NetReady: Network Readiness

Release draft (bc144a7)

OPNFV

July 11, 2016

1 Introduction **3**
1.1 Scope . . . 3
1.2 Problem Description . . . 3
1.3 Goals . . . 3

2 Use cases **5**
2.1 L3VPN Use Cases . . . 5
2.2 Port Abstraction . . . 12
2.3 Programmable Provisioning of Provider networks . . . 14
2.4 Georedundancy . . . 14

3 Summary **19**

Bibliography **21**

Index **23**

Project NetReady, https://wiki.opnfv.org/display/netready/NetReady

Editors TBD

Authors Bin Hu (AT&T), Gergely Csatari (Nokia), Georg Kunz (Ericsson) and others

Abstract OPNFV provides an infrastructure with different SDN controller options to realize NFV functionality on the platform it builds. As OPNFV uses OpenStack as VIM, we need to analyze the capabilities this component offers us. The networking functionality is provided by a single component called Neutron, which hides the controller under it, let it be Neutron itself or any supported SDN controller. As NFV wasn't taken into consideration at the time when Neutron was designed we are already facing several bottlenecks and architectural shortcomings while implementing our use cases.

The NetReady project aims at evolving OpenStack networking step-by-step to find the most efficient way to fulfill the requirements of the identified NFV use cases, taking into account the NFV mindset and the capabilities of SDN controllers.

History

Date	Description
22.03.2016	Project creation
19.04.2016	Initial version of the deliverable uploaded to Gerrit

Definition of terms

Different standards developing organizations and communities use different terminology related to Network Function Virtualization, Cloud Computing, and Software Defined Networking. This list defines the terminology in the contexts of this document.

API Application Programming Interface.

Cloud Computing A model that enables access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Edge Computing Edge computing pushes applications, data and computing power (services) away from centralized points to the logical extremes of a network.

Instance Refers in OpenStack terminology to a running VM, or a VM in a known state such as suspended, that can be used like a hardware server.

NFV Network Function Virtualization.

NFVI Network Function Virtualization Infrastructure. Totality of all hardware and software components which build up the environment in which VNFs are deployed.

SDN Software-Defined Networking. Emerging architecture that decouples the network control and forwarding functions, enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

Server Computer that provides explicit services to the client software running on that system, often managing a variety of computer operations. In OpenStack terminology, a server is a VM instance.

VIM Virtualized Infrastructure Manager. Functional block that is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain, e.g. NFVI Point of Presence (NFVI-PoP).

VM Virtual Machine. Virtualized computation environment that behaves like a physical computer/server by modeling the computing architecture of a real or hypothetical computer.

Virtual network Virtual network routes information among the network interfaces of VM instances and physical network interfaces, providing the necessary connectivity.

VNF Virtualized Network Function. Implementation of an Network Function that can be deployed on a Network Function Virtualization Infrastructure (NFVI).

WAN Wide Area Network.

CHAPTER

ONE

INTRODUCTION

This document represents and describes the results of the OPNFV NetReady (Network Readiness) project. Specifically, the document comprises a selection of NFV-related networking use cases and their networking requirements, a corresponding gap analysis of the aforementioned requirements with respect to the current OpenStack networking architecture and a description of potential solutions and improvements.

1.1 Scope

NetReady is a project within the OPNFV initiative. Its focus is on NFV (Network Function Virtualization) related networking use cases and their requirements on the underlying NFVI (Network Function Virtualization Infrastructure).

The NetReady project addresses the OpenStack networking architecture, specifically OpenStack Neutron, from a NFV perspective. Its goal is to identify gaps in the current OpenStack networking architecture with respect to NFV requirements and to propose and evaluate improvements and potential complementary solutions.

1.2 Problem Description

Traditionally, OpenStack networking, represented typically by the OpenStack Neutron project, targets virtualized data center networking. This comprises primarily establishing and managing layer 2 network connectivity among VMs (Virtual Machines). Over the past releases of OpenStack, Neutron has grown to provide an extensive feature set, covering L2 and L3 networking features such as virtual layer 2 networks, virtual routers, BGP VPNs and others.

It is often stated that NFV imposes additional requirements on the networking architecture and feature set of the underlying NFVI beyond those of data center networking. For instance, the NFVI needs to establish and manage connectivity beyond the data center to the WAN (Wide Area Network). Moreover, NFV networking use cases often abstract from L2 connectivity and instead focus on L3-only connectivity. Hence, the NFVI networking architecture needs to be flexible enough to be able to meet the requirements of NFV-related use cases in addition to traditional data center networking.

It is an ongoing debate how well the current OpenStack networking architecture can meet the additional requirements of NFV networking. Hence, a thorough analysis of NFV networking requirements and their relation to the OpenStack networking architecture is needed.

1.3 Goals

The goals of the NetReady project and correspondingly this document are the following:

- This document comprises a collection of relevant NFV networking use cases and clearly describes their requirements on the NFVI. These requirements are stated independently of a particular implementation, for instance

OpenStack Neutron. Instead, requirements are formulated in terms of APIs (Application Programming Interfaces) and data models needed to realize a given NFV use case.

- The list of use cases is not considered to be all-encompassing but it represents a carefully selected set of use cases that are considered to be relevant at the time of writing. More use cases may be added over time. The authors are very open to suggestions, reviews, clarifications, corrections and feedback in general.
- This document contains a thorough analysis of the gaps in the current OpenStack networking architecture with respect to the requirements imposed by the selected NFV use cases. To this end, we analyze existing functionality in OpenStack networking.
- This document will in future revisions describe the proposed improvements and complementary solutions needed to enable OpenStack to fulfill the identified NFV requirements.

CHAPTER TWO

USE CASES

The following sections address networking use cases that have been identified to be relevant in the scope of NFV and NetReady.

2.1 L3VPN Use Cases

Service Providers' virtualized network infrastructure may consist of one or more SDN Controllers from different vendors. Those SDN Controllers may be managed within one cloud or multiple clouds. Jointly, those VIMs (e.g. OpenStack instances) and SDN Controllers work together in an interoperable framework to create L3 services in Service Providers' virtualized network infrastructure.

Three use cases of creating L3VPN service by multiple SDN Controllers are described as follows.

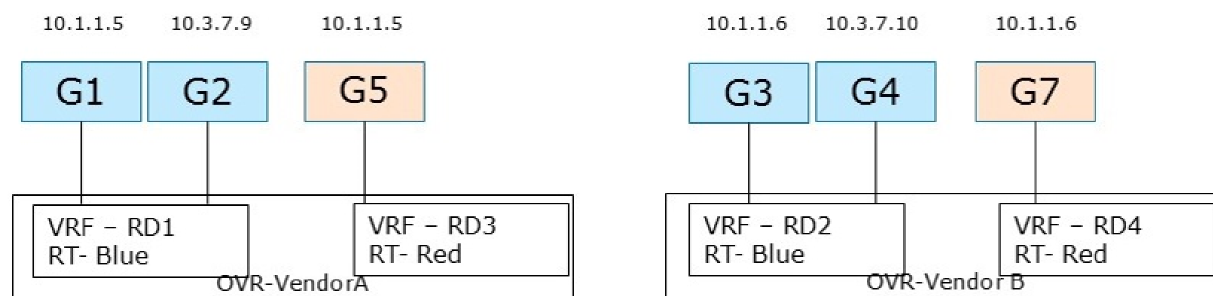
2.1.1 Any-to-Any Base Case

Description

There are 2 hosts (compute nodes). SDN Controller A and vRouter A are provided by Vendor A, and run on host A. SDN Controller B and vRouter B are provided by Vendor B, and run on host B.

There are 2 tenants. Tenant 1 creates L3VPN Blue with 2 subnets: 10.1.1.0/24 and 10.3.7.0/24. Tenant 2 creates L3VPN Red with 1 subnet, overlapping address space: 10.1.1.0/24.

The network topology is shown in Fig. 2.1.1:



In L3VPN Blue, VMs G1 (10.1.1.5) and G2 (10.3.7.9) are spawned on host A, and attached to 2 subnets (10.1.1.0/24 and 10.3.7.0/24) and assigned IP addresses respectively. VMs G3 (10.1.1.6) and G4 (10.3.7.10) are spawned on host B, and attached to 2 subnets (10.1.1.0/24 and 10.3.7.0/24) and assigned IP addresses respectively.

In L3VPN Red, VM G5 (10.1.1.5) is spawned on host A, and attached to subnet 10.1.1.0/24. VM G6 (10.1.1.6) is spawned on host B, and attached to the same subnet 10.1.1.0/24.

VRF Lets us do:

1. Overlapping Addresses
2. Segregation of Traffic

Derrived Requirements

Northbound API / Workflow

Exemplary workflow is described as follows:

1. Create Network
2. Create Network VRF Policy Resource `Any-to-Any`

   2.1. This sets up that when this tenant is put on a HOST that:

      2.1.1. There will be a RD assigned per VRF

      2.1.2. There will be a RT used for the common any-to-any communication

3. Create Subnet
4. Create Port (subnet, network vrf policy resource). This causes controller to:

   4.1. Create vrf in vRouter's FIB, or Update vrf if already exists

   4.2. Install an entry for Guest's HOST-Route in FIBs of Vrouters serving this tenant Virtual Network

   4.3. Announce Guest HOST-Route to WAN-GW via MP-BGP

Data model objects

- TBD

Orchestration

- TBD

Dependencies on compute services

- TBD

Current implementation

Support for creating and managing L3VPNs is available in OpenStack Neutron by means of the BGPVPN project *[BGPVPN]*. In order to create the L3VPN network configuration described above using the API BGPVPN API, the following workflow is needed:

1. Create a Neutron network "blue"

```
neutron net-create blue
```

2. Create the first Neutron subnet of the network "blue"

   ```
   neutron subnet-create <blue network UUID> 10.1.1.0/24
   ```

3. Create the second Neutron subnet of the network "blue"

   ```
   neutron subnet-create <blue network UUID> 10.3.7.0/24
   ```

4. Create a L3VPN by means of the BGPVPN API

   ```
   neutron bgpvpn-create --route-targets 64512:1 --tenant-id <tenant-id>
   --name blue
   ```

5. Associate the L3VPN with the previously created network

   ```
   neutron bgpvpn-net-assoc-create blue --network <network-UUID>
   ```

   This command associates the given Neutron network with the L3VPN. The semantic of this operation is that all subnets bound to the network are getting interconnected through the BGP VPN and hence VMs located in either subnet can communicate with each other.

Gaps in the current solution

TBD

Conclusion

TBD

2.1.2 ECMP Load Splitting Case (Anycast)

Description

There are 2 hosts (compute nodes). SDN Controller A and vRouter A are provided by Vendor A, and run on host A. SDN Controller B and vRouter B are provided by Vendor B, and run on host B.

There is 1 tenant. Tenant 1 creates L3VPN Blue with subnet 10.1.1.0/24.

The network topology is shown in Fig. 2.1.2:

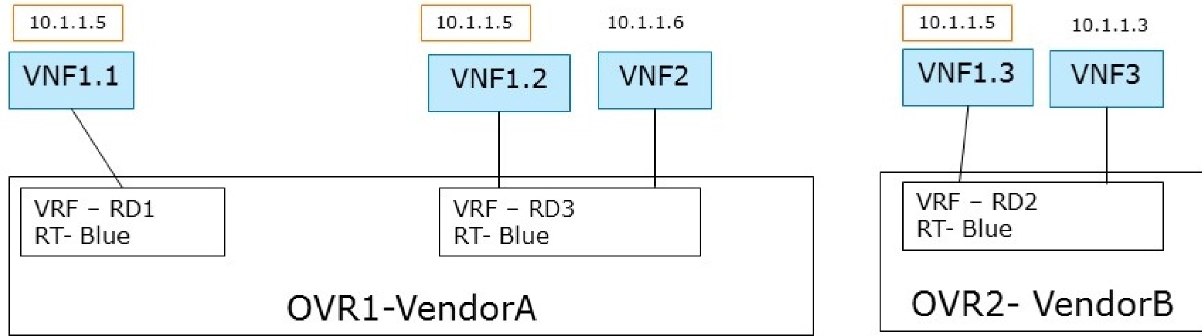
In L3VPN Blue, VNF1.1 and VNF1.2 are spawned on host A, attached to subnet 10.1.1.0/24 and assigned the same IP address 10.1.1.5. VNF1.3 is spawned on host B, attached to subnet 10.1.1.0/24 and assigned the same IP addresses 10.1.1.5. VNF 2 and VNF 3 are spawned on host A and B respectively, attached to subnet 10.1.1.0/24, and assigned different IP addresses 10.1.1.6 and 10.1.1.3 respectively.

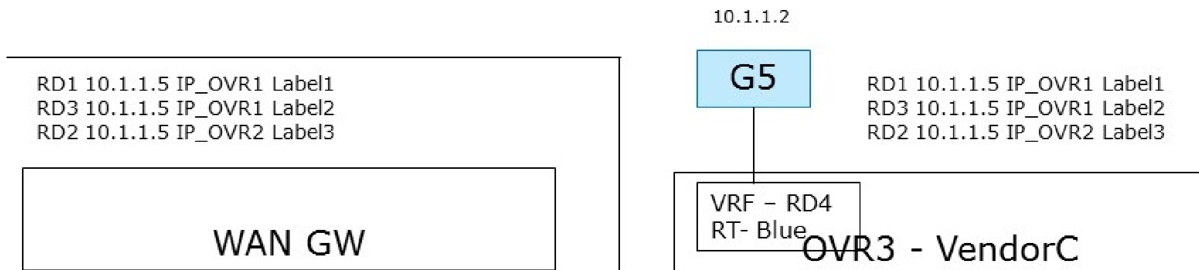
Here, the Network VRF Policy Resource is `ECMP/AnyCast`. Traffic to **Anycast 10.1.1.5** can be load split from either WAN GW or another VM like G5.

Derrived Requirements

Northbound API / Workflow

- TBD





Data model objects

- TBD

Orchestration

- TBD

Dependencies on compute services

- TBD

Current implementation

Support for creating and managing L3VPNs is in general available in OpenStack Neutron by means of the BGPVPN project *[BGPVPN]*. However, the BGPVPN API does not yet support ECMP, but this feature is on the project roadmap. Hence, it is currently not possible to configure the networking use case as described above.

Gaps in the current solution

Given the use case description and the currently available implementation in OpenStack provided by BGPVPN project, we identify the following gaps:

- [L3VPN-ECMP-GAP1] ECMP is current not yet supported by the BGPVPN API. The Development of this feature is on the roadmap of the project, however. TODO: add timeline and planned API

- [L3VPN-ECMP-GAP2] It is not possible to assign the same IP to multiple Neutron ports within the same Neutron subnet. This is due to the fundamental requirement of avoiding IP collisions within the L2 domain which is a Neutron network. A potential workaround is to create two subnets with the same IP ranges and associate both with the same BGP VPN.

Conclusion

TBD

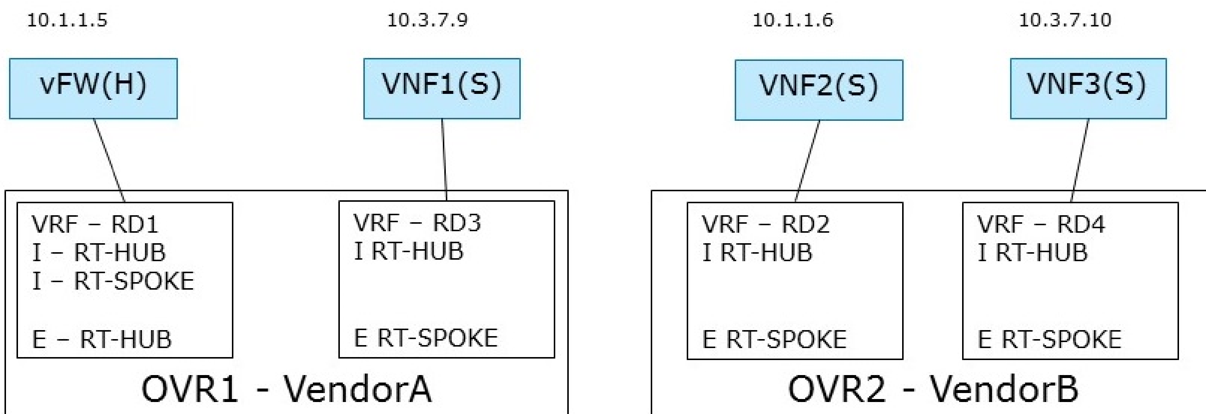
2.1.3 Hub and Spoke Case

Description

There are 2 hosts (compute nodes). SDN Controller A and vRouter A are provided by Vendor A, and run on host A. SDN Controller B and vRouter B are provided by Vendor B, and run on host B.

There is 1 tenant. Tenant 1 creates L3VPN Blue with 2 subnets: 10.1.1.0/24 and 10.3.7.0/24.

The network topology is shown in Fig. 2.1.3:



In L3VPN Blue, vFW(H) is acting the role of `hub` (a virtual firewall). The other 3 VNFsVMs are `spoke`. vFW(H) and VNF1(S) are spawned on host A, and VNF2(S) and VNF3(S) are spawned on host B. vFW(H) (10.1.1.5) and VNF2(S) (10.1.1.6) are attached to subnet 10.1.1.0/24. VNF1(S) (10.3.7.9) and VNF3(S) (10.3.7.10) are attached to subnet 10.3.7.0/24.

Derrived Requirements

Northbound API / Workflow

Exemplary vFW(H) Hub VRF is as follows:

- RD1 10.1.1.5 IP_OVR1 Label1
- RD1 0/0 IP_OVR1 Label1
- Label 1 Local IF (10.1.1.5)
- RD3 10.3.7.9 IP_OVR1 Label2

- RD2 10.1.1.6 IP_OVR2 Label3
- RD4 10.3.7.10 IP_OVR2 Label3

Exemplary VNF1(S) Spoke VRF is as follows:

- RD1 0/0 IP_OVR1 Label1
- RD3 10.3.7.9 IP_OVR1 Label2

Exemplary workflow is described as follows:

1. Create Network
2. Create VRF Policy Resource

   2.1. Hub and Spoke
3. Create Subnet
4. Create Port

   4.1. Subnet

   4.2. VRF Policy Resource, [H | S]

Data model objects

- TBD

Orchestration

- TBD

Dependencies on compute services

- TBD

Current implementation

Different APIs have been developed to support creating a network topology and directing network traffic through specific network elements in specific order, for example, *[BGPVPN]* and *[NETWORKING-SFC]*. We analyzed those APIs regarding the Hub-and-Spoke use case.

BGPVPN Support for creating and managing L3VPNs is in general available in OpenStack Neutron by means of the BGPVPN API *[BGPVPN]*. However, the *[BGPVPN]* API does not support creating the Hub-and-Spoke topology as outlined above, i.e. setting up specific VRFs of vFW(H) and other VNFs(S) within one L3VPN to direct the traffic from vFW(H) to VNFs(S).

The *[BGPVPN]* API currently supports the concepts of network- and router-associations. An association in principle maps to a VRF that interconnects either subnets of a Neutron network (network association) or the networks connected by a router (router association). It does not yet allow for creating VRFs per VM port (port associations) as illustrated in Hub-and-Spoke use case. The functionality of port association is needed, however, to create separate VRFs per VM in order to implement the Hub-and-Spoke use case. Furthermore, the functionality of setting up next-hop routing table, labels, I-RT and E-RT etc in VRF is also required to enable traffic direction from Hub to Spokes.

It may be argued that given the current network- and router-association mechanisms, the following workflow establishes a network topology which aims to achieve the desired traffic flow from Hub to Spokes. The basic idea is to model separate VRFs per VM by creating a dedicated Neutron network with two subnets for each VRF in the Hub-and-Spoke topology.

1. Create Neutron network "hub"

   ```
   neutron net-create hub
   ```

2. Create a separate Neutron network for every "spoke"

   ```
   neutron net-create spoke-i
   ```

3. For every network (hub and spokes), create two subnets

   ```
   neutron subnet-create <hub/spoke-i network UUID> 10.1.1.0/24   neutron
   subnet-create <hub/spoke-i network UUID> 10.3.7.0/24
   ```

4. Create a BGPVPN object (VRF) for the hub network with the corresponding import and export targets

   ```
   neutron bgpvpn-create --name hub-vrf --import-targets <RT-hub
   RT-spoke> --export-targets <RT-hub>
   ```

5. Create a BGPVPN object (VRF) for every spoke network with the corresponding import and export targets

   ```
   neutron bgpvpn-create --name spoke-i-vrf --import-targets <RT-hub>
   --export-targets <RT-spoke>
   ```

6. Associate the hub network with the hub VRF

   ```
   bgpvpn-net-assoc-create hub --network <hub network-UUID>
   ```

7. Associate each spoke network with the corresponding spoke VRF

   ```
   bgpvpn-net-assoc-create spoke-i --network <spoke-i network-UUID>
   ```

After step 7, VMs can be booted on the corresponding networks.

The resulting network topology tries to resemble our target topology as shown in Fig. 2.1.3, and achieve the desired traffic direction from Hub to Spoke. However, it deviates significantly from the essence of the Hub-and-Spoke use case as described above in terms of desired network topology, i.e. one L3VPN with multiple VRFs associated with vFW(H) and other VNFs(S) separately. And this method of using the current network- and router-association mechanism is not scalable when there are large number of Spokes, and in case of scale-in and scale-out of Hub and Spokes.

The gap analysis in the next section describes the technical reasons for this.

Network SFC Support of Service Function Chaining is in general available in OpenStack Neutron through the Neutron API for Service Insertion and Chaining project *[NETWORKING-SFC]*. However, the *[NETWORKING-SFC]* API is focused on creating service chaining through NSH at L2, although it intends to be agnostic of backend implementation. It is unclear whether or not the service chain from vFW(H) to VNFs(S) can be created in the way of L3VPN-based VRF policy approach using *[NETWORKING-SFC]* API.

Hence, it is currently not possible to configure the networking use case as described above.

Gaps in Current Solution

Given the use case description and the currently available implementation in OpenStack provided by *[BGPVPN]* project and *[NETWORKING-SFC]* project, we identify the following gaps:

- [L3VPN-HS-GAP1] The *[BGPVPN]* project lacks port-associations

The workflow described above intents to mimic port associations by means of separate Neutron networks. Hence, the resulting workflow is overly complicated and not intuitive by requiring to create additional Neutron entities (networks) which are not present in the target topology. This method is also not scalable.

Within the *[BGPVPN]* project, design work on port-association has started. The timeline for this feature is however not defined yet. As a result, creating a clean Hub-and-Spoke topology is current not yet supported by the *[BGPVPN]* API.

- [L3VPN-HS-GAP2] Creating a clean hub-and-spoke topology is current not yet supported by the *[NETWORKING-SFC]* API.

2.2 Port Abstraction

2.2.1 Description

This use case aims at binding multiple networks or network services to a single vNIC (port) of a given VM. There are several specific application scenarios for this use case:

- Shared Service Functions: A service function connects to multiple networks of a tenant by means of a single vNIC.

  Typically, a vNIC is bound to a single network. Hence, in order to directly connect a service function to multiple networks at the same time, multiple vNICs are needed - each vNIC binding the service function to a separate network. For service functions requiring connectivity to a large number of networks, this approach does not scale as the number of vNICs per VM is limited and additional vNICs occupy additional resources on the hypervisor.

  A more scalable approach is to bind multiple networks to a single vNIC and let the service function, which is now shared among multiple networks, handle the separation of traffic itself.

- Multiple network services: A service function connects to multiple different network types such as a L2 network, a L3(-VPN) network, a SFC domain or services such as DHCP, IPAM, firewall/security, etc.

In order to achieve a flexible binding of multiple services to vNICs, a logical separation between a vNIC (instance port) - that is, the entity that is used by the compute service as hand-off point between the network and the VM - and a service interface - that is, the interface a service binds to - is needed.

Furthermore, binding network services to service interfaces instead of to the vNIC directly enables a more dynamic management of the network connectivity of network functions as there is no need to add or remove vNICs.

2.2.2 Requirements

Data model

The envisioned data model of the port abstraction model is as follows:

- `instance-port`

  An instance port object represents a vNIC which is bindable to an OpenStack instance by the compute service (Nova).

  Attributes: Since an instance-port is a layer 2 device, its attributes include the MAC address, MTU and others (TBD).

- `interface`

An interface object is a logical abstraction of an instance-port. It allows to build hierachies of interfaces by means of a reference to a parent interface. Each interface represents a subset of the packets traversing a given port or parent interface after applying a layer 2 segmentation mechanism specific to the interface type.

Attributes: The attributes are specific to the type of interface.

Examples: trunk interface, VLAN interface, VxLAN interface, MPLS interface

- `service`

  A service object represents a specific networking service.

  Attributes: The attributes of the service objects are service specific and valid for given service instance.

  Examples: L2, L3VPN, SFC

- `service-port`

  A service port object binds an interface to a service.

  Attributes: The attributes of a service-port are specific for the bound service.

  Examples: port services (IPAM, DHCP, security), L2 interfaces, L3VPN interfaces, SFC interfaces.

Northbound API

The API for manipulating the data model is as follows:

- `instance-port-{create,delete} <name>`

  Creates or deletes an instance port object that represents a vNIC in a VM.

- `interface-{create,delete} <name> [interface type specific parameters]`

  Creates or deletes an interface object.

- `service-{create,delete} <name> [service specific parameters]`

  Create a specific service object, for instance a L3VPN, a SFC domain, or a L2 network.

- `service-port-{create,delete} <service-id> <interface-id> [service specific parameters]`

  Creates a service port object, thereby binding an interface to a given service.

Orchestration

None.

Dependencies on other resources

The compute service needs to be enabled to consume instance ports instead of classic Neutron ports.

2.2.3 Implementation Proposal

TBD

2.3 Programmable Provisioning of Provider networks

2.3.1 Description

In NFV environment the VNFM (consumer of OpenStack IaaS API) have no administrative rights, however in the telco domain provider networks are used in some cases. When a provider network is ceated administrative rights are needed what in the case of a VNFM without administrative rights needs manual work. It shall be possible to configure provider networks without administrative rights. It should be possible to assign the capability to create provider networks to any roles.

2.3.2 Derrived Requirements

- Authorize the possibility of provider network creation based on policy
- There should be a new entry in `policy.json` which controls the provider network creation
- Default policy of this new entry should be `rule:admin_or_owner`.
- This policy should be respected by neutron API

Northbound API / Workflow

- No changes in the API

Data model objects

- No changes in the data model

2.3.3 Current implementation

Only admin users can manage provider networks *[OS-NETWORKING-GUIDE-ML2]*.

2.3.4 Potential implementation

- Policy engine shall be able to handle a new provider network creation and modification related policy
- When a provider network is created or modified neutron should check the authority with the policy engine instead of requesting administrative rights

2.4 Georedundancy

Georedundancy refers to a configuration which ensures the service continuity of the VNF-s even if a whole datacenter fails.

It is possible that the VNF application layer provides additional redundancy with VNF pooling on top of the georedundancy functionality described here.

It is possible that either the VNFC-s of a single VNF are spread across several datacenters (this case is covered by the OPNFV multisite project *[MULTISITE]* or different, redundant VNF-s are started in different datacenters.

When the different VNF-s are started in different datacenters the redundancy can be achieved by redundant VNF-s in a hot (spare VNF is running its configuration and internal state is synchronised to the active VNF), warm (spare VNF is running, its configuration is synchronised to the active VNF) or cold (spare VNF is not running, active VNF-s configuration is stored in a persistent, central store and configured to the spare VNF during its activation) standby state in a different datacenter from where the active VNF-s are running. The synchronisation and data transfer can be handled by the application or by the infrastructure.

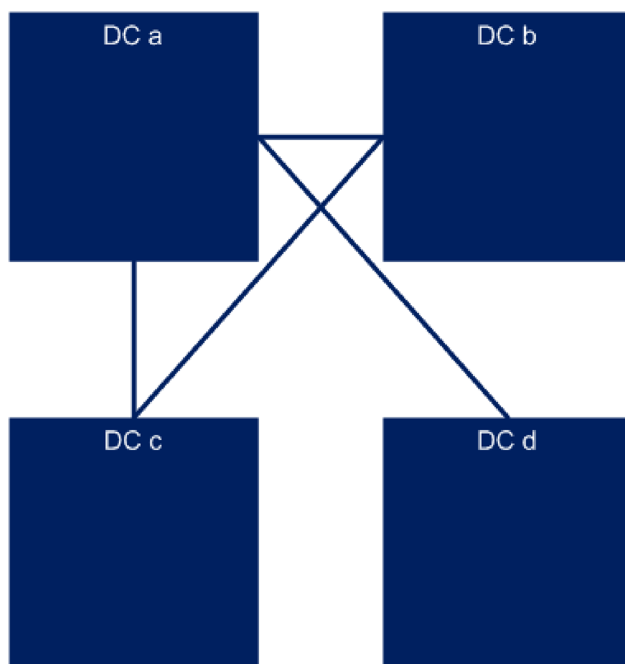
In all of these georedundancy setups there is a need for a network connection between the datacenter running the active VNF and the datacenter running the spare VNF.

In case of a distributed cloud it is possible that the georedundant cloud of an application is not predefined or changed and the change requires configuration in the underlay networks when the network operator uses network isolation. Isolation of the traffic between the datacenters might be needed due to the multitenant usage of NFVI/VIM or due to the IP pool management of the network operator.

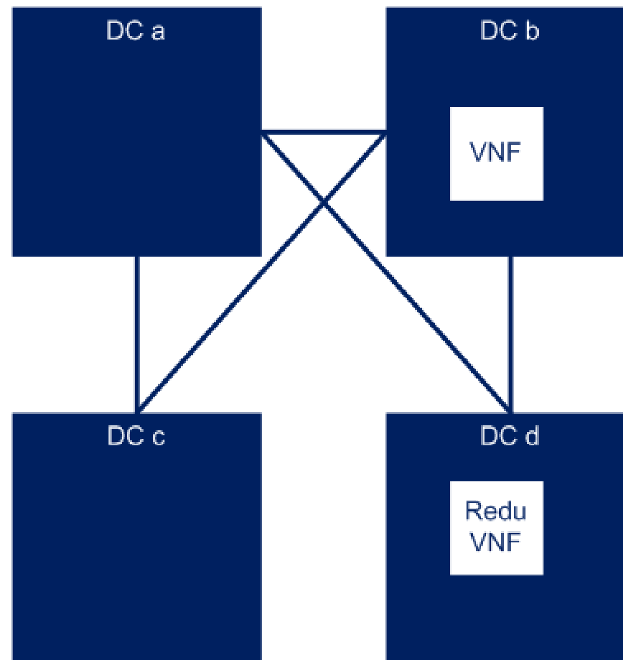
This set of georedundancy use cases is about enabling the possiblity to select a datacenter as backup datacenter and build the connectivity between the NFVI-s in the different datacenters in a programmable way.

The focus of these uses cases is on the functionality of OpenStack it is not considered how the provisioning of physical resources is handled by the SDN controllers to interconnect the two datacenters.

As an example the following picture (Fig. 2.4) shows a multicell cloud setup where the underlay network is not fully mashed.



Each datacenter (DC) is a separate OpenStack cell, region or instance. Let’s assume that a new VNF is started in DC b with a Redundant VNF in DC d. In this case a direct underlay network connection is needed between DC b and DC d. The configuration of this connection should be programable in both DC b and DC d. The result of the deployment is shown in the following figure (Fig. 2.4):



2.4.1 Connection between different OpenStack cells

Description

There should be an API to manage the infrastructure-s networks between two OpenStack cells. (Note: In the Mitaka release of OpenStack cells v1 are considered as experimental, while cells v2 functionality is under implementation). Cells are considered to be problematic from maintainability perspective as the sub-cells are using only the internal message bus and there is no API (and CLI) to do maintenance actions in case of a network connectivity problem between the main cell and the sub cells.

The functionality behind the API depends on the underlying network providers (SDN controllers) and the networking setup. (For example OpenDaylight has an API to add new BGP neighbour.)

OpenStack Neutron should provide an abstracted API for this functionality what calls the underlying SDN controllers API.

Derrived Requirements

- Possibility to define a remote and a local endpoint
- As in case of cells the nova-api service is shared it should be possible to identify the cell in the API calls

Northbound API / Workflow

- An infrastructure network management API is needed
- API call to define the remote and local infrastructure endpoints
- When the endpoints are created neutron is configured to use the new network.

Dependencies on compute services

None.

Data model objects

- local and remote endpoint objects (Most probably IP addresses with some

additional properties).

Current implementation

Current OpenStack implementation provides no way to set up the underlay network connection. OpenStack Tricicle project *[TRICICLE]* has plans to build up inter datacenter L2 and L3 networks.

Gaps in the current solution

An infrastructure management API is missing from Neutron where the local and remote endpoints of the underlay network could be configured.

2.4.2 Connection between different OpenStack regions or cloud instances

Description

There should be an API to manage the infrastructure-s networks between two OpenStack regions or instances.

The functionality behind the API depends on the underlying network providers (SDN controllers) and the networking setup. (For example OpenDaylight has an API to add new BGP neighbour.)

OpenStack Neutron should provide an abstracted API for this functionality what calls the underlying SDN controllers API.

Derrived Requirements

- Possibility to define a remote and a local endpoint
- As in case of cells the nova-api service is shared it should be possible to identify the cell in the API calls

Northbound API / Workflow

- An infrastructure network management API is needed
- API call to define the remote and local infrastructure endpoints
- When the endpoints are created neutron is configured to use the new network.

Data model objects

- local and remote endpoint objects (Most probably IP addresses with some

additional properties).

Current implementation

Current OpenStack implementation provides no way to set up the underlay network connection. OpenStack Tricicle project *[TRICICLE]* has plans to build up inter datacenter L2 and L3 networks.

Gaps in the current solution

An infrastructure management API is missing from Neutron where the local and remote endpoints of the underlay network could be configured.

2.4.3 Conclusion

An API is needed what provides possibility to set up the local and remote endpoints for the underlay network. This API present in the SDN solutions, but OpenStack does not provides and abstracted API for this functionality to hide the differences of the SDN solutions.

CHAPTER THREE

SUMMARY

The use cases and identified gaps in current OpenStack networking described in this document are formulated with the aim to start a discussion about the possibility to close these gaps. The contents of the current document are the selected use cases and their derived requirements and identified gaps for OPNFV C release.

OPNFV NetReady is open to take any further use cases under analyzis in later OPNFV releases. The project already have use cases in the following topics in the project backlog:

- API for programmable virtual network elements
- Networking requirements for P2P links from the cloud to places outside
- Networking requirements for VXLAN EVPN and eVPN DCI
- High available networking solution
- NAT-less routing
- Relevant use cases of *[NFV001]*
- Networking requirements for Edge Computing use cases
- Networking requirements for TE P2P links in the cloud
- Networking requirements for ECMP to VMs
- Networking requirements for LACP
- Neutron IETF Service Function Chaining API (see: *[TACKER-FOR-NEUTRON]*)

The actual version of the document can be reached at *[SELF]*.

[BGPVPN] http://docs.openstack.org/developer/networking-bgpvpn/

[NETWORKING-SFC] https://wiki.openstack.org/wiki/Neutron/ServiceInsertionAndChaining

[MULTISITE] https://wiki.opnfv.org/display/multisite/Multisite

[TRICICLE] https://wiki.openstack.org/wiki/Tricircle#Requirements

[OS-NETWORKING-GUIDE-ML2] http://docs.openstack.org/mitaka/networking-guide/config-ml2-plug-in.html

[SELF] http://artifacts.opnfv.org/netready/docs/requirements/index.html

[NFV001] http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[TACKER-FOR-NEUTRON] https://review.openstack.org/#/c/308453/

INDEX

A

API, **2**

C

Cloud Computing, **2**

E

Edge Computing, **2**

I

Instance, **2**

N

NFV, **2**
NFVI, **2**

S

SDN, **2**
Server, **2**

V

VIM, **2**
Virtual network, **2**
VM, **2**
VNF, **2**

W

WAN, **2**