Contrail Cloud: Lab Example

vSRX L3 service instance and vMX L3 gateway

Curtis Call
Juniper Networks
ccall@juniper.net

Version 1.01
February 18, 2016

Overview

This paper documents a basic Contrail Cloud lab topology in which end to end traffic is passed through both a vSRX service instance as well as a vMX gateway. The intention of this paper is to show the configuration required on the vSRX, the vMX, and OpenContrail to accomplish the following:

- Insert a vSRX firewall into a service chain at layer 3 with no routes required on the vSRX other than the default routes provided by OpenContrail via DHCP.
- Connect a vMX to OpenContrail via EBGP and thereby connect a remote endpoint into the OpenContrail virtual network

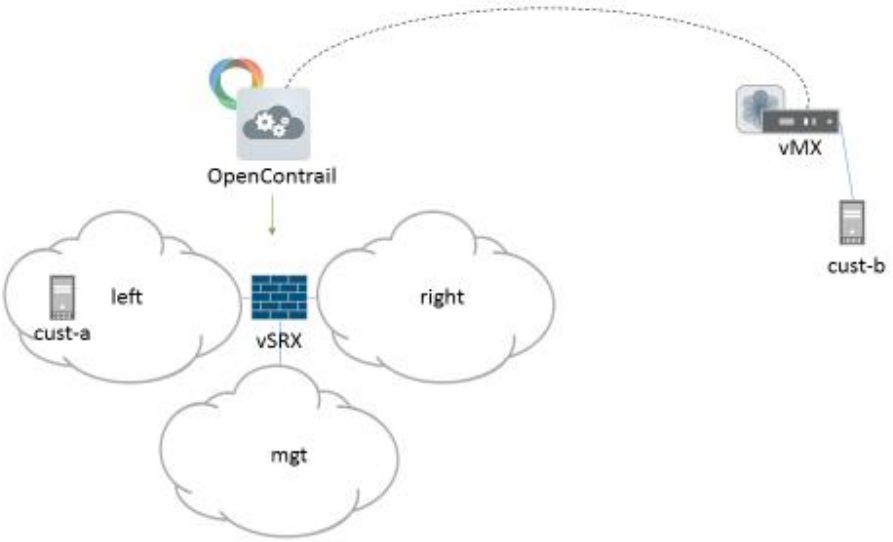
Note: This paper assumes you already have Contrail Cloud installed and are familiar with creating images and instances in OpenStack.

Topology

A basic left to right topology is used with two OpenContrail virtual networks included. The left network contains a single VM: cust-a, which is a Cirros host. A vSRX service instance is created between the left and right networks, with a policy allowing all traffic in both directions. The right network has a route target configured and the vMX is configured to have a VRF use that same route target. The vMX and the OpenContrail control node can then advertise routes back and forth, connecting the vMX's attached host, cust-b, into the OpenContrail virtual network.

A mgt network is also used, and a route target assigned, allowing the vMX to connect to the vSRX fxp0 interface.

The OpenContrail topology itself is a single-node setup, with the vMX running directly via KVM (outside of OpenStack) on a separate server.



Versions

Cirros: 0.3.4

Contrail Cloud 2.20 on Ubuntu 14.04 with kernel 3.16.0-30. (single-node topology)

vSRX: 15.1X49-D20.2

vMX: 15.1F4.15

GUI Web Address

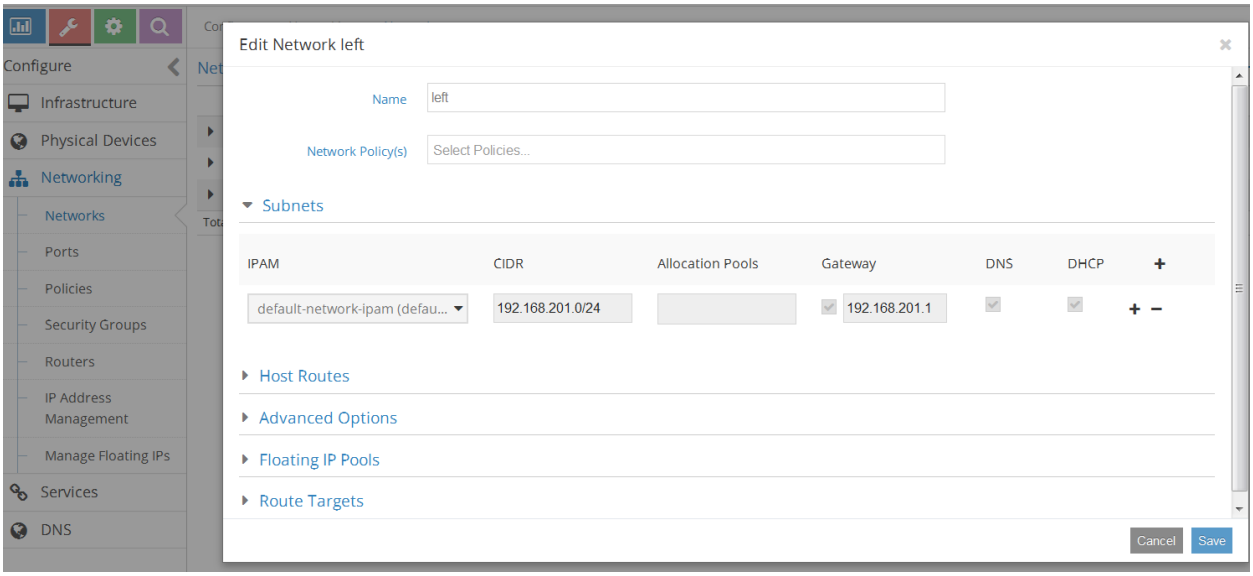
OpenStack: http://x.x.x.x/horizon

OpenContrail: http://x.x.x.x:8080

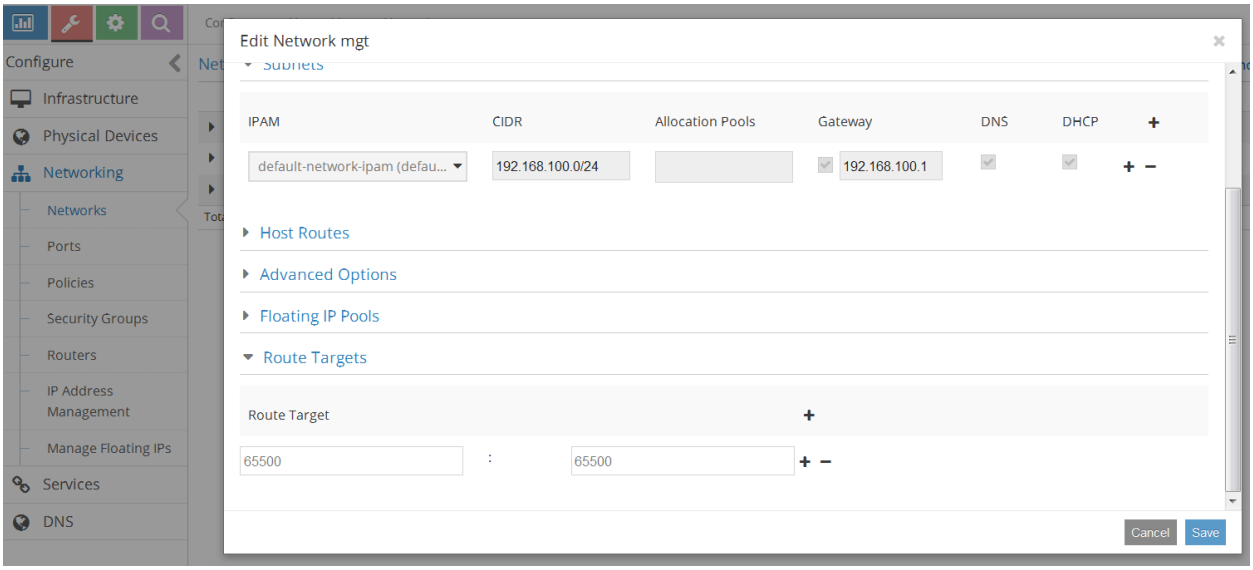
(Where x.x.x.x is the web UI IP address for your topology).

OpenContrail Virtual Network Configuration

The first thing you must create are your OpenContrail virtual networks. Create one for left, right, and mgt. At a minimum, you must add a network address (IPAM):



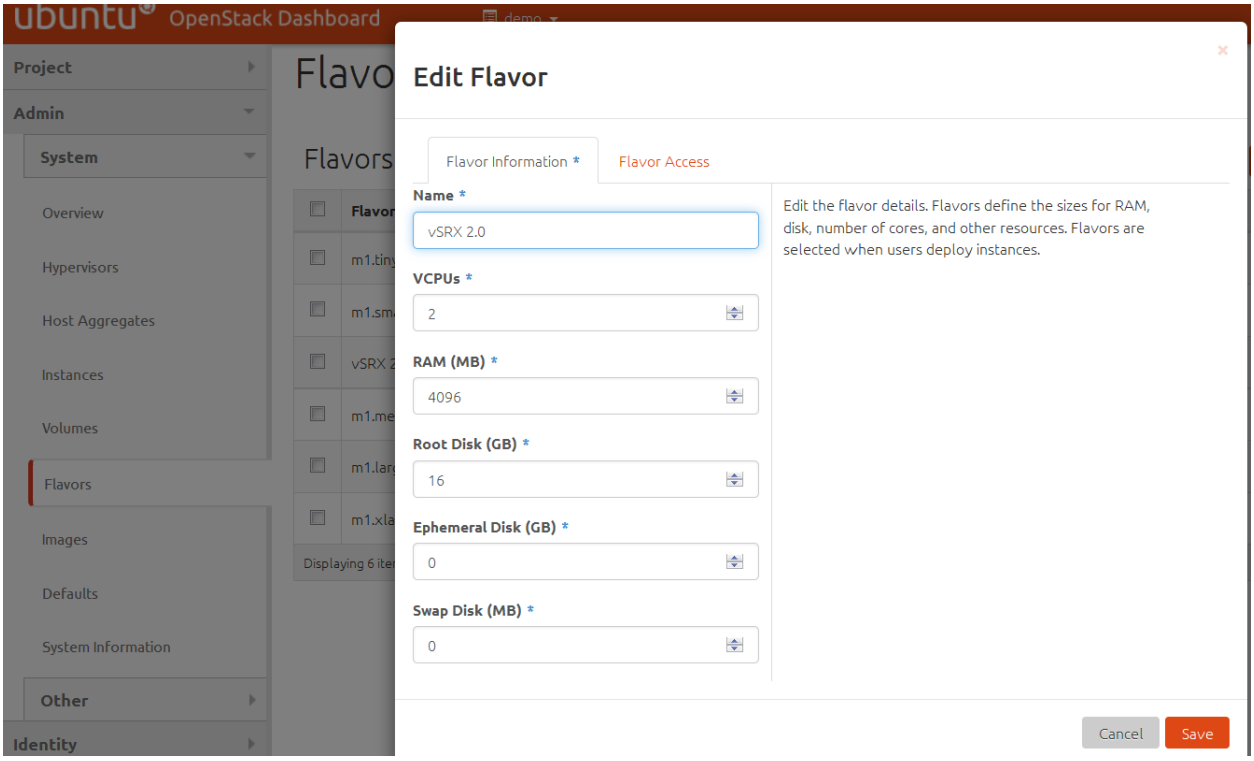
In addition, for mgt and right, specify a route target so they can be shared with the vMX:



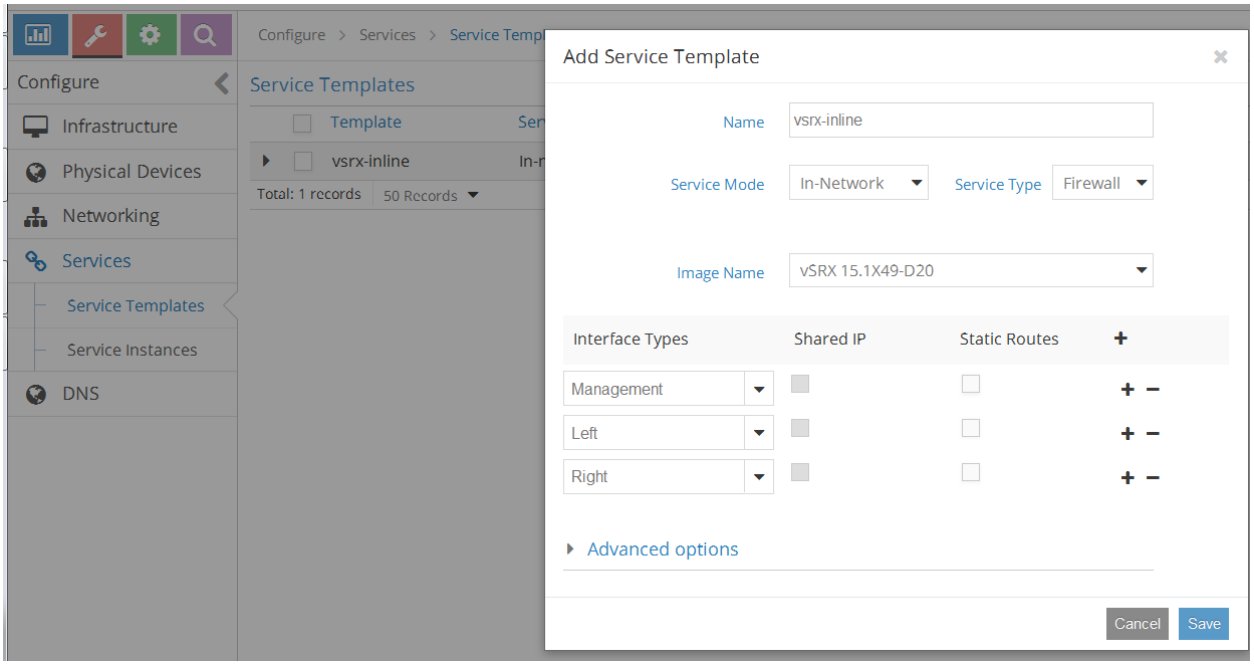
OpenStack/OpenContrail vSRX Service Instance Configuration

Before the vSRX service instance can itself be configured, its image must first be loaded into OpenStack, a flavor created for it, a service template created, and finally a service instance created.

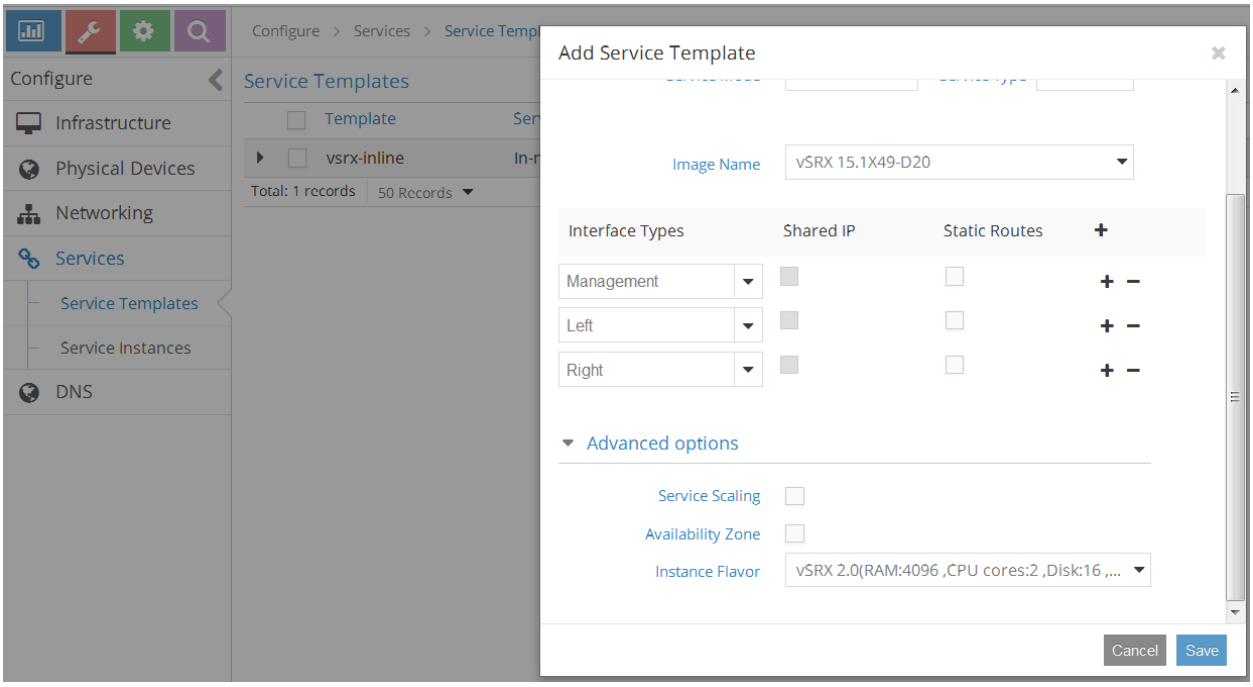
After loading the image into OpenStack, create a new flavor (in OpenStack) that specifies the correct CPU (2 VCPUS), memory (4GB), and disk requirements (16GB), as shown here (for a vSRX 15.1X49 image):



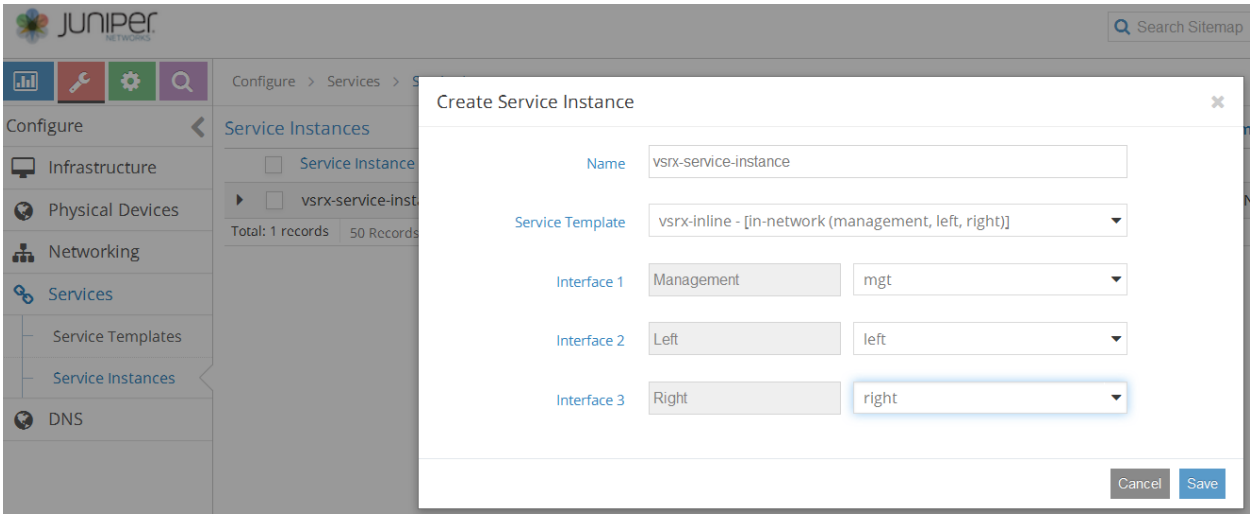
Next, create a service template within OpenContrail. This should be configured as an In-Network Firewall service instance and should use the vSRX image as well as the vSRX flavor (configured under advanced options). In addition, it should be assigned three interfaces: Management, Left, and Right:



You have to expand “Advanced options” to configure the flavor:

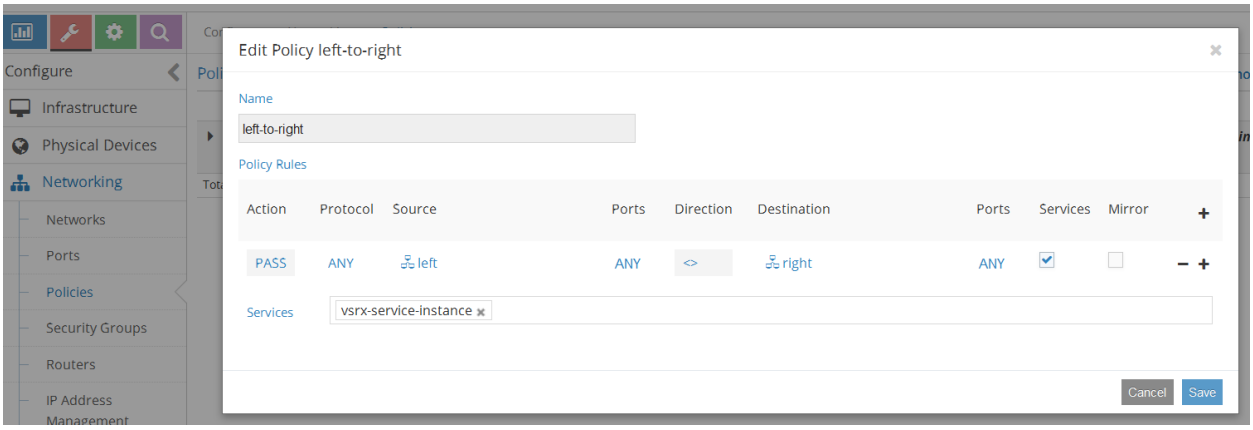


After a service template is created, you can use it to create a service instance. Select the service template and specify the mgt, left, and right networks:



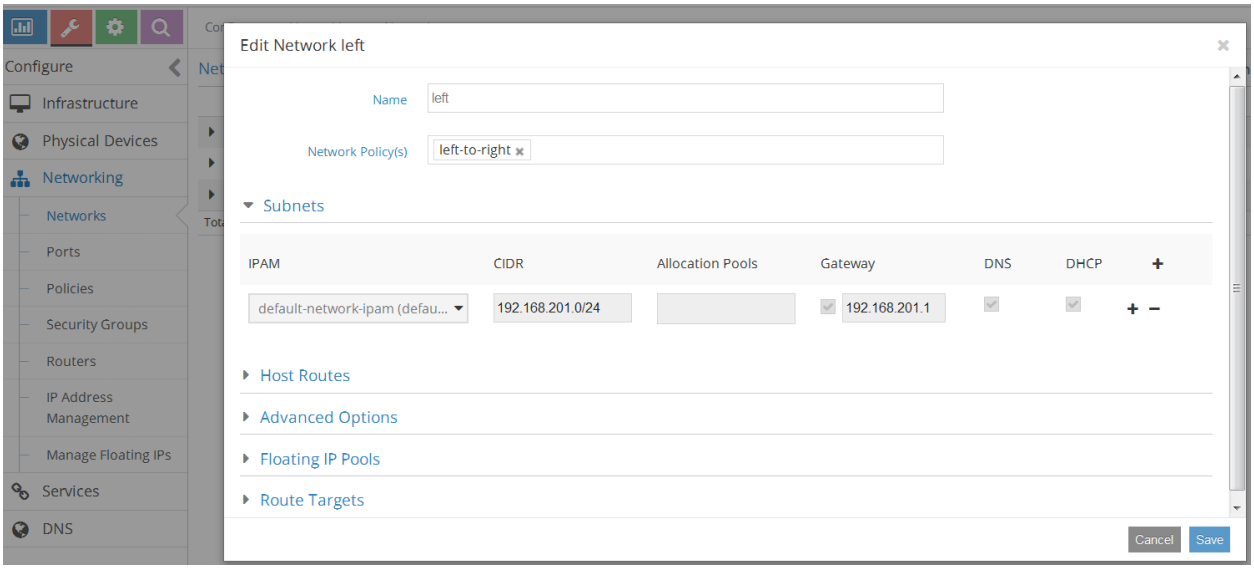
OpenContrail Policy Configuration

With the service instance created, the network policy can now be created. (These steps can be performed out of order, but then you would have to go back and edit the policy to add the service instance later.) Select left as the source and right as the destination (with direction set as bidirectional). Also select “Services” and then select your created service instance.



Attach OpenContrail Policy to the Virtual Networks

After the policy is created, it still needs to be attached to the virtual networks, which requires you to edit your previously created networks and add it. You must do this for both the left and right network:



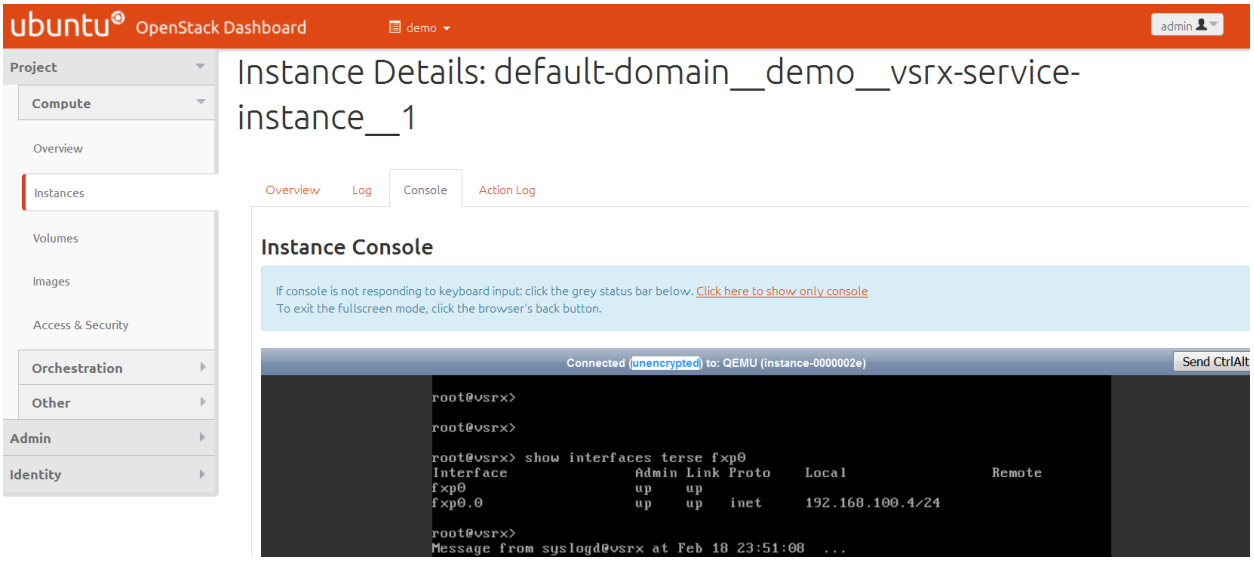
Cirros Configuration

The Cirros VM, which acts as cust-a in the left network, requires no specific configuration. It uses the standard Cirros image, which can be retrieved from here: http://docs.openstack.org/image-guide/obtain-images.html#cirros-test-images

After being loaded into OpenStack, this image can be instantiated using the m1.tiny flavor and the Cirros host will then automatically configure its attached interface via DHCP.

vSRX Configuration

Note: if you are using a fresh vSRX image, then DHCP will not be preconfigured and you will need to first connect to it from the console within OpenStack:



The vSRX is configured to receive DHCP addresses and default routes on its three interfaces and it is configured to policy route (filter-based forwarding) from the left interface to the right interface through the use of virtual-routers. Each configuration section will be explained separately.

Interfaces

```
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                dhcp-client;
                filter {
                    input left-to-right;
                }
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                dhcp-client;
                filter {
                    input right-to-left;
                }
            }
        }
    }
    fxp0 {
        unit 0 {
            family inet {
                dhcp-client;
            }
        }
    }
}
```

Each interface has dhcp-client configured (not the old “dhcp” configuration command). In addition, both the left and the right interfaces have an ingress firewall filter configured, directing traffic to the respective routing-instance.

Note: The vSRX assigns interfaces in the order they are specified in OpenContrail: first, fxp0, then ge-0/0/0, then ge-0/0/1, etc. So in this case, the mgt interface will be fxp0, the left interface will be ge-0/0/0, and the right interface will be ge-0/0/1.

Firewall

```
firewall {
    family inet {
        filter left-to-right {
            term fbf {
                then {
                    routing-instance right;
                }
            }
        }
        filter right-to-left {
            term fbf {
                then {
                    routing-instance left;
                }
            }
        }
    }
}
```

Two firewall filters are defined, each directing traffic to the opposite routing-instance.

Routing-Instances

```
routing-instances {
    left {
        instance-type virtual-router;
        interface ge-0/0/0.0;
    }
    right {
        instance-type virtual-router;
        interface ge-0/0/1.0;
    }
}
```

Separate left and right routing-instances are created, both virtual-routers, and the left and right interface assigned to the appropriate routing-instance.

Security

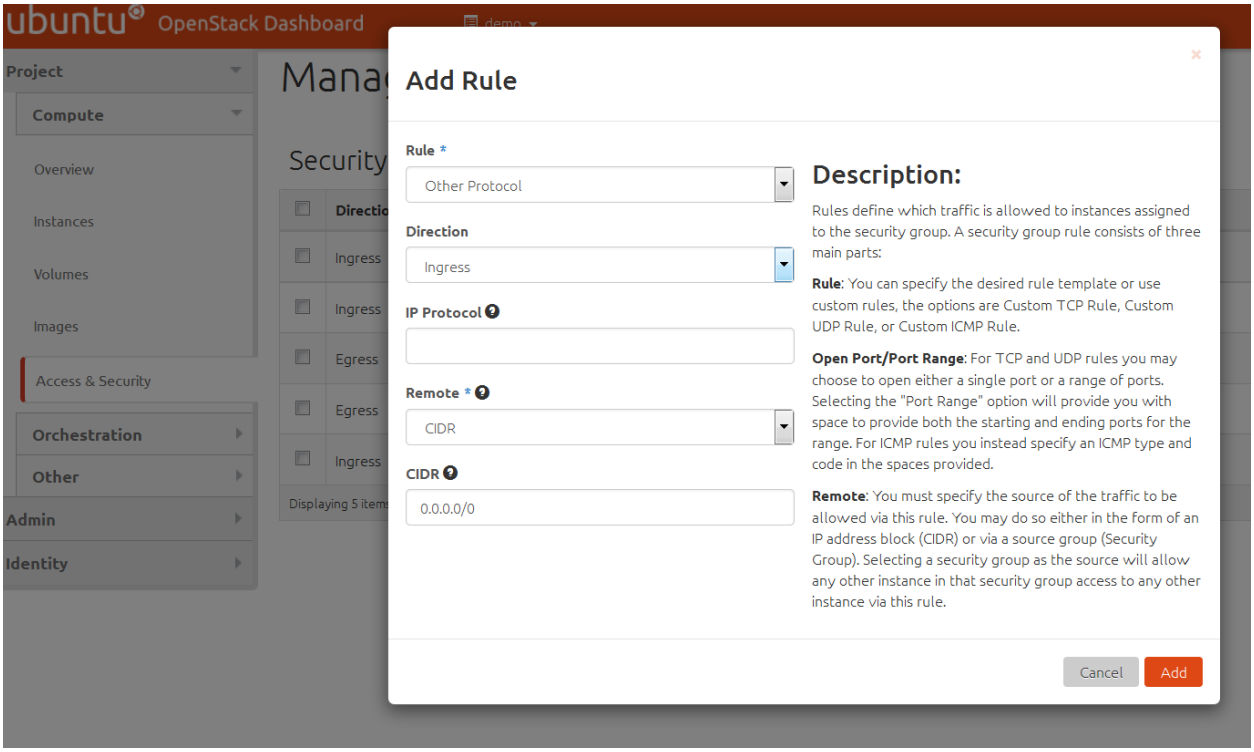
```
security {
    policies {
        from-zone left to-zone right {
            policy allow-ping {
                match {
                    source-address any;
                    destination-address any;
                    application junos-ping;
                }
                then {
                    permit;
                }
            }
        }
        from-zone right to-zone left {
            policy allow-ssh {
                match {
                    source-address any;
```

```
                    destination-address any;
                    application junos-ssh;
                }
                then {
                    permit;
                }
            }
        }
    }
    zones {
        security-zone left {
            interfaces {
                ge-0/0/0.0 {
                    host-inbound-traffic {
                        system-services {
                            dhcp;
                        }
                    }
                }
            }
        }
        security-zone right {
            interfaces {
                ge-0/0/1.0 {
                    host-inbound-traffic {
                        system-services {
                            dhcp;
                        }
                    }
                }
            }
        }
    }
}
```

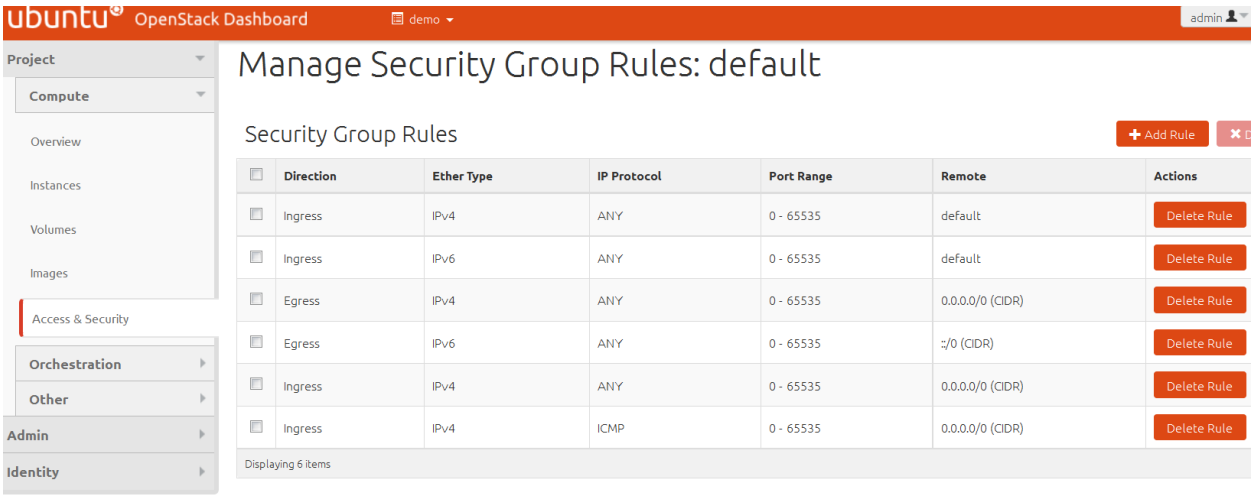
A security-zone left and security-zone right are created and the appropriate interfaces added with dhcp permitted. Security policies are created from left to right and right to left permitting the desired traffic. (In this case, ping from left to right and ssh from right to left.)

OpenStack Security Group

By default, the Cirros and vSRX VMs will be assigned to the Default Security Group, which will not allow SSH connections from cust-b. To allow this, you can add a new rule to the Default Security Group in OpenStack:

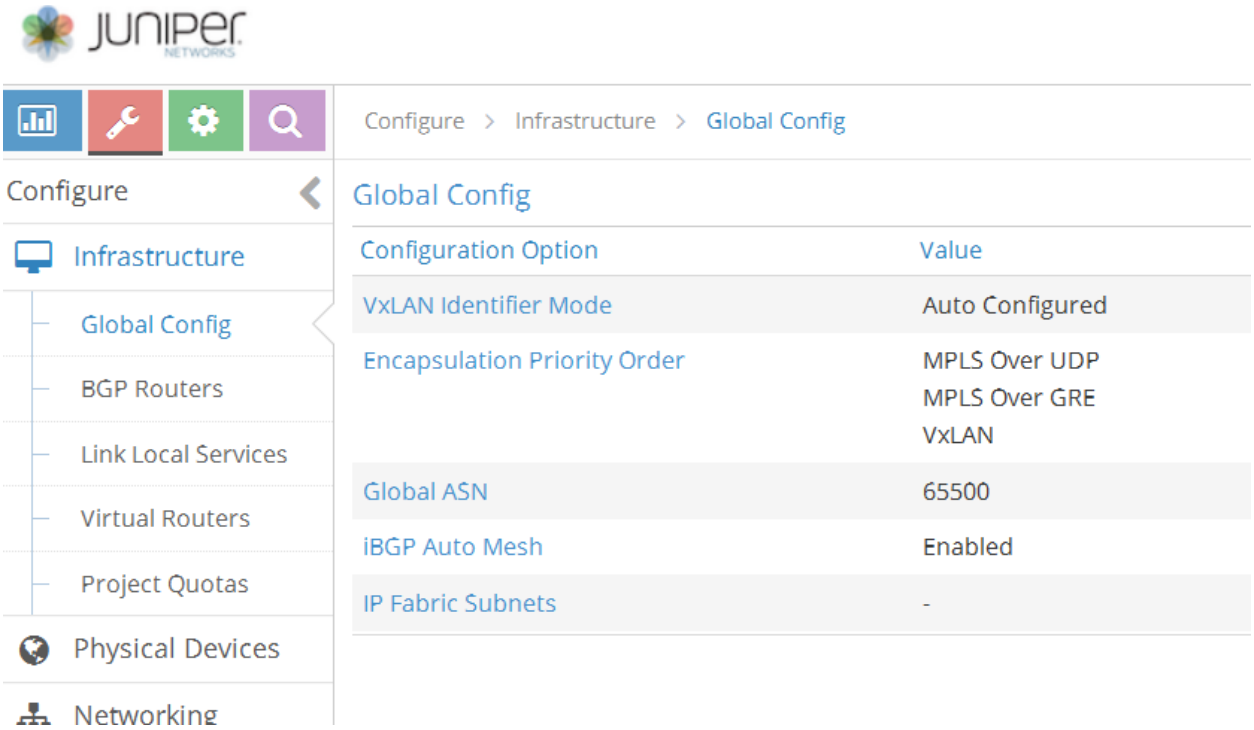


As shown above, if you select Other Protocol, leave the IP Protocol blank, and leave a 0.0.0.0/0 CIDR, then it results in allowing all inbound traffic (in production, you'd want to be more selective). The newly added rule is the second to last rule shown below, where ANY protocol is allowed from CIDR 0.0.0.0/0:

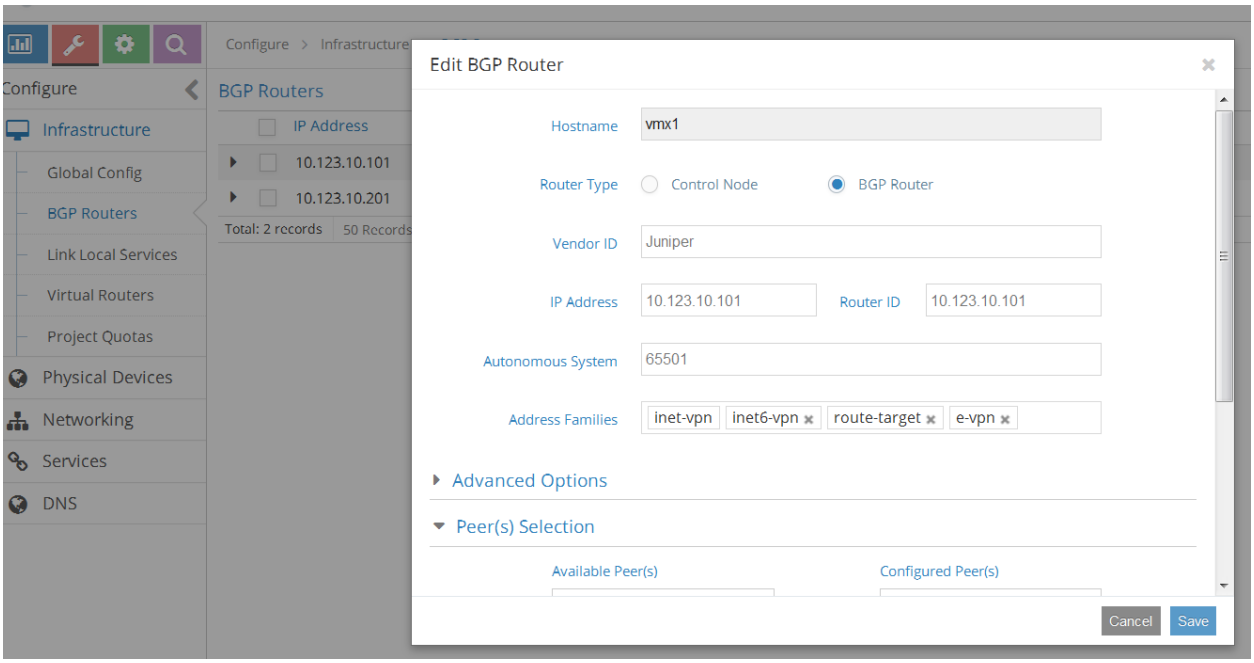


OpenContrail BGP Configuration

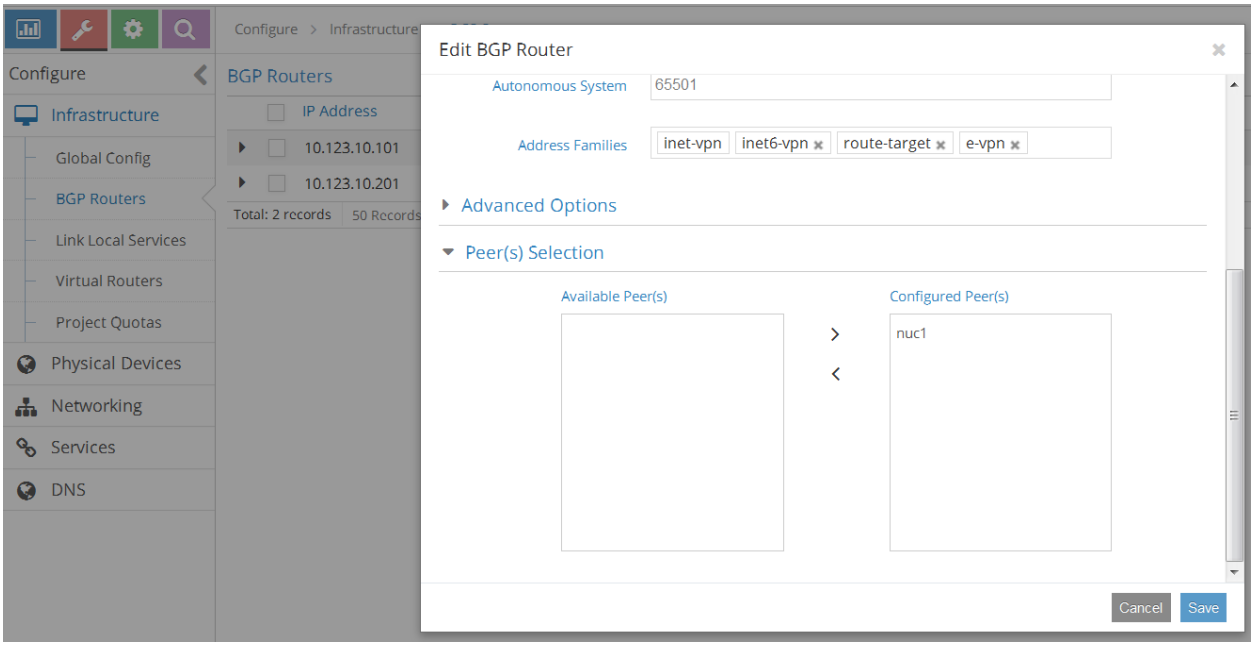
The ASN used by OpenContrail should have been configured as part of the installation process. It can be viewed here:



A BGP Router must be added for the vMX. The Vendor ID doesn't appear to be significant (however it must be filled with something), but the IP Address, Router ID, and ASN must be configured correctly for the vMX peer. Also, ensure that at least the inet-vpn family is included as an address family:



And the OpenContrail control node specified as its configured peer:



vMX Configuration

The vMX servers as a layer 3 gateway between the OpenContrail virtual networks and the vMX VRFs. In this example, a virtual-router is configured on the vMX to act as cust-b and a logical tunnel is created within the vMX to connect the VRF and virtual-router. Dynamic GRE tunnels are configured, allowing GRE tunnels to be created toward the OpenContrail compute node. A multihop EBGP session is established to OpenContrail and routing-instances created with the matching route targets to connect them to the OpenContrail virtual networks.

Each configuration section will be explained separately.

(Note: A feature license had to be loaded onto the vMX, allowing its network mode to become enhanced IP instead of the enhanced Ethernet, before this configuration was successful.)

Chassis

```
chassis {
    fpc 0 {
        pic 0 {
            /* Required for GRE dynamic tunnels */
            tunnel-services;
        }
    }
}
```

Tunnel-services must be enabled so that dynamic GRE tunnels can be used.

Interfaces

```
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 10.123.10.101/24;
            }
```

```
        }
    }
    lt-0/0/0 {
        /* PE side of local tunnel */
        unit 0 {
            encapsulation ethernet;
            peer-unit 1;
            family inet {
                address 192.168.0.1/24;
            }
        }
        /* cust-b side of local tunnel */
        unit 1 {
            encapsulation ethernet;
            peer-unit 0;
            family inet {
                address 192.168.0.2/24;
            }
        }
    }
    lo0 {
        /* Used for connectivity to mgt virtual network */
        unit 1 {
            family inet {
                address 192.168.101.1/32;
            }
        }
    }
}
```

ge-0/0/0 connects to the lab network. The lt tunnels are used for convenience, allowing the VRF to be connected to a virtual-router and thereby have cust-b on the vMX itself. The lo0.1 interface is used for the management VRF.

Routing-Options

```
routing-options {
    router-id 10.123.10.101;
    route-distinguisher-id 10.123.10.101;
    autonomous-system 65501;
    /* Required when using MPLS/GRE to create tunnels to compute nodes */
    dynamic-tunnels {
        example {
            source-address 10.123.10.101;
            gre;
            destination-networks {
                10.123.10.0/24;
            }
        }
    }
}
```

In addition to the standard BGP configuration items, dynamic-tunnels must be configured, with the source matching the BGP peer address and the destination-networks covering all of the OpenContrail compute nodes.

Protocols

```
protocols {
    bgp {
        group contrail-ebgp {
            type external;
            /* Required to connect to Contrail */
            multihop;
            local-address 10.123.10.101;
            /* Routes are exchanged as L3VPN routes */
            family inet-vpn {
                unicast;
```

```
            }
            peer-as 65500;
            neighbor 10.123.10.201;
        }
    }
}
```

A multihop EBGP peering session is connected to the OpenContrail control nodes. (It must be multihop even if directly connected.) Specify family inet-vpn unicast at a minimum. That is the address family used by OpenContrail to advertise virtual network routes

Routing-Instances

```
routing-instances {
    contrail-left {
        instance-type vrf;
        interface lt-0/0/0.0;
        /* Matches route target configured for left virtual network */
        vrf-target target:65500:1;
        vrf-table-label;
    }
    contrail-mgt {
        instance-type vrf;
        interface lo0.1;
        /* Matches route target configured for mgt virtual network */
        vrf-target target:65500:65500;
        vrf-table-label;
    }
    cust-b {
        instance-type virtual-router;
        interface lt-0/0/0.1;
        routing-options {
            static {
                route 0.0.0.0/0 next-hop 192.168.0.1;
            }
        }
    }
}
```

The two VRFs must have the same route target configured as the OpenContrail virtual network they are intended to connect to. In addition, a virtual-router instance is created to create a simulated cust-b, with a static route pointing back over the tunnel to the VRF that is connected to OpenContrail.

Routing Tables

Cirros

```
$ ip route
default via 192.168.201.1 dev eth0
192.168.201.0/24 dev eth0  src 192.168.201.4
$
```

vSRX

```
root@vsrx> show route

inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Access-internal/12] 06:24:51
                    > to 192.168.100.1 via fxp0.0
192.168.100.0/24   *[Direct/0] 06:24:52
                    > via fxp0.0
192.168.100.4/32   *[Local/0] 06:24:52
```

```
                      Local via fxp0.0

left.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Access-internal/12] 04:55:26
                    > to 192.168.201.1 via ge-0/0/0.0
192.168.201.0/24   *[Direct/0] 04:55:27
                    > via ge-0/0/0.0
192.168.201.5/32   *[Local/0] 04:55:27
                      Local via ge-0/0/0.0

right.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Access-internal/12] 04:55:26
                    > to 192.168.202.1 via ge-0/0/1.0
192.168.202.0/24   *[Direct/0] 04:55:27
                    > via ge-0/0/1.0
192.168.202.4/32   *[Local/0] 04:55:27
                      Local via ge-0/0/1.0

root@vsrx>
```

vMX

```
root@vmx> show route | no-more

inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.123.10.0/24     *[Direct/0] 01:02:19
                    > via ge-0/0/0.0
10.123.10.101/32   *[Local/0] 01:02:20
                      Local via ge-0/0/0.0

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.123.10.0/24     *[Tunnel/300] 01:05:40
                      Tunnel
10.123.10.201/32   *[Tunnel/300] 01:02:15
                    > via gr-0/0/0.32769

contrail-left.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.0.0/24     *[Direct/0] 01:02:22
                    > via lt-0/0/0.0
192.168.0.1/32     *[Local/0] 01:05:12
                      Local via lt-0/0/0.0
192.168.201.4/32   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                      AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
192.168.201.5/32   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                      AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
192.168.202.4/32   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                      AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44

contrail-mgt.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.100.4/32   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                      AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 40
192.168.101.1/32   *[Direct/0] 01:05:38
                    > via lo0.1

cust-b.inet.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
```

```
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 01:02:22
                    > to 192.168.0.1 via lt-0/0/0.1
192.168.0.0/24     *[Direct/0] 01:02:22
                    > via lt-0/0/0.1
192.168.0.2/32     *[Local/0] 01:05:12
                       Local via lt-0/0/0.1

mpls.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

16                 *[VPN/0] 01:05:38
                    > via lsi.0 (contrail-left), Pop
17                 *[VPN/0] 01:05:38
                    > via lsi.1 (contrail-mgt), Pop

bgp.l3vpn.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.123.10.101:7:192.168.0.0/24
                   *[Direct/0] 01:02:22
                    > via lt-0/0/0.0
10.123.10.101:7:192.168.0.1/32
                   *[Local/0] 01:05:12
                       Local via lt-0/0/0.0
10.123.10.101:8:192.168.101.1/32
                   *[Direct/0] 01:05:38
                    > via lo0.1
10.123.10.201:1:192.168.201.4/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 38
10.123.10.201:1:192.168.201.5/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 42
10.123.10.201:2:192.168.0.0/24
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 42
10.123.10.201:2:192.168.202.4/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 42
10.123.10.201:3:192.168.202.4/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.123.10.201:4:192.168.201.4/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.123.10.201:4:192.168.201.5/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 44
10.123.10.201:5:192.168.100.4/32
                   *[BGP/170] 01:02:15, localpref 100, from 10.123.10.201
                       AS path: 65500 ?, validation-state: unverified
                    > via gr-0/0/0.32769, Push 40

root@vmx>
```

Traffic Demonstration

Ping from cust-a to cust-b is successful:

```
$ ping 192.168.0.2 -c 5
```

```
PING 192.168.0.2 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=57 time=27.914 ms
64 bytes from 192.168.0.2: seq=1 ttl=57 time=3.850 ms
64 bytes from 192.168.0.2: seq=2 ttl=57 time=186.865 ms
64 bytes from 192.168.0.2: seq=3 ttl=57 time=3.160 ms
64 bytes from 192.168.0.2: seq=4 ttl=57 time=1.354 ms

--- 192.168.0.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.354/44.628/186.865 ms
$
```

Ping from cust-b to cust-a fails:

```
root@vmx> ping 192.168.201.4 routing-instance cust-b count 5
PING 192.168.201.4 (192.168.201.4): 56 data bytes

--- 192.168.201.4 ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss

root@vmx>
```

SSH from cust-b to cust-a is successful:

```
root@vmx> ssh cirros@192.168.201.4 routing-instance cust-b
cirros@192.168.201.4's password:
$ uname
Linux
$
```