# OPNFV User Guide

***Release brahmaputra.3.0 (605ebda)***

**OPNFV**

April 27, 2016

CONTENTS

CHAPTER

ONE

# ABSTRACT

OPNFV is a collaborative project aimed at providing a variety of virtualisation deployments intended to host applications serving the networking and carrier industry. This document provides guidance and instructions for using platform features designed to support these applications, made available in the Brahmaputra release of OPNFV.

This document is not intended to replace or replicate documentation from other open source projects such as OpenStack or OpenDaylight, rather highlight the features and capabilities delivered through the OPNFV project.

CHAPTER

TWO

# OVERVIEW

OPNFV provides a variety of virtual infrastructure deployments designed to host virtualised network functions (VNFs). This guide intends to help users of the platform leverage the features and capabilities delivered by the OPNFV project.

OPNFV Continuous Integration builds, deploys and tests combinations of virtual infrastructure components in what are defined as scenarios. A scenario may include components such as OpenStack, OpenDaylight, OVS, KVM etc. where each scenario will include different source components or configurations. Scenarios are designed to enable specific features and capabilities in the platform that can be leveraged by the OPNFV user community.

## 2.1 OPNFV Features

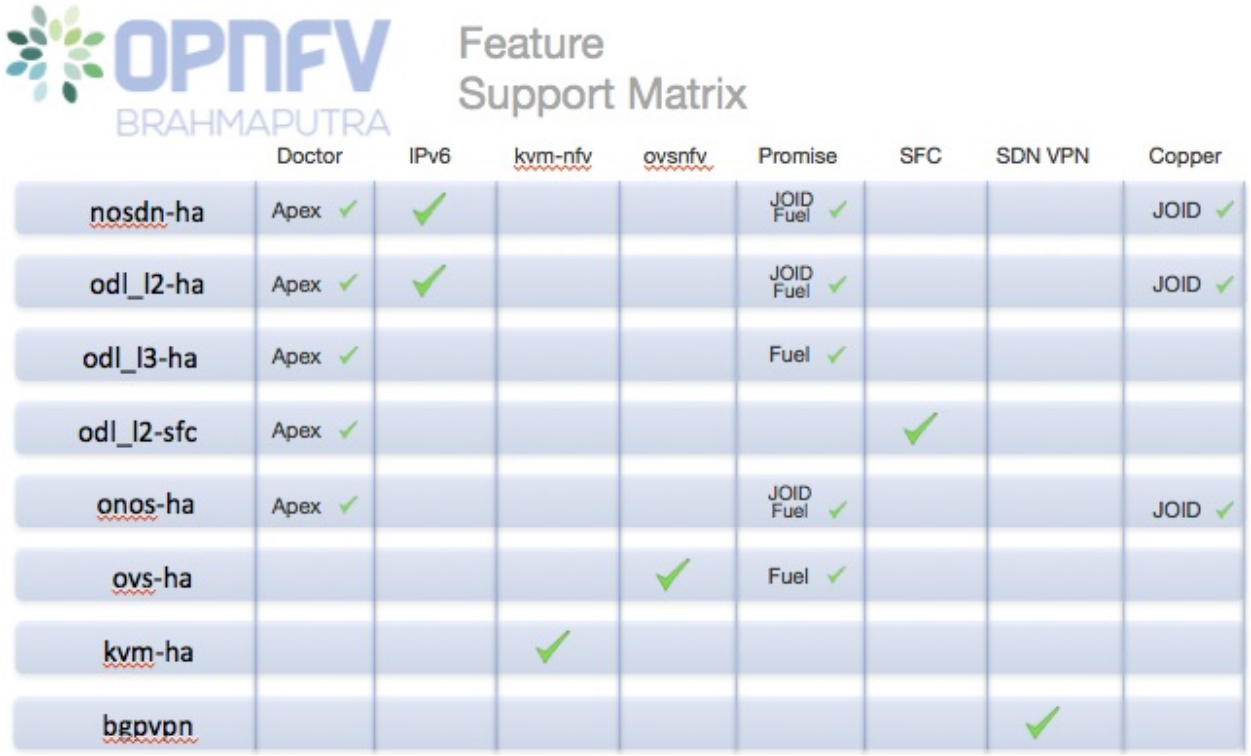
Each OPNFV scenario provides unique features and capabilities, it is important to ensure you have a scenario deployed on your infrastructure that provides the right capabilities for your needs before working through the user guide.

This user guide outlines how to work with key components and features in the platform, each feature description section will indicate the scenarios that provide the components and configurations required to use it.

Each scenario provides a set of platform capabilities and features that it supports. It is possible to identify which features are provided by reviewing the scenario name, however not all features and capabilities are discernible from the name itself.

### 2.1.1 Brahmaputra feature support matrix

The following table provides an overview of the available scenarios and supported features in the Brahmaputra release of OPNFV.




The table above provides an overview of which scenarios will support certain feature capabilities. The table does not indicate if the feature or scenario has limitations. Refer to the Configuration Guide for details on the state of each scenario and further information.

Feature development in the Brahmaputra release often consisted of the development of specific requirements and the further integration and validation of those requirements. This results in some features only being supported on the platform when a specific scenario, providing the capabilities necessary to run the feature, is deployed.

### 2.1.2 Scenario Naming

In OPNFV, scenarios are identified by short scenario names. These names follow a scheme that identifies the key components and behaviours of the scenario, the rules for scenario naming are as follows:

os-[controller]-[feature]-[mode]-[option]

For example: *os-nosdn-kvm-noha* provides an OpenStack based deployment using neutron including the OPNFV enhanced KVM hypervisor.

The [feature] tag in the scenario name describes the main feature provided by the scenario. This scenario may also provide support for features, such as advanced fault management, which are not apparent in the scenario name. The following section describes the features available in each scenario.

For details on which scenarios are best for you and how to install and configure them on your infrastructure the OPNFV Configuration Guide provides a valuable reference.

The user guide will describe how to enable and utilise features and use cases implemented and tested on deployed OPNFV scenarios. For details of the use cases and tests that have been run you should check the validation procedures section of the OPNFV Configuration Guide. This will provide information about the specific use cases that have been validated and are working on your deployment.

## 2.2 General usage guidelines

The user guide for OPNFV features and capabilities provide step by step instructions for using features that have been configured according to the installation and configuration instructions.

This guide is structured in a manner that will provide usage instructions for each feature in its own section. Start by identifying the feature capability you would like to leverage, then read through the relevant user guide section to understand how to work with the feature. The combination of platform features, if available in a given scenario and not otherwise indicated, should operate according to the documentation. Dependencies between features will be highlighted in the user guide text.

You may wish to use the platform in a manner that the development teams have not foreseen, or exercise capabilities not fully validated on the platform. If you experience issues leveraging the platform for the uses you have envisioned, the OPNFV user mailing list provides a mechanism to establish a dialog with the community to help you overcome any issues identified.

It may be that you have identified a bug in the system, or that you are trying to execute a use case that has not yet been implemented. In either case it is important for OPNFV to learn about it as we are in essence a development project looking to ensure the required capabilities for our users are available.

CHAPTER

THREE

# USING COMMON PLATFORM COMPONENTS

This section outlines basic usage principals and methods for some of the commonly deployed components of supported OPNFV scenario's in Brahmaputra. The subsections provide an outline of how these components are commonly used and how to address them in an OPNFV deployment. The components derive from autonomous upstream communities and where possible this guide will provide direction to the relevant documentation made available by those communities to better help you navigate the OPNFV deployment.

## 3.1 Common VIM components

### 3.1.1 Brahmaputra OpenStack User Guide

OpenStack is a cloud operating system developed and released by the OpenStack project. OpenStack is used in OPNFV for controlling pools of compute, storage, and networking resources in a Pharos compliant infrastructure.

OpenStack is used in Brahmaputra to manage tenants (known in OpenStack as projects), users, services, images, flavours, and quotas across the Pharos infrastructure. The OpenStack interface provides the primary interface for an operational Brahmaputra deployment and it is from the "horizon console" that an OPNFV user will perform the majority of administrative and operational activities on the deployment.

#### OpenStack references

The OpenStack user guide provides details and descriptions of how to configure and interact with the OpenStack deployment. This guide can be used by lab engineers and operators to tune the OpenStack deployment to your liking.

Once you have configured OpenStack to your purposes, or the Brahmaputra deployment meets your needs as deployed, an operator, or administrator, will find the best guidance for working with OpenStack in the OpenStack administration guide.

#### Connecting to the OpenStack instance

Once familiar with the basic of working with OpenStack you will want to connect to the OpenStack instance via the Horizon Console. The Horizon console provide a Web based GUI that will allow you operate the deployment. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

http://{Controller-VIP}:80/index.html> username: admin password: admin

Other methods of interacting with and configuring OpenStack,, like the REST API and CLI are also available in the Brahmaputra deployment, see the OpenStack administration guide for more information on using those interfaces.

## 3.2 Common SDN components

### 3.2.1 OpenDaylight User Guide

OpenDaylight is an SDN controller platform developed and released by the OpenDaylight project. The OpenDaylight controller is installed and configured in OPNFV as the networking component of a variety of OPNFV NVFi scenarios using the neutron ODL device driver as an integration point toward OpenStack.

OpenDaylight runs within a JVM and is installed in OPNFV within a container and integrated with OpenStack. The OpenDaylight instance can be configured through the OpenStack Horizon interface, or accessed directly from the OPNFV Jumphost. The Brahmaputra release of OPNFV integrates the latest Beryllium release.

#### OpenDaylight references

For an overview of the OpenDaylight controller a good reference is the Getting Started Guide. For more detailed information about using the platform the OpenDaylight User Guide provides a good feature by feature reference.

It is important when working on your Brahmaputra deployment to be aware of the configured state of the OpenDaylight controller in the scenario you have deployed, installing an SFC scenario will for instance configure the OpenDaylight controller with the required SFC Karaf features in the OpenDaylight controller. Make sure you read the installation and configuration guide carefully to understand the state of the deployed system.

#### Connecting to the OpenDaylight instance

Once you are familiar with the OpenDaylight controller and its configuration you will want to connect to the OpenDaylight instance from the Jumphost. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

> http://{Controller-VIP}:8181/index.html> username: admin password: admin

Other methods of interacting with and configuring the controller, like the REST API and CLI are also available in the Brahmaputra deployment, see the OpenDaylight User Guide for more information on using those interfaces.

It is important to be aware that when working directly on the OpenDaylight controller the OpenStack instance will not always be aware of the changes you are making to the networking controller. This may result in unrecoverable inconsistencies in your deployment.

### 3.2.2 ONOS User Guide

ONOS is an SDN controller platform developed and released by the ONOS project. The ONOS controller is installed and configured in OPNFV as the networking component of a variety of OPNFV NFVI scenarios.

ONOS runs within a JVM instance and is integrated with OpenStack via a Neutron ML2 plugin. The ONOS instance can be configured through the OpenStack Neutron interface, or through native ONOS tools from the OPNFV jumphost. The Brahmaputra release of OPNFV integrates the latest ONOS 1.4 (EMU) release version.

#### ONOS references

For an overview of the ONOS controller, please see User Guide. For more detailed information about the EMU version of ONOS, documentation is available on the ONOS download page.

### Connecting to the ONOS instance

Once you are familiar with the ONOS controller and its configuration you will want to connect to the ONOS instance from the Jumphost. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

http://{Controller-VIP}:8282/index.html> username: karaf password: karaf

Other methods of interacting with and configuring the controller, like the REST API and CLI are also available in the Brahmaputra deployment, see the ONOS User Guide for more information on using those interfaces.

It is important to be aware that when working directly on the ONOS controller the OpenStack instance will not always be aware of the changes you are making to the networking controller. This may result in unrecoverable inconsistencies in your deployment.

If you have any questions or need further assistance, you may also direct your queries to *ONOSFW Forum <http://forum.onosfw.com>*

CHAPTER

FOUR

# USING BRAHMAPUTRA FEATURES

The following sections of the user guide provide feature specific usage guidelines and references. Providing users the necessary information to leveraging the features in the platform, some operation in this section may refer back to the guides in the general system usage section.

## 4.1 Copper capabilities and usage

This release focused on use of the OpenStack Congress service for managing configuration policy. See the Congress intro guide on readthedocs for information on the capabilities and usage of Congress.

## 4.2 Doctor capabilities and usage

### 4.2.1 Immediate Notification

Immediate notification can be used by creating 'event' type alarm via OpenStack Alarming (Aodh) API with relevant internal components support.

See, upstream spec document: http://specs.openstack.org/openstack/ceilometer-specs/specs/liberty/event-alarm-evaluator.html

You can find an example of consumer of this notification in doctor repository. It can be executed as follows:

```
git clone https://gerrit.opnfv.org/gerrit/doctor -b stable/brahmaputra
cd doctor/tests
CONSUMER_PORT=12346
python consumer.py "$CONSUMER_PORT" > consumer.log 2>&1 &
```

### 4.2.2 Consistent resource state awareness (Compute/host-down)

Resource state of compute host can be fixed according to an input from a monitor sitting out side of OpenStack Compute (Nova) by using force-down API.

See http://artifacts.opnfv.org/doctor/brahmaputra/docs/manuals/mark-host-down_manual.html for more detail.

## 4.3 Using IPv6 Feature of Brahmaputra Release

This section provides the users with gap analysis regarding IPv6 feature requirements with OpenStack Liberty Official Release and Open Daylight Beryllium Official Release. The gap analysis serves as feature specific user guides and

references when as a user you may leverage the IPv6 feature in the platform and need to perform some IPv6 related operations.

### 4.3.1 IPv6 Gap Analysis with OpenStack Liberty

This section provides users with IPv6 gap analysis regarding feature requirement with OpenStack Neutron in Liberty Official Release. The following table lists the use cases / feature requirements of VIM-agnostic IPv6 functionality, including infrastructure layer and VNF (VM) layer, and its gap analysis with OpenStack Neutron in Liberty Official Release.

| Use Case / Requirement | Supported in Liberty | Notes |
|---|---|---|
| All topologies work in a multi-tenant environment | Yes | The IPv6 design is following the Neutron tenant networks model; dnsmasq is being used inside DHCP network namespaces, while radvd is being used inside Neutron routers namespaces to provide full isolation between tenants. Tenant isolation can be based on VLANs, GRE, or VXLAN encapsulation. In case of overlays, the transport network (and VTEPs) must be IPv4 based as of today. |
| IPv6 VM to VM only | Yes | It is possible to assign IPv6-only addresses to VMs. Both switching (within VMs on the same tenant network) as well as east/west routing (between different networks of the same tenant) are supported. |
| IPv6 external L2 VLAN directly attached to a VM | Yes | IPv6 provider network model; RA messages from upstream (external) router are forwarded into the VMs |
| IPv6 subnet routed via L3 agent to an external IPv6 network<br>1. Both VLAN and overlay (e.g. GRE, VXLAN) subnet attached to VMs;<br>2. Must be able to support multiple L3 agents for a given external network to support scaling (neutron scheduler to assign vRouters to the L3 agents) | 1. Yes<br>2. Yes | Configuration is enhanced since Kilo to allow easier setup of the upstream gateway, without the user being forced to create an IPv6 subnet for the external network. |

Continued on next page

Table 4.1 – continued from previous page

| Use Case / Requirement | Supported in Liberty | Notes |
|---|---|---|
| Ability for a NIC to support both IPv4 and IPv6 (dual stack) address.<br>1. VM with a single interface associated with a network, which is then associated with two subnets.<br>2. VM with two different interfaces associated with two different networks and two different subnets. | 1. Yes<br>2. Yes | Dual-stack is supported in Neutron with the addition of `Multiple IPv6 Prefixes` Blueprint |
| Support IPv6 Address assignment modes.<br>1. SLAAC<br>2. DHCPv6 Stateless<br>3. DHCPv6 Stateful | 1. Yes<br>2. Yes<br>3. Yes | |
| Ability to create a port on an IPv6 DHCPv6 Stateful subnet and assign a specific IPv6 address to the port and have it taken out of the DHCP address pool. | Yes | |
| Ability to create a port with fixed_ip for a SLAAC/DHCPv6-Stateless Subnet. | **No** | The following patch disables this operation: https://review.openstack.org/#/c/129144/ |
| Support for private IPv6 to external IPv6 floating IP; Ability to specify floating IPs via Neutron API (REST and CLI) as well as via Horizon, including combination of IPv6/IPv4 and IPv4/IPv6 floating IPs if implemented. | **Rejected** | Blueprint proposed in upstream and got rejected. General expectation is to avoid NAT with IPv6 by assigning GUA to tenant VMs. See https://review.openstack.org/#/c/139731/ for discussion. |
| Provide IPv6/IPv4 feature parity in support for pass-through capabilities (e.g., SR-IOV). | **To-Do** | The L3 configuration should be transparent for the SR-IOV implementation. SR-IOV networking support introduced in Juno based on the `sriovnicswitch` ML2 driver is expected to work with IPv4 and IPv6 enabled VMs. We need to verify if it works or not. |
| Additional IPv6 extensions, for example: IPSEC, IPv6 Anycast, Multicast | **No** | It does not appear to be considered yet (lack of clear requirements) |
| VM access to the meta-data server to obtain user data, SSH keys, etc. using cloud-init with IPv6 only interfaces. | **No** | This is currently not supported. Config-drive or dual-stack IPv4 / IPv6 can be used as a workaround (so that the IPv4 network is used to obtain connectivity with the metadata service) |

Continued on next page

Table 4.1 – continued from previous page

| Use Case / Requirement | Supported in Liberty | Notes |
|---|---|---|
| Full support for IPv6 matching (i.e., IPv6, ICMPv6, TCP, UDP) in security groups. Ability to control and manage all IPv6 security group capabilities via Neutron/Nova API (REST and CLI) as well as via Horizon. | Yes | |
| During network/subnet/router create, there should be an option to allow user to specify the type of address management they would like. This includes all options including those low priority if implemented (e.g., toggle on/off router and address prefix advertisements); It must be supported via Neutron API (REST and CLI) as well as via Horizon | Yes | Two new Subnet attributes were introduced to control IPv6 address assignment options:<br>• `ipv6-ra-mode`: to determine who sends Router Advertisements;<br>• `ipv6-address-mode`: to determine how VM obtains IPv6 address, default gateway, and/or optional information. |
| Security groups anti-spoofing: Prevent VM from using a source IPv6/MAC address which is not assigned to the VM | Yes | |
| Protect tenant and provider network from rogue RAs | Yes | When using a tenant network, Neutron is going to automatically handle the filter rules to allow connectivity of RAs to the VMs only from the Neutron router port; with provider networks, users are required to specify the LLA of the upstream router during the subnet creation, or otherwise manually edit the security-groups rules to allow incoming traffic from this specific address. |
| Support the ability to assign multiple IPv6 addresses to an interface; both for Neutron router interfaces and VM interfaces. | Yes | |
| Ability for a VM to support a mix of multiple IPv4 and IPv6 networks, including multiples of the same type. | Yes | |
| Support for IPv6 Prefix Delegation. | Yes | Partial support in Liberty |
| Distributed Virtual Routing (DVR) support for IPv6 | **No** | Blueprint proposed upstream, pending discussion. |
| IPv6 First-Hop Security, IPv6 ND spoofing | Yes | |
| IPv6 support in Neutron Layer3 High Availability (keepalived+VRRP). | Yes | |

### 4.3.2 IPv6 Gap Analysis with Open Daylight Beryllium

This section provides users with IPv6 gap analysis regarding feature requirement with Open Daylight Beryllium Official Release. The following table lists the use cases / feature requirements of VIM-agnostic IPv6 functionality,

including infrastructure layer and VNF (VM) layer, and its gap analysis with Open Daylight Beryllium Official Release.

| Use Case / Requirement | Supported in ODL Beryllium | Notes |
|---|---|---|
| REST API support for IPv6 subnet creation in ODL | Yes | Yes, it is possible to create IPv6 subnets in ODL using Neutron REST API.<br>For a network which has both IPv4 and IPv6 subnets, ODL mechanism driver will send the port information which includes IPv4/v6 addresses to ODL Neutron northbound API. When port information is queried it displays IPv4 and IPv6 addresses. However, in Beryllium release, ODL net-virt provider does not support IPv6 features (i.e., the actual functionality is missing and would be available only in the later releases of ODL). |
| IPv6 Router support in ODL<br>1. Communication between VMs on same compute node<br>2. Communication between VMs on different compute nodes (east-west)<br>3. External routing (north-south) | **No** | ODL net-virt provider in Beryllium release only supports IPv4 Router.<br>In the meantime, if IPv6 Routing is necessary, we can use ODL for L2 connectivity and Neutron L3 agent for IPv4/v6 routing. |
| IPAM: Support for IPv6 Address assignment modes.<br>1. SLAAC<br>2. DHCPv6 Stateless<br>3. DHCPv6 Stateful | **No** | Although it is possible to create different types of IPv6 subnets in ODL, ODL_L3 would have to implement the IPv6 Router that can send out Router Advertisements based on the IPv6 addressing mode. Router Advertisement is also necessary for VMs to configure the default route. |
| When using ODL for L2 forwarding/tunneling, it is compatible with IPv6. | Yes | |
| Full support for IPv6 matching (i.e., IPv6, ICMPv6, TCP, UDP) in security groups. Ability to control and manage all IPv6 security group capabilities via Neutron/Nova API (REST and CLI) as well as via Horizon. | **No** | Security Groups for IPv6 is a work in progress. |
| Shared Networks support | **No** | ODL currently assumes a single tenant to network mapping and does not support shared networks among tenants. |
| IPv6 external L2 VLAN directly attached to a VM. | **ToDo** | |

Continued on next page

Table 4.2 – continued from previous page

| Use Case / Requirement | Supported in ODL Beryllium | Notes |
|---|---|---|
| ODL on an IPv6 only Infrastructure. | **ToDo** | Deploying OpenStack with ODL on an IPv6 only infrastructure where the API endpoints are all IPv6 addresses. |

## 4.4 Open vSwitch

Open vSwtich (OVS) is a software switch commonly used in OpenStack deployments to replace Linux bridges as it offers advantages in terms of mobility, hardware integration and use by network controllers.

### 4.4.1 Supported OPNFV Installers

Currently not all installers are supported.

#### Fuel Installer

OVSNFV project supplies a Fuel Plugin to upgrades Open vSwitch on an OPNFV installation to use user-space datapath.

As part of the upgrade the following changes are also made:

- change libvirt on compute node to 1.2.12
- change qemu on compute node to 2.2.1
- installs DPDK 2.0.0
- installs OVS 2.1 (specifically git tag 1e77bbe)
- removes existing OVS neutron plugin
- installs new OVS plugin as part of networking_ovs_dpdk OpenStack plugin version stable/kilo
- work around _set_device_mtu issue

#### Limitations

This release should be considered experimental. In particular:

- performance will be addressed specifically in subsequent releases.
- OVS and other components are updated only on compute nodes.

#### Bugs

- There may be issues assigning floating and public ip address to VMs.

## 4.5 Promise capabilities and usage

Promise is a resource reservation and management project to identify NFV related requirements and realize resource reservation for future usage by capacity management of resource pools regarding compute, network and storage.

The following are the key features provided by this module:

- Capacity Management
- Reservation Management
- Allocation Management

The Brahmaputra implementation of Promise is built with the YangForge data modeling framework [1] , using a shim-layer on top of OpenStack to provide the Promise features. This approach requires communication between Consumers/Administrators and OpenStack to pass through the shim-layer. The shim-layer intercepts the message flow to manage the allocation requests based on existing reservations and available capacities in the providers. It also extracts information from the intercepted messages in order to update its internal databases. Furthermore, Promise provides additional intent-based APIs to allow a Consumer or Administrator to perform capacity management (i.e. add providers, update the capacity, and query the current capacity and utilization of a provider), reservation management (i.e. create, update, cancel, query reservations), and allocation management (i.e. create, destroy, query instances).

Detailed information about Promise use cases, features, interface specifications, work flows, and the underlying Promise YANG schema can be found in the Promise requirement document [2] .

### 4.5.1 Promise usage

The yfc run command will load the primary application package from this repository along with any other dependency files/assets referenced within the YAML manifest and instantiate the opnfv-promise module and run REST/JSON interface by default listeningon port 5000.:

```
$ yfc run promise.yaml
```

You can also checkout the GIT repository (https://github.com/opnfv/promise/) or simply download the files into your local system and run the application.

### 4.5.2 Promise feature and API usage guidelines and examples

This section lists the Promise features and API implemented in OPNFV Brahmaputra.

Note 1: In contrast to ETSI NFV specifications and the detailed interface specification in Section 7, the Promise shim-layer implementation does not distinguish intent interfaces per resource type, i.e. the various capacity, reservations, etc. operations have different endpoints for each domain such as compute, storage, and network. The current shim-layer implementation does not separate the endpoints for performing the various operations.

Note 2: The listed parameters are optional unless explicitly marked as “mandatory”.

#### Reservation management

The reservation management allows a Consumer to request reservations for resource capacity or specific resource elements. Reservations can be for now or a later time window. After the start time of a reservation has arrived, the Consumer can issue create server instance requests against the reserved capacity / elements. Note, a reservation will expire after a predefined *expiry* time in case no allocation referring to the reservation is requested.

[1] YangForge framework, http://github.com/opnfv/yangforge
[2] Promise requirement document, http://http://artifacts.opnfv.org/promise/docs/requirements/index.html

The implemented workflow is well aligned with the described workflow in the Promise requirement document [1] (Clause 6.1) except for the "multi-provider" scenario as described in *(Multi-)provider management* .

#### *create-reservation*

This operation allows making a request to the reservation system to reserve resources. The Consumer can either request to reserve a certain capacity (*container*) or specific resource elements (*elements*), like a certain server instance.

The operation takes the following input parameters:

- start: start time of the requested reservation
- end: end time of the requested reservation
- container: request for reservation of capacity
  - instances: number of instances
  - cores: number of cores
  - ram: size of ram (in MB)
  - networks: number of networks
  - addresses: number of (public) IP addresses
  - ports: number of ports
  - routers: number of routers
  - subnets: number of subnets
  - gigabytes: size of storage (in GB)
  - volumes: number of volumes
  - snapshots: number of snapshots
- elements: reference to a list of 'pre-existing' resource elements that are required for fulfillment of the resource-usage-request
  - instance-identifier: identifier of a specific resource element
- zone: identifier of an Availability Zone

Promise will check the available capacity in the given time window and in case sufficient capacity exists to meet the reservation request, will mark those resources "reserved" in its reservation map.

#### *update-reservation*

This operation allows to update the reservation details for an existing reservation.

It can take the same input parameters as in *create-reservation* but in addition requires a mandatory reference to the *reservation-id* of the reservation that shall be updated.

#### *cancel-reservation*

This operation is used to cancel an existing reservation.

The operation takes the following input parameter:

- reservation-id (mandatory): identifier of the reservation to be canceled.

#### *query-reservation*

The operation queries the reservation system to return reservation(s) matching the specified query filter, e.g., reservations that are within a specified start/end time window.

The operation takes the following input parameters to narrow down the query results:

- zone: identifier of an Availability Zone
- without: excludes specified collection identifiers from the result
- elements:
  - some: query for ResourceCollection(s) that contain some or more of these element(s)
  - every: query for ResourceCollection(s) that contain all of these element(s)
- window: matches entries that are within the specified start/end time window
  - start: start time
  - end: end time
  - scope: if set to 'exclusive', only reservations with start AND end time within the time window are returned. Otherwise ('inclusive'), all reservation starting OR ending in the time windows are returned.
- show-utilization: boolean value that specifies whether to also return the resource utilization in the queried time window or not

#### *subscribe-reservation-events* / *notify-reservation-events*

Subscription to receive notifications about reservation-related events, e.g. a reservation is about to expire or a reservation is in conflict state due to a failure in the NFVI.

Note, this feature is not yet available in Brahmaputra release.

### Allocation management

#### *create-instance*

This operation is used to create an instance of specified resource(s) for immediate use utilizing capacity from the pool. *Create-instance* requests can be issued against an existing reservation, but also allocations without a reference to an existing reservation are allowed. In case the allocation request specifies a reservation identifier, Promise checks if a reservation with that ID exists, the reservation start time has arrived (i.e. the reservation is 'active'), and the required capacity for the requested flavor is within the available capacity of the reservation. If those conditions are met, Promise creates a record for the allocation (VMState="INITIALIZED") and update its databases. If no *reservation_id* was provided in the allocation request, Promise checks whether the required capacity to meet the request can be provided from the available, non-reserved capacity. If yes, Promise creates a record for the allocation with an unique *instance-id* and update its databases. In any other case, Promise rejects the *create-instance* request.

In case the *create-instance* request is rejected, Promise responds with a "status=rejected" providing the reason of the rejection. This will help the Consumer to take appropriate actions, e.g., send an updated *create-instance* request. In case the *create-instance* request was accepted and a related allocation record has been created, the shim-layer issues a *createServer* request to the VIM Controller providing all information to create the server instance.

The operation takes the following input parameters:

- name (mandatory): Assigned name for the instance to be created
- image (mandatory): the image to be booted in the new instance

- flavor (mandatory): the flavor of the requested server instance
- networks: the list of network uuids of the requested server instance
- provider-id: identifier of the provider where the instance shall be created
- reservation-id: identifier of a resource reservation the *create-instance* is issued against

The Brahamputra implementation of Promise has the following limitations:

- All create server instance requests shall pass through the Promise shim-layer such that Promise can keep track of all allocation requests. This is necessary as in the current release the sychronization between the VIM Controller and Promise on the available capacity is not yet implemented.
- *Create-allocation* requests are limited to "simple" allocations, i.e., the current workflow only supports the Nova compute service and *create-allocation* requests are limited to creating one server instance at a time
- Prioritization of reservations and allocations is yet not implemented. Future version may allow certain policy-based conflict resolution where, e.g., new allocation request with high priority can "forcefully" terminate lower priority allocations.

#### *destroy-instance*

This operation request to destroy an existing server instance and release it back to the pool.

The operation takes the following input parameter:

- instance-id: identifier of the server instance to be destroyed

#### *query-resource-collection*

This operation allows to query for resource collection(s) that are within the specified start/end time window.

#### *subscribe-allocation-events* / *notify-allocation-events*

Subscription to receive notifications about allocation-related events, e.g. an allocation towards the VIM that did not pass the Promise shim-layer

Note, this feature is not yet available in Brahmaputra release.

### Capacity management

The capacity management feature allows the Consumer or Administrator to do capacity planning, i.e. the capacity available to the reservation management can differ from the actual capacity in the registered provider(s). This feature can, e.g., be used to limit the available capacity for a given time window due to a planned downtime of some of the resources, or increase the capacity available to the reservation system in case of a plannes upgrade of the available capacity.

#### *increase/decrease-capacity*

This operations allows to increase/decrease the total capacity that is made available to the Promise reservation service between a specified window in time. It does NOT increase the actual capacity of a given resource provider, but is used for capacity management inside Promise.

This feature can be used in different ways, like

- Limit the capacity available to the reservation system to a value below 100% of the available capacity in the VIM, e.g., in order to leave "buffer" in the actual NFVI to be used outside the Promise reservation service.
- Inform the reservation system that, from a given time in the future, additional resources can be reserved, e.g., due to a planned upgrade of the available capacity of the provider.
- Similarily, the "decrease-capacity" can be used to reduce the consumable resources in a given time window, e.g., to prepare for a planned downtime of some of the resources.
- Expose multiple reservation service instances to different consumers sharing the same resource provider.

The operation takes the following input parameters:

- start: start time for the increased/decreased capacity
- end: end time for the increased/decreased capacity
- container: see *create-reservation*

Note, increase/decreasing the capacity in Promise is completely transparent to the VIM. As such, when increasing the virtual capacity in Promise (e.g. for a planned upgrade of the capacity), it is in the responsibility of the Consumer/Administrator to ensure sufficient resources in the VIM are available at the appropriate time, in order to prevent allocations against reservations to fail due to a lack of resources. Therefore, this operations should only be used carefully.

#### *query-capacity*

This operation is used to query the available capacity information of the specified resource collection. A filter attribute can be specified to narrow down the query results.

The current implementation supports the following filter criteria:

- time window: returns reservations matching the specified window
- window scope: if set to 'exclusive', only reservations with start AND end time within the time window are returned. Otherwise, all reservation starting OR ending in the time windows are returned.
- metric: query for one of the following capacity metrics:
  - 'total': resource pools
  - 'reserved': reserved resources
  - 'usage': resource allocations
  - 'available': remaining capacity, i.e. neither reserved nor allocated

#### *subscribe-capacity-events* / *notify-capacity-events*

These operations enable the Consumer to subscribe to receiving notifications about capacity-related events, e.g., increased/decreased capacity for a provider due to a failure or upgrade of a resource pool. In order to provide such notifications to its Consumers, Promise shim-layer has to subscribe itself to OpenStack Aodh to be notified from the VIM about any capacity related events.

Note, this feature is not yet available in Brahmaputra release.

#### (Multi-)provider management

This API towards OpenStack allows an Consumer/Administrator to add and remove resource providers to Promise. Note, Promise supports a multi-provider configuration, however, for Brahmaputra, multi-provider support is not yet fully supported.

##### *add-provider*

This operation is used to register a new resource provider into the Promise reservation system.

Note, for Brahmaputra, the add-provider operation should only be used to register one provider with the Promise shim-layer. Further note that currently only OpenStack is supported as a provider.

The operation takes the following input parameters:

- provider-type (mandatory) = 'openstack': select a specific resource provider type.
- endpoint (mandatory): target URL endpoint for the resource provider.
- username (mandatory)
- password (mandatory)
- region: specified region for the provider
- tenant
  - id
  - name

##### *remove-provider*

This operation removes a resource provider from the reservation system. Note, this feature is not yet available in Brahmaputra release.

## 4.6 SDN VPN capabilities and usage

The BGPVPN feature enables creation of BGP VPNs according to the OpenStack BGPVPN blueprint at https://blueprints.launchpad.net/neutron/+spec/neutron-bgp-vpn. In a nutshell, the blueprint defines a BGPVPN object and a number of ways how to associate it with the existing Neutron object model, including a unique definition of the related semantics. The BGPVPN framework supports a backend driver model with currently available drivers for Bagpipe, OpenContrail, Nuage and OpenDaylight.

Currently, in OPNFV only ODL is supported as a backend for BGPVPN. API calls are mapped onto the ODL VPN Service REST API through the BGPVPN ODL driver and the ODL Neutron Northbound module.

### 4.6.1 Feature and API usage guidelines and example

For the details of using OpenStack BGPVPN API, please refer to the documentation at http://docs.openstack.org/developer/networking-bgpvpn/.

CHAPTER

FIVE

# USING THE TEST FRAMEWORKS IN OPNFV

Testing is one of the key activities in OPNFV, validation can include component level testing, system testing, automated deployment validation and performance characteristics testing.

The following section outlines how to use the test projects that are delivered on the OPNFV platform for the purpose of testing components and VNFs in the context of a Brahmaputra deployment.