SDN Fundamentals & Techniques

Chapter 5 - Demo 4 - Using ONOS RESTful API, Linux containers, and Intent-based networking to orchestrate SFC

Jing Yan (yanj3, jing.yan@aalto.fi) Mar 10th, 2020

Chapter 5 - Demo 4 - Using ONOS RESTful API, Linux containers, and Intent-based networking to orchestrate SFC **1**

0. Env Preparation **3**
Code 3
Result 4

1. TASK **4**
1.1 Create a Host-to-Host Intent to allow communication between the “RED" and “GREEN" containers. 4
Code 4
Result 4
1.2 Using the TCPDUMP utility or other similar tool, try to identify the path taken by the traffic from the source, i.e., “RED”, to the destination, i.e., "GREEN" and report it. 5
1.3 What do you observe? 6
1.4 Do you think that Host-to-Host Intents can be useful in SFC deployments? 6
1.5 Delete all the created Intents using Python-based code and ONOS RESTful API. 6
1.6 What kind of Intents can be used to activate SFC communication in the current scenario? 6
1.7 Using the right type of Intents, create a python-based code to steer the traffic as shown in Fig.3. Note that you do not need an Intent between interfaces 10.0.0.111 and 10.0.0.112. 6
1.8 Explain the method used to create the end-to-end path. 6

1.1. Introduction

This assignment aims to develop hands-on experience on Service Function Chaining (SFC) and SDN-based system orchestration, through the use of the ONOS RESTful API. This demonstration is part of the application layer as it introduces students to general orchestration methods and programmability in SDN-based environments. By completing this assignment, students will be familiar with the basics and operations of creating static SFC instances and their related networking traffic. Besides, students will develop technical knowledge to understand complex implementations used in well-known open-source projects such as Open-Stack.

1.2. Pre-Task

Please use the python code created in the previous demonstration to access ONOS RESTful API interface and activate the required ONOS applications. The list of required applications may be found in the following file "list_applications" at http://www.mosaic-lab.org/SDN_Demos.aspx under "Chapter 5 Demos- ONOS-based orchestration system Demo4"

1.3. Tasks

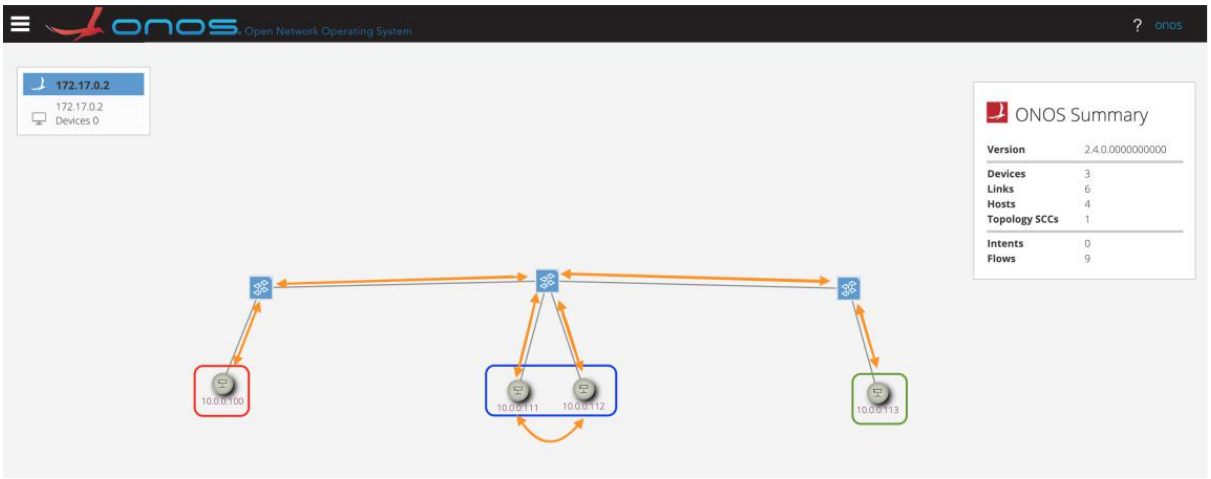
All tasks are performed in a pre-defined topology in which students are asked to create various Intents to achieve the desired SFC communication. Fig. 3 shows a simple SFC made up of a source, i.e., RED container, a destination, i.e., GREEN container, and a simplified Service Function (SF) dubbed BLUE. The purpose of BLUE SF is to redirect traffic received from one interface to another. For example, if the BLUE container receives ingress traffic from interface "10.0.0.112", it will output the same traffic from "10.0.0.111" and vice versa.

Figure 3: Task Topology.

1.3.1. Task 1

The file related to Task 1 is stored in the file "sfc.py" at http://www.mosaic-lab.org/SDN_Demos.aspx under "Chapter 5 Demos- ONOS-based orchestration system Demo4". This shell script creates a topology as depicted in Fig. 3. A host in the ONOS's GUI is an LXC, however, network namespaces or LXD containers can be used. In this task you are asked to:

Note that for each Intent, the student is asked to provide details and explanations. Please use only the ONOS RESTful API interface to enter the Intents and create the SFC traffic.



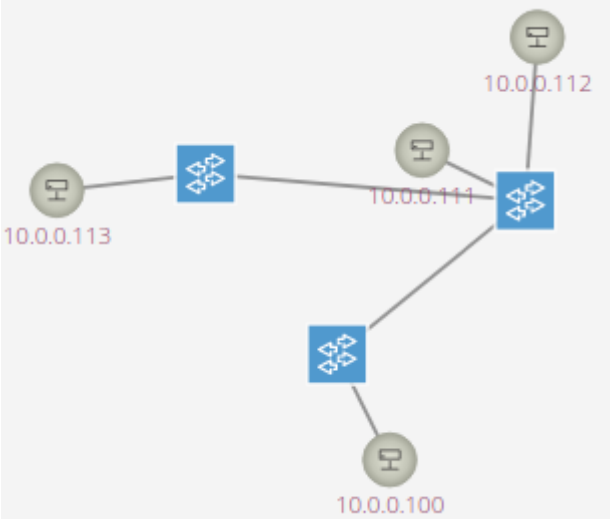
0. Env Preparation

Code

```
#!/usr/bin/env bash
create_ns() {
   echo "Creating the namespace $1"
   ip netns add $1
}

create_ovs_bridge() {
   echo "Creating the OVS bridge $1"
   ovs-vsctl add-br $1
}

attach_ns_to_ovs() {
   echo "Attaching the namespace $1 to the OVS $2"
   ip link add $3 type veth peer name $4
   ip link set $3 netns $1
   ovs-vsctl add-port $2 $4 -- set Interface $4 ofport_request=$5
   ip netns exec $1 ip addr add $6/24 dev $3
   ip netns exec $1 ip link set dev $3 up
   ip link set $4 up
}

attach_ovs_to_ovs() {
   echo "Attaching the OVS $1 to the OVS $2"
   ip link add name $3 type veth peer name $4
   ip link set $3 up
   ip link set $4 up
   ovs-vsctl add-port $1 $3 -- set Interface $3 ofport_request=$5
   ovs-vsctl add-port $2 $4 -- set Interface $4 ofport_request=$5
}

attach_ovs_to_sdn() {
   echo "Attaching the OVS bridge to the ONOS controller"
   ovs-vsctl set-controller $1 tcp:$(docker inspect -f '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -q  --filter
ancestor=onosproject/onos)):6653
}

create_ns red
create_ns blue1
create_ns blue2
create_ns green

create_ovs_bridge ovs-1
create_ovs_bridge ovs-2
create_ovs_bridge ovs-3
ovs-vsctl set bridge ovs-1
protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13,OpenFlow14
ovs-vsctl set bridge ovs-2
protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13,OpenFlow14
ovs-vsctl set bridge ovs-3
protocols=OpenFlow10,OpenFlow11,OpenFlow12,OpenFlow13,OpenFlow14

attach_ovs_to_ovs ovs-1 ovs-2 int-ovs12 int-ovs21 1
attach_ovs_to_ovs ovs-2 ovs-3 int-ovs23 int-ovs32 2

attach_ovs_to_sdn ovs-1
attach_ovs_to_sdn ovs-2
attach_ovs_to_sdn ovs-3

attach_ns_to_ovs red ovs-1 veth-red veth-red-br 4 10.0.0.100
attach_ns_to_ovs blue1 ovs-2 veth-blue1 veth-blue1-br 5 10.0.0.111
```

```
attach_ns_to_ovs blue2 ovs-2 veth-blue2 veth-blue2-br 6 10.0.0.112
attach_ns_to_ovs green ovs-3 veth-green veth-green-br 2 10.0.0.113
```

Result



1. TASK

1.1 Create a Host-to-Host Intent to allow communication between the “RED" and “GREEN" containers.

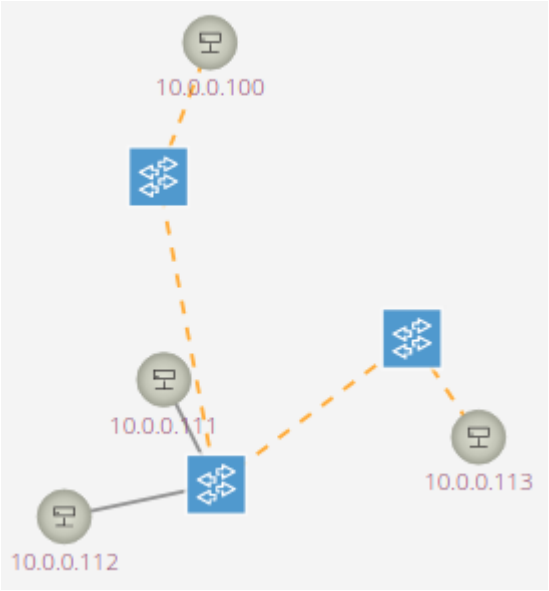
Code

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

if __name__ == "__main__":
   headers = {"Content-type": "application/json", "Accept": "application/json"}
   rule0 =
{"type":"HostToHostIntent","appId":"org.onosproject.cli","resources":["96:E8:3C:42:
33:B3/None","A2:61:B4:B2:47:2A/None"],"selector":{"criteria":[]},"treatment":{"inst
ructions":[{"type":"NOACTION"}],"deferred":[]},"priority":100,"constraints":[{"incl
usive":"false","types":["OPTICAL"],"type":"LinkTypeConstraint"}],"one":"96:E8:3C:42
:33:B3/None","two":"A2:61:B4:B2:47:2A/None"}
   res0 = requests.post("http://100.109.0.1:8181/onos/v1/intents",
json.dumps(rule0), headers = headers, auth = HTTPBasicAuth('onos', 'rocks'))
```

Result

```
# python task11.py
# curl -u onos:rocks -X GET http://100.109.0.1:8181/onos/v1/intents
{"intents":[{"type":"HostToHostIntent","id":"0xa","key":"0xa","appId":"org.onosproj
ect.cli","resources":["FE:E5:05:F7:EF:30/None","E6:8F:D8:D7:41:89/None"],"state":"I
NSTALLED"}]}
# curl -u onos:rocks -X GET
http://100.109.0.1:8181/onos/v1/intents/org.onosproject.cli/0
{"type":"HostToHostIntent","id":"0x0","key":"0x0","appId":"org.onosproject.cli","re
sources":["96:E8:3C:42:33:B3/None","A2:61:B4:B2:47:2A/None"],"state":"INSTALLED","s
elector":{"criteria":[]},"treatment":{"instructions":[{"type":"NOACTION"}],"deferre
d":[]},"priority":100,"constraints":[{"inclusive":false,"types":["OPTICAL"],"type":
```

```
"LinkTypeConstraint"}],"one":"96:E8:3C:42:33:B3/None","two":"A2:61:B4:B2:47:2A/None
"}
```

1.2 Using the TCPDUMP utility or other similar tool, try to identify the path taken by the traffic from the source, i.e., “RED”, to the destination, i.e., "GREEN" and report it.

Note that TCPDUMP does not need to be installed inside containers. This utility can be used from the hosting machine by specifying the name of the interfaces used by the containers. Use the ICMP traffic generated by the ping command as an example.

```
# ip netns exec red ping -c1 10.0.0.113
PING 10.0.0.113 (10.0.0.113) 56(84) bytes of data.
64 bytes from 10.0.0.113: icmp_seq=1 ttl=64 time=1.52 ms

--- 10.0.0.113 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.523/1.523/1.523/0.000 ms

# tcpdump -i veth-blue1-br icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth-blue1-br, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel

# tcpdump -i veth-blue2-br icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth-blue2-br, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel

# tcpdump -i veth-green-br icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth-green-br, link-type EN10MB (Ethernet), capture size 262144 bytes
22:56:36.007807 IP 10.0.0.100 > 10.0.0.113: ICMP echo request, id 1217, seq 1,
length 64
22:56:36.007895 IP 10.0.0.113 > 10.0.0.100: ICMP echo reply, id 1217, seq 1, length
64
```

```
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
```

1.3 What do you observe?

From the tcpdump report above showed on the interface of the green namespace, the src is the IP of the red namespace, the dst is the green namespace, and no any pkg on blue namespaces.

1.4 Do you think that Host-to-Host Intents can be useful in SFC deployments?

No.

1.5 Delete all the created Intents using Python-based code and ONOS RESTful API.

Code

```
#!/usr/bin/env python3
import requests, json
from requests.auth import HTTPBasicAuth

if __name__ == "__main__":
   headers = {"Content-type": "application/json", "Accept": "application/json"}
   res0 = requests.get("http://100.109.0.1:8181/onos/v1/intents/", auth =
HTTPBasicAuth('onos', 'rocks'))
   # Get all intent rules and store in a list and print
   for index, value in enumerate(res0.json()["intents"]):
       requests.delete("http://100.109.0.1:8181/onos/v1/intents/%s/%s" %
(value["appId"], value["id"]), auth = HTTPBasicAuth('onos', 'rocks'))
```

Result

```
# python task15.py

# curl -u onos:rocks -X GET http://100.109.0.1:8181/onos/v1/intents/
{"intents":[]}
```

1.6 What kind of Intents can be used to activate SFC communication in the current scenario?

Point-to-Point intents

1.7 Using the right type of Intents, create a python-based code to steer the traffic as shown in Fig.3. Note that you do not need an Intent between interfaces 10.0.0.111 and 10.0.0.112.

Code

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

if __name__ == "__main__":
   headers = {"Content-type": "application/json", "Accept": "application/json"}
   # From red to OVS2
   rule0 =
{"type":"PointToPointIntent","appId":"org.onosproject.cli","resources":[],"selector
":{"criteria":[]},"treatment":{"instructions":[{"type":"NOACTION"}],"deferred":[]},
"priority":100,"constraints":[],"ingressPoint":{"port":"1","device":"of:00000a6890f
e9f46"},"egressPoint":{"port":"2","device":"of:0000e635ce801942"}}
   res0 = requests.post("http://100.109.0.1:8181/onos/v1/intents",
json.dumps(rule0), headers = headers, auth = HTTPBasicAuth('onos', 'rocks'))
   # From green to OVS2
   rule1 =
{"type":"PointToPointIntent","appId":"org.onosproject.cli","resources":[],"selector
":{"criteria":[]},"treatment":{"instructions":[{"type":"NOACTION"}],"deferred":[]},
"priority":100,"constraints":[],"ingressPoint":{"port":"5","device":"of:0000e635ce8
01942"},"egressPoint":{"port":"1","device":"of:00006a724f688f41"}}
   res1 = requests.post("http://100.109.0.1:8181/onos/v1/intents",
json.dumps(rule1), headers = headers, auth = HTTPBasicAuth('onos', 'rocks'))
   # From blue1 to OVS1
   rule2 =
{"type":"PointToPointIntent","appId":"org.onosproject.cli","resources":[],"selector
":{"criteria":[]},"treatment":{"instructions":[{"type":"NOACTION"}],"deferred":[]},
"priority":100,"constraints":[],"ingressPoint":{"port":"6","device":"of:0000e635ce8
01942"},"egressPoint":{"port":"2","device":"of:00000a6890fe9f46"}}
   res2 = requests.post("http://100.109.0.1:8181/onos/v1/intents",
json.dumps(rule2), headers = headers, auth = HTTPBasicAuth('onos', 'rocks'))
   # From blue2 to OVS1
   rule3 =
{"type":"PointToPointIntent","appId":"org.onosproject.cli","resources":[],"selector
":{"criteria":[]},"treatment":{"instructions":[{"type":"NOACTION"}],"deferred":[]},
"priority":100,"constraints":[],"ingressPoint":{"port":"4","device":"of:00006a724f6
88f41"},"egressPoint":{"port":"1","device":"of:0000e635ce801942"}}
   res3 = requests.post("http://100.109.0.1:8181/onos/v1/intents",
json.dumps(rule3), headers = headers, auth = HTTPBasicAuth('onos', 'rocks'))
```

Result

```
# curl -u onos:rocks -X GET
http://100.109.0.1:8181/onos/v1/intents/org.onosproject.cli/0x22
{"type":"PointToPointIntent","id":"0x22","key":"0x22","appId":"org.onosproject.cli"
,"resources":[],"state":"INSTALLED","selector":{"criteria":[]},"treatment":{"instru
ctions":[{"type":"NOACTION"}],"deferred":[]},"priority":100,"constraints":[],"ingre
ssPoint":{"port":"6","device":"of:0000e635ce801942"},"egressPoint":{"port":"2","dev
ice":"of:00000a6890fe9f46"}}root@elxa9698s73:/home/ejinyna/Desktop/SDN/chapter5_dem
o4#
# curl -u onos:rocks -X GET
http://100.109.0.1:8181/onos/v1/intents/org.onosproject.cli/0x21
{"type":"PointToPointIntent","id":"0x21","key":"0x21","appId":"org.onosproject.cli"
,"resources":[],"state":"INSTALLED","selector":{"criteria":[]},"treatment":{"instru
ctions":[{"type":"NOACTION"}],"deferred":[]},"priority":100,"constraints":[],"ingre
ssPoint":{"port":"5","device":"of:0000e635ce801942"},"egressPoint":{"port":"1","dev
ice":"of:00006a724f688f41"}}root@elxa9698s73:/home/ejinyna/Desktop/SDN/chapter5_dem
o4#
# curl -u onos:rocks -X GET
http://100.109.0.1:8181/onos/v1/intents/org.onosproject.cli/0x20
```

```
{"type":"PointToPointIntent","id":"0x20","key":"0x20","appId":"org.onosproject.cli"
,"resources":[],"state":"INSTALLED","selector":{"criteria":[]},"treatment":{"instru
ctions":[{"type":"NOACTION"}],"deferred":[]},"priority":100,"constraints":[],"ingre
ssPoint":{"port":"1","device":"of:00000a6890fe9f46"},"egressPoint":{"port":"2","dev
ice":"of:0000e635ce801942"}}root@elxa9698s73:/home/ejinyna/Desktop/SDN/chapter5_dem
o4#
# curl -u onos:rocks -X GET
http://100.109.0.1:8181/onos/v1/intents/org.onosproject.cli/0x23
{"type":"PointToPointIntent","id":"0x23","key":"0x23","appId":"org.onosproject.cli"
,"resources":[],"state":"INSTALLED","selector":{"criteria":[]},"treatment":{"instru
ctions":[{"type":"NOACTION"}],"deferred":[]},"priority":100,"constraints":[],"ingre
ssPoint":{"port":"4","device":"of:00006a724f688f41"},"egressPoint":{"port":"1","dev
ice":"of:0000e635ce801942"}}
```

1.8 Explain the method used to create the end-to-end path.

End-to-end path includes the type of traffic and the source and destination hosts, or ingress and egress ports to request connectivity. The service provisions this connectivity over the appropriate paths and then continuously monitors the network, changing the paths over time to continue meeting the objectives prescribed by the intent in the face of varying network conditions.