Promise: Resource Management

Release 2015.1.1 (82e0ab3)

OPNFV

August 22, 2016

1 Definition of terms 3

2 Introduction 5
2.1 Problem description 5

3 Use cases and scenarios 7
3.1 Use cases 7
3.2 Scenarios 8

4 High level architecture and general features 9
4.1 Architecture Overview 9
4.2 General Features 9
4.3 High level northbound interface specification 10
4.4 Information elements 13

5 Gap analysis in upstream projects 15
5.1 OpenStack 15

6 Detailed architecture and message flows 17
6.1 Shim-layer architecture 17
6.2 Integrated architecture 22

7 Detailed northbound interface specification 25
7.1 ETSI NFV IFA Information Models 25
7.2 Virtualised Compute Resources 25
7.3 Virtualised Network Resources 32
7.4 Virtualised Storage Resources 36

8 Summary and conclusion 41

9 References and bibliography 43

10 ANNEX A: Use case for OPNFV Brahmaputra 45

11 ANNEX B: Promise YANG schema based on YangForge 47

12 ANNEX C: Supported APIS 59
12.1 Add Provider 59
12.2 Create Reservation 60
12.3 Update Reservation 61
12.4 Cancel Reservation 62
12.5 Query Reservation 62

12.6 Create Instance 63
12.7 Destroy Instance 64
12.8 Decrease Capacity 64
12.9 Increase Capacity 65
12.10 Query Capacity 66

13 ANNEX D: Document revision **69**

Bibliography **71**

Index **73**

Project Promise, https://wiki.opnfv.org/promise

Editors Ashiq Khan (NTT DOCOMO), Bertrand Souville (NTT DOCOMO)

Authors Ravi Chunduru (ClearPath Networks), Peter Lee (ClearPath Networks), Gerald Kunzmann (NTT DOCOMO), Ryota Mibu (NEC), Carlos Goncalves (NEC), Arturo Martin De Nicolas (Ericsson)

History

Date	Description
04.12.2014	Project creation
20.04.2015	Initial version of the deliverable uploaded to gerrit
19.06.2015	Stable version of the Promise deliverable

Abstract Promise is an OPNFV requirement project. Its objective is to realize ETSI NFV defined resource reservation and NFVI capacity features within the scope of OPNFV. Promise provides the details of the requirements on resource reservation, NFVI capacity management at VIM, specification of the northbound interfaces from VIM relevant to these features, and implementation plan to realize these features in OPNFV.

CHAPTER

ONE

DEFINITION OF TERMS

Different SDOs and communities use different terminology related to NFV/Cloud/SDN. This list tries to define an OPNFV terminology, mapping/translating the OPNFV terms to terminology used in other contexts.

Administrator Administrator of the system, e.g. OAM in Telco context.

Consumer User-side Manager; consumer of the interfaces produced by the VIM; VNFM, NFVO, or Orchestrator in ETSI NFV *[NFV003]* terminology.

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.

NFVO Network Functions Virtualization Orchestrator; functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity.

Physical resource Actual resources in NFVI; not visible to Consumer.

Resource zone A set of NFVI hardware and software resources logically grouped according to physical isolation and redundancy capabilities or to certain administrative policies for the NFVI *[NFVIFA010]*

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).

Virtual Machine (VM) Virtualized computation environment that behaves very much like a physical computer/server.

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

Virtual resource A Virtual Machine (VM), a virtual network, or virtualized storage; Offered resources to "Consumer" as result of infrastructure virtualization; visible to Consumer.

Virtual Storage Virtualized non-volatile storage allocated to a VM.

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

VNFM Virtualized Network Function Manager; functional block that is responsible for the lifecycle management of VNF.

CHAPTER

TWO

INTRODUCTION

Resource reservation is a basic function for the operation of a virtualized telecom network. In resource reservation, VIM reserves resources for a certain period as requested by the NFVO. A resource reservation will have a start time which could be into the future. Therefore, the reserved resources shall be available for the NFVO requested purpose (e.g. for a VNF) at the start time for the duration asked by NFVO. Resources include all three resource types in an NFVI i.e. compute, storage and network.

Besides, NFVO requires abstracted NFVI resource capacity information in order to take decisions on VNF placement and other operations related to the virtual resources. VIM is required to inform the NFVO of NFVI resource state information for this purpose. Promise project aims at delivering the detailed requirements on these two features defined in ETSI NFV MAN GS *[NFVMAN]*, the list of gaps in upstream projects, potential implementation architecture and plan, and the VIM northbound interface specification for resource reservation and capacity management.

2.1 Problem description

OpenStack, a prominent candidate for the VIM, cannot reserve resources for future use. OpenStack requires immediate instantiation of Virtual Machines (VMs) in order to occupy resources intended to be reserved. Blazar can reserve compute resources for future by keeping the VMs in shelved mode. However, such reserved resources can also be used for scaling out rather than new VM instantiation. Blazar does not support network and storage resource reservation yet.

Besides, OpenStack does not provide a northbound interface through which it can notify an upper layer management entity e.g. NFVO about capacity changes in its NFVI, periodically or in an event driven way. Capacity management is a feature defined in ETSI NFV MAN GS *[NFVMAN]* and is required in network operation.

CHAPTER

THREE

USE CASES AND SCENARIOS

3.1 Use cases

Resource reservation is a basic feature in any virtualization-based network operation. In order to perform such resource reservation from NFVO to VIM, NFVI capacity information is also necessary at the NFVO side. Below, four use cases to show typical requirements and solutions for capacity management and resource reservation is presented. A typical use case as considered for the Brahmaputra release is described in *ANNEX A: Use case for OPNFV Brahmaputra*.

1. Resource capacity management
2. Resource reservation for immediate use
3. Resource reservation for future use
4. Co-existence of reservations and allocation requests without reservation

3.1.1 Resource capacity management

NFVO takes the first decision on in which NFVI it would instantiate a VNF. Along with NFVIs resource attributes (e.g. availability of hardware accelerators, particular CPU architectures etc.), NFVO needs to know available capacity of an NFVI in order to make an informed decision on selecting a particular NFVI. Such capacity information shall be in a coarser granularity than the respective VIM, as VIM maintains capacity information of its NFVI in fine details. However a very coarse granularity, like simply the number of available virtual CPU cores, may not be sufficient. In order to allow the NFVO to make well founded allocation decisions, an appropriate level to expose the available capacity may be per flavor. Capacity information may be required for the complete NFVI, or per partition or availability zone, or other granularities. Therefore, VIM requires to inform the NFVO about available capacity information regarding its NFVI at a pre-determined abstraction, either by a query-response, or in an event-based, or in a periodical way.

3.1.2 Resource reservation for immediate use

Reservation is inherently for the future. Even if some reserved resources are to be consumed instantly, there is a network latency between the issuance of a resource reservation request from the NFVO, a response from the VIM, and actual allocation of the requested resources to a VNF/VNFM. Within such latency, resource capacity in the NFVI in question could change, e.g., due to failure, allocation to a different request. Therefore, the response from a VIM to the NFVO to a resource reservation request for immediate use should have a validity period which shows until when this VIM can hold the requested resources. During this time, the NFVO should proceed to allocation if it wishes to consume the reserved requested. If allocation is not performed within the validity period, the response from VIM for a particular resource reservation request becomes invalid and VIM is not liable to provide those resources to NFVO/VNFM anymore. Reservations requests for immediate use do not have a start time but may have an end time.

3.1.3 Resource reservation for future use

Network operators may want to reserve extra resources for future use. Such necessity could arise from predicted congestion in telecom nodes e.g. due to local traffic spikes for concerts, natural disasters etc. In such a case, the NFVO, while sending a resource reservation request to the VIM, shall include a start time (and an end time if necessary). The start time indicates at what time the reserved resource shall be available to a designated consumer e.g. a VNF/VNFM. Here, the requirement is that the reserved resources shall be available when the start time arrives. After the start time has arrived, the reserved resources are allocated to the designated consumer(s). An explicit allocation request is needed. How actually these requested resources are held by the VIM for the period in between the arrival of the resource reservation request and the actual allocation is outside the scope of this requirement project.

3.1.4 Co-existence of reservations and allocation requests without reservation

In a real environment VIM will have to handle allocation requests without any time reference, i.e. time-unbound, together with time-bound reservations and allocation requests with an explicitly indicated end-time. A granted reservation for the future will effectively reduce the available capacity for any new time-unbound allocation request. The consequence is that reservations, even those far in the future, may result in denial of service for new allocation requests.

To alleviate this problem several approaches can be taken. They imply an implicit or explicit priority scheme:

- Allocation requests without reservation and which are time-unbound will be granted resources in a best-effort way: if there is instant capacity, but the resources may be later withdrawn due to the start time of a previously granted reservation
- Both allocation requests and reservation requests contain a priority which may be related to SLAs and contractual conditions between the tenant and the NFVI provider. Interactions may look like:
  - A reservation request for future use may cancel another, not yet started, reservation with lower priority
  - An allocation request without reservations and time-unbound [1] may be granted resources and prevent a future reservation with lower priority from getting resources at start time
  - A reservation request may result in terminating resources allocated to a request with no reservation, if the latter has lower priority

3.2 Scenarios

This section describes the expected behavior of the system in different scenarios.

As we are targeting a cloud platform with the above use cases, it is crucial to keep the flexibility in the system. By this means it is hard to explicitely block hardware resources for the future and ensure their availablility for allocation therefore it can happen that the reservation plan cannot be fulfilled due to certain changes in the underlying environment. We need to ensure that we are prepared for error and edge cases during the design period.

[1] In this case, the consumer (VNFM or NFVO) requests to immediately instantiate and assign virtualized resources without having reserved the resources beforehand

CHAPTER

FOUR

HIGH LEVEL ARCHITECTURE AND GENERAL FEATURES

4.1 Architecture Overview

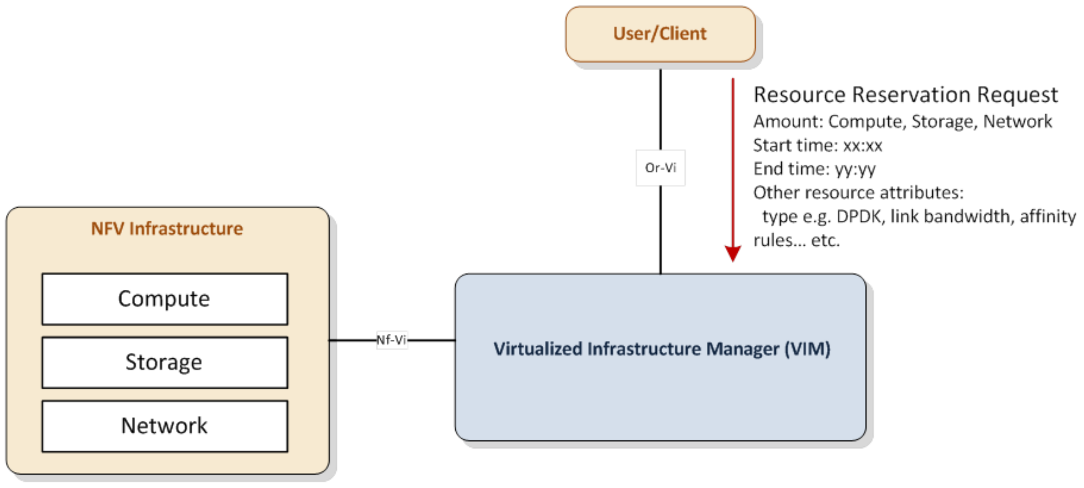

User/Client
Resource Reservation Request
Amount: Compute, Storage, Network
Start time: xx:xx
End time: yy:yy
Other resource attributes:
type e.g. DPDK, link bandwidth, affinity
rules... etc.
Or-Vi
NFV Infrastructure
Compute
Storage
Network
Nf-Vi
Virtualized Infrastructure Manager (VIM)

Fig. 4.1: Resource Reservation Architecture

Fig. 4.1 shows the high level architecture for the resource reservation use cases. Reserved resources are guaranteed for a given user/client for the period expressed by start and end time. User/client represents the requestor and the consequent consumer of the reserved resources and correspond to the NFVO or VNFM in ETSI NFV terminology.

Note: in this document only reservation requests from NFVO are considered.

4.2 General Features

This section provides a list of features that need to be developed in the Promise project.

- Resource capacity management
  - Discovery of available resource capacity in resource providers
  - Monitoring of available resource capacity in resource providers
  - Update available resource capacity as a result of new or expired reservations, addition/removal of resources. Note: this is a VIM internal function, not an operation in the VIM northbound interface.
- Resource reservation

    - Set start time and end time for allocation
    - Increase/decrease reserved resource's capacity
    - Update resource reservations, e.g. add/remove reserved resources
    - Terminate an allocated resource due to the end time of a reservation

- VIM northbound interfaces
    - Receive/Reply resource reservation requests
    - Receive/Reply resource capacity management requests
    - Receive/Reply resource allocation requests for reserved resources when start time arrives
    - Subscribe/Notify resource reservation event
        * Notify reservation error or process completion prior to reservation start
        * Notify remaining time until termination of a resource due to the end time of a reservation
        * Notify termination of a resource due to the end time of a reservation
    - Receive/Reply queries on available resource capacity
    - Subscribe/Notify changes in available resource capacity

4.3 High level northbound interface specification

4.3.1 Resource Capacity Management

Fig. 4.2 shows a high level flow for a use case of resource capacity management. In this example, the VIM notifies the NFVO of capacity change after having received an event regarding a change in capacity (e.g. a fault notification) from the NFVI. The NFVO can also retrieve detailed capacity information using the Query Capacity Request interface operation.

Fig. 4.3 shows a high level flow for another use case of resource capacity management. In this example, the NFVO queries the VIM about the actual capacity to instantiate a certain resource according to a certain template, for example a VM according to a certain flavor. In this case the VIM responds with the number of VMs that could be instantiated according to that flavor with the currently available capacity.

4.3.2 Resource Reservation

Fig. 4.4 shows a high level flow for a use case of resource reservation. The main steps are:

- The NFVO sends a resource reservation request to the VIM using the Create Resource Reservation Request interface operation.
- The NFVO gets a reservation identifier reservation associated with this request in the reply message
- Using the reservation identifier reservation, the NFVO can query/update/terminate a resource reservation using the corresponding interface operations
- The NFVO is notified that the resource reservation is terminated due to the end time of the reservation

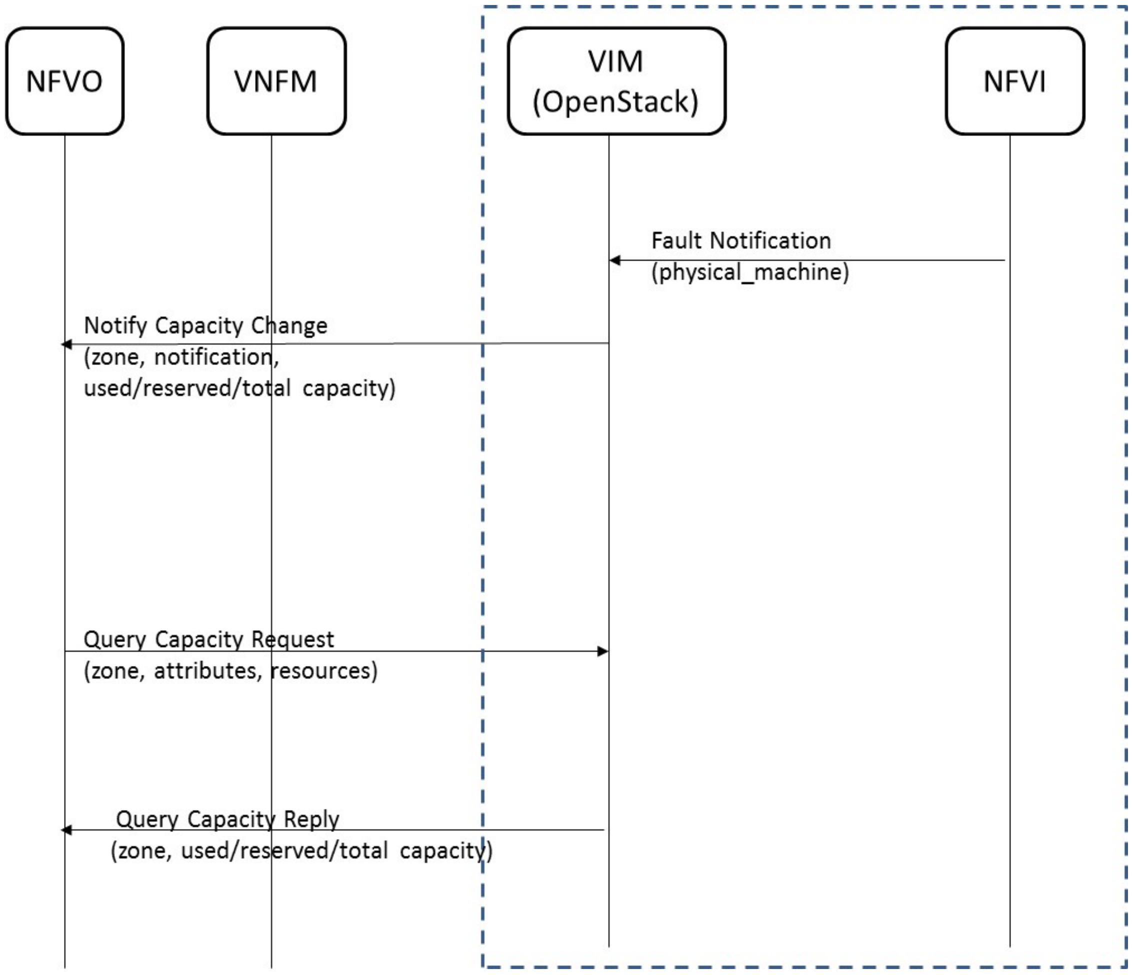

NFVO
VNFM
VIM (OpenStack)
NFVI
Fault Notification (physical_machine)
Notify Capacity Change (zone, notification, used/reserved/total capacity)
Query Capacity Request (zone, attributes, resources)
Query Capacity Reply (zone, used/reserved/total capacity)

Fig. 4.2: Resource capacity management message flow: notification of capacity change

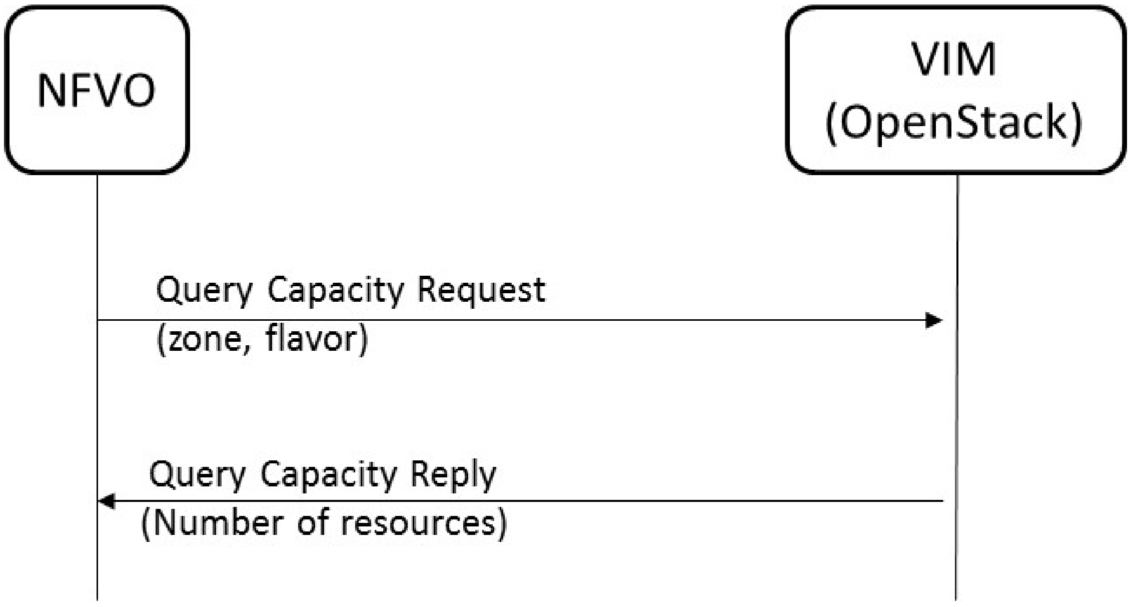

NFVO
VIM (OpenStack)
Query Capacity Request (zone, flavor)
Query Capacity Reply (Number of resources)

Fig. 4.3: Resource capacity management message flow: query of capacity density

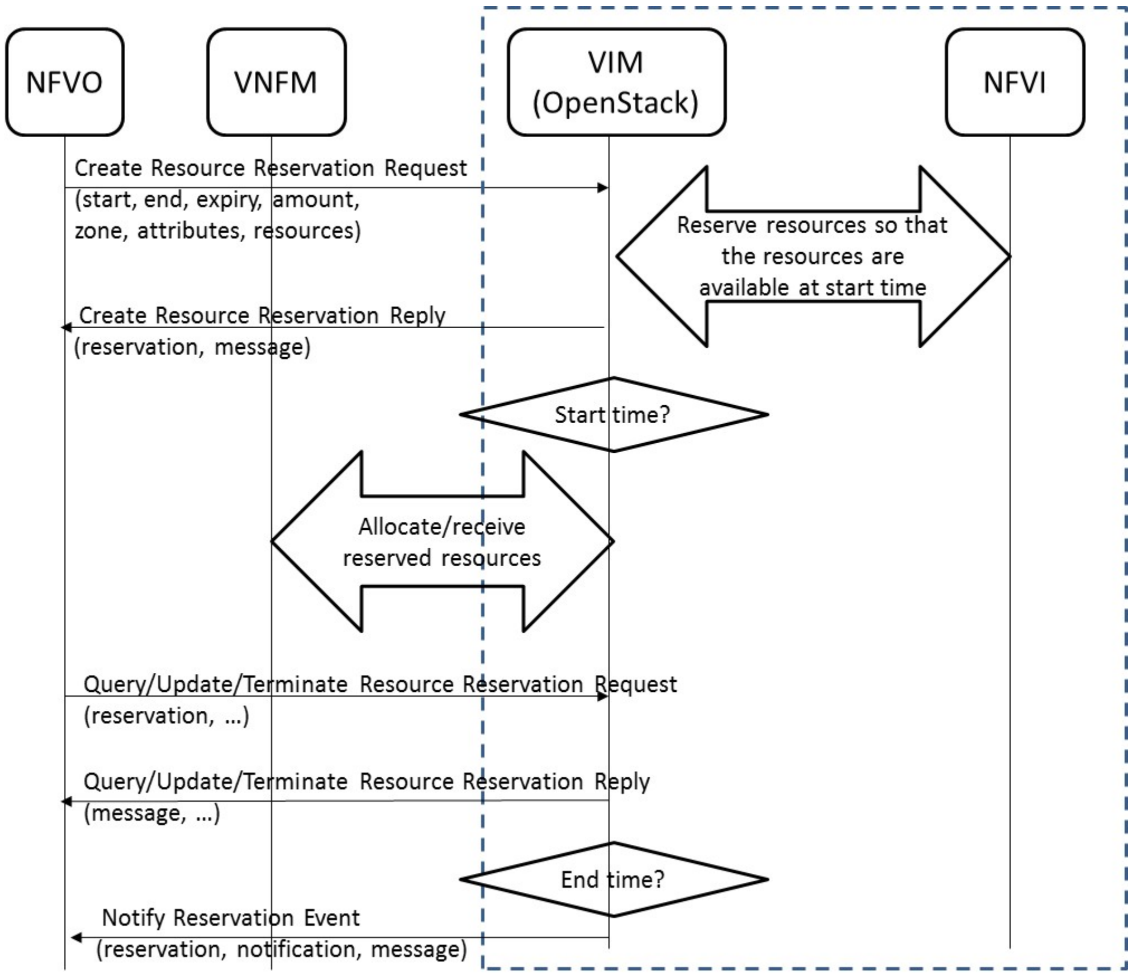

NFVO
VNFM
VIM (OpenStack)
NFVI
Create Resource Reservation Request (start, end, expiry, amount, zone, attributes, resources)
Reserve resources so that the resources are available at start time
Create Resource Reservation Reply (reservation, message)
Start time?
Allocate/receive reserved resources
Query/Update/Terminate Resource Reservation Request (reservation, …)
Query/Update/Terminate Resource Reservation Reply (message, …)
End time?
Notify Reservation Event (reservation, notification, message)

Fig. 4.4: Resource reservation flow

4.4 Information elements

4.4.1 Resource Capacity Management

Notify Capacity Change Event

The notification change message shall include the following information elements:

Name	Type	Description
Notification	Identifier	Identifier issued by the VIM for the capacity change event notification
Zone	Identifier	Identifier of the zone where capacity has changed
Used/Reserved/Total Capacity	List	Used, reserved and total capacity information regarding the resource items subscribed for notification for which capacity change event occurred

Query Resource Capacity Request

The capacity management query request message shall include the following information elements:

Name	Type	Description
Zone	Identifier	Identifier of the zone where capacity is requested
Attributes	List	Attributes of resource items to be notified regarding capacity change events
Resources	List	Identifiers of existing resource items to be queried regarding capacity info (such as images, flavors, virtual containers, networks, physical machines, etc.)

The capacity management query request message may also include the following information element:

Name	Type	Description
Flavor	Identifier	Identifier that is passed in the request to obtain information of the number of virtual resources that can be instantiated according to this flavor with the available capacity

Query Resource Capacity Reply

The capacity management query reply message shall include the following information elements:

Name	Type	Description
Zone	Identifier	Identifier of the zone where capacity is requested
Used/Reserved/Total Capacity	List	Used, reserved and total capacity information regarding each of the resource items requested to check for capacity

The detailed specification of the northbound interface for Capacity Management in provided in section 5.1.1.

4.4.2 Resource Reservation

Create Resource Reservation Request

The create resource reservation request message shall include the following information elements:

Name	Type	Description
Start	Times-tamp	Start time for consumption of the reserved resources
End	Times-tamp	End time for consumption of the reserved resources
Ex-piry	Times-tamp	If not all reserved resources are allocated between start time and expiry, the VIM shall release the corresponding resources [1]
Amount	Num-ber	Amount of the resources per resource item type (i.e. compute/network/storage) that need to be reserved
Zone	Iden-tifier	The zone where the resources need(s) to be reserved
At-tributes	List	Attributes of the resources to be reserved such as DPDK support, hypervisor, network link bandwidth, affinity rules, etc.
Re-sources	List	Identifiers of existing resource items to be reserved (such as images, flavors, virtual containers, networks, physical machines, etc.)

Create Resource Reservation Reply

The create resource reservation reply message shall include the following information elements:

Name	Type	Description
Reser-vation	Iden-tifier	Identification of the reservation instance. It can be used by a consumer to modify the reservation later, and to request the allocation of the reserved resources.
Mes-sage	Text	Output message that provides additional information about the create resource reservation request (e.g. may be a simple ACK if the request is being background processed by the VIM)

Notify Reservation Event

The notification reservation event message shall include the following information elements:

Name	Type	Description
Reservation	Identifier	Identification of the reservation instance triggering the event
Notification	Identifier	Identification of the resource event notification issued by the VIM
Message	Text	Message describing the event

The detailed specification of the northbound interface for Resource Reservation is provided in section 5.1.2.

[1] Expiry is a period around start time within which, the allocation process must take place. If allocation process does not start within the expiry period, the reservation becomes invalid and VIM should release the resources

CHAPTER

FIVE

GAP ANALYSIS IN UPSTREAM PROJECTS

This section provides a list of gaps in upstream projects for realizing resource reservation and management. The gap analysis work focuses on the current OpenStack Blazar project *[BLAZAR]* in this first release.

5.1 OpenStack

5.1.1 Resource reservation for future use

- Category: Blazar
- Type: 'missing' (lack of functionality)
- Description:
  - To-be: To reserve a whole set of compute/storage/network resources in the future
  - As-is: Blazar currently can do only compute resource reservation by using "Shelved VM"
- Related blueprints:
  - https://blueprints.launchpad.net/blazar/+spec/basic-volume-plugin
  - https://blueprints.launchpad.net/blazar/+spec/basic-network-plugin
  - It was planned in Blazar to implement volume and network/fixed ip reservations

5.1.2 Resource reservation update

- Category: Blazar
- Type: 'missing' (lack of functionality)
- Description:
  - To-be: Have the possibility of adding/removing resources to an existing reservation, e..g in case of NFVI failure
  - As-is: Currently in Blazar, a reservation can only be modified in terms of start/end time
- Related blueprints: N/A

5.1.3 Give me an offer

- Category: Blazar
- Type: 'missing' (lack of functionality)
- Description:
  - To-be: To have the possibility of giving a quotation to a requesting user and an expiration time. Reserved resources shall be released if they are not claimed before this expiration time.
  - As-is: Blazar can already send notification e.g. to inform a given user that a reservation is about to expire
- Related blueprints: N/A

5.1.4 StormStack StormForge

Stormify

- Stormify enables rapid web applications construction
- Based on Ember.js style Data stores
- Developed on Node.js using coffeescript/javascript
- Auto RESTful API generation based on Data Models
- Development starts with defining Data Models
- Code hosted at github : http://github.com/stormstack/stormify

StormForge

- Data Model driven management of Resource Providers
- Based on Stormify Framework and implemented as per the OPNFV Promise requirements
- Data Models are auto generated and RESTful API code from YANG schema
- Currently planned key services include Resource Capacity Management Service and Resource Reservation Service
- List of YANG schemas for Promise project is attached in the Appendix
- Code hosted at github: http://github.com/stormstack/stormforge

Resource Discovery

- Category: StormForge
- Type: 'planning' (lack of functionality)
- Description
  - To-be: To be able to discover resources in real time from OpenStack components. Planning to add OpenStack Project to interface with Promise for real time updates on capacity or any failures
  - As-is: Currently, resource capacity is learnt using NB APIs related to quota
- Related Blueprints: N/A

CHAPTER

SIX

DETAILED ARCHITECTURE AND MESSAGE FLOWS

Within the Promise project we consider two different architectural options, i.e. a *shim-layer* based architecture and an architecture targeting at full OpenStack *integration*.

6.1 Shim-layer architecture

The *shim-layer architecture* is using a layer on top of OpenStack to provide the capacity management, resource reservation, and resource allocation features.

6.1.1 Detailed Message Flows

Note, that only selected parameters for the messages are shown. Refer to *Detailed northbound interface specification* and Annex *ANNEX B: Promise YANG schema based on YangForge* for a full set of message parameters.

Resource Capacity Management

Fig. 6.1 shows a detailed message flow between the consumers and the capacity management functional blocks inside the shim-layer. It has the following steps:

- Step 1a: The Consumer sends a *query-capacity* request to Promise using some filter like time-windows or resource type. The capacity is looked up in the shim-layer capacity map.
- Step 1b: The shim-layer will respond with information about the total, available, reserved, and used (allocated) capacities matching the filter.
- Step 2a: The Consumer can send *increase/decrease-capacity* requests to update the capacity available to the reservation system. It can be 100% of available capacity in the given provider/source or only a subset, i.e., it can allow for leaving some “buffer” in the actual NFVI to be used outside the Promise shim-layer or for a different reservation service instance. It can also be used to inform the reservation system that from a certain time in the future, additional resources can be reserved (e.g. due to a planned upgrade of the capacity), or the available capacity will be reduced (e.g. due to a planned downtime of some of the resources).
- Step 2b: The shim-layer will respond with an ACK/NACK message.
- Step 3a: Consumers can subscribe for capacity-change events using a filter.
- Step 3b: Each successful subscription is responded with a subscription_id.
- Step 4: The shim-layer monitors the capacity information for the various types of resources by periodically querying the various Controllers (e.g. Nova, Neutron, Cinder) or by creating event alarms in the VIM (e.g. with Ceilometer for OpenStack) and updates capacity information in its capacity map.
- Step 5: Capacity changes are notified to the Consumer.

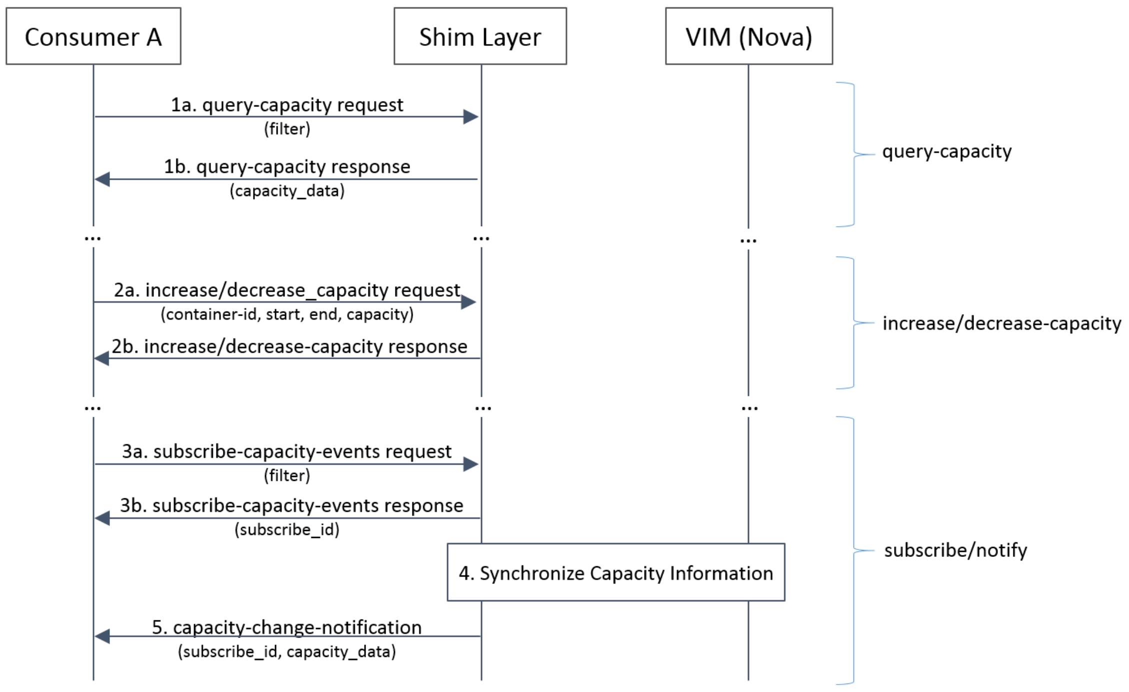

Consumer A
Shim Layer
VIM (Nova)
1a. query-capacity request
(filter)
1b. query-capacity response
(capacity_data)
query-capacity
2a. increase/decrease_capacity request
(container-id, start, end, capacity)
2b. increase/decrease-capacity response
increase/decrease-capacity
3a. subscribe-capacity-events request
(filter)
3b. subscribe-capacity-events response
(subscribe_id)
4. Synchronize Capacity Information
5. capacity-change-notification
(subscribe_id, capacity_data)
subscribe/notify

Fig. 6.1: Capacity Management Scenario

Resource Reservation

Fig. 6.2 shows a detailed message flow between the Consumer and the resource reservation functional blocks inside the shim-layer. It has the following steps:

- Step 1a: The Consumer creates a resource reservation request for future use by setting a start and end time for the reservation as well as more detailed information about the resources to be reserved. The Promise shim-layer will check the free 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.
- Step 1b: If the reservation was successful, a reservation_id and status of the reservation will be returned to the Consumer. In case the reservation cannot be met, the shim-layer may return information about the maximum capacity that could be reserved during the requested time window and/or a potential time window where the requested (amount of) resources would be available.
- Step 2a: Reservations can be updated using an *update-reservation*, providing the reservation_id and the new reservation_data. Promise Reservation Manageer will check the feasibility to update the reservation as requested.
- Step 2b: If the reservation was updated successfully, a reservation_id and status of the reservation will be returned to the Consumer. Otherwise, an appropriate error message will be returned.
- Step 3a: A *cancel-reservation* request can be used to withdraw an existing reservation. Promise will update the reservation map by removing the reservation as well as the capacity map by adding the freed capacity.
- Step 3b: The response message confirms the cancelation.
- Step 4a: Consumers can also issue *query-reservation* requests to receive a list of reservation. An input filter can be used to narrow down the query, e.g., only provide reservations in a given time window. Promise will query its reservation map to identify reservations matching the input filter.
- Step 4b: The response message contains information about all reservations matching the input filter. It also provides information about the utilization in the requested time window.

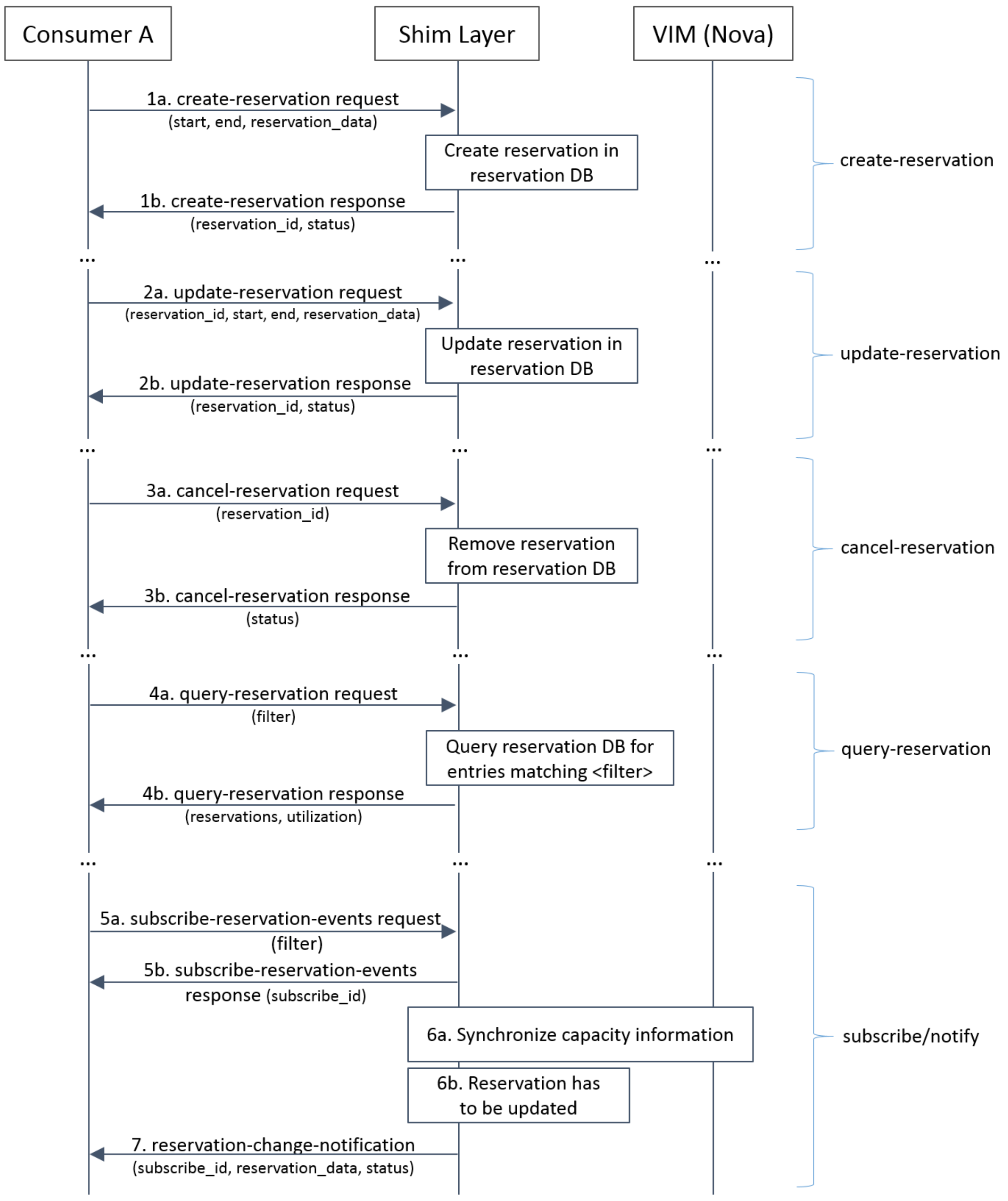

Consumer A
Shim Layer
VIM (Nova)
1a. create-reservation request (start, end, reservation_data)
Create reservation in reservation DB
1b. create-reservation response (reservation_id, status)
create-reservation
2a. update-reservation request (reservation_id, start, end, reservation_data)
Update reservation in reservation DB
2b. update-reservation response (reservation_id, status)
update-reservation
3a. cancel-reservation request (reservation_id)
Remove reservation from reservation DB
3b. cancel-reservation response (status)
cancel-reservation
4a. query-reservation request (filter)
Query reservation DB for entries matching <filter>
4b. query-reservation response (reservations, utilization)
query-reservation
5a. subscribe-reservation-events request (filter)
5b. subscribe-reservation-events response (subscribe_id)
6a. Synchronize capacity information
6b. Reservation has to be updated
7. reservation-change-notification (subscribe_id, reservation_data, status)
subscribe/notify

Fig. 6.2: Resource Reservation for Future Use Scenario

- Step 5a: Consumers can subscribe for reservation-change events using a filter.
- Step 5b: Each successful subscription is responded with a subscription_id.
- Step 6a: Promise synchronizes the available and used capacity with the underlying VIM.
- Step 6b: In certain cases, e.g., due a failure in the underlying hardware, some reservations cannot be kept up anymore and have to be updated or canceled. The shim-layer will identify affected reservations among its reservation records.
- Step 7: Subscribed Consumers will be informed about the updated reservations. The notification contains the updated reservation_data and new status of the reservation. It is then up to the Consumer to take appropriate actions in order to ensure high priority reservations are favored over lower priority reservations.

Resource Allocation

Fig. 6.3 shows a detailed message flow between the Consumer, the functional blocks inside the shim-layer, and the VIM. It has the following steps:

- Step 1a: The Consumer sends a *create-instance* request providing information about the resources to be reserved, i.e., provider_id (optional in case of only one provider), name of the instance, the requested flavour and image, etc. If the allocation is against an existing reservation, the reservation_id has to be provided.
- Step 1b: If a reservation_id was provided, 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 and update its databases. In any other case, Promise rejects the *create-instance* request.
- Step 2: In the case the *create-instance* request was 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. The allocation work flow will terminate at this step and the below steps are not executed.
- Step 3a: If 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.
- Step 3b: The VIM Controller sends an immediate reply with an instance_id and starts the VIM-internal allocation process.
- Step 4: The Consumer gets an immediate response message with allocation status "in progress" and the assigned instance_id.
- Step 5a+b: The consumer subscribes to receive notifications about allocation events related to the requested instance. Promise responds with an acknowledgment including a subscribe_id.
- Step 6: In parallel to the previous step, Promise shim-layer creates an alarm in Aodh to receive notifications about all changes to the VMState for instance_id.
- Step 7a: The VIM Controller notifies all instance related events to Ceilometer. After the allocation has been completed or failed, it sends an event to Ceilometer. This triggers the OpenStack alarming service Aodh to notify the new VMState (e.g. ACTIVE and ERROR) to the shim-layer that updates its internal allocation records.
- Step 7b: Promise sends a notification message to the subscribed Consumer with information on the allocated resources including their new VMState.
- Step 8a+b: Allocated instances can be terminated by the Consumer by sending a *destroy-instance* request to the shim-layer. Promise responds with an acknowledgment and the new status "DELETING" for the instance.
- Step 9a: Promise sends a *deleteServer* request for the instance_id to the VIM Controller.

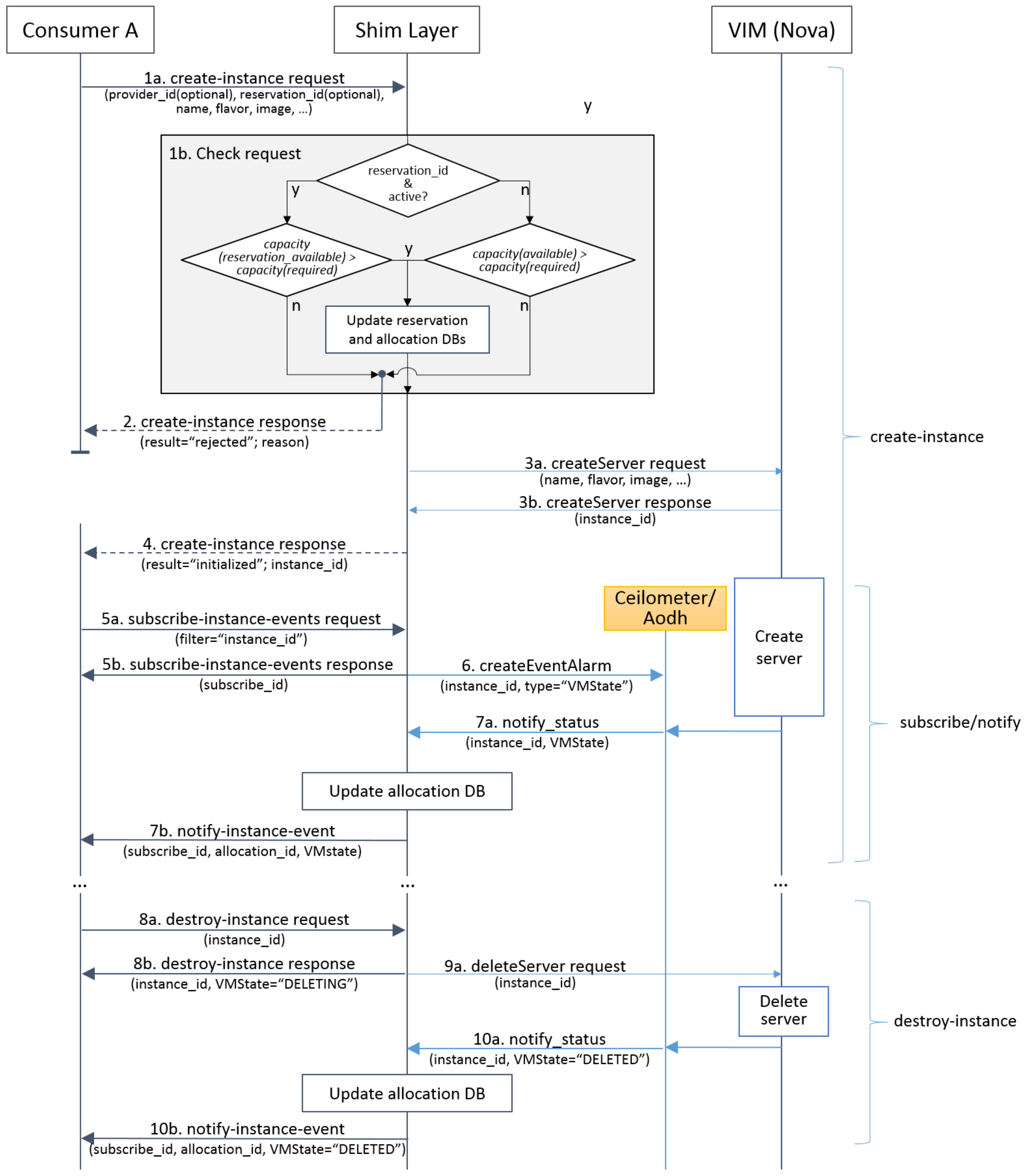

Consumer A
Shim Layer
VIM (Nova)
1a. create-instance request
(provider_id(optional), reservation_id(optional), name, flavor, image, …)
y
1b. Check request
reservation_id & active?
y
n
capacity (reservation_available) > capacity(required)
y
capacity(available) > capacity(required)
n
n
Update reservation and allocation DBs
2. create-instance response
(result="rejected"; reason)
3a. createServer request
(name, flavor, image, …)
3b. createServer response
(instance_id)
4. create-instance response
(result="initialized"; instance_id)
create-instance
Ceilometer/ Aodh
Create server
5a. subscribe-instance-events request
(filter="instance_id")
5b. subscribe-instance-events response
(subscribe_id)
6. createEventAlarm
(instance_id, type="VMState")
7a. notify_status
(instance_id, VMState)
subscribe/notify
Update allocation DB
7b. notify-instance-event
(subscribe_id, allocation_id, VMstate)
…
8a. destroy-instance request
(instance_id)
8b. destroy-instance response
(instance_id, VMState="DELETING")
9a. deleteServer request
(instance_id)
Delete server
destroy-instance
10a. notify_status
(instance_id, VMState="DELETED")
Update allocation DB
10b. notify-instance-event
(subscribe_id, allocation_id, VMState="DELETED")

Fig. 6.3: Resource Allocation

- Step 10a: After the instance has been deleted, an event alarm is sent to the shim-layer that updates its internal allocation records and capacity utilization.
- Step 10b: The shim-layer also notifies the subscribed Consumer about the successfully destroyed instance.

6.1.2 Internal operations

Note: This section is to be updated

In the following, the internal logic and operations of the shim-layer will be explained in more detail, e.g. the "check request" (step 1b in Fig. 6.3 of the allocation work flow).

6.2 Integrated architecture

The *integrated architecture* aims at full integration with OpenStack. This means that it is planned to use the already existing OpenStack APIs extended with the reservation capabilities.

The advantage of this approach is that we don't need to re-model the complex resource structure we have for the virtual machines and the corresponding infrastructure.

The atomic item is the virtual machine with the minimum set of resources it requires to be able to start it up. It is important to state that resource reservation is handled on VM instance level as opposed to standalone resources like CPU, memory and so forth. As the placement is an important aspect in order to be able to use the reserved resources it provides the constraint to handle resources in groups.

The placement constraint also makes it impossible to use a quota management system to solve the base use case described earlier in this document.

OpenStack had a project called Blazar, which was created in order to provide resource reservation functionality in cloud environments. It uses the Shelve API of Nova, which provides a sub-optimal solution. Due to the fact that this feature blocks the reserved resources this solution cannot be considered to be final. Further work is needed to reach a more optimal stage, where the Nova scheduler is intended to be used to schedule the resources for future use to make the reservations.

6.2.1 Phases of the work

The work has two main stages to reach the final solution. The following main work items are on the roadmap for this approach:

1. Sub-optimal solution by using the shelve API of Nova through the Blazar project:
    - Fix the code base of the Blazar project:

      Due to integration difficulties the Blazar project got suspended. Since the last activities in that repository the OpenStack code base and environment changed significantly, which means that the project's code base needs to be updated to the latest standards and has to be able to interact with the latest version of the other OpenStack services.
    - Update the Blazar API:

      The REST API needs to be extended to contain the attributes for the reservation defined in this document. This activity shall include testing towards the new API.
2. Use Nova scheduler to avoid blocking the reserved resources:

- Analyze the Nova scheduler:

  The status and the possible interface between the resource reservation system and the Nova scheduler needs to be identified. It is crucial to achieve a much more optimal solution than what the current version of Blazar can provide. The goal is to be able to use the reserved resources before the reservation starts. In order to be able to achieve this we need the scheduler to do scheduling for the future considering the reservation intervals that are specified in the request.

- Define a new design based on the analysis and start the work on it:

  The design for the more optimal solution can be defined only after analyzing the structure and capabilities of the Nova scheduler.

- This phase can be started in parallel with the previous one.

6.2.2 Detailed Message Flows

Note: to be done

Resource Reservation

Note: to be specified

CHAPTER

SEVEN

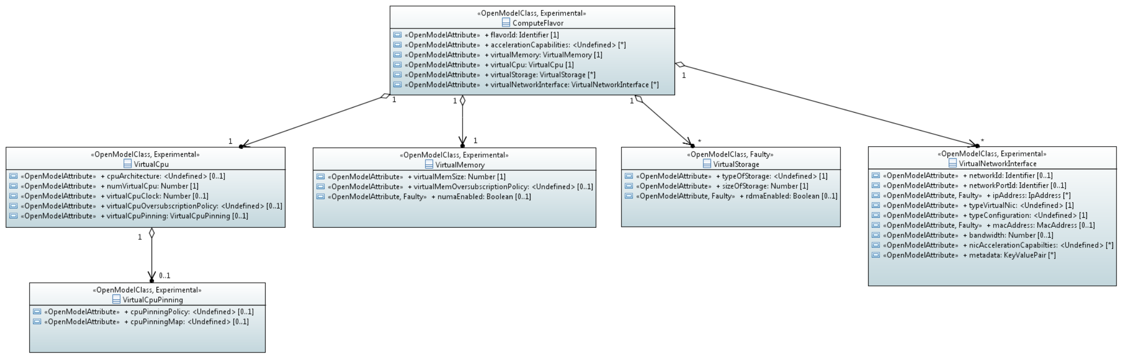
DETAILED NORTHBOUND INTERFACE SPECIFICATION

Note: This is Work in Progress.

7.1 ETSI NFV IFA Information Models

7.1.1 Compute Flavor

A compute flavor includes information about number of virtual CPUs, size of virtual memory, size of virtual storage, and virtual network interfaces *[NFVIFA005]*.


«OpenModelClass, Experimental»
ComputeFlavor
«OpenModelAttribute» + flavorId: Identifier [1]
«OpenModelAttribute» + accelerationCapabilities: <Undefined> [*]
«OpenModelAttribute» + virtualMemory: VirtualMemory [1]
«OpenModelAttribute» + virtualCpu: VirtualCpu [1]
«OpenModelAttribute» + virtualStorage: VirtualStorage [*]
«OpenModelAttribute» + virtualNetworkInterface: VirtualNetworkInterface [*]
1
1
1
1
1
1
*
*
«OpenModelClass, Experimental»
VirtualCpu
«OpenModelAttribute» + cpuArchitecture: <Undefined> [0..1]
«OpenModelAttribute» + numVirtualCpu: Number [1]
«OpenModelAttribute» + virtualCpuClock: Number [0..1]
«OpenModelAttribute» + virtualCpuOversubscriptionPolicy: <Undefined> [0..1]
«OpenModelAttribute» + virtualCpuPinning: VirtualCpuPinning [0..1]
«OpenModelClass, Experimental»
VirtualMemory
«OpenModelAttribute» + virtualMemSize: Number [1]
«OpenModelAttribute» + virtualMemOversubscriptionPolicy: <Undefined> [0..1]
«OpenModelAttribute, Faulty» + numaEnabled: Boolean [0..1]
«OpenModelClass, Faulty»
VirtualStorage
«OpenModelAttribute» + typeOfStorage: <Undefined> [1]
«OpenModelAttribute» + sizeOfStorage: Number [1]
«OpenModelAttribute, Faulty» + rdmaEnabled: Boolean [0..1]
«OpenModelClass, Experimental»
VirtualNetworkInterface
«OpenModelAttribute» + networkId: Identifier [0..1]
«OpenModelAttribute» + networkPortId: Identifier [0..1]
«OpenModelAttribute, Faulty» + ipAddress: IpAddress [*]
«OpenModelAttribute» + typeVirtualNic: <Undefined> [1]
«OpenModelAttribute» + typeConfiguration: <Undefined> [1]
«OpenModelAttribute, Faulty» + macAddress: MacAddress [0..1]
«OpenModelAttribute» + bandwidth: Number [0..1]
«OpenModelAttribute» + nicAccelerationCapabilties: <Undefined> [*]
«OpenModelAttribute» + metadata: KeyValuePair [*]
1
0..1
«OpenModelClass, Experimental»
VirtualCpuPinning
«OpenModelAttribute» + cpuPinningPolicy: <Undefined> [0..1]
«OpenModelAttribute» + cpuPinningMap: <Undefined> [0..1]

7.2 Virtualised Compute Resources

7.2.1 Compute Capacity Management

Subscribe Compute Capacity Change Event

Subscription from Consumer to VIM to be notified about compute capacity changes

POST /capacity/compute/subscribe

Example request:

```
POST /capacity/compute/subscribe HTTP/1.1
Accept: application/json

{
   "zoneId": "12345",
   "computeResourceTypeId": "vcInstances",
   "threshold": {
      "thresholdType" : "absoluteValue",
      "threshold": {
          "capacity_info": "available",
          "condition": "lt",
          "value": 5
      }
   }
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
   "created": "2015-09-21T00:00:00Z",
   "capacityChangeSubscriptionId": "abcdef-ghijkl-123456789"
}
```

Status Codes

- 400 Bad Request – computeResourceTypeId is missing

Query Compute Capacity for a defined resource type

Request to find out about available, reserved, total and allocated compute capacity.

GET /capacity/compute/query

Example request:

```
GET /capacity/compute/query HTTP/1.1
Accept: application/json

{
  "zoneId": "12345",
  "computeResourceTypeId": "vcInstances",
  "timePeriod":  {
       "startTime": "2015-09-21T00:00:00Z",
       "stopTime": "2015-09-21T00:05:30Z"
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "zoneId": "12345",
   "lastUpdate": "2015-09-21T00:03:20Z",
   "capacityInformation": {
      "available": 4,
      "reserved": 17,
      "total": 50,
      "allocated": 29
```

```
    }
}
```

Query Parameters

- **limit** – Default is 10.

Status Codes

- 404 Not Found – resource zone unknown

Query Compute Capacity with required attributes

Request to find out available compute capacity with given characteristics

GET /capacity/compute/query

Example request:

```
GET /capacity/compute/query HTTP/1.1
Accept: application/json

{
  "zoneId": "12345",
  "resourceCriteria":  {
       "virtualCPU": {
           "cpuArchitecture": "x86",
           "numVirtualCpu": 8
       }
  },
  "attributeSelector":  "available",
  "timePeriod":  {
       "startTime": "2015-09-21T00:00:00Z",
       "stopTime": "2015-09-21T00:05:30Z"
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "zoneId": "12345",
   "lastUpdate": "2015-09-21T00:03:20Z",
   "capacityInformation": {
      "available": 50
   }
}
```

Query Parameters

- **limit** – Default is 10.

Status Codes

- 404 Not Found – resource zone unknown

Notify Compute Capacity Change Event

Notification about compute capacity changes

POST /capacity/compute/notification

Example notification:

```
Content-Type: application/json

{
     "zoneId": "12345",
     "notificationId": "zyxwvu-tsrqpo-987654321",
     "capacityChangeTime": "2015-09-21T00:03:20Z",
     "resourceDescriptor": {
        "computeResourceTypeId": "vcInstances"
     },
     "capacityInformation": {
        "available": 4,
        "reserved": 17,
        "total": 50,
        "allocated": 29
     }
}
```

7.2.2 Compute Resource Reservation

Create Compute Resource Reservation

Request the reservation of compute resource capacity

POST /reservation/compute/create

Example request:

```
POST /reservation/compute/create HTTP/1.1
Accept: application/json

{
    "startTime": "2015-09-21T01:00:00Z",
    "computePoolReservation": {
        "numCpuCores": 20,
        "numVcInstances": 5,
        "virtualMemSize": 10
    }
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
   "reservationData": {
      "startTime": "2015-09-21T01:00:00Z",
      "reservationStatus": "initialized",
      "reservationId": "xxxx-yyyy-zzzz",
      "computePoolReserved": {
          "numCpuCores": 20,
          "numVcInstances": 5,
          "virtualMemSize": 10,
          "zoneId": "23456"
      }
   }
}
```

or virtualization containers

POST reservation/compute/create

Example request:

```
POST /reservation/compute/create HTTP/1.1
Accept: application/json

{
  "startTime": "2015-10-05T15:00:00Z",
  "virtualizationContainerReservation": [
    {
       "containerId": "myContainer",
       "containerFlavor": {
          "flavorId": "myFlavor",
          "virtualCpu": {
             "numVirtualCpu": 2,
             "cpuArchitecture": "x86"
          },
          "virtualMemory": {
              "numaEnabled": "False",
              "virtualMemSize": 16
          },
          "storageAttributes": {
              "typeOfStorage": "volume",
              "sizeOfStorage": 16
          }
       }
    }
  ]
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
   "reservationData": {
      "startTime": "2015-10-05T15:00:00Z",
      "reservationId": "aaaa-bbbb-cccc",
      "reservationStatus": "initialized",
      "virtualizationContainerReserved": [
          {
             "containerId": "myContainer",
             "flavorId": "myFlavor",
             "virtualCpu": {
                 "numVirtualCpu": 2,
                 "cpuArchitecture": "x86"
             },
             "virtualMemory": {
                 "numaEnabled": "False",
                 "virtualMemSize": 16
```

```
                  },
                  "virtualDisks": {
                      "storageId": "myStorage",
                      "flavourId": "myStorageFlavour",
                      "typeOfStorage": "volume",
                      "sizeOfStorage": 16,
                      "operationalState": "enabled"
                  }
               }
           ]
       }
   }
```

Query Compute Resource Reservation

Request to find out about reserved compute resources that the consumer has access to.

GET /reservation/compute/query

Example request:

```
GET /reservation/compute/query HTTP/1.1
Accept: application/json

{
   "queryReservationFilter": [
       {
           "reservationId": "xxxx-yyyy-zzzz"
       }
   ]

}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "queryResult":
   {
      "startTime": "2015-09-21T01:00:00Z",
      "reservationStatus": "active",
      "reservationId": "xxxx-yyyy-zzzz",
      "computePoolReserved":
      {
          "numCpuCores": 20,
          "numVcInstances": 5,
          "virtualMemSize": 10,
          "zoneId": "23456"
      }
   }
}
```

Status Codes

- 404 Not Found – reservation id unknown

Update Compute Resource Reservation

Request to update compute resource reservation

POST /reservation/compute/update

Example request:

```
POST /reservation/compute/update HTTP/1.1
Accept: application/json

{
    "startTime": "2015-09-14T16:00:00Z",
    "reservationId": "xxxx-yyyy-zzzz"
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
  "reservationData": {
      "startTime": "2015-09-14TT16:00:00Z",
      "reservationStatus": "active",
      "reservationId": "xxxx-yyyy-zzzz",
      "computePoolReserved": {
          "numCpuCores": 20,
          "numVcInstances": 5,
          "virtualMemSize": 10,
          "zoneId": "23456"
      }
   }
}
```

Terminate Compute Resource Reservation

Request to terminate a compute resource reservation

DELETE /reservation/compute/(*reservation_id*)

Example response:

```
HTTP/1.1 200
Content-Type: application/json

{
   "reservationId": "xxxx-yyyy-zzzz",
}
```

Subscribe Resource Reservation Change Event

Subscription from Consumer to VIM to be notified about changes related to a reservation or to the resources associated to it.

POST /reservation/subscribe

Example request:

```
 POST /reservation/subscribe HTTP/1.1
 Accept: application/json

 {
    "inputFilter": [
        {
           "reservationId": "xxxx-yyyy-zzzz",
        }
    ]
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
   "created": "2015-09-21T00:00:00Z",
   "reservationChangeSubscriptionId": "abcdef-ghijkl-123456789"
}
```

Status Codes

- 400 Bad Request – inputFilter is missing

Notify Resource Reservation Change Event

Notification about changes in a compute resource reservation

`POST /capacity/compute/notification`

Example notification:

```
Content-Type: application/json

{
     "changeId": "aaaaaa-btgxxx-987654321",
     "reservationId": "xxxx-yyyy-zzzz",
     "vimId": "vim-CX-03"
     "changeType": "Reservation time change"
     "changedReservationData": {
        "endTime": "2015-10-14TT16:00:00Z",
     }
}
```

7.3 Virtualised Network Resources

7.3.1 Network Capacity Management

Subscribe Network Capacity Change Event

Susbcription from Consumer to VIM to be notified about network capacity changes

`POST /capacity/network/subscribe`

Example request:

```
POST /capacity/network/subscribe HTTP/1.1
Accept: application/json

{
    "networkResourceTypeId": "publicIps",
    "threshold": {
       "thresholdType": "absoluteValue",
       "threshold": {
           "capacity_info": "available",
           "condition": "lt",
           "value": 5
       }
    }
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
   "created": "2015-09-28T00:00:00Z",
   "capacityChangeSubscriptionId": "bcdefg-hijklm-234567890"
}
```

Query Network Capacity

Request to find out about available, reserved, total and allocated network capacity.

GET /capacity/network/query

Example request:

```
GET /capacity/network/query HTTP/1.1
Accept: application/json

{
    "networkResourceTypeId": "publicIps",
    "timePeriod":  {
        "startTime": "2015-09-28T00:00:00Z",
        "stopTime": "2015-09-28T00:05:30Z"
    }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "lastUpdate": "2015-09-28T00:02:10Z",
    "capacityInformation": {
        "available": 4,
        "reserved": 10,
        "total": 64,
        "allocated": 50
    }
}
```

Notify Network Capacity Change Event

Notification about network capacity changes

POST /capacity/network/notification

Example notification:

```
Content-Type: application/json

{
    "notificationId": "yxwvut-srqpon-876543210",
    "capacityChangeTime": "2015-09-28T00:02:10Z",
    "resourceDescriptor": {
        "networkResourceTypeId": "publicIps"
    },
    "capacityInformation": {
        "available": 4,
        "reserved": 10,
        "total": 64,
        "allocated": 50
    }
}
```

7.3.2 Network Resource Reservation

Create Network Resource Reservation

Request the reservation of network resource capacity and/or virtual networks, network ports

POST /reservation/network/create

Example request:

```
POST /reservation/network/create HTTP/1.1
Accept: application/json

{
    "startTime": "2015-09-28T01:00:00Z",
    "networkReservation": {
        "numPublicIps": 2
    }
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
    "reservationData": {
        "startTime": "2015-09-28T01:00:00Z",
        "reservationStatus": "initialized",
        "reservationId": "wwww-xxxx-yyyy",
        "publicIps": [
            "10.2.91.60",
            "10.2.91.61"
        ]
    }
}
```

Query Network Resource Reservation

Request to find out about reserved network resources that the consumer has access to.

GET /reservation/network/query

Example request:

```
GET /reservation/network/query HTTP/1.1
Accept: application/json

{
    "queryReservationFilter": [
        {
            "reservationId": "wwww-xxxx-yyyy"
        }
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "queryResult": {
        "startTime": "2015-09-28T01:00:00Z",
        "reservationStatus": "active",
        "reservationId": "wwww-xxxx-yyyy",
        "publicIps": [
            "10.2.91.60",
            "10.2.91.61"
        ]
    }
}
```

Update Network Resource Reservation

Request to update network resource reservation

POST /reservation/network/update

Example request:

```
POST /reservation/network/update HTTP/1.1
Accept: application/json

{
    "startTime": "2015-09-21T16:00:00Z",
    "reservationId": "wwww-xxxx-yyyy"
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
    "reservationData": {
        "startTime": "2015-09-21T16:00:00Z",
        "reservationStatus": "active",
        "reservationId": "wwww-xxxx-yyyy",
        "publicIps": [
           "10.2.91.60",
           "10.2.91.61"
        ]
    }
}
```

Terminate Network Resource Reservation

Request to terminate a network resource reservation

DELETE /reservation/network/(*reservation_id*)

Example response:

```
HTTP/1.1 200
Content-Type: application/json

{
   "reservationId": "xxxx-yyyy-zzzz",
}
```

7.4 Virtualised Storage Resources

7.4.1 Storage Capacity Management

Subscribe Storage Capacity Change Event

Subscription from Consumer to VIM to be notified about storage capacity changes

POST /capacity/storage/subscribe

Example request:

```
POST /capacity/storage/subscribe HTTP/1.1
Accept: application/json

{
   "storageResourceTypeId": "volumes",
   "threshold": {
      "thresholdType": "absoluteValue",
      "threshold": {
          "capacity_info": "available",
          "condition": "lt",
          "value": 3
       }
   }
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
    "created": "2015-09-28T12:00:00Z",
    "capacityChangeSubscriptionId": "cdefgh-ijklmn-345678901"
}
```

Query Storage Capacity for a defined resource type

Request to find out about available, reserved, total and allocated storage capacity.

GET /capacity/storage/query

Example request:

```
    GET /capacity/storage/query HTTP/1.1
    Accept: application/json

    {
        "storageResourceTypeId": "volumes",
        "timePeriod":  {
            "startTime": "2015-09-28T12:00:00Z",
            "stopTime": "2015-09-28T12:04:45Z"
        }
    }
```

Example response:

```
    HTTP/1.1 200 OK
    Content-Type: application/json

    {
        "lastUpdate": "2015-09-28T12:01:35Z",
        "capacityInformation": {
            "available": 2,
            "reserved": 4,
            "total": 10,
            "allocated": 4
        }
    }
```

Query Storage Capacity with required attributes

Request to find out available capacity.

GET /capacity/storage/query

Example request:

```
    GET /capacity/storage/query HTTP/1.1
    Accept: application/json

    {
        "resourceCriteria": {
            "typeOfStorage" : "volume",
            "sizeOfStorage" : 200,
            "rdmaSupported" : "True",
        },
        "attributeSelector": "available",
        "timePeriod":  {
            "startTime": "2015-09-28T12:00:00Z",
            "stopTime": "2015-09-28T12:04:45Z"
        }
    }
```

Example response:

```
    HTTP/1.1 200 OK
    Content-Type: application/json

    {
        "lastUpdate": "2015-09-28T12:01:35Z",
        "capacityInformation": {
            "available": 2
        }
    }
```

Notify Storage Capacity Change Event

Notification about storage capacity changes

POST /capacity/storage/notification

Example notification:

```
     Content-Type: application/json

     {
         "notificationId": "xwvuts-rqponm-765432109",
         "capacityChangeTime": "2015-09-28T12:01:35Z",
         "resourceDescriptor": {
             "storageResourceTypeId": "volumes"
         },
         "capacityInformation": {
             "available": 2,
             "reserved": 4,
             "total": 10,
             "allocated": 4
         }
    }
```

7.4.2 Storage Resource Reservation

Create Storage Resource Reservation

Request the reservation of storage resource capacity

POST /reservation/storage/create

Example request:

```
    POST /reservation/storage/create HTTP/1.1
    Accept: application/json

    {
        "startTime": "2015-09-28T13:00:00Z",
        "storagePoolReservation": {
            "storageSize": 10,
            "numSnapshots": 3,
            "numVolumes": 2
        }
    }
```

Example response:

```
    HTTP/1.1 201 CREATED
    Content-Type: application/json

    {
        "reservationData": {
            "startTime": "2015-09-28T13:00:00Z",
            "reservationStatus": "initialized",
            "reservationId": "vvvv-wwww-xxxx",
            "storagePoolReserved": {
                "storageSize": 10,
                "numSnapshots": 3,
                "numVolumes": 2
            }
        }
    }
```

Query Storage Resource Reservation

Request to find out about reserved storage resources that the consumer has access to.

`GET /reservation/storage/query`

Example request:

```
    GET /reservation/storage/query HTTP/1.1
    Accept: application/json

    {
        "queryReservationFilter": [
            {
                "reservationId": "vvvv-wwww-xxxx"
            }
        ]
    }
```

Example response:

```
    HTTP/1.1 200 OK
    Content-Type: application/json

    {
        "queryResult": {
            "startTime": "2015-09-28T13:00:00Z",
            "reservationStatus": "active",
            "reservationId": "vvvv-wwww-xxxx",
            "storagePoolReserved": {
                "storageSize": 10,
                "numSnapshots": 3,
                "numVolumes": 2
            }
        }
    }
```

Update Storage Resource Reservation

Request to update storage resource reservation

POST /reservation/storage/update

Example request:

```
POST /reservation/storage/update HTTP/1.1
Accept: application/json


{
    "startTime": "2015-09-20T23:00:00Z",
    "reservationId": "vvvv-wwww-xxxx"

}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
    "reservationData": {
        "startTime": "2015-09-20T23:00:00Z",
        "reservationStatus": "active",
        "reservationId": "vvvv-wwww-xxxx",
        "storagePoolReserved": {
            "storageSize": 10,
            "numSnapshots": 3,
            "numVolumes": 2
        }
    }
}
```

Terminate Storage Resource Reservation

Request to terminate a storage resource reservation

DELETE /reservation/storage/(*reservation_id*)

Example response:

```
HTTP/1.1 200
Content-Type: application/json

{
   "reservationId": "xxxx-yyyy-zzzz",
}
```

CHAPTER

EIGHT

SUMMARY AND CONCLUSION

Resource Reservation and Resource Capacity Management are features to be supported by the VIM and exposed to the consumer via the VIM NBI. These features have been specified by ETSI NFV.

This document has described several use cases and corresponding high level flows where Resource Reservation and Capacity Management are of great benefit for the consumer of the virtualised resource management interface: the NFVO or the VNFM. The use cases include:

- Notification of changes in capacity in the NFVI
- Query of available resource capacity
- Reservation of a resource or set of resources for immediate use
- Reservation of a resource or set of resources for future use

The Promise project has performed a gap analysis in order to fulfill the required functionality. Based on the gap analysis an implementation plan and way forward has been proposed, including a possible design architecture and high level information model. Immediate next steps of this project is to deliver a working Proof-of-Concepts (PoC) and engage upstream communities to fill out the gaps identified by Promise.

CHAPTER

NINE

REFERENCES AND BIBLIOGRAPHY

CHAPTER

TEN

ANNEX A: USE CASE FOR OPNFV BRAHMAPUTRA

A basic resource reservation use case to be realized for OPNFV B-release may look as follows:

- Step 0: Shim-layer is monitoring/querying available capacity at NFVI
  - Step 0a: Cloud operator creates a new OpenStack tenant user and updates quota values for this user
  - Step 0b: The tenant user is creating and instantiating a simple VNF (e.g. 1 network, 2 VMs)
  - Step 0c: OpenStack is notifying shim-layer about capacity change for this new tenant user
  - Step 0d: Cloud operator can visualize the changes using the GUI
- Step 1: Consumer(NFVO) is sending a reservation request for future use to shim-layer
- Step 2: Shim-layer is checking for available capacity at the given time window
- Step 3: Shim-layer is responding with reservation identifier
- Step 4 (optional): Consumer(NFVO) is sending an update reservation request to shim-layer (startTime set to now) -> continue with Steps 2 and 3.
- Step 5: Consumer(VNFM) is requesting the allocation of virtualised resources using the reservation identifier in Step 3

CHAPTER ELEVEN

ANNEX B: PROMISE YANG SCHEMA BASED ON YANGFORGE

```
module opnfv-promise {
namespace "urn:opnfv:promise";
prefix promise;

import complex-types { prefix ct; }
import ietf-yang-types { prefix yang; }
import ietf-inet-types { prefix inet; }
import access-control-models { prefix acm; }
import nfv-infrastructure { prefix nfvi; }
import nfv-mano { prefix mano; }

description
  "OPNFV Promise Resource Reservation/Allocation controller module";

revision 2015-10-05 {
  description "Complete coverage of reservation related intents";
}

revision 2015-08-06 {
  description "Updated to incorporate YangForge framework";
}

revision 2015-04-16 {
  description "Initial revision.";
}

feature reservation-service {
  description "When enabled, provides resource reservation service";
}

feature multi-provider {
  description "When enabled, provides resource management across multiple providers";
}

typedef reference-identifier {
  description "defines valid formats for external reference id";
  type union {
    type yang:uuid;
    type inet:uri;
    type uint32;
  }
}

grouping resource-utilization {
  container capacity {
    container total     { description 'Conceptual container that should be extended'; }
    container reserved  { description 'Conceptual container that should be extended';
                          config false; }
    container usage     { description 'Conceptual container that should be extended';
                          config false; }
    container available { description 'Conceptual container that should be extended';
                          config false; }
  }
}

grouping temporal-resource-collection {
  description
    "Information model capturing resource-collection with start/end time window";

  leaf start { type yang:date-and-time; }
  leaf end   { type yang:date-and-time; }

  uses nfvi:resource-collection;
}

grouping resource-usage-request {
  description
    "Information model capturing available parameters to make a resource
     usage request.";
  reference "OPNFV-PROMISE, Section 3.4.1";

  uses temporal-resource-collection {
    refine elements {
      description
        "Reference to a list of 'pre-existing' resource elements that are
       required for fulfillment of the resource-usage-request.

       It can contain any instance derived from ResourceElement,
       such as ServerInstances or even other
       ResourceReservations. If the resource-usage-request is
       accepted, the ResourceElement(s) listed here will be placed
       into 'protected' mode as to prevent accidental removal.

       If any of these resource elements become 'unavailable' due to
       environmental or administrative activity, a notification will
       be issued informing of the issue.";
    }
  }

  leaf zone {
    description "Optional identifier to an Availability Zone";
    type instance-identifier { ct:instance-type nfvi:AvailabilityZone; }
  }
}

grouping query-start-end-window {
  container window {
    description "Matches entries that are within the specified start/end time window";
    leaf start { type yang:date-and-time; }
    leaf end   { type yang:date-and-time; }
    leaf scope {
      type enumeration {
        enum "exclusive" {
          description "Matches entries that start AND end within the window";
        }
        enum "inclusive" {
          description "Matches entries that start OR end within the window";
        }
      }
      default "inclusive";
    }
  }
}

grouping query-resource-collection {
  uses query-start-end-window {
    description "Match for ResourceCollection(s) that are within the specified
                 start/end time window";
  }
  leaf-list without {
    description "Excludes specified collection identifiers from the result";
    type instance-identifier { ct:instance-type ResourceCollection; }
  }
  leaf show-utilization { type boolean; default true; }
  container elements {
    leaf-list some {
      description "Query for ResourceCollection(s) that contain some or more of
                   these element(s)";
      type instance-identifier { ct:instance-type nfvi:ResourceElement; }
    }
    leaf-list every {
      description "Query for ResourceCollection(s) that contain all of
                   these element(s)";
      type instance-identifier { ct:instance-type nfvi:ResourceElement; }
    }
  }
}

grouping common-intent-output {
  leaf result {
    type enumeration {
      enum "ok";
      enum "conflict";
      enum "error";
    }
  }
  leaf message { type string; }
}

grouping utilization-output {
  list utilization {
    key 'timestamp';
    leaf timestamp { type yang:date-and-time; }
    leaf count { type int16; }
    container capacity { uses nfvi:resource-capacity; }
  }
}

ct:complex-type ResourceCollection {
  ct:extends nfvi:ResourceContainer;
  ct:abstract true;

  description
    "Describes an abstract ResourceCollection data model, which represents
     a grouping of capacity and elements available during a given
     window in time which must be extended by other resource
     collection related models";

  leaf start { type yang:date-and-time; }
  leaf end   { type yang:date-and-time; }

  leaf active {
    config false;
    description
      "Provides current state of this record whether it is enabled and within
       specified start/end time";
    type boolean;
  }
}

ct:complex-type ResourcePool {
  ct:extends ResourceCollection;

  description
    "Describes an instance of an active ResourcePool record, which
     represents total available capacity and elements from a given
     source.";

  leaf source {
    type instance-identifier {
      ct:instance-type nfvi:ResourceContainer;
      require-instance true;
    }
    mandatory true;
  }

  refine elements {
    // following 'must' statement applies to each element
    // NOTE: just a non-working example for now...
    must "boolean(/source/elements/*[@id=id])" {
      error-message "One or more of the ResourceElement(s) does not exist in
                     the provider to be reserved";
    }
  }
}

ct:complex-type ResourceReservation {
  ct:extends ResourceCollection;

  description
    "Describes an instance of an accepted resource reservation request,
     created usually as a result of 'create-reservation' request.

     A ResourceReservation is a derived instance of a generic
     ResourceCollection which has additional parameters to map the
     pool(s) that were referenced to accept this reservation as well
     as to track allocations made referencing this reservation.
     Contains the capacities of various resource attributes being
     reserved along with any resource elements that are needed to be
     available at the time of allocation(s).";

  reference "OPNFV-PROMISE, Section 3.4.1";

  leaf created-on  { type yang:date-and-time; config false; }
  leaf modified-on { type yang:date-and-time; config false; }

  leaf-list pools {
    config false;
    description
      "Provides list of one or more pools that were referenced for providing
       the requested resources for this reservation.  This is an
       important parameter for informing how/where allocation
       requests can be issued using this reservation since it is
       likely that the total reserved resource capacity/elements are
       made availble from multiple sources.";
    type instance-identifier {
      ct:instance-type ResourcePool;
      require-instance true;
    }
  }

  container remaining {
    config false;
    description
      "Provides visibility into total remaining capacity for this
       reservation based on allocations that took effect utilizing
       this reservation ID as a reference.";

    uses nfvi:resource-capacity;
  }

  leaf-list allocations {
    config false;
    description
      "Reference to a collection of consumed allocations referencing
       this reservation.";
    type instance-identifier {
      ct:instance-type ResourceAllocation;
      require-instance true;
    }
  }
}

ct:complex-type ResourceAllocation {
  ct:extends ResourceCollection;

  description
    "A ResourceAllocation record denotes consumption of resources from a
     referenced ResourcePool.

     It does not reflect an accepted request but is created to
     represent the actual state about the ResourcePool. It is
     created once the allocation(s) have successfully taken effect
     on the 'source' of the ResourcePool.
     The 'priority' state indicates the classification for dealing
     with resource starvation scenarios. Lower priority allocations
     will be forcefully terminated to allow for higher priority
     allocations to be fulfilled.

     Allocations without reference to an existing reservation will
     receive the lowest priority.";

  reference "OPNFV-PROMISE, Section 3.4.3";

  leaf reservation {
    description "Reference to an existing reservation identifier (optional)";

    type instance-identifier {
      ct:instance-type ResourceReservation;
      require-instance true;
    }
  }

  leaf pool {
    description "Reference to an existing resource pool from which allocation is drawn";

    type instance-identifier {
      ct:instance-type ResourcePool;
      require-instance true;
    }
  }

  container instance-ref {
    config false;
    description
      "Reference to actual instance identifier of the provider/server
      for this allocation";
    leaf provider {
      type instance-identifier { ct:instance-type ResourceProvider; }
    }
    leaf server { type yang:uuid; }
  }

  leaf priority {
    config false;
    description
      "Reflects current priority level of the allocation according to
       classification rules";
    type enumeration {
      enum "high"   { value 1; }
      enum "normal" { value 2; }
      enum "low"    { value 3; }
    }
    default "normal";
  }
}

ct:complex-type ResourceProvider {
  ct:extends nfvi:ResourceContainer;

  key "name";
  leaf token { type string; mandatory true; }
  container services { // read-only
    config false;
    container compute {
      leaf endpoint { type inet:uri; }
      ct:instance-list flavors { ct:instance-type nfvi:ComputeFlavor; }
    }
  }

  leaf-list pools {
    config false;
    description
      "Provides list of one or more pools that are referencing this provider.";

    type instance-identifier {
      ct:instance-type ResourcePool;
      require-instance true;
    }
  }
}

// MAIN CONTAINER
container promise {

  uses resource-utilization {
    description "Describes current state info about capacity utilization info";

    augment "capacity/total"     { uses nfvi:resource-capacity; }
    augment "capacity/reserved"  { uses nfvi:resource-capacity; }
    augment "capacity/usage"     { uses nfvi:resource-capacity; }
    augment "capacity/available" { uses nfvi:resource-capacity; }
  }

  ct:instance-list providers {
    if-feature multi-provider;
    description "Aggregate collection of all registered ResourceProvider instances
                 for Promise resource management service";
    ct:instance-type ResourceProvider;
  }

  ct:instance-list pools {
    if-feature reservation-service;
    description "Aggregate collection of all ResourcePool instances";
    ct:instance-type ResourcePool;
  }

  ct:instance-list reservations {
    if-feature reservation-service;
    description "Aggregate collection of all ResourceReservation instances";
    ct:instance-type ResourceReservation;
  }

  ct:instance-list allocations {
    description "Aggregate collection of all ResourceAllocation instances";
    ct:instance-type ResourceAllocation;
  }

  container policy {
    container reservation {
      leaf max-future-start-range {
        description
          "Enforce reservation request 'start' time is within allowed range from now";
        type uint16 { range 0..365; }
        units "days";
      }
      leaf max-future-end-range {
        description
          "Enforce reservation request 'end' time is within allowed range from now";
        type uint16 { range 0..365; }
        units "days";
      }
      leaf max-duration {
        description
          "Enforce reservation duration (end-start) does not exceed specified threshold";
        type uint16;
        units "hours";
        default 8760; // for now cap it at max one year as default
      }
      leaf expiry {
        description
          "Duration in minutes from start when unallocated reserved resources
           will be released back into the pool";
        type uint32;
        units "minutes";
      }
    }
  }
}

//-------------------
// INTENT INTERFACE
//-------------------

// RESERVATION INTENTS
rpc create-reservation {
  if-feature reservation-service;
  description "Make a request to the reservation system to reserve resources";
  input {
    uses resource-usage-request;
  }
  output {
    uses common-intent-output;
    leaf reservation-id {
      type instance-identifier { ct:instance-type ResourceReservation; }
    }
  }
}

rpc update-reservation {
  description "Update reservation details for an existing reservation";
  input {
    leaf reservation-id {
      type instance-identifier {
        ct:instance-type ResourceReservation;
        require-instance true;
      }
      mandatory true;
    }
    uses resource-usage-request;
  }
  output {
    uses common-intent-output;
  }
}

rpc cancel-reservation {
  description "Cancel the reservation and be a good steward";
  input {
    leaf reservation-id {
      type instance-identifier { ct:instance-type ResourceReservation; }
      mandatory true;
    }
  }
  output {
    uses common-intent-output;
  }
}

rpc query-reservation {
  if-feature reservation-service;
  description "Query the reservation system to return matching reservation(s)";
  input {
    leaf zone { type instance-identifier { ct:instance-type nfvi:AvailabilityZone; } }
    uses query-resource-collection;
  }
  output {
    leaf-list reservations { type instance-identifier
                             { ct:instance-type ResourceReservation; } }
    uses utilization-output;
  }
}

// CAPACITY INTENTS
rpc increase-capacity {
  description "Increase total capacity for the reservation system
               between a window in time";
  input {
    uses temporal-resource-collection;
    leaf source {
      type instance-identifier {
        ct:instance-type nfvi:ResourceContainer;
      }
    }
  }
  output {
    uses common-intent-output;
    leaf pool-id {
      type instance-identifier { ct:instance-type ResourcePool; }
    }
  }
}

rpc decrease-capacity {
  description "Decrease total capacity for the reservation system
               between a window in time";
  input {
    uses temporal-resource-collection;
    leaf source {
      type instance-identifier {
        ct:instance-type nfvi:ResourceContainer;
      }
    }
  }
  output {
    uses common-intent-output;
    leaf pool-id {
      type instance-identifier { ct:instance-type ResourcePool; }
    }
  }
}

rpc query-capacity {
  description "Check available capacity information about a specified
               resource collection";
  input {
    leaf capacity {
      type enumeration {
        enum 'total';
        enum 'reserved';
        enum 'usage';
        enum 'available';
      }
      default 'available';
    }
    leaf zone { type instance-identifier { ct:instance-type nfvi:AvailabilityZone; } }
    uses query-resource-collection;
    // TBD: additional parameters for query-capacity
  }
  output {
    leaf-list collections { type instance-identifier
                            { ct:instance-type ResourceCollection; } }
    uses utilization-output;
  }
}

// ALLOCATION INTENTS (should go into VIM module in the future)
rpc create-instance {
  description "Create an instance of specified resource(s) utilizing capacity
               from the pool";
  input {
    leaf provider-id {
      if-feature multi-provider;
      type instance-identifier { ct:instance-type ResourceProvider;
                                 require-instance true; }
    }
    leaf name   { type string; mandatory true; }
    leaf image  {
      type reference-identifier;
      mandatory true;
    }
    leaf flavor {
      type reference-identifier;
      mandatory true;
    }
    leaf-list networks {
      type reference-identifier;
      description "optional, will assign default network if not provided";
    }

    // TODO: consider supporting a template-id (such as HEAT) for more complex instantiation

    leaf reservation-id {
      type instance-identifier { ct:instance-type ResourceReservation;
                                 require-instance true; }
    }
  }
  output {
    uses common-intent-output;
    leaf instance-id {
      type instance-identifier { ct:instance-type ResourceAllocation; }
    }
  }
}

rpc destroy-instance {
  description "Destroy an instance of resource utilization and release it
               back to the pool";
  input {
    leaf instance-id {
      type instance-identifier { ct:instance-type ResourceAllocation;
                                 require-instance true; }
    }
  }
  output {
    uses common-intent-output;
  }
}

// PROVIDER INTENTS (should go into VIM module in the future)
rpc add-provider {
  description "Register a new resource provider into reservation system";
  input {
    leaf provider-type {
      description "Select a specific resource provider type";
      mandatory true;
      type enumeration {
        enum openstack;
        enum hp;
        enum rackspace;
        enum amazon {
          status planned;
        }
        enum joyent {
          status planned;
        }
        enum azure {
          status planned;
        }
      }
      default openstack;
    }
    uses mano:provider-credentials {
      refine endpoint {
        default "http://localhost:5000/v2.0/tokens";
      }
    }
    container tenant {
      leaf id { type string; }
      leaf name { type string; }
    }
  }
  output {
    uses common-intent-output;
    leaf provider-id {
      type instance-identifier { ct:instance-type ResourceProvider; }
    }
  }
}

// TODO...
notification reservation-event;
notification capacity-event;
notification allocation-event;
}
```

CHAPTER

TWELVE

ANNEX C: SUPPORTED APIS

12.1 Add Provider

Register a new resource provider (e.g. OpenStack) into reservation system.

Request parameters

Name	Type	Description
provider-type	Enumeration	Name of the resource provider
endpoint	URI	Targer URL end point for the resource provider
username	String	User name
password	String	Password
region	String	Specified region for the provider
tenant.id	String	Id of the tenant
tenant.name	String	Name of the tenant

Response parameters

Name	Type	Description
provider-id	String	Id of the new resource provider
result	Enumeration	Result info

POST /add-provider

Example request:

```
POST /add-provider HTTP/1.1
Accept: application/json

{
  "provider-type": "openstack",
  "endpoint": "http://10.0.2.15:5000/v2.0/tokens",
  "username": "promise_user",
  "password": "******",
  "tenant": {
     "name": "promise"
  }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "provider-id": "f25ed9cb-de57-43d5-9b4a-a389a1397302",
    "result": "ok"
}
```

12.2 Create Reservation

Make a request to the reservation system to reserve resources.

Request parameters

Name	Type	Description
zone	String	Id to an availability zone
start	DateTime	Timestamp when the consumption of reserved resources can begin
end	DateTime	Timestamp when the consumption of reserved resources should end
capacity.cores	int16	Amount of cores to be reserved
capacity.ram	int32	Amount of RAM to be reserved
capacity.instances	int16	Amount of instances to be reserved
capacity.addresses	int32	Amount of public IP addresses to be reserved
elements	ResourceElement	List of pre-existing resource elements to be reserved

Response parameters

Name	Type	Description
reservation-id	String	Id of the reservation
result	Enumeration	Result info
message	String	Output message

POST /create-reservation

Example request:

```
POST /create-reservation HTTP/1.1
Accept: application/json

{
   "capacity": {
      "cores": "5",
      "ram": "25600",
      "addresses": "3",
      "instances": "3"
   },
   "start": "2016-02-02T00:00:00Z",
   "end": "2016-02-03T00:00:00Z"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "reservation-id": "269b2944-9efc-41e0-b067-6898221e8619",
  "result": "ok",
  "message": "reservation request accepted"
}
```

12.3 Update Reservation

Update reservation details for an existing reservation.

Request parameters

Name	Type	Description
reservation-id	String	Id of the reservation to be updated
zone	String	Id to an availability zone
start	DateTime	Updated timestamp when the consumption of reserved resources can begin
end	DateTime	Updated timestamp when the consumption of reserved resources should end
capacity.cores	int16	Updated amount of cores to be reserved
capacity.ram	int32	Updated amount of RAM to be reserved
capacity.instances	int16	Updated amount of instances to be reserved
capacity.addresses	int32	Updated amount of public IP addresses to be reserved
elements	ResourceElement	Updated list of pre-existing resource elements to be reserved

Response parameters

Name	Type	Description
result	Enumeration	Result info
message	String	Output message

POST /update-reservation

Example request:

```
POST /update-reservation HTTP/1.1
Accept: application/json

{
   "reservation-id": "269b2944-9efv-41e0-b067-6898221e8619",
   "capacity": {
      "cores": "1",
      "ram": "5120",
      "addresses": "1",
      "instances": "1"
   }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "result": "ok",
  "message": "reservation update successful"
}
```

12.4 Cancel Reservation

Cancel the reservation.

Request parameters

Name	Type	Description
reservation-id	String	Id of the reservation to be canceled

Response parameters

Name	Type	Description
result	Enumeration	Result info
message	String	Output message

POST /cancel-reservation

Example request:

```
POST /cancel-reservation HTTP/1.1
Accept: application/json

{
  "reservation-id": "269b2944-9efv-41e0-b067-6898221e8619"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "result": "ok",
  "message": "reservation canceled"
}
```

12.5 Query Reservation

Query the reservation system to return matching reservation(s).

Request parameters

Name	Type	Description
zone	String	Id to an availability zone
show-utilization	Boolean	Show capacity utilization
without	ResourceCollection	Excludes specified collection identifiers from the result
elements.some	ResourceElement	Query for ResourceCollection(s) that contain some or more of these element(s)
elements.every	ResourceElement	Query for ResourceCollection(s) that contain all of these element(s)
window.start	DateTime	Matches entries that are within the specified start/end window
window.end	DateTime	
wndow.scope	Enumeration	Matches entries that start {and/or} end within the time window

Response parameters

Name	Type	Description
reservations	ResourceReservation	List of matching reservations
utilization	CapacityUtilization	Capacity utilization over time

POST /query-reservation

Example request:

```
POST /query-reservation HTTP/1.1
Accept: application/json

{
   "show-utilization": false,
   "window": {
      "start": "2016-02-01T00:00:00Z",
      "end": "2016-02-04T00:00:00Z"
   }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "reservations": [
    "269b2944-9efv-41e0-b067-6898221e8619"
  ],
  "utilization": []
}
```

12.6 Create Instance

Create an instance of specified resource(s) utilizing capacity from the pool.

Request parameters

Name	Type	Description
provider-id	String	Id of the resource provider
reservation-id	String	Id of the resource reservation
name	String	Name of the instance
image	String	Id of the image
flavor	String	Id of the flavor
networks	Uuid	List of network uuids

Response parameters

Name	Type	Description
instance-id	String	Id of the instance
result	Enumeration	Result info
message	String	Output message

POST /create-instance

Example request:

```
POST /create-instance HTTP/1.1
Accept: application/json

{
  "provider-id": "f25ed9cb-de57-43d5-9b4a-a389a1397302",
  "name": "vm1",
  "image": "ddffc6f5-5c86-4126-b0fb-2c71678633f8",
  "flavor": "91bfdf57-863b-4b73-9d93-fc311894b902"
}
```

Example response:

```
  HTTP/1.1 200 OK
  Content-Type: application/json

  {
    "instance-id": "82572779-896b-493f-92f6-a63008868250",
    "result": "ok",
    "message": "created-instance request accepted"
  }
```

12.7 Destroy Instance

Destroy an instance of resource utilization and release it back to the pool.

Request parameters

Name	Type	Description
instance-id	String	Id of the instance to be destroyed

Response parameters

Name	Type	Description
result	Enumeration	Result info
message	String	Output message

POST /destroy-instance

Example request:

```
  POST /destroy-instance HTTP/1.1
  Accept: application/json

  {
     "instance-id": "82572779-896b-493f-92f6-a63008868250"
  }
```

Example response:

```
  HTTP/1.1 200 OK
  Content-Type: application/json

  {
    "result": "ok",
    "message": "instance destroyed and resource released back to pool"
  }
```

12.8 Decrease Capacity

Decrease total capacity for the reservation system for a given time window.

Request parameters

Name	Type	Description
source	String	Id of the resource container
start	DateTime	Start/end defines the time window when total capacity is decreased
end	DateTime	
capacity.cores	int16	Decreased amount of cores
capacity.ram	int32	Decreased amount of RAM
capacity.instances	int16	Decreased amount of instances
capacity.addresses	int32	Decreased amount of public IP addresses

Response parameters

Name	Type	Description
pool-id	String	Id of the resource pool
result	Enumeration	Result info
message	String	Output message

POST /decrease-capacity

Example request:

```
POST /decrease-capacity HTTP/1.1
Accept: application/json

{
   "source": "ResourcePool:4085f0da-8030-4252-a0ff-c6f93870eb5f",
   "capacity": {
      "cores": "3",
      "ram": "5120",
      "addresses": "1"
   }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "pool-id": "c63b2a41-bcc6-42f6-8254-89d633e1bd0b",
   "result": "ok",
   "message": "capacity decrease successful"
}
```

12.9 Increase Capacity

Increase total capacity for the reservation system for a given time window.

Request parameters

Name	Type	Description
source	String	Id of the resource container
start	DateTime	Start/end defines the time window when total capacity is increased
end	DateTime	
capacity.cores	int16	Increased amount of cores
capacity.ram	int32	Increased amount of RAM
capacity.instances	int16	Increased amount of instances
capacity.addresses	int32	Increased amount of public IP addresses

Response parameters

Name	Type	Description
pool-id	String	Id of the resource pool
result	Enumeration	Result info
message	String	Output message

POST /increase-capacity

Example request:

```
POST /increase-capacity HTTP/1.1
Accept: application/json

{
   "source": "ResourceProvider:f6f13fe3-0126-4c6d-a84f-15f1ab685c4f",
   "capacity": {
       "cores": "20",
       "ram": "51200",
       "instances": "10",
       "addresses": "10"
   }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "pool-id": "279217a4-7461-4176-bf9d-66770574ca6a",
   "result": "ok",
   "message": "capacity increase successful"
}
```

12.10 Query Capacity

Query for capacity information about a specified resource collection.

Request parameters

Name	Type	Description
capacity	Enumeration	Return total or reserved or available or usage capacity information
zone	String	Id to an availability zone
show-utilization	Boolean	Show capacity utilization
without	ResourceCollection	Excludes specified collection identifiers from the result
elements.some	ResourceElement	Query for ResourceCollection(s) that contain some or more of these element(s)
elements.every	ResourceElement	Query for ResourceCollection(s) that contain all of these element(s)
window.start	DateTime	Matches entries that are within the specified start/end window
window.end	DateTime	
window.scope	Enumeration	Matches entries that start {and/or} end within the time window

Response parameters

Name	Type	Description
collections	ResourceCollection	List of matching collections
utilization	CapacityUtilization	Capacity utilization over time

POST /query-capacity

Example request:

```
POST /query-capacity HTTP/1.1
Accept: application/json

{
  "show-utilization": false
}
```

Example response:

```
HTTP/1.1 201 CREATED
Content-Type: application/json

{
  "collections": [
    "ResourcePool:279217a4-7461-4176-bf9d-66770574ca6a"
  ],
  "utilization": []
}
```

ANNEX D: DOCUMENT REVISION

Version	Description
1.0.1	**JIRA: PROMISE-23** • Reference to YangForge Framework • Corrections to figure 3.1
1.0.2	**JIRA: PROMISE-11** • OPNFV Logo Added • Alignment to ETSI NFV Specs
1.0.3	**JIRA: PROMISE-54** • Use case for Brahmaputra
1.0.4	**JIRA: PROMISE-60** • Editorial fixes **JIRA: PROMISE-57** • Split shim-layer architecture and integrated architecture sections
1.0.5	**JIRA: PROMISE-61** • Updated Promise Yang Schema
1.0.6	**JIRA: PROMISE-62** • Supported APIs for Brahmaputra
1.0.7	**JIRA: PROMISE-63** • Update message flow for shim-layer **JIRA: PROMISE-58** • Integrated approach description **JIRA: PROMISE-64** • Promise userguide
1.0.8	**JIRA: PROMISE-11** • Alignment to ETSI NVF Specs

[PROMISE] OPNFV, “Promise” requirements project, [Online]. Available at https://wiki.opnfv.org/promise

[BLAZAR] OpenStack Blazar Project, [Online]. Available at https://wiki.openstack.org/wiki/Blazar

[PROMOSS] Promise reference implementation, [Online]. Available at https://github.com/opnfv/promise

[YANGFO] Yangforge Project, [Online]. Available at https://github.com/opnfv/yangforge

[NFVMAN] ETSI GS NFV MAN 001, “Network Function Virtualisation (NFV); Management and Orchestration”

[NFV003] ETSI GS NFV 003, “Network Function Virtualisation (NFV); Terminology for Main Concepts in NFV”

[NFVIFA010] ETSI GS NFV IFA 010, “Network Function Virtualisation (NFV); Management and Orchestration; Functional Requirements Specification”

[NFVIFA005] ETSI GS NFV IFA 005, “Network Function Virtualisation (NFV); Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification”

[NFVIFA006] ETSI GS NFV IFA 006 “Network Function Virtualisation (NFV); Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification”

[ETSINFV] ETSI NFV, [Online]. Available at http://www.etsi.org/technologies-clusters/technologies/nfv

A

Administrator, **3**

C

Consumer, **3**

N

NFV, **3**
NFVI, **3**
NFVO, **3**

P

Physical resource, **3**

R

Resource zone, **3**

V

VIM, **3**
Virtual Machine (VM), **3**
Virtual network, **3**
Virtual resource, **3**
Virtual Storage, **3**
VNF, **3**
VNFM, **3**