Promise User Guide

Release brahmaputra.1.0 (9921dd3)

OPNFV

February 25, 2016

CHAPTER

ONE

PROMISE CAPABILITIES AND USAGE

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

The following are the key features provided by this module:

- Capacity Management
- Reservation Management
- Allocation Management

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

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

1.1 Promise usage

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

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

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

1.2 Promise feature and API usage guidelines and examples

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

Note 1: In contrast to ETSI NFV specifications and the detailed interface specification in Section 7, the Promise shim-layer implementation does not distinguish intent interfaces per resource type, i.e. the various capacity, reservations, etc.

[1] YangForge framework, http://github.com/opnfv/yangforge

[2] Promise requirement document, http://http://artifacts.opnfv.org/promise/docs/requirements/index.html

operations have different endpoints for each domain such as compute, storage, and network. The current shim-layer implementation does not separate the endpoints for performing the various operations.

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

1.2.1 Reservation management

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

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

create-reservation

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

The operation takes the following input parameters:

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

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

update-reservation

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

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

cancel-reservation

This operation is used to cancel an existing reservation.

The operation takes the following input parameter:

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

query-reservation

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

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

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

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

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

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

1.2.2 Allocation management

create-instance

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

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

The operation takes the following input parameters:

- name (mandatory): Assigned name for the instance to be created
- image (mandatory): the image to be booted in the new instance
- flavor (mandatory): the flavor of the requested server instance
- networks: the list of network uuids of the requested server instance
- provider-id: identifier of the provider where the instance shall be created
- reservation-id: identifier of a resource reservation the *create-instance* is issued against

The Brahamputra implementation of Promise has the following limitations:

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

destroy-instance

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

The operation takes the following input parameter:

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

query-resource-collection

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

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

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

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

1.2.3 Capacity management

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

increase/decrease-capacity

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

This feature can be used in different ways, like

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

The operation takes the following input parameters:

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

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

query-capacity

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

The current implementation supports the following filter criteria:

- time window: returns reservations matching the specified window
- window scope: if set to 'exclusive', only reservations with start AND end time within the time window are returned. Otherwise, all reservation starting OR ending in the time windows are returned.
- metric: query for one of the following capacity metrics:
  - 'total': resource pools
  - 'reserved': reserved resources
  - 'usage': resource allocations

  - 'available': remaining capacity, i.e. neither reserved nor allocated

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

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

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

1.2.4 (Multi-)provider management

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

add-provider

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

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

The operation takes the following input parameters:

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

remove-provider

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