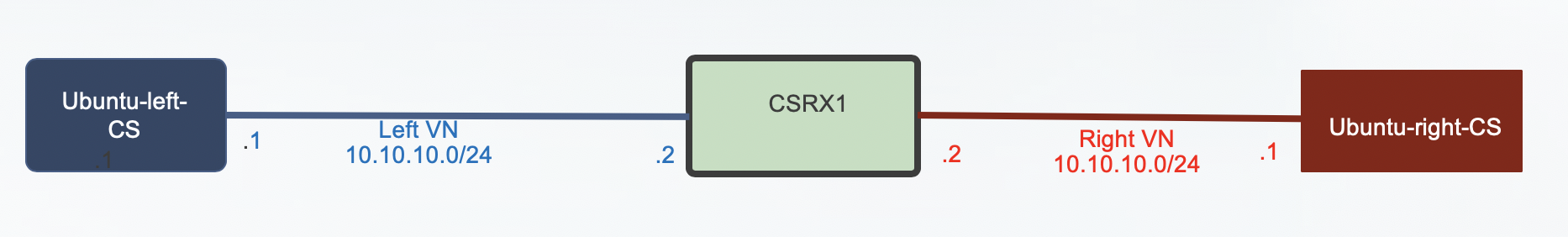
Service Chaining   
service chaining is the idea of forwarding traffic through multiple network entity in a certain order, each network entity do specific function such as firewall, IPS , NAT , LB , …,etc   
the legacy way of doing service chaining would use standalone HW appliances which made service chaining inflexible, expensive and takes a long time to setup  
Dynamic service chaining is where network functions deployed as VM or Container and could be chained automatically in a logical way.  
in the next example we use contrail for services chaining between two PODs in two different networking using CSRX container L4-L7 firewall to secure the traffic between these two networks as shown in the diagram   
  
  
  
Tips :

* left and right networks are just a common name used for simplicity and expected the traffic to follow from left to right but you can use your own names
* make sure to configure the network before you attached a POD to it otherwise POD would fail to be created

so let’s start create two networks using this YAML files

[root@cent11]# cat vn-left.yaml

apiVersion: k8s.cni.cncf.io/v1

kind: NetworkAttachmentDefinition

metadata:

annotations:

opencontrail.org/cidr: “10.10.10.0/24"

opencontrail.org/ip\_fabric\_forwarding: "false"

opencontrail.org/ip\_fabric\_snat: "false"

name: vn-left

namespace: default

spec:

config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'

[root@cent11]# cat vn-left.yaml

apiVersion: k8s.cni.cncf.io/v1

kind: NetworkAttachmentDefinition

metadata:

annotations:

opencontrail.org/cidr: “10.20.20.0/24"

opencontrail.org/ip\_fabric\_forwarding: "false"

opencontrail.org/ip\_fabric\_snat: "false"

name: vn-right

namespace: default

spec:

config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'

[root@cent11]# kubectl create -f vn-left.yaml

[root@cent11]# kubectl create -f vn-right.yaml

Verify using Kubectl

[root@cent11 ~]# kubectl get network-attachment-definition

NAME AGE

vn-left 19d

vn-right 17d

[root@cent11 ~]# kubectl describe network-attachment-definition

Name: vn-left

Namespace: default

Labels: <none>

Annotations: opencontrail.org/cidr: 10.10.10.0/24

opencontrail.org/ip\_fabric\_forwarding: false

opencontrail.org/ip\_fabric\_snat: false

API Version: k8s.cni.cncf.io/v1

Kind: NetworkAttachmentDefinition

Metadata:

Creation Timestamp: 2019-05-25T20:28:22Z

Generation: 1

Resource Version: 83111

Self Link: /apis/k8s.cni.cncf.io/v1/namespaces/default/network-attachment-definitions/vn-left

UID: a44fe276-7f2b-11e9-9ff0-0050569e2171

Spec:

Config: { "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }

Events: <none>

Name: vn-right

Namespace: default

Labels: <none>

Annotations: opencontrail.org/cidr: 10.20.20.0/24

opencontrail.org/ip\_fabric\_forwarding: false

opencontrail.org/ip\_fabric\_snat: false

API Version: k8s.cni.cncf.io/v1

Kind: NetworkAttachmentDefinition

Metadata:

Creation Timestamp: 2019-05-28T07:14:02Z

Generation: 1

Resource Version: 380427

Self Link: /apis/k8s.cni.cncf.io/v1/namespaces/default/network-attachment-definitions/vn-right

UID: 2b8d394f-8118-11e9-b36d-0050569e2171

Spec:

Config: { "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }

Events: <none>

It’s a good practice to confirm these two networks are seen now in contrail before proceeding.   
From the Contrail Controller module control node (<http://10.85.188.16:8143> in our setup), select Configure > Networking > Networks > default-domain > k8s-default  
As shown in the diagram which focus on left network

A screenshot of a computer

Description automatically generated

Tip: using namespace: default object in the YAML file for a network will create it n in domain “default-domain” and project “K8s-default”

Create two ubuntu Pods, one in each network using the annotation object

[root@cent11 ~]# cat left-ubuntu-sc.yaml

apiVersion: v1

kind: Pod

metadata:

name: left-ubuntu-sc

labels:

app: webapp-sc

annotations:

k8s.v1.cni.cncf.io/networks: '[

{ "name": "vn-left" }]'

spec:

containers:

- name: ubuntu-left-pod-sc

image: virtualhops/ato-ubuntu:latest

securityContext:

privileged: true

capabilities:

add:

- NET\_ADMIN

[root@cent11 ~]# cat right-ubuntu-sc.yaml

apiVersion: v1

kind: Pod

metadata:

name: right-ubuntu-sc

labels:

app: webapp-sc

annotations:

k8s.v1.cni.cncf.io/networks: '[

{ "name": "vn-right" }]'

spec:

containers:

- name: ubuntu-right-pod-sc

image: virtualhops/ato-ubuntu:latest

securityContext:

privileged: true

capabilities:

add:

- NET\_ADMIN

[root@cent11 ~]# kubectl create -f right-ubuntu-sc.yaml

[root@cent11 ~]# kubectl create -f left-ubuntu-sc.yaml

[root@cent11 ~]# kubectl get pod

NAME READY STATUS RESTARTS AGE

left-ubuntu-sc 1/1 Running 0 25h

right-ubuntu-sc 1/1 Running 0 25h

create Juniper CSRX container that have one interface on the left network and one interface on the right network using this YAML file

[root@cent11 ~]# cat csrx1-sc.yaml

apiVersion: v1

kind: Pod

metadata:

name: csrx1-sc

labels:

app: webapp-sc

annotations:

k8s.v1.cni.cncf.io/networks: '[

{ "name": "vn-left" },

{ "name": "vn-right" }

]'

spec:

containers:

- name: csrx1-sc

image: csrx

ports:

- containerPort: 22

imagePullPolicy: Never

stdin: true

tty: true

securityContext:

privileged: true

[root@cent11 ~]# kubectl create -f csrx1-sc.yaml

Confirm the interface placement in the correct network

[root@cent11 ~]# kubectl describe pod

Name: csrx1-sc

Namespace: default

Priority: 0

PriorityClassName: <none>

Node: cent22/10.85.188.17

Start Time: Thu, 13 Jun 2019 03:40:31 -0400

Labels: app=webapp-sc

Annotations: k8s.v1.cni.cncf.io/network-status:

[

{

"ips": "10.10.10.2",

"mac": "02:84:71:f4:f2:8d",

"name": "vn-left"

},

{

"ips": "10.20.20.2",

"mac": "02:84:8b:4c:18:8d",

"name": "vn-right"

},

{

"ips": "10.47.255.248",

"mac": "02:84:59:7e:54:8d",

"name": "cluster-wide-default"

}

]

k8s.v1.cni.cncf.io/networks: [ { "name": "vn-left" }, { "name": "vn-right" } ]

Status: Running

IP: 10.47.255.248

Containers:

csrx1-sc:

Container ID: docker://82b7605172d937895269d76850d083b6dc6e278e41cb45b4cb8cee21283e4f17

Image: csrx

Image ID: docker://sha256:329e805012bdf081f4a15322f994e5e3116b31c90f108a19123cf52710c7617e

Port: 22/TCP

Host Port: 0/TCP

State: Running

Started: Thu, 13 Jun 2019 03:40:46 -0400

Ready: True

Restart Count: 0

Environment: <none>

Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-m75c5 (ro)

Conditions:

Type Status

Initialized True

Ready True

ContainersReady True

PodScheduled True

Volumes:

default-token-m75c5:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-m75c5

Optional: false

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events: <none>

Name: left-ubuntu-sc

Namespace: default

Priority: 0

PriorityClassName: <none>

Node: cent22/10.85.188.17

Start Time: Thu, 13 Jun 2019 03:40:20 -0400

Labels: app=webapp-sc

Annotations: k8s.v1.cni.cncf.io/network-status:

[

{

"ips": "10.10.10.1",

"mac": "02:7d:b1:09:00:8d",

"name": "vn-left"

},

{

"ips": "10.47.255.249",

"mac": "02:7d:99:ff:62:8d",

"name": "cluster-wide-default"

}

]

k8s.v1.cni.cncf.io/networks: [ { "name": "vn-left" }]

Status: Running

IP: 10.47.255.249

Containers:

ubuntu-left-pod-sc:

Container ID: docker://2f9a22568d844c68a1c4a45de4a81478958233052e08d4473742827482b244cd

Image: virtualhops/ato-ubuntu:latest

Image ID: docker-pullable://virtualhops/ato-ubuntu@sha256:fa2930cb8f4b766e5b335dfa42de510ecd30af6433ceada14cdaae8de9065d2a

Port: <none>

Host Port: <none>

State: Running

Started: Thu, 13 Jun 2019 03:40:27 -0400

Ready: True

Restart Count: 0

Environment: <none>

Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-m75c5 (ro)

Conditions:

Type Status

Initialized True

Ready True

ContainersReady True

PodScheduled True

Volumes:

default-token-m75c5:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-m75c5

Optional: false

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events: <none>

Name: right-ubuntu-sc

Namespace: default

Priority: 0

PriorityClassName: <none>

Node: cent22/10.85.188.17

Start Time: Thu, 13 Jun 2019 04:09:18 -0400

Labels: app=webapp-sc

Annotations: k8s.v1.cni.cncf.io/network-status:

[

{

"ips": "10.20.20.1",

"mac": "02:89:cc:86:48:8d",

"name": "vn-right"

},

{

"ips": "10.47.255.252",

"mac": "02:89:b0:8e:98:8d",

"name": "cluster-wide-default"

}

]

k8s.v1.cni.cncf.io/networks: [ { "name": "vn-right" }]

Status: Running

IP: 10.47.255.252

Containers:

ubuntu-right-pod-sc:

Container ID: docker://4e0b6fa085905be984517a11c3774517d01f481fa43aadd76a633ef15c58cbfe

Image: virtualhops/ato-ubuntu:latest

Image ID: docker-pullable://virtualhops/ato-ubuntu@sha256:fa2930cb8f4b766e5b335dfa42de510ecd30af6433ceada14cdaae8de9065d2a

Port: <none>

Host Port: <none>

State: Running

Started: Thu, 13 Jun 2019 04:09:25 -0400

Ready: True

Restart Count: 0

Environment: <none>

Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-m75c5 (ro)

Conditions:

Type Status

Initialized True

Ready True

ContainersReady True

PodScheduled True

Volumes:

default-token-m75c5:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-m75c5

Optional: false

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events: <none>

Note: each container has one interface belong to “cluster-wide-default” network regardless the use of the annotations object because annotations object above creates and put one extra interface in a specific network

Login to the left, right Pods and the CSRX to confirm the IP/MAC address

[root@cent11 ~]# kubectl exec -it left-ubuntu-sc bash

root@left-ubuntu-sc:/# ip a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:7d:99:ff:62:8d brd ff:ff:ff:ff:ff:ff

inet 10.47.255.249/12 scope global eth0

valid\_lft forever preferred\_lft forever

15: eth1@if16: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:7d:b1:09:00:8d brd ff:ff:ff:ff:ff:ff

inet 10.10.10.1/24 scope global eth1

valid\_lft forever preferred\_lft forever

[root@cent11 ~]# kubectl exec -it right-ubuntu-sc bash

root@right-ubuntu-sc:/# ip a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:89:b0:8e:98:8d brd ff:ff:ff:ff:ff:ff

inet 10.47.255.252/12 scope global eth0

valid\_lft forever preferred\_lft forever

25: eth1@if26: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc noqueue state UP group default

link/ether 02:89:cc:86:48:8d brd ff:ff:ff:ff:ff:ff

inet 10.20.20.1/24 scope global eth1

valid\_lft forever preferred\_lft forever

[root@cent11 ~]# kubectl exec -it csrx1-sc cli  
root@csrx1-sc>

root@csrx1-sc> show interfaces

Physical interface: ge-0/0/1, Enabled, Physical link is Up

Interface index: 100

Link-level type: Ethernet, MTU: 1514

Current address: 02:84:71:f4:f2:8d, Hardware address: 02:84:71:f4:f2:8d

Physical interface: ge-0/0/0, Enabled, Physical link is Up

Interface index: 200

Link-level type: Ethernet, MTU: 1514

Current address: 02:84:8b:4c:18:8d, Hardware address: 02:84:8b:4c:18:8d

Note : unlike other PODs the CSRX didn’t acquire IP with DHCP and it start with factory default configuration hence it need to be configured.

Tip: By default, CSRX eth0 is visible only from shell and used for management.

And when attaching networks, the first attach network is mapped to eth1 which is GE-0/0/1

And the second attach is mapped to eth2 which is GE-0/0/0

Configure this basic setup on the CSRX, to assign the correct IP address use the MAC/IP address mapping from the “ kubectl describe pod” command show output as well configure default security policy to allow everything for now

set interfaces ge-0/0/1 unit 0 family inet address 10.10.10.2/24

set interfaces ge-0/0/0 unit 0 family inet address 10.20.20.2/24

set security zones security-zone trust interfaces ge-0/0/0

set security zones security-zone untrust interfaces ge-0/0/1

set security policies default-policy permit-all

commit

verify the IP address assigned on the CSRX

root@csrx1-sc> show interfaces

Physical interface: ge-0/0/1, Enabled, Physical link is Up

Interface index: 100

Link-level type: Ethernet, MTU: 1514

Current address: 02:84:71:f4:f2:8d, Hardware address: 02:84:71:f4:f2:8d

Logical interface ge-0/0/1.0 (Index 100)

Flags: Encapsulation: ENET2

Protocol inet

Destination: 10.10.10.0/24, Local: 10.10.10.2

Physical interface: ge-0/0/0, Enabled, Physical link is Up

Interface index: 200

Link-level type: Ethernet, MTU: 1514

Current address: 02:84:8b:4c:18:8d, Hardware address: 02:84:8b:4c:18:8d

Logical interface ge-0/0/0.0 (Index 200)

Flags: Encapsulation: ENET2

Protocol inet

Destination: 10.20.20.0/24, Local: 10.20.20.2

From the Left POD try to ping the left POD, ping would fail as there is no route

root@left-ubuntu-sc:/# ping 10.20.20.1

PING 10.20.20.1 (10.20.20.1) 56(84) bytes of data.

^C

--- 10.20.20.1 ping statistics ---

3 packets transmitted, 0 received, 100% packet loss, time 1999ms

root@left-ubuntu-sc:/# ip r

default via 10.47.255.254 dev eth0

10.10.10.0/24 dev eth1 proto kernel scope link src 10.10.10.1

10.32.0.0/12 dev eth0 proto kernel scope link src 10.47.255.249

Adding static route to the left and right PODs and try to ping again

root@left-ubuntu-sc:/# ip r add 10.20.20.0/24 via 10.10.10.2

root@right-ubuntu-sc:/# ip r add 10.10.10.0/24 via 10.20.20.2

root@left-ubuntu-sc:/# ping 10.20.20.1

PING 10.20.20.1 (10.20.20.1) 56(84) bytes of data.

^C

--- 10.20.20.1 ping statistics ---

4 packets transmitted, 0 received, 100% packet loss, time 2999ms

Still ping failed, as we didn’t create the service chaining which will also take care of the routing  
let’s see what happen to our packets

root@csrx1-sc# run show security flow session

Total sessions: 0

No session on the CSRX.

Login to the compute node “cent22” that host this container to dump the traffic using tshark and check the routing

To get the interface linking the containers

[root@cent22 ~]# vif -l

Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror

Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2

D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged

Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface is Monitored

Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC Learning Enabled

Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, Ig=Igmp Trap Enabled

vif0/0 OS: ens160 (Speed 10000, Duplex 1)

Type:Physical HWaddr:00:50:56:9e:bb:98 IPaddr:0.0.0.0

Vrf:0 Mcast Vrf:65535 Flags:TcL3L2VpEr QOS:-1 Ref:7

RX packets:896021 bytes:291885987 errors:0

TX packets:885150 bytes:851902554 errors:0

Drops:20

vif0/1 OS: vhost0

Type:Host HWaddr:00:50:56:9e:bb:98 IPaddr:10.85.188.17

Vrf:0 Mcast Vrf:65535 Flags:PL3DEr QOS:-1 Ref:8

RX packets:883626 bytes:851837334 errors:0

TX packets:912984 bytes:292597411 errors:0

Drops:17

vif0/2 OS: pkt0

Type:Agent HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0

Vrf:65535 Mcast Vrf:65535 Flags:L3Er QOS:-1 Ref:3

RX packets:151865 bytes:13060426 errors:0

TX packets:465234 bytes:52562395 errors:0

Drops:0

vif0/3 OS: tapeth0-89a4e2

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.47.255.252

Vrf:3 Mcast Vrf:3 Flags:PL3DEr QOS:-1 Ref:6

RX packets:10760 bytes:452800 errors:0

TX packets:14239 bytes:598366 errors:0

Drops:10744

vif0/4 OS: tapeth1-89a4e2

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.20.20.1

Vrf:5 Mcast Vrf:5 Flags:PL3DEr QOS:-1 Ref:6

RX packets:13002 bytes:867603 errors:0

TX packets:16435 bytes:1046981 errors:0

Drops:10805

vif0/5 OS: tapeth0-7d8e06

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.47.255.249

Vrf:3 Mcast Vrf:3 Flags:PL3DEr QOS:-1 Ref:6

RX packets:10933 bytes:459186 errors:0

TX packets:14536 bytes:610512 errors:0

Drops:10933

vif0/6 OS: tapeth1-7d8e06

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.10.1

Vrf:6 Mcast Vrf:6 Flags:PL3DEr QOS:-1 Ref:6

RX packets:12625 bytes:1102433 errors:0

TX packets:15651 bytes:810689 errors:0

Drops:10957

vif0/7 OS: tapeth0-844f1c

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.47.255.248

Vrf:3 Mcast Vrf:3 Flags:PL3DEr QOS:-1 Ref:6

RX packets:20996 bytes:1230688 errors:0

TX packets:27205 bytes:1142610 errors:0

Drops:21226

vif0/8 OS: tapeth1-844f1c

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.10.10.2

Vrf:6 Mcast Vrf:6 Flags:PL3DEr QOS:-1 Ref:6

RX packets:13908 bytes:742243 errors:0

TX packets:29023 bytes:1790589 errors:0

Drops:10514

vif0/9 OS: tapeth2-844f1c

Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:10.20.20.2

Vrf:5 Mcast Vrf:5 Flags:PL3DEr QOS:-1 Ref:6

RX packets:16590 bytes:1053659 errors:0

TX packets:31321 bytes:1635153 errors:0

Drops:10421

vif0/4350 OS: pkt3

Type:Stats HWaddr:00:00:00:00:00:00 IPaddr:0.0.0.0

Vrf:65535 Mcast Vrf:65535 Flags:L3L2 QOS:0 Ref:1

RX packets:0 bytes:0 errors:0

TX packets:0 bytes:0 errors:0

Drops:0

vif0/4351 OS: pkt1

Type:Stats HWaddr:00:00:00:00:00:00 IPaddr:0.0.0.0

Vrf:65535 Mcast Vrf:65535 Flags:L3L2 QOS:0 Ref:1

RX packets:8 bytes:552 errors:0

TX packets:8 bytes:552 errors:0

Drops:0

Note that Vif0/3 and Vif0/4 are bounded with the right POD and both linked to tapeth0-89a4e2 and tapeth1-89a4e2 respectively same goes for the left POD for Vif0/5 and vif0/6 while vif0/7, vif 0/8 and vif0/9 are bound with CSRX1.|  
from that you can also see the number of packets/bytes hits that interface as well the VRF which is this interface belong   
in here VRF 3 is for the default-cluster-network while VRF 6 for the left network and VRF 5 for the right network   
in this diagram you can see the interface mapping from the all prospective (container, Linux , vr-agent)

A screenshot of a cell phone

Description automatically generated

try to ping again from the left POD to the right POD and use tshark on the tap interface for the right POD for further inspection

[root@cent22 ~]# tshark -i tapeth1-89a4e2

Running as user "root" and group "root". This could be dangerous.

Capturing on 'tapeth1-89a4e2'

1 0.000000000 IETF-VRRP-VRID\_00 -> 02:89:cc:86:48:8d ARP 42 Gratuitous ARP for 10.20.20.254 (Request)

2 0.000037656 IETF-VRRP-VRID\_00 -> 02:89:cc:86:48:8d ARP 42 Gratuitous ARP for 10.20.20.253 (Request)

3 1.379993896 IETF-VRRP-VRID\_00 -> 02:89:cc:86:48:8d ARP 42 Who has 10.20.20.1? Tell 10.20.20.253

Looks like the ping isn’t reaching the right POD at all , lets see on the CSRX left network tap interface

[root@cent22 ~]# tshark -i tapeth1-844f1c

Running as user "root" and group "root". This could be dangerous.

Capturing on 'tapeth1-844f1c'

1 0.000000000 IETF-VRRP-VRID\_00 -> 02:84:71:f4:f2:8d ARP 42 Who has 0.255.255.252? Tell 0.0.0.0

2 0.201392098 10.10.10.1 -> 10.20.20.1 ICMP 98 Echo (ping) request id=0x020a, seq=410/39425, ttl=63

3 0.201549430 10.10.10.2 -> 10.10.10.1 ICMP 70 Destination unreachable (Port unreachable)

4 1.201444156 10.10.10.1 -> 10.20.20.1 ICMP 98 Echo (ping) request id=0x020a, seq=411/39681, ttl=63

5 1.201600074 10.10.10.2 -> 10.10.10.1 ICMP 70 Destination unreachable (Port unreachable)

6 1.394074095 IETF-VRRP-VRID\_00 -> 02:84:71:f4:f2:8d ARP 42 Gratuitous ARP for 10.10.10.254 (Request)

7 1.394108344 IETF-VRRP-VRID\_00 -> 02:84:71:f4:f2:8d ARP 42 Gratuitous ARP for 10.10.10.253 (Request)

8 2.201462515 10.10.10.1 -> 10.20.20.1 ICMP 98 Echo (ping) request id=0x020a, seq=412/39937, ttl=63

We can see the packet but there is nothing in the CSRX security prospective to drop this packet  
  
checking the routing table of the left network VRF by logging to the vrouter\_vrouter-agent\_1 in the compute node

[root@cent22 ~]# docker ps | grep vrouter

9a737df53abe ci-repo.englab.juniper.net:5000/contrail-**vrouter**-agent:master-latest "/entrypoint.sh /usr…" 2 weeks ago Up 47 hours **vrouter**\_**vrouter**-agent\_1

e25f1467403d ci-repo.englab.juniper.net:5000/contrail-nodemgr:master-latest "/entrypoint.sh /bin…" 2 weeks ago Up 47 hours **vrouter**\_nodemgr\_1

[root@cent22 ~]# docker exec -it vrouter\_vrouter-agent\_1 bash

(vrouter-agent)[root@cent22 /]$

**(vrouter-agent)**[root@cent22 /]$ rt --dump 6 | grep 10.20.20.

**(vrouter-agent)**[root@cent22 /]$

Note that 6 is the routing table VRF of the left network, same would goes for the right network VRF routing table there is missing route

**(vrouter-agent)**[root@cent22 /]$ rt --dump 5 | grep 10.10.10.

**(vrouter-agent)**[root@cent22 /]$

So even if all the PODs are hosted on the same compute nodes, they can’t reach each other. And if these PODs are hosted on different compute nodes then you have a bigger problem to solve   
Service chaining isn’t about adjusting the routes on the containers but mainly about exchange routes between the vrouter-agent between the compute nodes regardless of the location of the POD as well adjust that automatically if the POD moved to another compute node

Before we build service chaining lets address an important concerns for network administrator who are not fan of this kind of CLI troubleshooting, can we do the same troubleshooting using contrail controller GUI   
the answer is yes and lets do it

From the Contrail Controller module control node (<http://10.85.188.16:8143> in oursetup), select monitor > infrastructure > virtual router then select the node the that host the POD , in our case “Cent22.local”

A screenshot of a computer

Description automatically generated

as shown in the diagram from the interface tab which is equivalent to running “vif -l” command on the vrouter\_vrouter-agent-1 container and even showing more information   
notice the mapping between the instance ID and tap interface naming where the first 6 character of the instance ID are always reflected in the tap interface naming

to check the routing tables of each VRF move to the “routes” tab and select the VRF you want to see

A screenshot of a cell phone

Description automatically generated

If we select the left network ( the name is longer as it include the domain , project ) we can confirm there is not 10.20.20.0/24 prefix from the right network

We can also check the mac address learned in the left network by selecting L2 ( which is equvilant to “rt --dump 6 --family bridge” command

A screenshot of a social media post

Description automatically generated

Now lets utilize the CSRX to service chaining using contrail command GUI

creating Service chaining is 4 steps make sure to do them in this order

1. create Service template
2. creating service instance based on the service template you created before
3. creating network policy and select the service instance you created before
4. apply this network policy on network

Note: since contrail command GUI is the solution to provide a single point of management for all environments, we will use it to build service changing but you still can use the normal contrail controller GUI to build service changing

Login to contrail command GUI ( in our setup <https://10.85.188.16:9091/>) then select service > catalog > create

A screenshot of a computer

Description automatically generated

A picture containing screenshot

Description automatically generated

insret a name of services template “myweb-CSRX-CS” in here   
then chose v2 , virtual machine ( no other option available)

for service mode we will work with In-network and firewall as service type

A screenshot of a cell phone

Description automatically generated

Select interfaces management, left and right then click create

A close up of a logo

Description automatically generated

Now select deployment and click create to create the service instances

A screenshot of a cell phone

Description automatically generated

Insert a name for this service instance then select from the drop down menu the name of the template you created before   
then chose the proper network from the prospective of the CSRX being the instance (container in that case) that will do the service chaining and click on port tuples to expand it

A screenshot of a cell phone

Description automatically generated  
then for each of the three interface bound one interface of the CSRX then click create  
  
tip : the name of the virtual machine interface isn’t shown in the drop down menu instead the instance ID, you can identify that from the tap interface name as we showed before.  
In other word all you have to know is most 6 left character for any interface belong to that container as all the interface in a given instance ( VM or container) share the same first characters from the left

Before you procced make sure the status of the three interfaces are up and they are showing the correct IP address of the CSRX instance

A screenshot of a social media post

Description automatically generated

To create network policy go to overlay > network policies > create

A screenshot of a cell phone

Description automatically generated

Insert a name for your network policy then in the first rule add left network as source network and right network as destination with action pass

A screenshot of a cell phone

Description automatically generated

Select advanced option to attached the service instance you create before and click create

A screenshot of a cell phone

Description automatically generated

To attach this network policy to network click virtual network and select the left network and edit

A screenshot of a cell phone

Description automatically generated

In network policies select the network policy you just created from the drop down menu then click save   
do the same for the right network

A screenshot of a cell phone

Description automatically generated

Now lets check the effect of this service changing on routing

From the Contrail Controller module control node (<http://10.85.188.16:8143> in oursetup), select monitor > infrastructure > virtual router then select the node the that host the POD , in our case “Cent22.local” then select the “routes” tab and select the left VRF

A screenshot of a social media post

Description automatically generated

Now we can the right networks host routes has been leaked to the left network (10.20.20.1/32 , 10.20.20.2/32 in this case)

Now let’s try to ping the right pod from the left pod to see the session created on the CSRX

root@left-ubuntu-sc:/# ping 10.20.20.1

PING 10.20.20.1 (10.20.20.1) 56(84) bytes of data.

64 bytes from 10.20.20.1: icmp\_seq=1 ttl=61 time=0.863 ms

64 bytes from 10.20.20.1: icmp\_seq=2 ttl=61 time=0.290 ms

^C

--- 10.20.20.1 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1000ms

rtt min/avg/max/mdev = 0.290/0.576/0.863/0.287 ms

root@csrx1-sc# run show security flow session

Session ID: 5378, Policy name: default-policy-logical-system-00/2, Timeout: 2, Valid

In: 10.10.10.1/2 --> 10.20.20.1/534;icmp, Conn Tag: 0x0, If: ge-0/0/1.0, Pkts: 1, Bytes: 84,

Out: 10.20.20.1/534 --> 10.10.10.1/2;icmp, Conn Tag: 0x0, If: ge-0/0/0.0, Pkts: 1, Bytes: 84,

Session ID: 5379, Policy name: default-policy-logical-system-00/2, Timeout: 2, Valid

In: 10.10.10.1/3 --> 10.20.20.1/534;icmp, Conn Tag: 0x0, If: ge-0/0/1.0, Pkts: 1, Bytes: 84,

Out: 10.20.20.1/534 --> 10.10.10.1/3;icmp, Conn Tag: 0x0, If: ge-0/0/0.0, Pkts: 1, Bytes: 84,

Total sessions: 2

Now let try to create security policy on the CSRX to allow only http and https

root@csrx1-sc# show security

policies {

traceoptions {

file ayma;

flag all;

}

from-zone trust to-zone untrust {

policy only-http-s {

match {

source-address any;

destination-address any;

application [ junos-http junos-https ];

}

then {

permit;

log {

session-init;

session-close;

}

}

}

policy deny-ping {

match {

source-address any;

destination-address any;

application any;

}

then {

reject;

log {

session-init;

session-close;

}

}

}

}

default-policy {

deny-all;

}

}

zones {

security-zone trust {

interfaces {

ge-0/0/0.0;

}

}

security-zone untrust {

interfaces {

ge-0/0/1.0;

}

}

}

root@left-ubuntu-sc:/# ping 10.20.20.1

PING 10.20.20.1 (10.20.20.1) 56(84) bytes of data.

^C

--- 10.20.20.1 ping statistics ---

3 packets transmitted, 0 received, 100% packet loss, time 2000ms  
  
the ping failed as the policy on the CSRX drop it

root@csrx1-sc> show log syslog | last 20

Jun 14 23:04:01 csrx1-sc flowd-0x2[374]: RT\_FLOW: RT\_FLOW\_SESSION\_DENY: session denied 10.10.10.1/8->10.20.20.1/575 0x0 icmp 1(8) deny-ping trust untrust UNKNOWN UNKNOWN N/A(N/A) ge-0/0/1.0 No policy reject 5394 N/A N/A -1

Jun 14 23:04:02 csrx1-sc flowd-0x2[374]: RT\_FLOW: RT\_FLOW\_SESSION\_DENY: session denied 10.10.10.1/9->10.20.20.1/575 0x0 icmp 1(8) deny-ping trust untrust UNKNOWN UNKNOWN N/A(N/A) ge-0/0/1.0 No policy reject 5395 N/A N/A -1

Try to send http traffic from the left to the right POD and verify the session status on the CSRX

root@left-ubuntu-sc:/# wget 10.20.20.1

--2019-06-14 23:07:34-- http://10.20.20.1/

Connecting to 10.20.20.1:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: 11510 (11K) [text/html]

Saving to: 'index.html.4'

100%[======================================>] 11,510 --.-K/s in 0s

2019-06-14 23:07:34 (278 MB/s) - 'index.html.4' saved [11510/11510]

And in the CSRX we can see the session creation

root@csrx1-sc> show log syslog | last 20

Jun 14 23:07:31 csrx1-sc flowd-0x2[374]: csrx\_l3\_add\_new\_resolved\_unicast\_nexthop: Adding resolved unicast NH. dest: 10.20.20.1, proto v4 (peer initiated)

Jun 14 23:07:31 csrx1-sc flowd-0x2[374]: csrx\_l3\_add\_new\_resolved\_unicast\_nexthop: Sending resolve request for stale ARP entry (b). NH: 5507 dest: 10.20.20.1

Jun 14 23:07:34 csrx1-sc flowd-0x2[374]: RT\_FLOW: RT\_FLOW\_SESSION\_CREATE: session created 10.10.10.1/47190->10.20.20.1/80 0x0 junos-http 10.10.10.1/47190->10.20.20.1/80 0x0 N/A N/A N/A N/A 6 only-http-s trust untrust 5434 N/A(N/A) ge-0/0/1.0 UNKNOWN UNKNOWN UNKNOWN N/A N/A -1

Jun 14 23:07:35 csrx1-sc flowd-0x2[374]: RT\_FLOW: RT\_FLOW\_SESSION\_CLOSE: session closed TCP FIN: 10.10.10.1/47190->10.20.20.1/80 0x0 junos-http 10.10.10.1/47190->10.20.20.1/80 0x0 N/A N/A N/A N/A 6 only-http-s trust untrust 5434 14(940) 12(12452) 2 UNKNOWN UNKNOWN N/A(N/A) ge-0/0/1.0 UNKNOWN N/A N/A -1