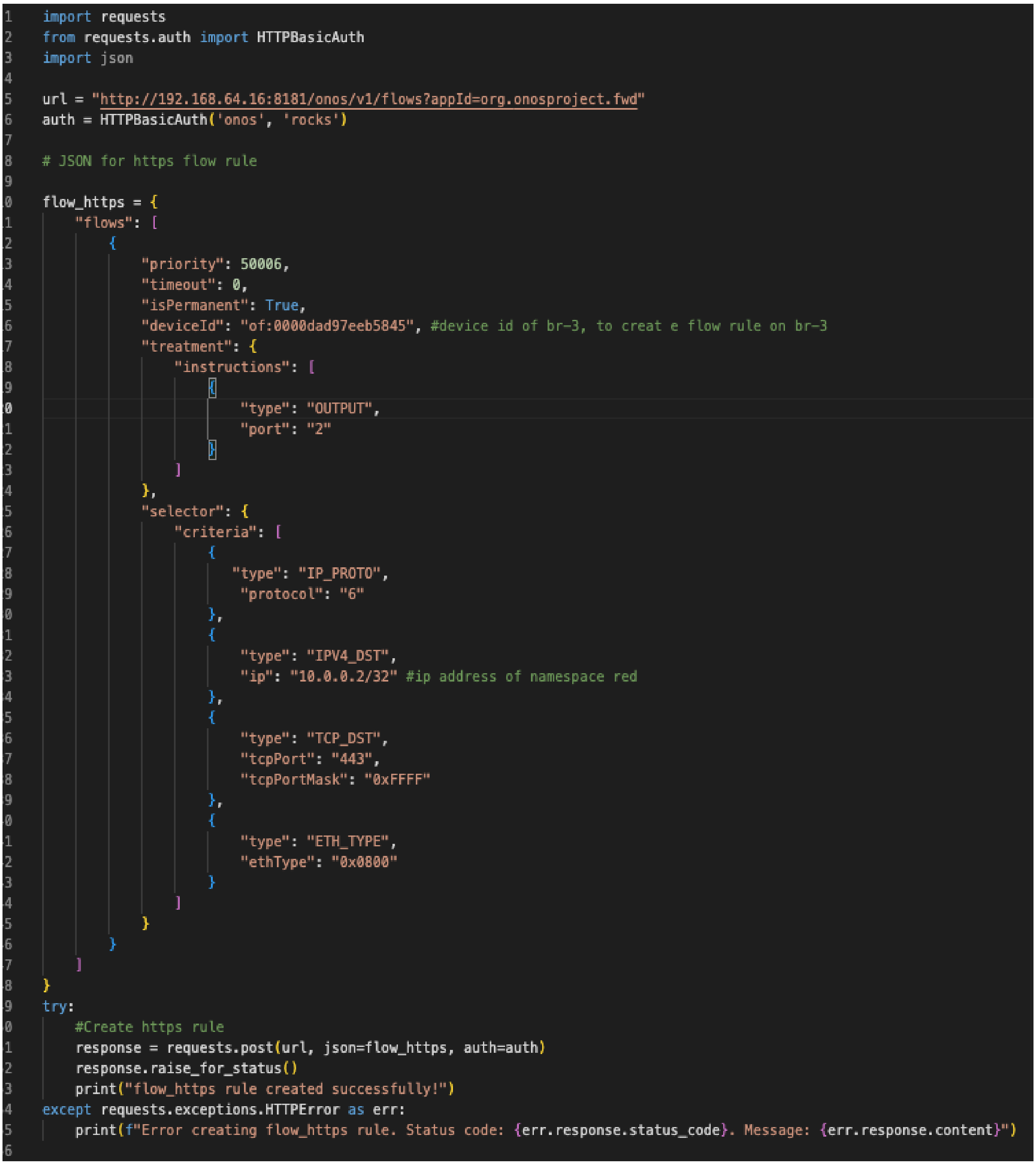
Added	2	98	50003	0	ETH_TYPE:ipv4, IPV4_DST:10.0.0.2/32	imm[OUTPUT:2], cleared:false	*fwd

- Do you need to delete the previously created flow rules to activate flow rule number 3?

NO. W edo not need to delete any flow rule and to apply a different rule with higher priority works fine without deleting any rule which may have any conflict with new rule.

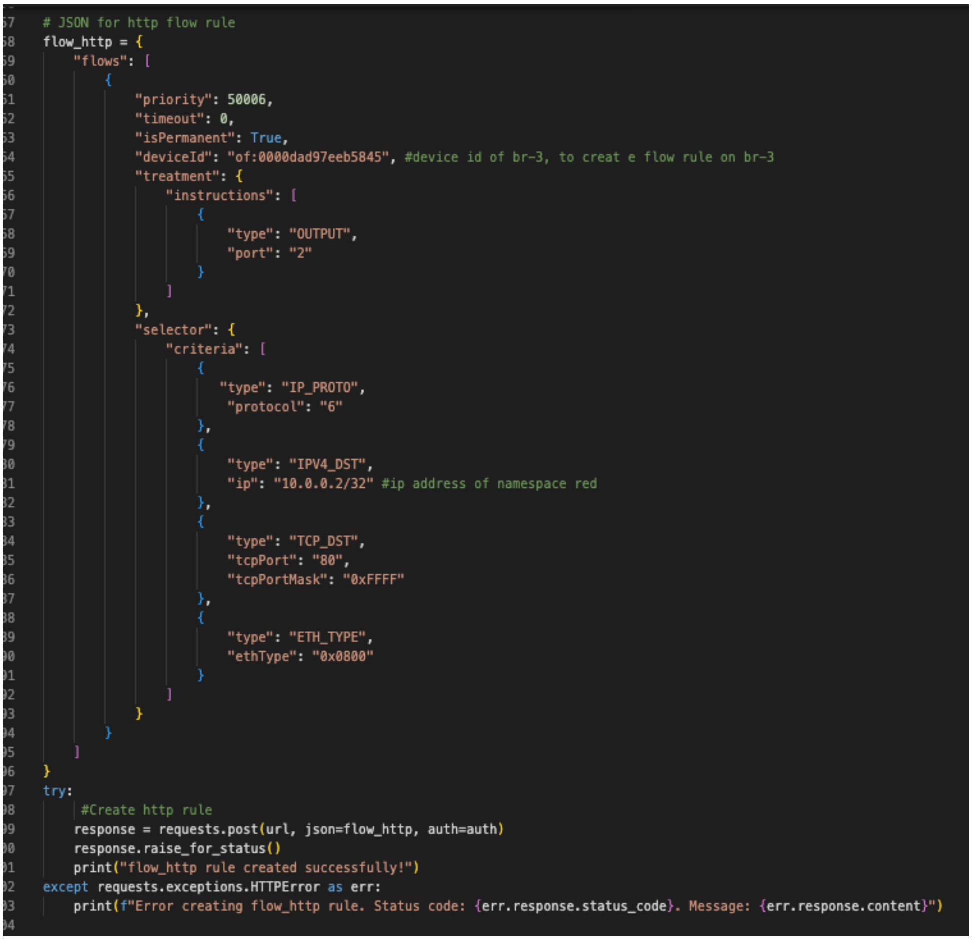
- Create a flow rule to allow only Http and Https traffic to the “Red" namespace.

I used same python script to create required flow rules for allowing http, HTTPS and block other traffic but I used different JSON payload to create required rules and below is the part of the script for HTTPS flow rule creation. Mentioned in the selector ”IP_PROTO” to consider only TCP packets and used ”protocol:6” which indicates TCP. Also for the ”TCP_DST” to specify Port.



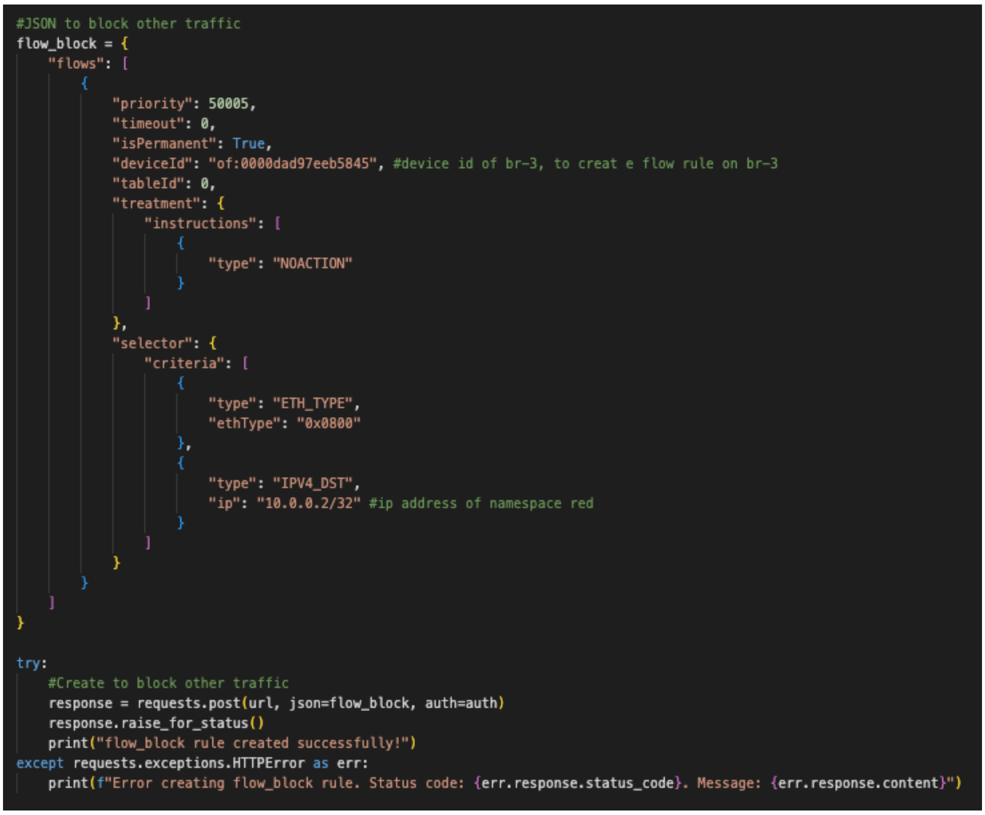
```python
import requests
from requests.auth import HTTPBasicAuth
import json

url = "http://192.168.64.16:8181/onos/v1/flows?appId=org.onosproject.fwd"
auth = HTTPBasicAuth('onos', 'rocks')

# JSON for https flow rule

flow_https = {
    "flows": [
        {
            "priority": 50006,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "treatment": {
                "instructions": [
                    {
                        "type": "OUTPUT",
                        "port": "2"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                       "type": "IP_PROTO",
                        "protocol": "6"
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    },
                    {
                        "type": "TCP_DST",
                        "tcpPort": "443",
                        "tcpPortMask": "0xFFFF"
                    },
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    }
                ]
            }
        }
    ]
}
try:
    #Create https rule
    response = requests.post(url, json=flow_https, auth=auth)
    response.raise_for_status()
    print("flow_https rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow_https rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

Similar way below is the part of the script for HTTP flow rule creation.



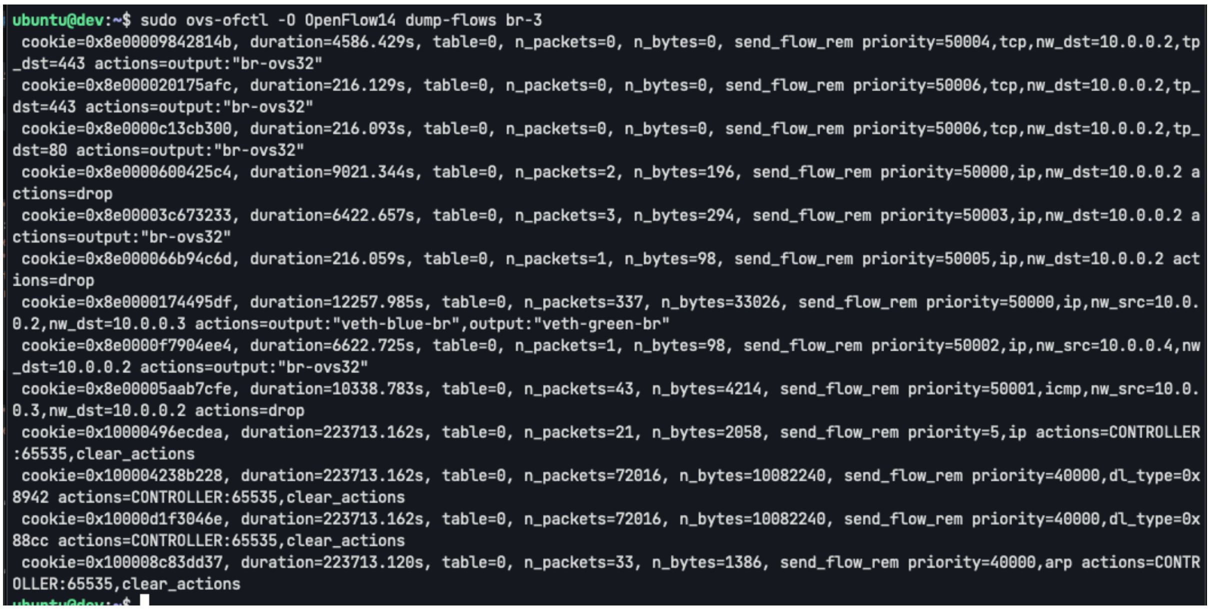
```
# JSON for http flow rule
flow_http = {
    "flows": [
        {
            "priority": 50006,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "treatment": {
                "instructions": [
                    {
                        "type": "OUTPUT",
                        "port": "2"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                       "type": "IP_PROTO",
                        "protocol": "6"
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    },
                    {
                        "type": "TCP_DST",
                        "tcpPort": "80",
                        "tcpPortMask": "0xFFFF"
                    },
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    }
                ]
            }
        }
    ]
}
try:
     #Create http rule
    response = requests.post(url, json=flow_http, auth=auth)
    response.raise_for_status()
    print("flow_http rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow_http rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

Also this part of the script used to create rule to block all other traffic.



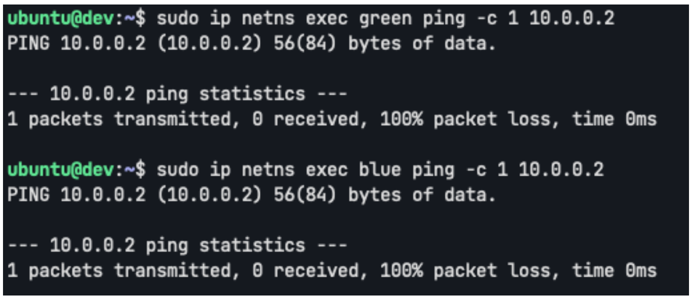
```
#JSON to block other traffic
flow_block = {
    "flows": [
        {
            "priority": 50005,
            "timeout": 0,
            "isPermanent": True,
            "deviceId": "of:0000dad97eeb5845", #device id of br-3, to creat e flow rule on br-3
            "tableId": 0,
            "treatment": {
                "instructions": [
                    {
                        "type": "NOACTION"
                    }
                ]
            },
            "selector": {
                "criteria": [
                    {
                        "type": "ETH_TYPE",
                        "ethType": "0x0800"
                    },
                    {
                        "type": "IPV4_DST",
                        "ip": "10.0.0.2/32" #ip address of namespace red
                    }
                ]
            }
        }
    ]
}

try:
    #Create to block other traffic
    response = requests.post(url, json=flow_block, auth=auth)
    response.raise_for_status()
    print("flow_block rule created successfully!")
except requests.exceptions.HTTPError as err:
    print(f"Error creating flow_block rule. Status code: {err.response.status_code}. Message: {err.response.content}")
```

From the dump-flow we can verify that all required rules added with priority. This time also used higher pririty accordingly to apply this rule so that any other previous rules that may conflict with this new rule will not applicable anymore.



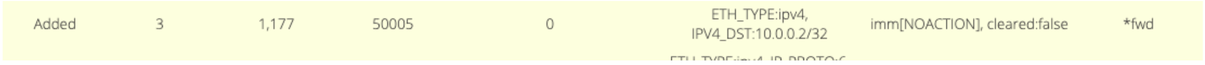
```
ubuntu@dev:~$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x8e00009842814b, duration=4586.429s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50004,tcp,nw_dst=10.0.0.2,tp
_dst=443 actions=output:"br-ovs32"
 cookie=0x8e000020175afc, duration=216.129s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50006,tcp,nw_dst=10.0.0.2,tp_
dst=443 actions=output:"br-ovs32"
 cookie=0x8e0000c13cb300, duration=216.093s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=50006,tcp,nw_dst=10.0.0.2,tp_
dst=80 actions=output:"br-ovs32"
 cookie=0x8e0000600425c4, duration=9021.344s, table=0, n_packets=2, n_bytes=196, send_flow_rem priority=50000,ip,nw_dst=10.0.0.2 a
ctions=drop
 cookie=0x8e00003c673233, duration=6422.657s, table=0, n_packets=3, n_bytes=294, send_flow_rem priority=50003,ip,nw_dst=10.0.0.2 a
ctions=output:"br-ovs32"
 cookie=0x8e000066b94c6d, duration=216.059s, table=0, n_packets=1, n_bytes=98, send_flow_rem priority=50005,ip,nw_dst=10.0.0.2 act
ions=drop
 cookie=0x8e0000174495df, duration=12257.985s, table=0, n_packets=337, n_bytes=33026, send_flow_rem priority=50000,ip,nw_src=10.0.
0.2,nw_dst=10.0.0.3 actions=output:"veth-blue-br",output:"veth-green-br"
 cookie=0x8e0000f7904ee4, duration=6622.725s, table=0, n_packets=1, n_bytes=98, send_flow_rem priority=50002,ip,nw_src=10.0.0.4,nw
_dst=10.0.0.2 actions=output:"br-ovs32"
 cookie=0x8e00005aab7cfe, duration=10338.783s, table=0, n_packets=43, n_bytes=4214, send_flow_rem priority=50001,icmp,nw_src=10.0.
0.3,nw_dst=10.0.0.2 actions=drop
 cookie=0x10000496ecdea, duration=223713.162s, table=0, n_packets=21, n_bytes=2058, send_flow_rem priority=5,ip actions=CONTROLLER
:65535,clear_actions
 cookie=0x100004238b228, duration=223713.162s, table=0, n_packets=72016, n_bytes=10082240, send_flow_rem priority=40000,dl_type=0x
8942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d1f3046e, duration=223713.162s, table=0, n_packets=72016, n_bytes=10082240, send_flow_rem priority=40000,dl_type=0x
88cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100008c83dd37, duration=223713.120s, table=0, n_packets=33, n_bytes=1386, send_flow_rem priority=40000,arp actions=CONTR
OLLER:65535,clear_actions
ubuntu@dev:~$
```

Used ping from blue and Green to red and verified that other traffic is blocked as defined by the rule.



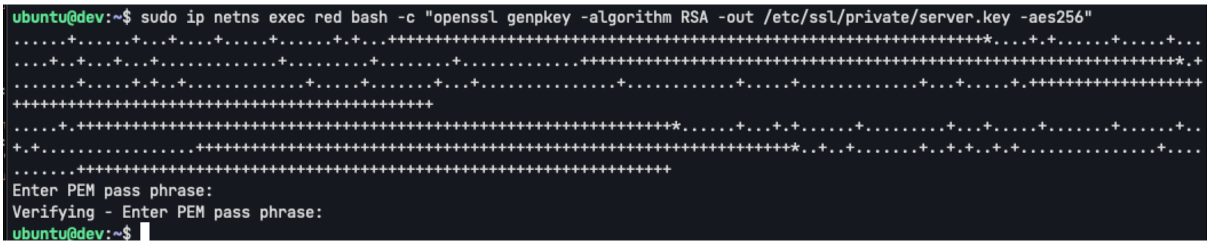
```
ubuntu@dev:~$ sudo ip netns exec green ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@dev:~$ sudo ip netns exec blue ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

We can check from ONOS GUI that it dropped packet as expected by. The rule.



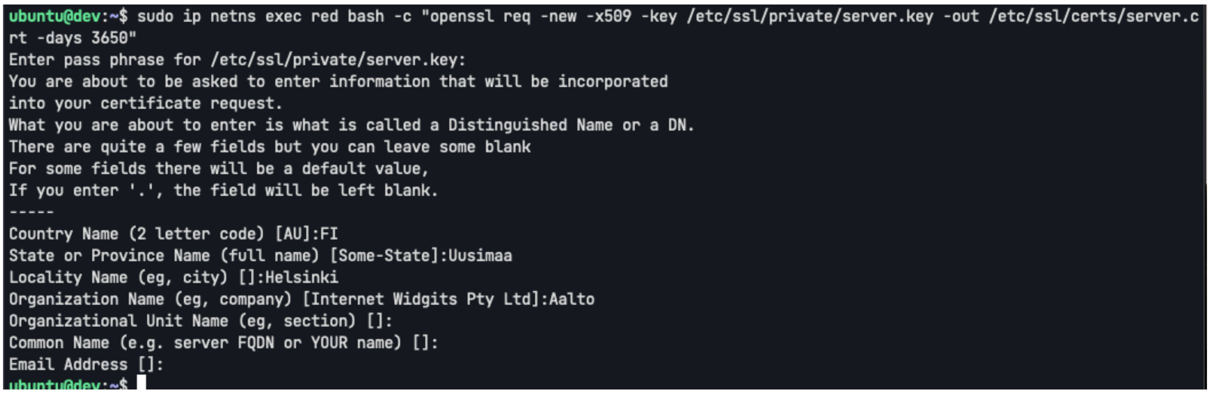
Added	3	1,177	50005	0	ETH_TYPE:ipv4, IPV4_DST:10.0.0.2/32	imm[NOACTION], cleared:false	*fwd

To test and verify HTTPS, I have generated a private key on red host



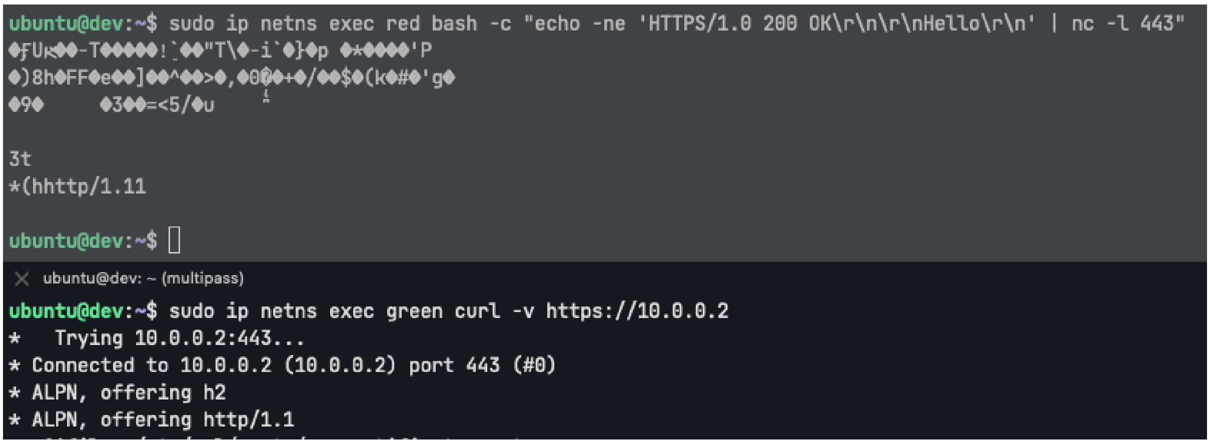
```
ubuntu@dev:~$ sudo ip netns exec red bash -c "openssl genpkey -algorithm RSA -out /etc/ssl/private/server.key -aes256"
......+......+...+....+.....+......+.+...+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*....+.+......+.....+...
....+..+...+...+.............+.........+........+.............+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.+
........+.....+.+..+..............+.....+.......+...+...............+...........+.....+.............+...+.....+.++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++
.....+.+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*......+...+.+......+.........+...+.....+.......+......+..
+.+................+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+..+.......+..+.+..+.+................+....
.......+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
ubuntu@dev:~$
```

Also generated a self signed Certificate for testing HTTPS traffic.



```
ubuntu@dev:~$ sudo ip netns exec red bash -c "openssl req -new -x509 -key /etc/ssl/private/server.key -out /etc/ssl/certs/server.c
rt -days 3650"
Enter pass phrase for /etc/ssl/private/server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FI
State or Province Name (full name) [Some-State]:Uusimaa
Locality Name (eg, city) []:Helsinki
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Aalto
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
ubuntu@dev:~$
```

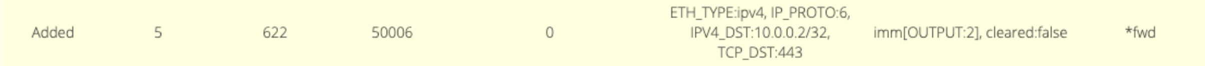
Finally to test and validate HTTPS I run http server on red namespace and from another terminal sent a curl request from Green to red to test HTTPS working as expected.



```
ubuntu@dev:~$ sudo ip netns exec red bash -c "echo -ne 'HTTPS/1.0 200 OK\r\n\r\nHello\r\n' | nc -l 443"
�ƒUĸ��-T�����!`��"T\�-i`�}�p �*����'P
�)8h�FF�e��]��^��>�,�0��+�/��$�(k�#�'g�
�9�     �3��=<5/�u

3t
*(hhttp/1.11

ubuntu@dev:~$
```

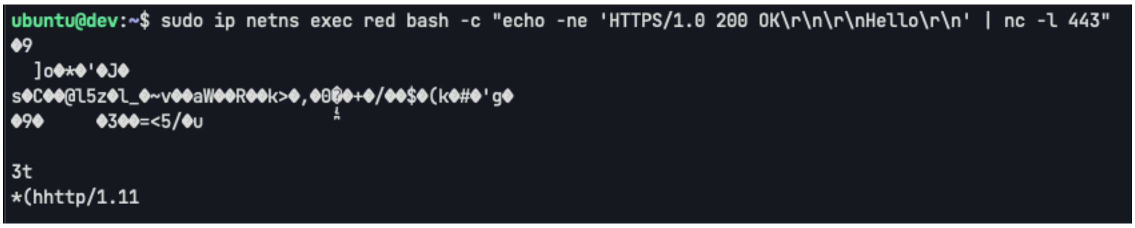
```
ubuntu@dev:~$ sudo ip netns exec green curl -v https://10.0.0.2
*   Trying 10.0.0.2:443...
* Connected to 10.0.0.2 (10.0.0.2) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
```

Also checked from GUI that data transmitting packet and the flow rule in Added state.

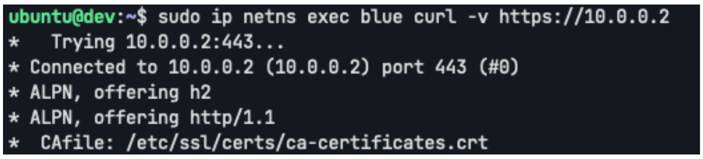


Added	5	622	50006	0	ETH_TYPE:ipv4, IP_PROTO:6, IPV4_DST:10.0.0.2/32, TCP_DST:443	imm[OUTPUT:2], cleared:false	*fwd

Similar way tested and validated that its working as expected from blue to red.

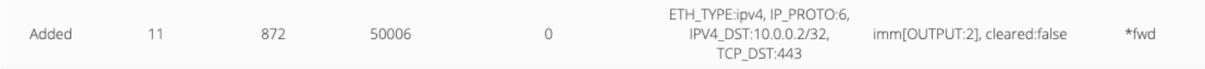


```
ubuntu@dev:~$ sudo ip netns exec red bash -c "echo -ne 'HTTPS/1.0 200 OK\r\n\r\nHello\r\n' | nc -l 443"
�9
  ]o�*�'�J�
s�C��@l5z�l_�~v��aW��R��k>�,�0��+�/��$�(k�#�'g�
�9�     �3��=<5/�u

3t
*(hhttp/1.11
```



```
ubuntu@dev:~$ sudo ip netns exec blue curl -v https://10.0.0.2
*   Trying 10.0.0.2:443...
* Connected to 10.0.0.2 (10.0.0.2) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
```

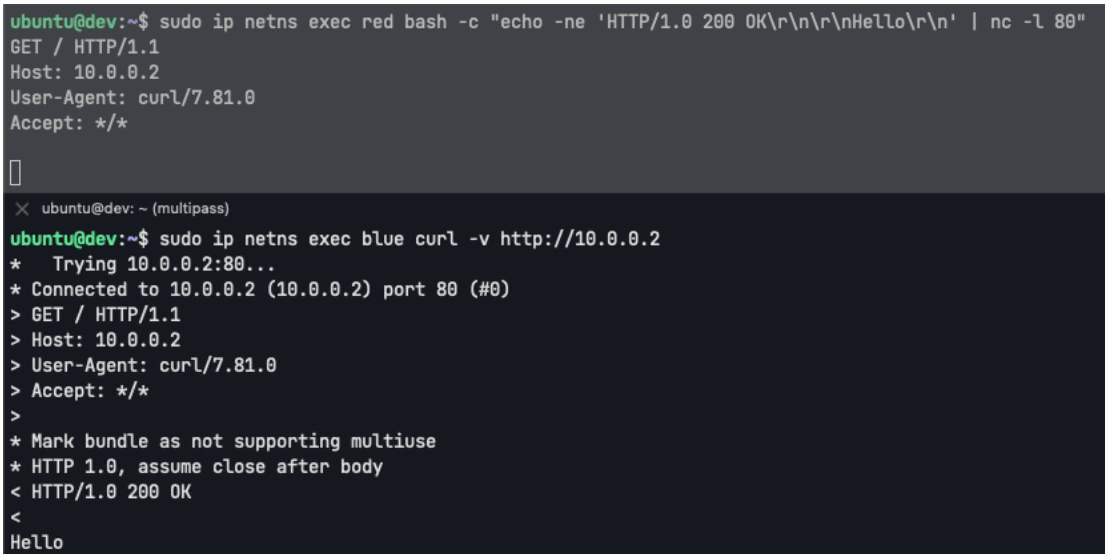
We also checked from GUI that packet transmitted by the same rule.



Added	11	872	50006	0	ETH_TYPE:ipv4, IP_PROTO:6, IPV4_DST:10.0.0.2/32, TCP_DST:443	imm[OUTPUT:2], cleared:false	*fwd

Similar way tested and validated for http and it is working as expected bu the rules created.

We tested http traffic from blue to red.

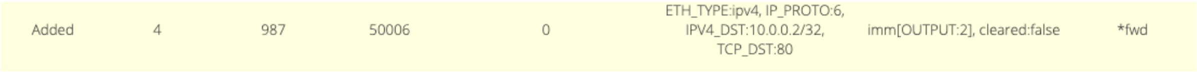


```
ubuntu@dev:~$ sudo ip netns exec red bash -c "echo -ne 'HTTP/1.0 200 OK\r\n\r\nHello\r\n' | nc -l 80"
GET / HTTP/1.1
Host: 10.0.0.2
User-Agent: curl/7.81.0
Accept: */*
```

ubuntu@dev: ~ (multipass)

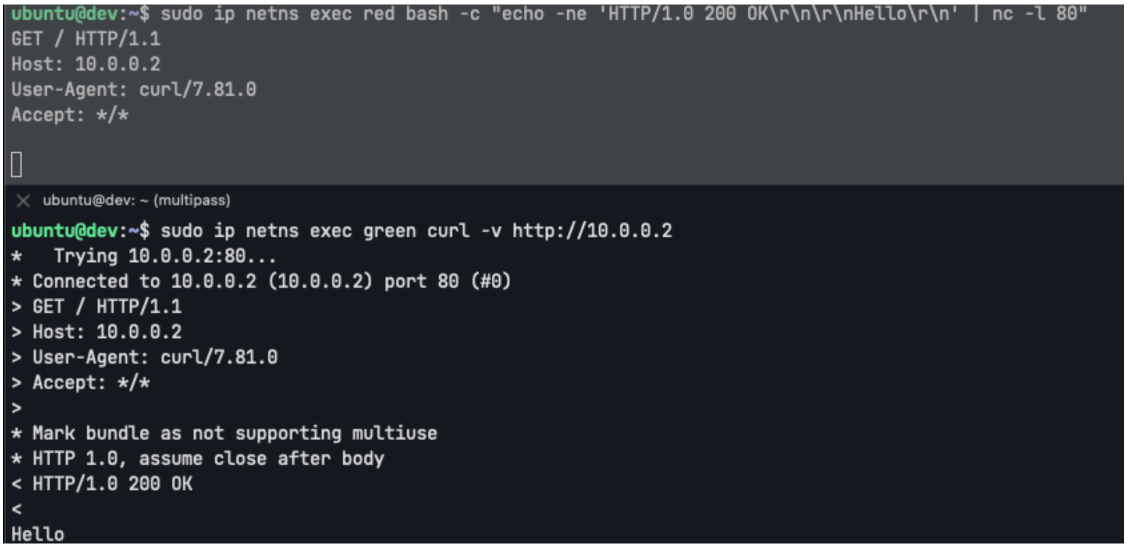
```
ubuntu@dev:~$ sudo ip netns exec blue curl -v http://10.0.0.2
*   Trying 10.0.0.2:80...
* Connected to 10.0.0.2 (10.0.0.2) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.0.2
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
<
Hello
```

We also checked from GUI.



Added	4	987	50006	0	ETH_TYPE:ipv4, IP_PROTO:6, IPV4_DST:10.0.0.2/32, TCP_DST:80	imm[OUTPUT:2], cleared:false	*fwd

We tested http traffic from green to red.



```
ubuntu@dev:~$ sudo ip netns exec red bash -c "echo -ne 'HTTP/1.0 200 OK\r\n\r\nHello\r\n' | nc -l 80"
GET / HTTP/1.1
Host: 10.0.0.2
User-Agent: curl/7.81.0
Accept: */*
```

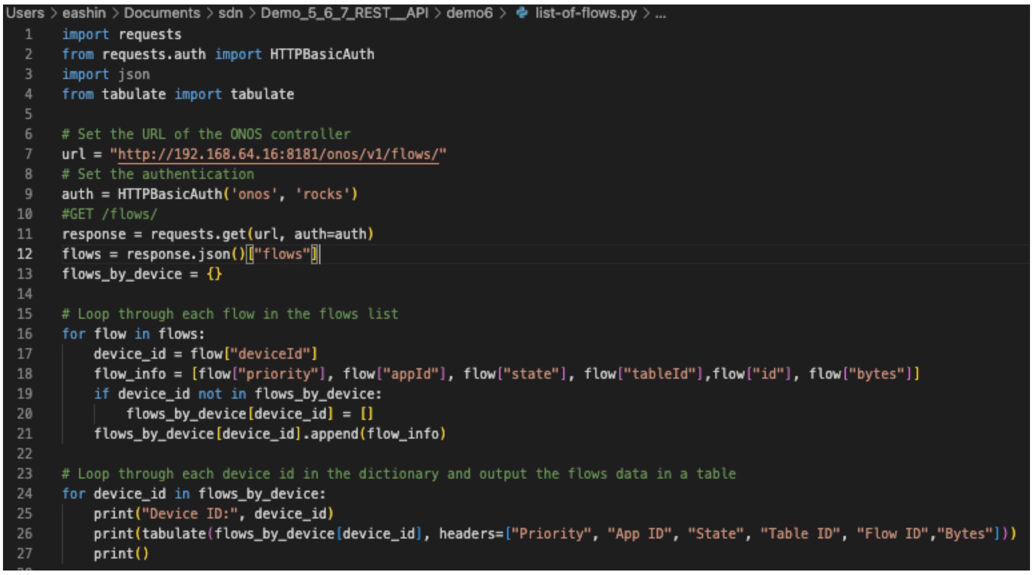
ubuntu@dev: ~ (multipass)

```
ubuntu@dev:~$ sudo ip netns exec green curl -v http://10.0.0.2
*   Trying 10.0.0.2:80...
* Connected to 10.0.0.2 (10.0.0.2) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.0.2
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
<
Hello
```

2.3.2. Task 2

- Create a python program to list all flow rules per device id, the code must be generic for any given network topology.

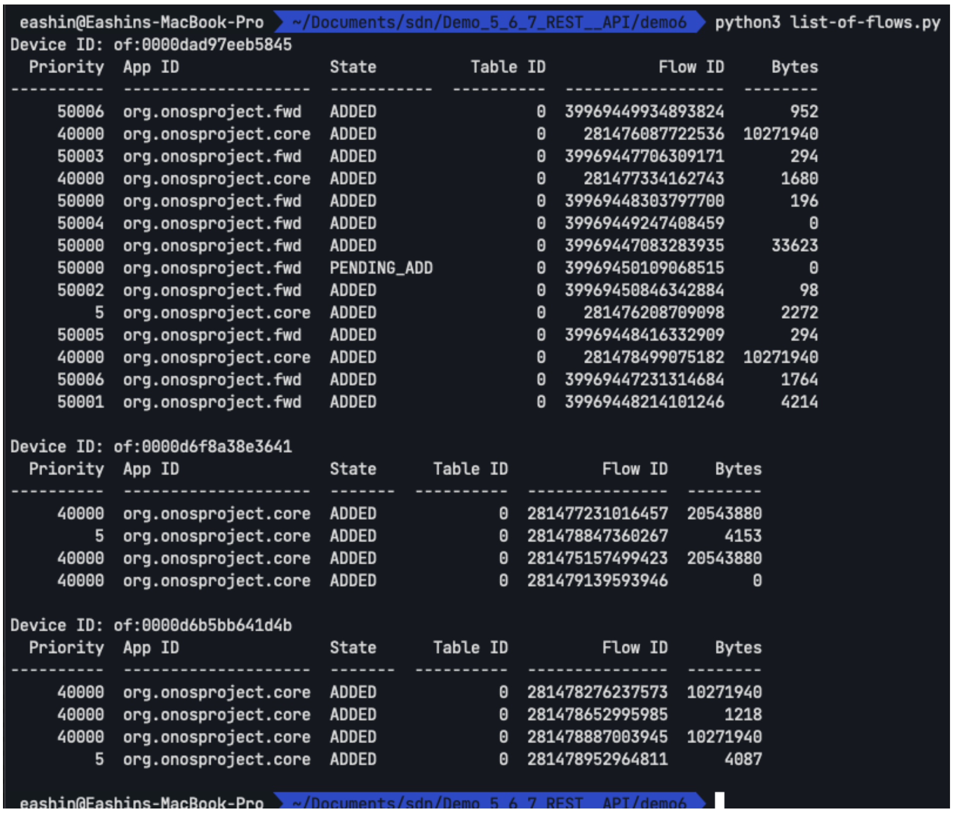
Here is the script to list all flow rules and list it by device id. Here it first GET all flow rules and then reformat it to output as a table by device id using some important field of the flow.



Users > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > list-of-flows.py > ...

```python
import requests
from requests.auth import HTTPBasicAuth
import json
from tabulate import tabulate

# Set the URL of the ONOS controller
url = "http://192.168.64.16:8181/onos/v1/flows/"
# Set the authentication
auth = HTTPBasicAuth('onos', 'rocks')
#GET /flows/
response = requests.get(url, auth=auth)
flows = response.json()["flows"]
flows_by_device = {}

# Loop through each flow in the flows list
for flow in flows:
    device_id = flow["deviceId"]
    flow_info = [flow["priority"], flow["appId"], flow["state"], flow["tableId"],flow["id"], flow["bytes"]]
    if device_id not in flows_by_device:
        flows_by_device[device_id] = []
    flows_by_device[device_id].append(flow_info)

# Loop through each device id in the dictionary and output the flows data in a table
for device_id in flows_by_device:
    print("Device ID:", device_id)
    print(tabulate(flows_by_device[device_id], headers=["Priority", "App ID", "State", "Table ID", "Flow ID","Bytes"]))
    print()
```

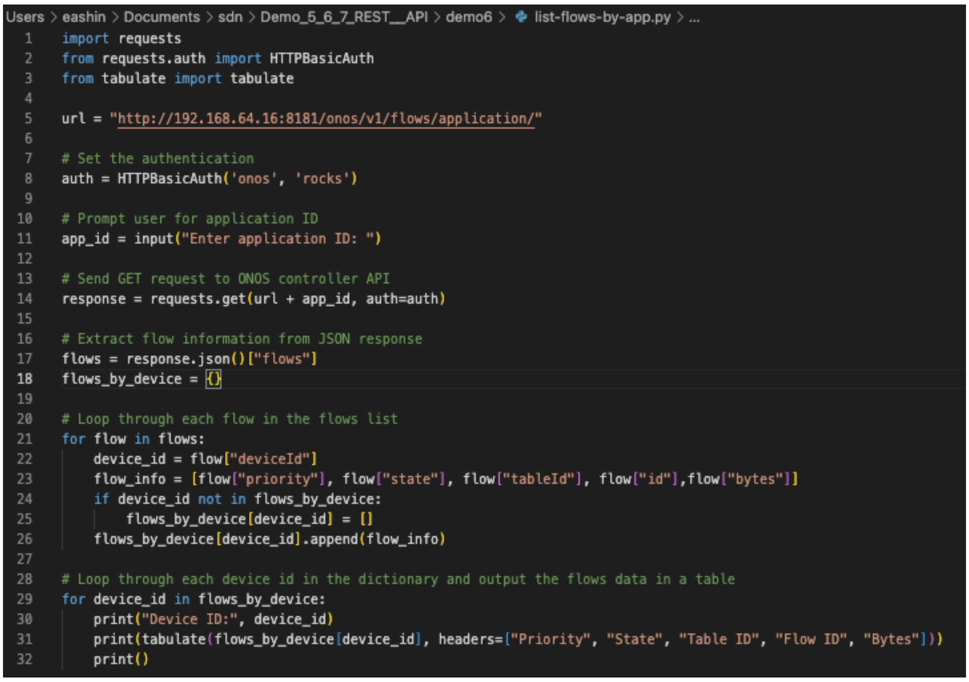
Here is the output of the program as table and all flow rules bu device id.



```
 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 list-of-flows.py
Device ID: of:0000dad97eeb5845
  Priority  App ID                State          Table ID            Flow ID     Bytes
----------  --------------------  -----------  ----------  -----------------  --------
     50006  org.onosproject.fwd   ADDED                 0  39969449934893824       952
     40000  org.onosproject.core  ADDED                 0    281476087722536  10271940
     50003  org.onosproject.fwd   ADDED                 0  39969447706309171       294
     40000  org.onosproject.core  ADDED                 0    281477334162743      1680
     50000  org.onosproject.fwd   ADDED                 0  39969448303797700       196
     50004  org.onosproject.fwd   ADDED                 0  39969449247408459         0
     50000  org.onosproject.fwd   ADDED                 0  39969447083283935     33623
     50000  org.onosproject.fwd   PENDING_ADD           0  39969450109068515         0
     50002  org.onosproject.fwd   ADDED                 0  39969450846342884        98
         5  org.onosproject.core  ADDED                 0    281476208709098      2272
     50005  org.onosproject.fwd   ADDED                 0  39969448416332909       294
     40000  org.onosproject.core  ADDED                 0    281478499075182  10271940
     50006  org.onosproject.fwd   ADDED                 0  39969447231314684      1764
     50001  org.onosproject.fwd   ADDED                 0  39969448214101246      4214

Device ID: of:0000d6f8a38e3641
  Priority  App ID                State      Table ID          Flow ID     Bytes
----------  --------------------  -------  ----------  ---------------  --------
     40000  org.onosproject.core  ADDED             0  281477231016457  20543880
         5  org.onosproject.core  ADDED             0  281478847360267      4153
     40000  org.onosproject.core  ADDED             0  281475157499423  20543880
     40000  org.onosproject.core  ADDED             0  281479139593946         0

Device ID: of:0000d6b5bb641d4b
  Priority  App ID                State      Table ID          Flow ID     Bytes
----------  --------------------  -------  ----------  ---------------  --------
     40000  org.onosproject.core  ADDED             0  281478276237573  10271940
     40000  org.onosproject.core  ADDED             0  281478652995985      1218
     40000  org.onosproject.core  ADDED             0  281478887003945  10271940
         5  org.onosproject.core  ADDED             0  281478952964811      4087

 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6
```

- Create a python program to list flow rules by application id.

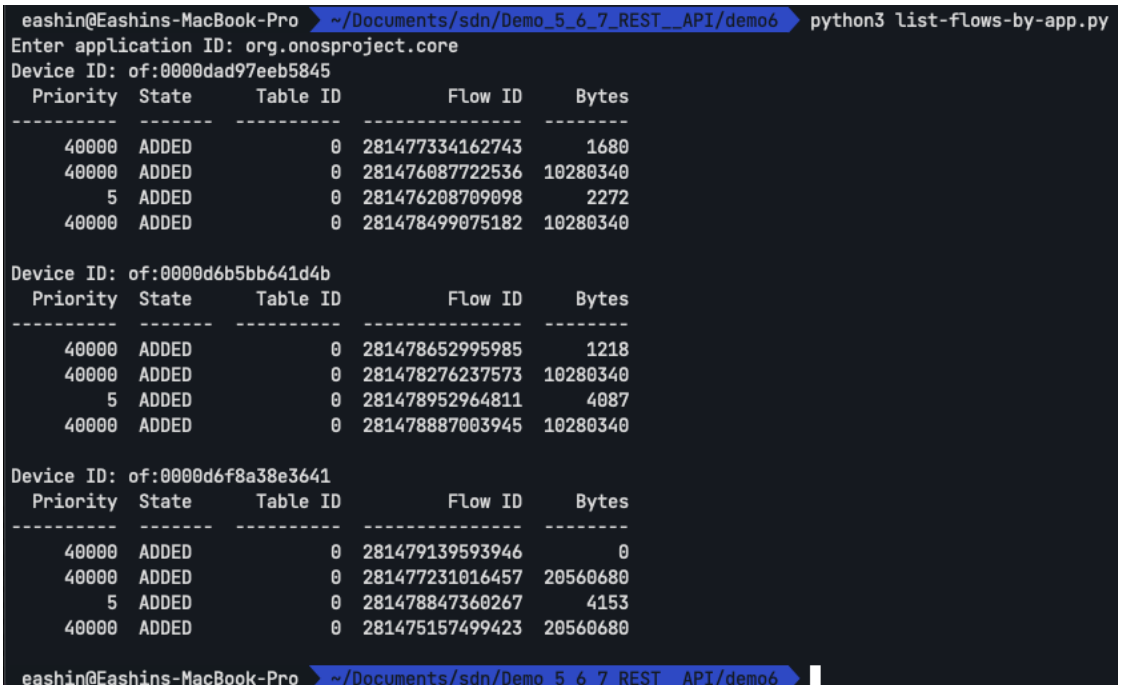
Here is the script to list flow rules by application Id and the script prompt user to input application id and used that to retrieve flow rules by the app ID.



Users > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > list-flows-by-app.py > ...

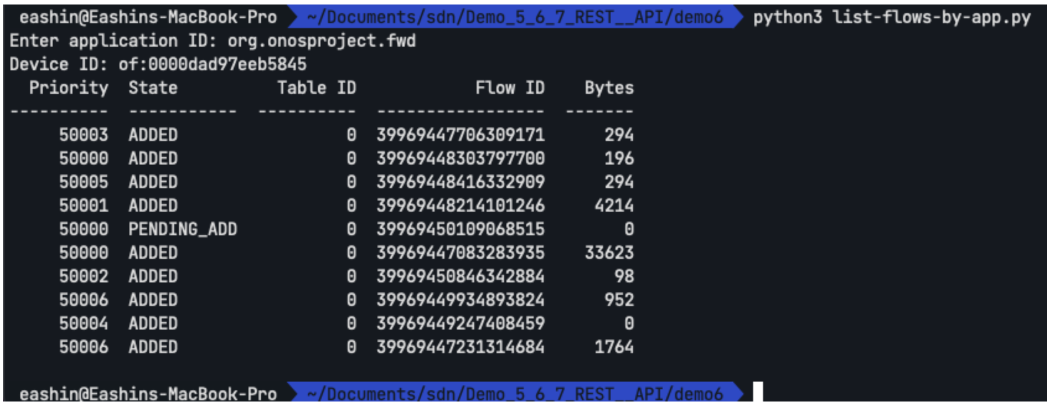
```python
import requests
from requests.auth import HTTPBasicAuth
from tabulate import tabulate

url = "http://192.168.64.16:8181/onos/v1/flows/application/"

# Set the authentication
auth = HTTPBasicAuth('onos', 'rocks')

# Prompt user for application ID
app_id = input("Enter application ID: ")

# Send GET request to ONOS controller API
response = requests.get(url + app_id, auth=auth)

# Extract flow information from JSON response
flows = response.json()["flows"]
flows_by_device = {}

# Loop through each flow in the flows list
for flow in flows:
    device_id = flow["deviceId"]
    flow_info = [flow["priority"], flow["state"], flow["tableId"], flow["id"],flow["bytes"]]
    if device_id not in flows_by_device:
        flows_by_device[device_id] = []
    flows_by_device[device_id].append(flow_info)

# Loop through each device id in the dictionary and output the flows data in a table
for device_id in flows_by_device:
    print("Device ID:", device_id)
    print(tabulate(flows_by_device[device_id], headers=["Priority", "State", "Table ID", "Flow ID", "Bytes"]))
    print()
```

Expected output of the program below by using `org.onosproject.core`



```
 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 list-flows-by-app.py
Enter application ID: org.onosproject.core
Device ID: of:0000dad97eeb5845
  Priority  State      Table ID          Flow ID     Bytes
----------  -------  ----------  ---------------  --------
     40000  ADDED             0  281477334162743      1680
     40000  ADDED             0  281476087722536  10280340
         5  ADDED             0  281476208709098      2272
     40000  ADDED             0  281478499075182  10280340

Device ID: of:0000d6b5bb641d4b
  Priority  State      Table ID          Flow ID     Bytes
----------  -------  ----------  ---------------  --------
     40000  ADDED             0  281478652995985      1218
     40000  ADDED             0  281478276237573  10280340
         5  ADDED             0  281478952964811      4087
     40000  ADDED             0  281478887003945  10280340

Device ID: of:0000d6f8a38e3641
  Priority  State      Table ID          Flow ID     Bytes
----------  -------  ----------  ---------------  --------
     40000  ADDED             0  281479139593946         0
     40000  ADDED             0  281477231016457  20560680
         5  ADDED             0  281478847360267      4153
     40000  ADDED             0  281475157499423  20560680

 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6 
```

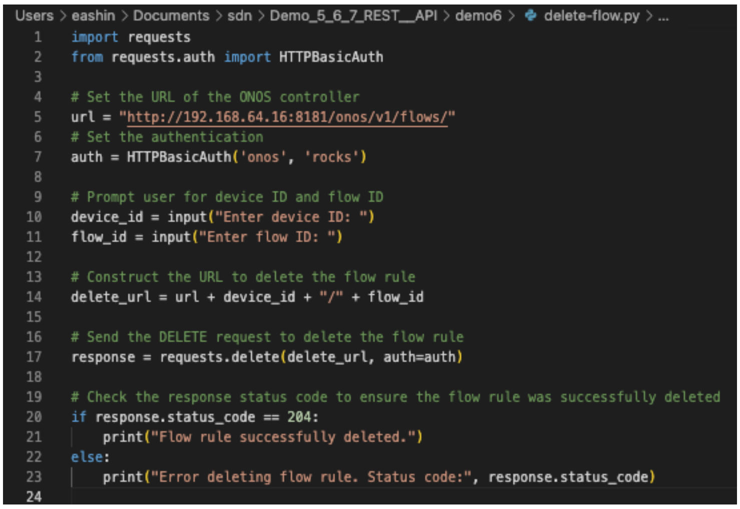
Expected output of the program below by using `org.onosproject.fwd`



```
 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 list-flows-by-app.py
Enter application ID: org.onosproject.fwd
Device ID: of:0000dad97eeb5845
  Priority  State          Table ID            Flow ID    Bytes
----------  -----------  ----------  -----------------  -------
     50003  ADDED                 0  39969447706309171      294
     50000  ADDED                 0  39969448303797700      196
     50005  ADDED                 0  39969448416332909      294
     50001  ADDED                 0  39969448214101246     4214
     50000  PENDING_ADD           0  39969450109068515        0
     50000  ADDED                 0  39969447083283935    33623
     50002  ADDED                 0  39969450846342884       98
     50006  ADDED                 0  39969449934893824      952
     50004  ADDED                 0  39969449247408459        0
     50006  ADDED                 0  39969447231314684     1764

 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6 
```

- Create a python program to delete flow rules knowing the device id and the flow-id.

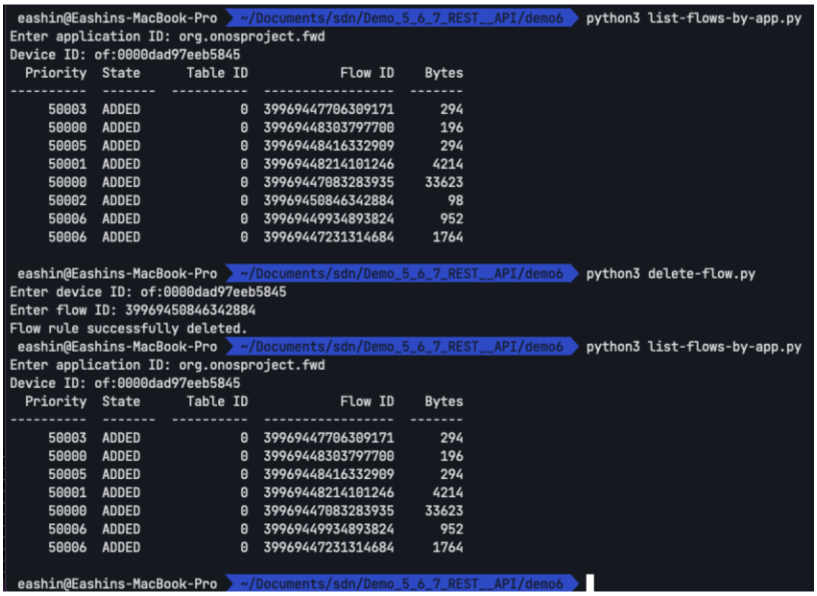
Here is the script to DELETE flow rules by device Id and flow Id:



Users > eashin > Documents > sdn > Demo_5_6_7_REST__API > demo6 > delete-flow.py > ...

```python
import requests
from requests.auth import HTTPBasicAuth

# Set the URL of the ONOS controller
url = "http://192.168.64.16:8181/onos/v1/flows/"
# Set the authentication
auth = HTTPBasicAuth('onos', 'rocks')

# Prompt user for device ID and flow ID
device_id = input("Enter device ID: ")
flow_id = input("Enter flow ID: ")

# Construct the URL to delete the flow rule
delete_url = url + device_id + "/" + flow_id

# Send the DELETE request to delete the flow rule
response = requests.delete(delete_url, auth=auth)

# Check the response status code to ensure the flow rule was successfully deleted
if response.status_code == 204:
    print("Flow rule successfully deleted.")
else:
    print("Error deleting flow rule. Status code:", response.status_code)

```

Here tested that list all flow by device Id that has flow ID `39969450846342884,` we deleted that using the script and we list again the flow rules not exist anymore.



```
 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 list-flows-by-app.py
Enter application ID: org.onosproject.fwd
Device ID: of:0000dad97eeb5845
  Priority  State      Table ID            Flow ID    Bytes
----------  -------  ----------  -----------------  -------
     50003  ADDED             0  39969447706309171      294
     50000  ADDED             0  39969448303797700      196
     50005  ADDED             0  39969448416332909      294
     50001  ADDED             0  39969448214101246     4214
     50000  ADDED             0  39969447083283935    33623
     50002  ADDED             0  39969450846342884       98
     50006  ADDED             0  39969449934893824      952
     50006  ADDED             0  39969447231314684     1764

 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 delete-flow.py
Enter device ID: of:0000dad97eeb5845
Enter flow ID: 39969450846342884
Flow rule successfully deleted.
 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6  python3 list-flows-by-app.py
Enter application ID: org.onosproject.fwd
Device ID: of:0000dad97eeb5845
  Priority  State      Table ID            Flow ID    Bytes
----------  -------  ----------  -----------------  -------
     50003  ADDED             0  39969447706309171      294
     50000  ADDED             0  39969448303797700      196
     50005  ADDED             0  39969448416332909      294
     50001  ADDED             0  39969448214101246     4214
     50000  ADDED             0  39969447083283935    33623
     50006  ADDED             0  39969449934893824      952
     50006  ADDED             0  39969447231314684     1764

 eashin@Eashins-MacBook-Pro  ~/Documents/sdn/Demo_5_6_7_REST__API/demo6 
```